Algorithms and Hardness for Approximating the Diameter of a Graph

by

Nicole Spence Wein

B.S., Harvey Mudd College (2015)
M.S., Stanford University (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
August 20, 2021

Certified by .....
Virginia Vassilevska Williams
Steven and Renee Finn Career Development Associate Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Algorithms and Hardness for Approximating the Diameter of a Graph

by

Nicole Spence Wein

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The *diameter* of a graph is one of the most basic and fundamental attributes of a graph. It is defined as the distance between the pair of vertices that is the farthest apart. The diameter of a graph is a meaningful parameter for many applications such as distributed computation and social networks. We seek fast algorithms for computing the diameter of a graph. This is one of the central problems in the area of *fine-grained complexity*.

The naive algorithm for computing the diameter of a graph is to find the distance between *all* pairs of vertices and return the largest one. Interestingly, no better algorithm is known. Furthermore, there is evidence from fine-grained complexity, that no *subquadratic time* algorithm exists for computing the diameter of a graph exactly. In particular, such an algorithm would falsify the *Strong Exponential Time Hypothesis (SETH)*. For applications with very large graphs, even quadratic time can be prohibitively slow. Thus, we turn to *approximation* algorithms with faster running times. Prior work establishes a hierarchy of algorithms that trade-off time and accuracy, as well as a single lower bound conditioned on SETH.

Our first main contribution is the development of a hierarchy of conditional lower bounds under SETH for approximating the diameter of a graph, establishing a time vs. accuracy trade-off. These lower bounds show that several of the known algorithms on the trade-off curve are conditionally tight.

Second, we study the approximability of the diameter of a graph in a variety of natural settings, such as when the graph is changing over time, or when we only care about the distances between particular subsets of vertices, or when we only care about one-way distances in a directed graph. For these variants, we develop both approximation algorithms and conditional lower bounds that are often tight.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Steven and Renee Finn Career Development Associate Professor

Acknowledgments

First and foremost, I would like to thank my advisor Virginia Vassilevska Williams. I feel extremely lucky to have been her student for the past 6 years. She has shaped me as a researcher by giving me many interesting problems to think about, sharing her research vision with me, and being available to discuss research ideas frequently. She has prepared me for a future in this field by giving me career advice and introducing me to many collaborators. I can't imagine where I would be without her.

I would also like to thank her and Ryan Williams for making my PhD experience fun by organizing weekly music nights, giving me exceptionally generous amounts of food, and just being friends with their students. I will always remember when our music night band performed parody songs at FOCS 2019, with Dylan McKay and Ryan on the guitar, Virginia and me on the kazoo, and Josh Alman on the sandpaper.

I would also like to thank all of the other mentors I have had over the years. In chronological order: Ran Libeskind-Hadas sparked my interest in algorithms as an undergraduate through his enthusiastic teaching and genuine concern for his students. He taught me three courses ranging from introductory computer science to advanced algorithms, and took me on as a researcher the summer after my first year of college. If I can pass on only a fraction of his commitment and care towards my future students, that will be a success. Glencora Borradaile took me on as an REU student the summer before my last year of college, which exposed me to graph algorithms research and set me up to apply for a PhD. Moses Charikar and Tim Roughgarden helped me explore my interests by being my research rotation advisors during the first year of my PhD at Stanford. Tim introduced me to the problem that became my first theory paper. Shay Solomon took me under his wing, generously shared his ideas with me, and believed in my ability as a researcher before I had any publications. Erik Demaine and Monika Henzinger supported my career through collaboration and letters of recommendation. Erik Demaine hosted a number of fruitful open problem sessions that I attended. He shaped my perspective on collaboration and I hope to take his spirit of "supercollaboration" with me into the future. Piotr Indyk was the faculty advisor to the Algorithms Office Hours initiative that I organized. Lastly, I would like to thank Erik Demaine and Piotr Indyk for being on my thesis committee.

I would also like to thank all of my wonderful collaborators. This dissertation would not exist without them. They came up with beautiful ideas, introduced me to many of my favorite problems, went through the ups and downs of research with me, and were simply a pleasure to be around. In alphabetical order they are: Oswin Aichholzer, Shyan Akmal, Bertie Ancona, Arturs Backurs, Thiago Bergamaschi, Aaron Berger, Mina Dalirrooyfard, Erik Demaine, Jacob Fox, Monika Henzinger, Ce Jin, Matias Korman, William Kuszmaul, Kevin Lu, Anna Lubiw, Jayson Lynch, Zuzana Masárová, Adir Morgan, Krzysztof Onak, Adam Polak, Maximilian Probst Gutenberg, Liam Roditty, Tim Roughgarden, Mikhail Rudoy, Baruch Schieber, Gilad Segal, C. Seshadhri, Shay Solomon, Hsin-Hao Su, Jonathan Tidor, Virginia Vassilevska Williams, Nikhil Vyas, Fan Wei, Yinzhan Xu, Zixuan Xu, and Yuancheng Yu. I would also like to thank the many others who I worked with on research without having a publication together.

Thank you to the entire MIT and Stanford theory groups for being fun and vibrant communities. Thank you especially to the MIT theory group for warmly welcoming the Williams cohort when we moved from Stanford to MIT in the middle of the year. I already feel nostalgic for the days of coming into the Stata Center and being surrounded by so many awesome people to talk to about research and non-research alike. We had many memorable times together including going on theory retreats to New Hampshire, having weekly theory lunches, playing games and doing crosswords at theory tea, and playing sports in the hallways of Stata. I really felt like part of a community here, and I will miss you all.

I would like to thank the theory group administrators at both MIT and Stanford, especially Rebecca Yadegar, Debbie Goodwin, and Joanne Hanley. They made things run smoothly behind the scenes and organized fun community events like Corn Fest and TOC Day of Arts. They were a warm presence around the office, and even gave me a plant for my office and named a fish after me.

I would like to acknowledge all of the free food lying around the computer science buildings at both MIT and Stanford for being the basis of my diet.

Beyond computer science, I would like to thank the ballet and tap dancing teachers I had during my PhD including Marny Trounson, Julia Boynton, Roseann Ridings, and Marcus Schulkind, and the student dance groups I was in including Cardinal Ballet Company, Stanford Tap Th@t, and Harvard Ballet Company. Dance has been a big part of my life, and during my PhD it has offered the type of instant gratification that doesn't often come from research.

I would like to thank my parents Anne and Larry, for raising me with love and always encouraging me to pursue whatever I am interested in. Thank you to my parents, my brother Alex, my sister Natasha, and my sister-in-law Melissa Zhang for providing lots of love, laughs, joy, and creative celebrations. Thank you to my nana, Jan Spence, for providing encouragement from afar.

Thank you to my partner Jirka Hladiš for just being the absolute best for the past 10 years. He moved with me to Stanford to start my PhD, then to MIT to finish my PhD, and now to Rutgers for my postdoc.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

One of the most basic questions one can ask about a graph is "what is the distance between the pair of vertices that is the farthest apart?" This quantity is known as the *diameter* of the graph. The diameter of a graph is a fundamental and meaningful parameter for many applications.

For example, the diameter of a distributed network measures how quickly information can spread across the entire network, so low diameter distributed networks are desired in practice (see e.g. [KMX+04]). In particular, if D is the diameter of the network, then in D rounds of communication, every vertex can receive information from every other vertex in the graph. In fact, the complexity of algorithms for distributed networks is often expressed in terms of the diameter.

Relatedly, *hop-constrained network design* seeks to compute low-cost low-diameter sub-structures of graphs that connect all terminals (see e.g. [HHZ21]). The low-diameter condition facilitates faster and more reliable communication.

In addition to distributed algorithms, there are a number of other models of computation where solving distance-based graph problems is easier when the diameter is small, for example in parallel algorithms, streaming algorithms, and dynamic algorithms. In these settings, algorithms for distance-based problems often proceed by adding a set of edges called a *hopset* that decreases the diameter of the graph without distorting the distances too much (see e.g. the survey [EN+20]).

Another example of the applicability of diameter is for social networks, where the diameter measures the maximum number of *degrees of separation* between any two people. There has been great popular and academic interest in the idea of the "small world phenomenon" or "six degrees of separation" which hypothesizes that the global friendship network has diameter 6. This line of inquiry has expanded towards asking whether other types of graphs besides social networks also exhibit a "small world" structure, for instance linguistic networks [VSPB19, CM09], road networks [MC11], and the graph of the internet [AJB99].

As a consequence of the applicability of diameter, there are a number of papers that develop algorithms and heuristics for computing the diameter of real-world

graphs [CPPU16, BCH+15, LWCW16, MLH09, TK11, CGLM12, CPPU20].

1.1.1 Fine-Grained Complexity

Computing the diameter of a graph is one of the central questions in the area of *fine-grained complexity*. In contrast to the standard theory of NP-hardness, where the goal is to distinguish between problems that are solvable in polynomial time and those that are likely not, fine-grained complexity seeks to distinguish between different running times at a more fine-grained level. Oftentimes, these running times fall within polynomial time, such as distinguishing linear time versus quadratic time versus cubic time. These distinctions are especially important for modern applications with very large amounts of data where, for example, even an algorithm that runs in quadratic time could be infeasible.

In the standard theory of NP-hardness, proofs of hardness are based on the assumption that $\mathsf{P} \neq \mathsf{NP}$. In contrast, in fine-grained complexity we wish to prove much stronger hardness results, so the hardness assumptions that we use are stronger than $\mathsf{P} \neq \mathsf{NP}$. Several different assumptions have been used in fine-grained complexity, but the most standard assumptions are based on hardness of the following three problems: 3-SUM, CNF-SAT, and All-Pairs Shortest Paths (APSP). In this dissertation, we will obtain hardness proofs conditioned on the hardness of CNF-SAT; specifically we will assume the *Strong Exponential Time Hypothesis* (SETH), which is defined in Chapter 2.

Since the above hardness assumptions are plausible, but less established than $\mathsf{P} \neq \mathsf{NP}$, the goal of fine-grained complexity can be rephrased as understanding *why* we are stuck on long-standing problems, that is, asking whether we can blame any of the above three problems for the fact that we are stuck. By reducing from the above three problems to many problems of interest, fine-grained complexity has shown that we are stuck on many seemingly very different problems for the same reason.

Among the problems that fine-grained complexity has shown hardness for are some of the most basic problems in P such as string problems like Edit Distance, geometric problems like Colinearity Testing, and of course, distance problems like computing the diameter of a graph. Each of these problems can be solved using standard approaches such as a greedy algorithm or dynamic programming, but no better algorithm has been discovered despite many efforts. Fine-grained complexity has shown that, under appropriate assumptions, there do not exist significantly better algorithms than these basic standard approaches, for any of these problems [BI15, GO95, RV13].

1.1.2 Computing the Diameter of a Graph

We let "Diameter" denote the problem of computing the diameter of a graph. When faced with this problem, the naive approach is to simply compute the distance between *all* pairs of vertices and output the largest one. One might expect there to exist a faster algorithm for Diameter using some clever trick — after all, the diameter is only a single number, whereas the distance between *all* pairs of vertices consists of n^2

numbers. However, amazingly, the fastest known algorithm for Diameter is the naive algorithm!

Moreover, there is evidence from fine-grained complexity suggesting that there does *not* exist a faster algorithm [RV13] for sparse graphs. In particular, All-Pairs Shortest Paths (APSP) on a sparse graph with $\tilde{O}(n)$ edges takes time $\tilde{O}(n^2)$, and SETH implies that there is no significantly faster algorithm. For dense graphs with $\Omega(n^2)$ edges, computing APSP takes time $\tilde{O}(n^3)$ and it is a major open problem to determine whether or not there is a barrier from fine-grained complexity to preclude a faster algorithm.

Most real-world graphs of practical interest are *very large sparse graphs*. For such graphs, an algorithm that runs in quadratic time may be prohibitively slow. That is, computing the diameter of such graphs may be infeasible. However, we would still like to obtain meaningful information about the diameter of a graph using only *subquadratic* time computation. To do so, we need to relax the requirement that our algorithm solves Diameter *exactly*. This motivates the following question:

For sparse graphs, is there a subquadratic-time algorithm that produces an approximation for Diameter?

The answer to this question is "yes". For example, a very simple folklore algorithm gives a multiplicative 2-approximation for Diameter in near-linear time. The algorithm simply picks an arbitrary vertex v, runs Dijkstra's algorithm from v (in both directions if the graph is directed), and returns the largest distance found. By the triangle inequality, the value returned is at least half of the true diameter. More involved techniques provide a series of algorithms that provide a time vs. accuracy trade-off. These results raise the following question:

Does there exist a single algorithm for Diameter that achieves the best possible running time and approximation factor simultaneously? Or is there an inherent time vs. accuracy trade-off?

The answer to this question is that a time vs. accuracy trade-off *is* inherent, assuming SETH. A central focus of this dissertation is to precisely understand this time vs. accuracy trade-off.

Another central focus of this dissertation is to address the approximability of Diameter in a variety of natural settings, such as when the graph is changing over time, or when we only care about the distances between particular subsets of vertices, or when we only care about one-way distances in a directed graph.

1.2 Summary of Results

In this dissertation, we provide both algorithms and conditional lower bounds for approximating Diameter in a variety of different settings. Part I focuses obtaining conditional lower bounds that establish the aforementioned time vs. accuracy trade-off for approximating Diameter in the standard setting. Part II introduces a number of different natural settings to study Diameter, and contains both algorithms and conditional lower bounds for approximating Diameter in these settings.

Part I: Hardness of Approximating Diameter

In Chapter 2, we give a detailed survey of prior work on the problem of approximating Diameter.

Diameter exhibits a time vs. accuracy trade-off. In Chapter 3, we present the first time vs. accuracy trade-off conditional lower bounds for approximating Diameter (for directed/undirected and weighted/unweighted graphs). Our conditional lower bounds are based on SETH. Prior work established a time vs. accuracy trade-off of *upper* bounds [ACIM99, RV13, CLR$^+$14, CGR16] as well as a single lower bound [RV13], and together with this work, our results show that under SETH, the existence of a time vs. accuracy trade-off is inherent. We note that our results do not give a tight characterization of the entire trade-off; this question is further addressed in Chapter 4. However, our results do show tightness for a particular approximation algorithm on the trade-off curve.

As a starting point, we use a construction of a conditional lower bound from [BRS$^+$18] for a variant of Diameter, called *ST-Diameter*, for which approximation is potentially much harder than Diameter. In ST-Diameter, S and T are subsets of vertices and the goal is to find the largest distance between a vertex in S and a vertex in T. That paper proves a hierarchy of time vs. accuracy trade-off conditional lower bounds for ST-Diameter.

To give some context relevant to several chapters of this dissertation, we briefly outline the ideas from the ST-Diameter conditional lower bound of [BRS$^+$18]. Many conditional lower bounds based on SETH (including ours) use an intermediate problem, called the *orthogonal vectors* (OV) problem, where we are given a set of binary vectors and the goal is to determine whether there exists a pair of vectors that are orthogonal. It is known that SETH implies hardness for OV, as well as hardness for a generalization of OV called k-OV, where the goal is to find a set of k orthogonal vectors. The key approach towards getting time vs. accuracy trade-off conditional lower bounds for ST-Diameter is to construct a reduction from k-OV to ST-Diameter, rather than just a reduction from OV. Each value of k produces a different value on time vs. accuracy trade-off. In particular, increasing k corresponds to algorithms with better running times but worse approximation factors.

The obstacle in extending these conditional lower bounds for ST-Diameter to conditional lower bounds for Diameter is that we need to ensure that, if the k-OV instance has no solution, then *all* pairs of vertices have small enough distance. For ST-Diameter, it sufficed for only the pairs $s \in S, t \in T$ have small distance. In particular, in the construction for ST-Diameter, pairs of vertices $s, s' \in S$ can be very far from one another. The challenge is to add extra gadgetry to make such pairs close when the k-OV instance has no solution while maintaining that the diameter is large when the k-OV instance has a solution.

Chapter 3 is based on the previously published paper "Towards Tight Approximation Bounds for Graph Diameter and Eccentricities" [BRS$^+$18] with Arturs Backurs, Liam Roditty, Gilad Segal, and Virginia Vassilevska Williams, which appeared in STOC 2018 and SICOMP.

Tightening the time vs. accuracy trade-off for Diameter in directed graphs. In Chapter 4, we build upon the conditional lower bounds from Chapter 3. In Chapter 3, we only established one step of the time vs. accuracy trade-off of conditional lower bounds for Diameter (though we established an infinite collection of bounds for ST-Diameter). Another step of the time vs. accuracy trade-off was established in [Bon21b] for directed weighted graphs, and in Chapter 4 we establish an infinite collection of time vs. accuracy trade-off conditional lower bounds for Diameter for directed unweighted graphs. This work answers a question that was posed as Open Question 2.2 in the survey [RV19] by Rubinstein and Vassilevska W., and has also been explicitly asked in several other works [BRS$^+$18, Bon21b]. Our work addresses the directed case, and the undirected case has recently been addressed in subsequent work [Bon21a, DLV21].

Our collection of conditional lower bounds is tight with all three of the known algorithms for directed graphs. Known results for this problem are discussed in more detail in Chapter 2 and depicted in Figure 2-1. In particular, our result shows that a very simple folklore algorithm that gives a 2-approximation for Diameter in near-linear time is *tight* for directed graphs. That is, one cannot achieve a better-than-2-approximation in near-linear time for directed graphs, assuming SETH. This folklore algorithm consists of simply performing a single-source shortest paths algorithm both into and out of an arbitrary vertex and returning the largest distance found. A simple application of the triangle inequality shows that this algorithm gives a 2-approximation for Diameter. It is interesting to know that such a simple algorithm is conditionally optimal.

We achieve this result by refining the aforementioned reduction from k-OV to ST-Diameter from [BRS$^+$18] by adding extra gadgetry to have more precise control over the distances in the constructed graph.

Chapter 4 is based on the previously published paper "Tight Conditional Lower Bounds for Approximating Diameter in Directed Graphs" [DW21] with Mina Dalirrooyfard, which appeared in STOC 2021. Concurrent and independent work by Ray Li [Li21] proved the same result which also appeared in STOC 2021.

Part II: Approximating the Diameter of a Graph in Various Settings

Bichromatic Diameter, ST-Diameter, and Subset Diameter. In Chapter 5, we study several variants of Diameter where we only care about distances between specified vertices. One such problem is ST-Diameter, which was introduced above. Another problem is *Bichromatic Diameter*, which is the variant of ST-Diameter where every vertex belongs to exactly one of S or T. A third related problem is *Subset Diameter*, which is the variant of ST-Diameter where $S = T$. There are many natural ways to interpret these variants of Diameter. For example, for Bichromatic Diameter, each vertex could represent either a school or a child's home, and the Bichromatic Diameter of the graph is the maximum distance that a child needs to travel to get to school.

All three of these variants exhibit the same phenomenon as Diameter where solving APSP solves these problems exactly, but SETH implies that there is no truly subquadratic time algorithm to solve them exactly, so we seek subquadratic time approximation algorithms.

We provide a comprehensive study of the approximability of ST-Diameter, Bichromatic Diameter, and Subset Diameter for directed/undirected, weighted/unweighted graphs. We give algorithms and conditional lower bounds under SETH and all of them are *tight* except for one.

For undirected graphs, Bichromatic Diameter exhibits a similar time vs. accuracy trade-off as Diameter and ST-Diameter, but interestingly, the optimal approximation factor is inherently different for each problem. For example, in $\tilde{O}(m^{3/2})$ time, Diameter admits a 3/2-approximation, Bichromatic Diameter admits a 5/3-approximation, and ST-Diameter admits a 2-approximation, and all of these approximation factors are tight under SETH.

For directed graphs however, the picture is completely different. For ST-Diameter in directed graphs, under SETH one cannot achieve *any* finite approximation in subquadratic time. On the other hand, for Bichromatic Diameter in directed graphs, there is a subquadratic-time approximation algorithm. Bichromatic Diameter in directed graphs is the only variant where our results are not completely tight: we provide an algorithm for with tight approximation factor but the running time is not tight.

Subset Diameter behaves completely differently from any of the other variants: it does not exhibit a time vs. accuracy trade-off and instead exhibits a sharp threshold behavior for both directed and undirected graphs: there exists a near-linear time 2-approximation algorithm, but SETH implies that getting any approximation factor better than 2 requires quadratic time.

We also obtain parameterized approximation algorithms and conditional lower bounds for Bichromatic Diameter, parameterized by the size of the *boundary* between S and T.

Chapter 5 is based on the previously published paper "Tight Approximation Algorithms for Bichromatic Graph Diameter and Related Problems" [DVVW19] with Mina Dalirrooyfard, Virginia Vassilevska Williams, and Nikhil Vyas, which appeared in ICALP 2019. This chapter also contains some content from the previously published paper "Towards Tight Approximation Bounds for Graph Diameter and Eccentricities" [BRS+18] with Arturs Backurs, Liam Roditty, Gilad Segal, and Virginia Vassilevska Williams, which appeared in STOC 2018 and SICOMP.

Min-Diameter. In Chapter 6, we study a variant of Diameter where the graph is directed and we only care about one-way distances. Specifically, given a directed graph, the *min-distance* between a pair of vertices u, v is defined as minimum between the $u \to v$ distance and the $v \to u$ distance. Min-distances are appropriate for situations where it suffices to traverse a path in *either* direction. For example, suppose two parties wish to share a secret and it suffices for either party A to communicate to party B or vice versa, or suppose a person needs health services and either they

can go to the doctor or the doctor can come to them.

Given the notion of min-distances, the *min-diameter* is accordingly defined as the maximum over all min-distances in the graph. Min-Diameter exhibits the same phenomenon as Diameter where solving APSP computes the min-diameter exactly, but SETH implies that there is no truly subquadratic time exact algorithm for Min-Diameter, so we seek subquadratic time approximation algorithms.

A constant-factor approximation algorithm for Min-Diameter was previously known only for DAGs [AVW16]. We obtain the first constant-factor approximation for Min-Diameter for general (weighted) graphs. We also obtain a hierarchy of algorithms that give a time vs. accuracy trade-off (which is not tight with the known conditional lower bounds).

The most intriguing technical aspect of min-distances as compared to standard distance measures, is that min-distances do not obey the *triangle inequality*. For example, consider a vertex v with directed edges to a vertex u and a vertex w. See Figure 1.2.

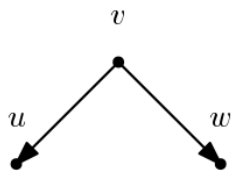

Figure 1-1: Min-distances do not obey the triangle inequality.

The min-distance between v and both u and w is 1, but there is no directed path between u and w, so the min-distance between them is infinite. Not being able to apply the triangle inequality is a significant barrier to developing approximation algorithms for Min-Diameter. For every other variant of Diameter, including all of the Bichromatic, ST, and Subset variants from Chapter 5, as well as the standard variant on directed/undirected weighted/unweighted graphs, all known approximation algorithms rely heavily on the triangle inequality. For example, as mentioned previously, the folklore 2-approximation algorithm for the standard Diameter problem simply returns the largest distance from an arbitrary vertex, and the analysis follows from a single application of the triangle inequality. For Min-Diameter, there is no algorithm analogous to this folklore algorithm, and it was previously completely unclear whether there exists *any* algorithm that provides a constant-factor approximation.

One of the key ideas in our approximation algorithm for Min-Diameter is as follows. Because there already exists an algorithm for Min-Diameter on DAGs, it would be ideal to get a direct reduction from Min-Diameter on general graphs to Min-Diameter on DAGs, however it is unclear how to do this. Instead, we recognize that it suffices to define a *DAG-like* structure on general graphs. In particular, we define an ordering of the vertices that shares some key properties with the topological ordering of a DAG. In particular, we pick a vertex v and partition the rest of the vertices as follows: vertices whose distance to v is smaller than its distance from v are placed before v in the ordering, and the rest are placed after v. Applying this partition recursively gives an ordering over all of the vertices. This ordering provides enough

information to allow us to wisely choose which distances in the graph to compute exactly so that we can approximate Min-Diameter within a constant factor.

Chapter 6 is based on the previously published paper "Approximation Algorithms for Min-Distance Problems" [DVV+19] with Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, Yinzhan Xu, and Yuancheng Yu, which appeared in ICALP 2019.

Dynamic Diameter. In Chapter 7, we study the *dynamic* variant of Diameter where the graph is changing over time. In general, dynamic graph algorithms are specified as follows. At each time step an adversary either inserts an edge into the graph or deletes an edge from the graph, and after every edge insertion/deletion the algorithm must output the quantity of interest. The naive solution is to recompute the output from scratch after every edge insertion/deletion. Accordingly, the goal of dynamic algorithms is to store a data structure that allows the output to be updated faster than recomputation from scratch.

There are two main settings for dynamic algorithms. A *fully* dynamic algorithm supports an intermingled sequence of edge insertions and deletions, while a *partially* dynamic algorithm supports either only edge insertions or only edge deletions. The ideal of a partially dynamic algorithm is that its total running time matches the running time of the best static (non-dynamic) algorithm for the problem.

We study the problem of approximating Diameter in both the partially and fully dynamic settings, providing both algorithms and conditional lower bounds. For the partially dynamic setting, we obtain a 3/2-approximation algorithm for Diameter whose total running time matches the best-known static algorithm (which is optimal under SETH). For the fully dynamic setting, the picture is completely different. We show that under SETH, there is no such fully dynamic algorithm, and in particular there is no algorithm that is significantly better than recomputing the answer from scratch after every insertion/deletion or maintaining dynamic APSP.

Chapter 7 is based on the previously published paper "Algorithms and Hardness for Diameter in Dynamic Graphs" [AHR+19] with Bertie Ancona, Monika Henzinger, Liam Roditty, and Virginia Vassilevska Williams, which appeared in ICALP 2019.

Open Problems. Lastly, in Chapter 8 we provide a list of open problems pertaining to all chapters of this dissertation.

1.3 Related Problems

Most of the previously published papers whose content is included in this dissertation also study two graph parameters that are strongly related to the diameter: *radius* and *eccentricities*. The eccentricity of a vertex v is defined as the largest distance from v to another vertex. The diameter and radius are the extremal eccentricities — the diameter is the largest eccentricity and the radius is the smallest eccentricity. (To clarify the computational problems implicit in these parameters, the output of the eccentricities problem is the eccentricity of all n vertices, while the output of Diameter

and Radius are only a single distance.) Many of the techniques for both algorithms and conditional lower bounds for Diameter also apply to Radius and Eccentricities, and vice versa. In this dissertation, we focus only on Diameter, in the interest of including a wider breadth of different settings where one can study these problems.

1.4 Preliminaries

Notation

Let $G = (V, E)$ be a weighted or unweighted, directed or undirected graph, where $|V| = n$ and $|E| = m$. For every $u, v \in V$ let $d_G(u, v)$ be the length of the shortest path from u to v. When the graph G is clear from the context we omit the subscript G. A distance $d(u, v)$ is considered to be ∞ if v is not reachable from u.

The *diameter* D of a graph is defined as $\max_{u,v \in V} d(u, v)$. Given $S, T \subseteq V$, the *ST-diameter* D_{ST} of a graph is defined as $\max_{s \in S, t \in T} d(s, t)$. If $S \cup T = V$ and $S \cap T = \emptyset$, then the ST-diameter is called the *bichromatic* diameter. If $S = T$, then the ST-diameter is called the *subset diameter*. For vertices u, v, their *min-distance* $d_{min}(u, v)$ is defined as $\min\{d(u, v), d(v, u)\}$. The *min-diameter* D_{min} of a graph is defined as $\max_{u,v \in V} d_{min}(u, v)$.

For $\alpha \geq 1$, an α-approximation algorithm for the diameter D of a graph is defined as an algorithm that returns a value D' such that $D/\alpha \leq D' \leq D$. We note that all of our approximation algorithms also return a *witness*, that is, a pair of vertices of distance at least D'.

For an algorithm with input size n we use *with high probability* to denote the probability $> 1 - 1/n^c$ for all constants c. We say some quantity is *poly*(n) to mean it is $O(n^c)$ for some fixed constant c. We use $\tilde{O}$ notation to hide polylogarithmic factors.

SETH and k-Orthogonal Vectors

Let $k \geq 2$. The *k-Orthogonal Vectors Problem (k-OV)* is as follows: Given a set S of n vectors in $\{0, 1\}^d$, determine whether there exist $v_1, \ldots, v_k \in S$ so that their generalized inner product is 0, i.e. $\sum_{i=1}^{d} \prod_{j=1}^{k} v_j[i] = 0$, where $v_j[i]$ is the ith bit of the vector v_j. If $k = 2$, we simply say OV instead of 2-OV.

Our conditional lower bounds are based on the *k-OV Hypothesis*, defined as follows:

Hypothesis 1 (k-OV Hypothesis)**.** *For all constants $k \geq 2$ and all $\epsilon > 0$, there exists $c_k > 0$ such that k-OV on $d = c_k \log n$ bit vectors requires $n^{k-\epsilon}$ time on a word-RAM with $O(\log n)$ bit words.*

Williams [Wil05] showed that if the k-OV Hypothesis is false, then CNF-SAT on formulas with N variables and m clauses can be solved in $2^{N(1-\varepsilon/k)}\text{poly}\,(m)$ time. In particular, such an algorithm would contradict the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, and Zane [IPZ01] (the name SETH was introduced by Calabro, Impagliazzo, and Paturi[CIP09]) which is the following: For

every $\varepsilon > 0$ there is a K such that K-SAT on N variables cannot be solved in $2^{(1-\varepsilon)N}\text{poly}\,(N)$ time (say, on a word-RAM with $O(\log N)$ bit words). This means that SETH implies the k-OV Hypothesis.

A main motivation behind SETH is that despite decades of research, the best upper bounds for K-SAT on N' variables and M clauses remain of the form $2^{N'(1-c/K)}\text{poly}\,(M)$ for constant c (see e.g. [Hir98, PPSZ05, Sch99]). The best algorithms for the k-OV problem for any constant $k \geq 2$ on N vectors and dimension $c \log N$ run in time $N^{k-1/O(\log c)}$ (Abboud, Williams, and Yu [AWY15] and Chan and Williams [CW16]).

Part I

Hardness of Approximating the Diameter of a Graph

Chapter 2

Prior Work on Approximating the Diameter of a Graph

The fastest known algorithms [Wil18, PR05, Pet04] for Diameter in n-vertex m-edge graphs are only slightly faster (by $n^{o(1)}$ factors) than the simple $\tilde{O}(mn)$ time algorithm of running Dijkstra's algorithm from every vertex and then taking the largest distance. For dense graphs with small integer weights there are improved algorithms [Sei95, Zwi02, CGS15] using fast matrix multiplication, but these algorithms are not faster than mn for sparser graphs or graphs with large weights. Furthermore, under the Strong Exponential Time Hypothesis (SETH), there is no $O(m^{2-\varepsilon})$ time algorithm for any constant $\varepsilon > 0$ for Diameter even in unweighted, undirected graphs [RV13]. Since quadratic time can be prohibitively slow on very large graphs, finding efficient *approximation* algorithms for Diameter is desirable.

The following approximation and hardness of approximation results are shown in Figure 2-1.

A folklore $\tilde{O}(m)$ time algorithm gives a 2-approximation for Diameter in directed weighted graphs. The first non-trivial approximation algorithm for Diameter was by Aingworth, Chekuri, Indyk, and Motwani [ACIM99], who presented an almost-3/2-approximation[1] algorithm for Diameter in unweighted directed graphs running in $\tilde{O}(n^2 + m\sqrt{n})$ time. Roditty and Vassilevska W. [RV13] then improved the running time to $\tilde{O}(m\sqrt{n})$ in expectation. This was extended in [CLR+14] to obtain a (genuine) 3/2-approximation algorithm for Diameter in weighted directed graphs running in $\tilde{O}(\min\{m^{3/2}, mn^{2/3}\})$ time. Cairo, Grossi, and Rizzi [CGR16] generalized the above results for undirected graphs with small weights and obtained a time-accuracy trade-off: for every $k \geq 1$ they obtained an $\tilde{O}(mn^{1/(k+1)})$ time algorithm that achieves an almost-$2 - 1/2^k$-approximation.

The above 3/2-approximation algorithm in $\tilde{O}(m^{3/2})$ time is conditionally tight for sparse graphs in terms of both its approximation factor and its running time [RV13, BRS+18]. In particular, Roditty and Vassilevska W. [RV13] proved that the approximation factor is tight under SETH by showing that any $(3/2 - \varepsilon)$-approximation algorithm (for $\varepsilon > 0$) in undirected unweighted graphs requires $m^{2-o(1)}$ time[2]. Later,

[1] An almost-c-approximation of X is an estimate X' so that $X/c - O(1) \leq X' \leq X$.

[2] All of the conditional lower bounds are expressed in terms of m and hold for sparse graphs where

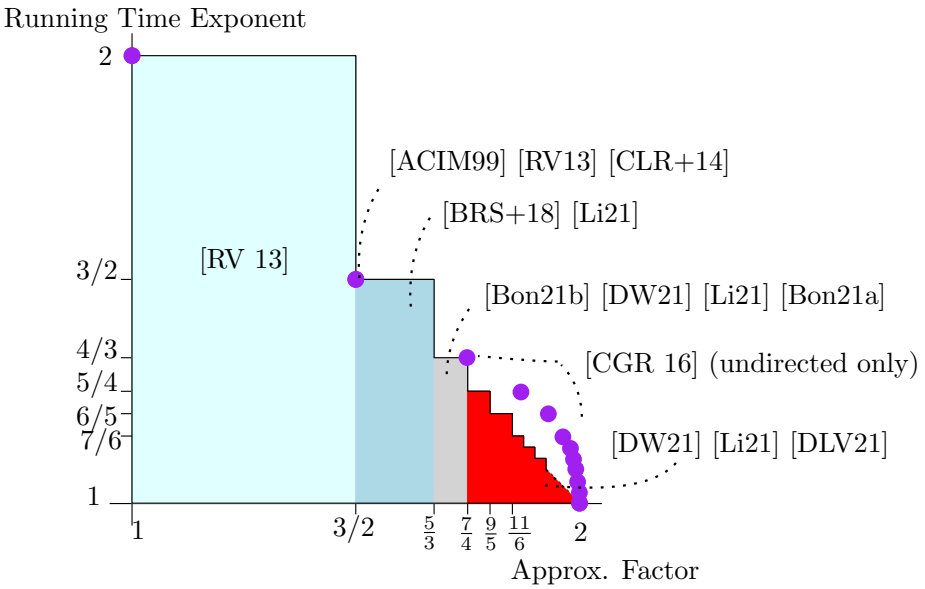

Figure 2-1: Running time exponent versus approximation factor for Diameter. Algorithms are represented by points and conditional lower bounds (under SETH) are represented by shaded boxes. All conditional lower bounds hold for undirected unweighted graphs, and all algorithms hold for weighted directed graphs, unless otherwise specified. Lists of citations often mean that the result was first proved only for graphs with a particular combination of directed/undirected, weighted/unweighted.

Backurs, Roditty, Segal, Vassilevska W., and Wein [BRS+18] proved that the running time is tight under SETH by showing that any $(8/5-\varepsilon)$-approximation algorithm in undirected unweighted graphs, or any $(5/3-\varepsilon)$-approximation in undirected weighted graphs requires $m^{3/2-o(1)}$ time, and for directed unweighted graphs any $(\frac{5k-7}{3k-4}-\varepsilon)$-approximation requires $\Omega(n^{1+1/(k-1)-o(1)})$ time. (The proofs of these bounds are included in Chapter 3.) Later, Li [Li21] (in a earlier version of his paper) improved the unweighted undirected construction of [BRS+18] to match the weighted undirected construction of [BRS+18]. That is, he showed that under SETH, any $(5/3-\varepsilon)$-approximation algorithm for Diameter in undirected unweighted graphs requires $m^{3/2-o(1)}$ time.

Recently, Bonnet [Bon21b] surpassed this 5/3 barrier for directed weighted graphs, showing another step of the time vs. accuracy trade-off, by showing that under SETH, any $(7/4-\varepsilon)$-approximation algorithm requires $m^{4/3-o(1)}$ time. Then, concurrent and independent work by Dalirrooyfard and Wein [DW21] and Li [Li21] extended this time vs. accuracy trade-off to a hierarchy of infinitely many bounds for directed unweighted graphs. In particular they showed that for any fixed integer $k \geq 2$, SETH implies that any $(\frac{2k-1}{k}-\varepsilon)$-approximation algorithm requires $m^{\frac{k}{k-1}-o(1)}$ time, for unweighted directed graphs. (The proof of these bounds are included in Chapter 4.) These bounds show in particular that the folklore near-linear time 2-approximation algorithm is tight under SETH. Li [Li21] additionally showed limitations on getting a

$m = \tilde{O}(n)$.

better deterministic SETH-based reduction assuming some nondeterministic versions of SETH.

Very recently, the above bounds for directed graphs were extended to undirected graphs. Bonnet [Bon21a] showed that under SETH, any $(7/4-\varepsilon)$-approximation algorithm requires $m^{4/3-o(1)}$ time for undirected unweighted graphs. Then, Dalirrooyfard, Li, and Vassilevska W. [DLV21] showed that SETH implies that any $(\frac{2k-1}{k}-\varepsilon)$-approximation algorithm requires $m^{\frac{k}{k-1}-o(1)}$ time for undirected unweighted graphs.

2.0.1 Techniques from Prior Work

All of our time vs. accuracy conditional lower bounds for Diameter use as a starting point the time vs. accuracy conditional lower bounds for ST-Diameter of Buckers, Roditty, Segal, Vassilevska W., and Wein [BRS+18]. This work, in turn, uses as a starting point the result of Roditty and Vassilevska W. [RV13] that any $(3/2-\varepsilon)$-approximation algorithm (for $\varepsilon > 0$) in undirected unweighted graphs requires $m^{2-o(1)}$ time under SETH. The latter result is a reduction from OV to Diameter, and the former result is a reduction from k-OV to ST-Diameter. Here, we will describe the reduction from OV to Diameter as well as the reduction from 3-OV to ST-Diameter. These reductions will give intuition for our constructions, especially our reduction from k-OV to Diameter in directed graphs in Chapter 4.

Reduction from OV to Diameter

The construction of [RV13] is for the standard Diameter problem on undirected unweighted graphs, but it implicitly gives a construction for the ST-Diameter problem with approximation factor 2 instead of 3/2. Here, we will describe their construction for ST-Diameter, and then note how it can be extended to Diameter. The construction for ST-Diameter is shown in Figure 2-2.

We reduce from the OV problem to ST-Diameter. We are given an OV instance consisting of sets $W_0, W_1 \subseteq \{0,1\}^d$, each of size N. Our goal is to construct a graph on $\tilde{O}(N)$ vertices and edges so that if the OV instance is a NO instance then the ST-diameter is 2, and if the OV instance is a YES instance then the ST-diameter is at least 4.

We construct a layered graph G on three layers L_0, L_1, L_2 where edges only go between adjacent layers. We set $S = L_0$ and $T = L_2$ for the ST-Diameter instance. L_0 consists of one vertex for each vector $a \in W_0$, and L_2 consists of one vertex for each vector $b \in W_1$. L_1 consists of one vertex for each coordinate in $[d]$.

There is an edge between $a \in L_0$ and $x \in L_1$ if and only if a is 1 in coordinate x. There is an edge between $b \in L_2$ and $x \in L_1$ if and only if b is 1 in coordinate x. This completes the description of the construction.

If the OV instance is a NO instance, then by definition, for every pair $a \in W_0$, $b \in W_1$, there exists a coordinate x that is 1 for both a and b. Thus, there is a path of length 2 in G from $a \in L_0$ to $b \in L_2$ through $x \in L_1$. On the other hand, if the OV instance is a YES instance with orthogonal pair $a \in W_0$, $b \in W_1$, then by definition there is no coordinate such that a and b are both 1. Therefore, the distance between

$a \in L_0$ and $b \in L_2$ is more than 2, and it must be at least 4 due to the layered structure of the graph.

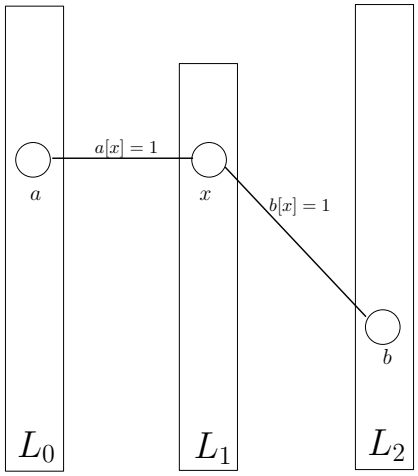

Figure 2-2: Reduction from OV to ST-Diameter. $S = L_0$ and $T = L_2$.

To extend this construction to the standard Diameter problem with approximation factor 3/2, additional vertices u and v are added along with the edge (u, v), such that u is adjacent to every vertex in L_0, v is adjacent to every vertex in L_2, and both u and v are adjacent to every vertex in L_1.

Reduction from 3-OV to ST-Diameter

The input is a 3-OV instance consisting of sets $W_0, W_1, W_2 \subseteq \{0,1\}^d$, each of size N. The goal is to construct a graph on $\tilde{O}(N^2)$ vertices and edges so that if the 3-OV instances is a NO instance, the ST-diameter is 3, and if the 3-OV instance is a YES instance, the ST-diameter is at least 7. The construction is shown in Figure 2-3.

Construct a layered graph G on four layers L_0, L_1, L_2, L_3 where the edges go only between adjacent layers. Set $S = L_0$ and $T = L_3$ for the ST-Diameter instance. L_0 consists of one vertex for each pair of vectors $a_0 \in W_0$, $a_1 \in W_1$, and L_3 consists of one vertex for each pair of vectors $b_1 \in W_1$, $b_2 \in W_2$.

Now, the goal is to define L_1, L_2, and the edges so that the ST-diameter is 3 if and only if the 3-OV instance is a NO instance. To provide some intuition, fix a pair of vertices $(a_0, a_1) \in S$, $(b_1, b_2) \in T$ and ask the question: what can we say about the vectors a_0, a_1, b_1, and b_2 in a NO instance? By definition, in a NO instance the vectors a_0, a_1, and b_2 are all 1 at some coordinate x_0. Similarly, the vectors a_0, b_1, and b_2 are all 1 at some coordinate x_1. Because the S side of the graph concerns the vectors a_0 and a_1 and the T side of the graph concerns the vectors b_1 and b_2, the conditions on a_0, a_1, b_1, and b_2 are separate according to each side of the graph. For the S side, it holds that $a_0[x_0] = a_1[x_0] = a_0[x_1] = 1$. For the T side, it holds that $b_1[x_1] = b_2[x_1] = b_2[x_0] = 1$.

This motivates a first attempt for how to define the rest of the graph. Suppose L_1 and L_2 both consist of one vertex for every pair of coordinates $x_0, x_1 \in [d]$. Add an

edge from $(a_0, a_1) \in L_0$ to $(x_0, x_1) \in L_1$ if $a_0[x_0] = a_1[x_0] = a_0[x_1] = 1$. Add an edge from $(b_1, b_2) \in L_3$ to $(x_0, x_1) \in L_2$ if $b_1[x_1] = b_2[x_1] = b_2[x_0] = 1$. Finally, add an edge from $(x_0, x_1) \in L_1$ to $(x'_0, x'_1) \in L_2$ if $x_0 = x'_0$ and $x_1 = x'_1$. While this construction has ST-diameter 3 for a NO instance of 3-OV, it does not have ST-diameter 7 for a YES instance, as is desired. In particular, suppose $a_0 \in W_0$, $a_1 \in W_1$, $a_2 \in W_2$ is an orthogonal triple. We would like the distance between $(a_0, a_1) \in L_0$ and $(a_1, a_2) \in L_3$ to be at least 7, however, with the current construction, there could be a path of length 5 from $(a_0, a_1) \in L_0$ to some $(x_0, x_1) \in L_1$, to some $(a'_0, a_1) \in L_0$, to some $(x'_0, x'_1) \in L_1$, to $(x'_0, x'_1) \in L_2$, to $(a_1, a_2) \in L_3$. The issue is that from $(a_0, a_1) \in L_0$ we can reach $(a'_0, a_1) \in L_0$ with a path of length 2 for some convenient choice of a'_0.

To overcome this issue, they also include the vector a_0 in the representation of vertices in L_1; that is, L_1 consists of one vertex for every triple $(a_0 \in W_0, x_0 \in [d], x_1 \in [d])$. There is an edge from $(a_0, a_1) \in L_0$ to $(a'_0, x_0, x_1) \in L_1$ if and only if $a = a'$ and $a_0[x_0] = a_1[x_0] = a_0[x_1] = 1$. Symmetrically, L_2 consists of one vertex for every triple $(b_2 \in W_2, x_0 \in [d], x_1 \in [d])$ and there is an edge from $(b_1, b_2) \in L_3$ to $(b'_2, x_0, x_1) \in L_2$ if and only if $b_2 = b'_2$ and $b_1[x_1] = b_2[x_1] = b_2[x_0] = 1$. Lastly, there is an edge between $(a_0, x_0, x_1) \in L_1$ and $(b_2, x'_0, x'_1) \in L_2$ if and only if $x_0 = x'_0$ and $x_1 = x'_1$. This completes the description of the construction. One can verify that this construction indeed satisfies the property that the ST-diameter is 3 for a NO instance of 3-OV, and the ST-diameter is 7 for a YES instance of 3-OV.

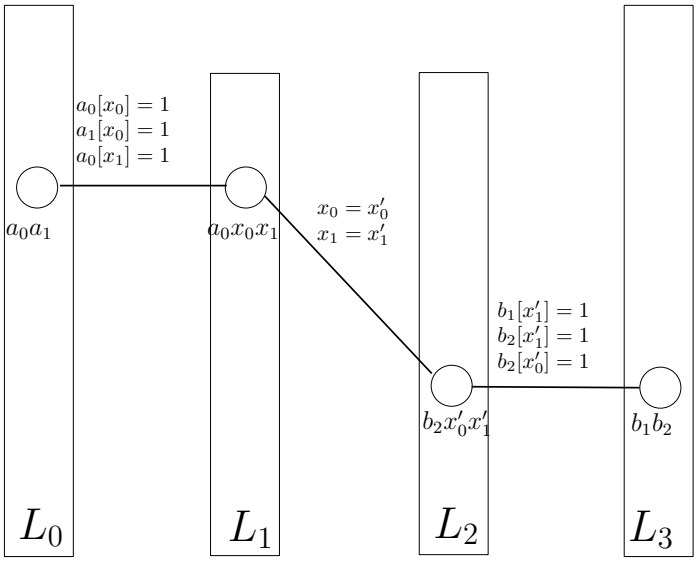

Figure 2-3: Reduction from 3-OV to ST-Diameter. $S = L_0$ and $T = L_3$.

Chapter 3

Diameter Exhibits a Time vs. Accuracy Trade-off

3.1 Results

In this chapter we will prove the following conditional lower bounds for approximating Diameter inunweighted/weighted undirected/directed graphs.

Theorem 3.1.1 (unweighted undirected)**.** *Assuming SETH, for all $\delta > 0$, any $(8/5-\delta)$-approximation algorithm for Diameter in an unweighted undirected graph on m edges requires $m^{3/2-o(1)}$ time.*

Theorem 3.1.2 (weighted undirected)**.** *Assuming SETH, for all $\delta > 0$, any $(5/3-\delta)$-approximation algorithm for Diameter in a weighted undirected graph on m edges requires $m^{3/2-o(1)}$ time.*

Theorem 3.1.3 (unweighted directed)**.** *Let $k \geq 2$ be a fixed integer. Assuming SETH, for all $\delta > 0$, any $(\frac{5k-7}{3k-4} - \delta)$-approximation algorithm for Diameter in an unweighted directed graph on m edges requires $m^{\frac{k}{k-1}-o(1)}$ time.*

We note that subsequent work [Li21] builds upon our work by obtaining a simpler construction showing that Theorem 3.1.2 also holds for unweighted undirected graphs (which subsumes Theorems 3.1.1 and 3.1.3).

3.2 Techniques

To prove our conditional lower bounds for Diameter, we use conditional lower bounds for ST-Diameter from [BRS$^+$18] as a starting point. That paper obtains a hierarchy of conditional lower bounds that establish a time vs. accuracy trade-off for ST-Diameter:

Theorem 3.2.1 ([BRS$^+$18])**.** *Let $k \geq 2$ be a fixed integer. Assuming SETH, for all $\delta > 0$, any $(\frac{3k-2}{k} - \delta)$-approximation algorithm for ST-Diameter in an unweighted undirected graph on m edges requires $m^{\frac{k}{k-1}-o(1)}$ time.*

Theorem 3.2.1 for $k = 3$ is used as a basis for the proofs of Theorems 3.1.1 and 3.1.2. Theorem 3.2.1 for general k is used as the basis for Theorem 3.1.3, our conditional lower bounds for Diameter in Chapter 4, our conditional lower bounds for Bichromatic Diameter in Chapter 5, and all subsequent work by other authors on conditional lower bounds for Diameter.

We will now state a more detailed version of Theorem 3.2.1, which will be useful for our conditional lower bounds for Diameter as well as our conditional lower bounds for Bichromatic Diameter in Chapter 5.

Theorem 3.2.2 ([BRS+18]). *Given a k-OV instance consisting of $k \geq 2$ sets $W_0, W_1, \ldots, W_{k-1} \subseteq \{0,1\}^d$, each of size N, we can in $O(kN^{k-1}d^{k-1})$ time construct an unweighted, undirected graph with $O(N^{k-1}+kN^{k-2}d^{k-1})$ vertices and $O(kN^{k-1}d^{k-1})$ edges that satisfies the following properties.*

1. *The graph consists of $k+1$ layers of vertices $S = L_0, L_1, L_2, \ldots, L_k = T$. The number of vertices in the sets is $|S| = |T| = N^{k-1}$ and $|L_1|, |L_2|, \ldots, |L_{k-1}| \leq N^{k-2}d^{k-1}$.*

2. *S consists of all tuples $(a_0, a_1, \ldots, a_{k-2})$ where for each i, $a_i \in W_i$. Similarly, T consists of all tuples $(b_1, b_2, \ldots, b_{k-1})$ where for each i, $b_i \in W_i$.*

3. *If the k-OV instance has no solution, then $d(u, v) = k$ for all $u \in S$ and $v \in T$.*

4. *If the k-OV instance has a solution $a_0, a_1, \ldots, a_{k-1}$ where for each i, $a_i \in W_i$ then if $\alpha = (a_0, \ldots a_{k-2}) \in S$ and $\beta = (a_1, \ldots, a_{k-1}) \in T$, then $d(\alpha, \beta) \geq 3k-2$.*

5. *If the k-OV instance has a solution $a_0, a_1, \ldots, a_{k-1}$ where for each i, $a_i \in W_i$ then for any tuple $(b_1, \ldots, b_{k-2})$, if $\alpha = (a_0, b_1, \ldots, b_{k-2}) \in S$ and $\beta = (a_1, \ldots, a_{k-1}) \in T$, then $d(\alpha, \beta) \geq k+2$. Symmetrically, if $\alpha = (a_0, a_1, \ldots, a_{k-2}) \in S$ and $\beta = (b_1, \ldots, b_{k-2}, a_{k-1}) \in T$, then $d(\alpha, \beta) \geq k+2$.*

6. *For all i from 1 to $k-1$, for all $v \in L_i$ there exists a vertex in L_{i-1} adjacent to v and a vertex in L_{i+1} adjacent to v. We can assume that this property holds because we can remove all vertices that do not satisfy this property from the graph and the resulting graph will still satisfy the previous three properties.*

We now state the special case of Theorem 3.2.2 where $k = 3$, which is used for Theorems 3.1.1 and 3.1.2

Theorem 3.2.3 ([BRS+18]). *Given a 3-OV instance consisting of three sets $A, B, C \subseteq \{0,1\}^d$, $|A| = |B| = |C| = N$, we can in $O(N^2d^2)$ time construct an unweighted, undirected graph with $O(N^2+Nd^2)$ vertices and $O(N^2d^2)$ edges that satisfies the following properties.*

1. *The graph consists of 4 layers of vertices S, L_1, L_2, T. The number of vertices in the sets is $|S| = |T| = N^2$ and $|L_1|, |L_2| \leq Nd^2$.*

2. *S consists of all tuples (a, b) of vertices $a \in A$ and $b \in B$. Similarly, T consists of all tuples (b, c) of vertices $b \in B$ and $c \in C$.*

3. *If the 3-OV instance has no solution, then $d(u, v) = 3$ for all $u \in S$ and $v \in T$.*

4. *If the 3-OV instance has a solution $a \in A, b \in B, c \in C$ with a, b, c orthogonal, then $d((a, b) \in S, (b, c) \in T) \geq 7$.*

5. *If the 3-OV instance has a solution $a \in A, b \in B, c \in C$ with a, b, c orthogonal, then for any $b' \in B$ we have $d((a, b) \in S, (b', c) \in T) \geq 5$ and $d((a, b') \in S, (b, c) \in T) \geq 5$.*

6. *For any vertex $u \in L_1$ there exists a vertex $s \in S$ that is adjacent to u. Similarly, for any vertex $v \in L_2$ there exists a vertex $t \in T$ that is adjacent to v. We can assume that this property holds because we can remove all vertices that do not satisfy this property from the graph and the resulting graph will still satisfy the other properties.*

3.2.1 Overview

For all of our constructions we begin with the ST-Diameter lower bound construction from Theorem 3.2.2. Here, we will provide intuition for extending the reduction from 3-OV to ST-Diameter from Theorem 3.2.3 to get a reudction from 3-OV to Diameter. In this description we treat the reduction from 3-OV to ST-Diameter as a black box.

In the ST-Diameter construction from 3-OV, if the 3-OV instance has no solution, $D_{S,T} = 3$ and if the instance has a solution $D_{S,T} \geq 7$. To adapt this construction to Diameter, we need to ensure that if the OV instance has no solution then *all* pairs of vertices have small enough distance. We begin by augmenting the ST-Diameter construction by adding a matching between S and a new set S' as well as a matching between T and a new set T'. Without any further modifications, pairs of vertices $u, v \in S \cup S'$ (or $u, v \in T \cup T'$) could be far from one another. The challenge is to add extra gadgetry to make these pairs close for NO instances while maintaining that in YES instances the distance between the diameter endpoints $s' \in S', t' \in T'$ is large. That is, for YES instances, we want a shortest path between the diameter endpoints s' and t' to contain the vertex $s \in S$ matched to s' and the vertex $t \in T$ matched to t' so that we can use use the fact that $d(s, t) \geq 7$. In other words, we do not want there to be a shortcut from s' to some vertex in S that allows us to use a path of length 3 from S to T. For example, we cannot simply create a vertex x and connect it to all vertices in $S \cup S'$ because this would introduce shortcuts from S' to S.

To achieve these goals, we augment the graph as follows. Recall that $s' \in S', t' \in T'$ are the endpoints of the diameter and let t be the vertex matched to t'. To solve the problem outlined in the above paragraph, we observe that in the YES case there are three types of vertices $s \in S$:

1. close: $d(s, t) = 3$ (property 3 of Theorem 3.2.3),

2. far: $d(s, t) \geq 7$ (property 4 of Theorem 3.2.3), and

3. intermediate: $d(s, t) \geq 5$ (property 5 of Theorem 3.2.3).

For close s, we need $d(s', s)$ to be large so that there is no shortcut from s' to t' through s. For far s, it is acceptable if $d(s', s)$ is small because $d(s, t)$ is large enough to ensure that paths from s' to t' through s are still long enough. For intermediate s, $d(s', s)$ cannot be small, but it also need not be large. To fulfill these specifications, we add a small clique (the graph is still sparse) and connect each of its vertices to only *some* of the vertices in S and/or S' according to the implications of property 5 of Theorem 3.2.2. When s is close, we ensure that $d(s', s)$ is large by requiring that a shortest path from s' to s goes from s' to the clique, uses an edge inside of the clique, and then goes from the clique to s. When s is intermediate, we ensure that $d(s', s)$ is not too small by requiring that a shortest path from s' to s goes from s' to the clique and then from the clique to s (without using an edge inside of the clique). These intermediate s are important as they allow every vertex in the clique to have an edge to some vertex in S and thus be close enough to the T side of the graph in the NO case.

3.2.2 5 vs 8 unweighted undirected construction

In this section we will prove Theorem 3.1.1.

In particular, we will use Theorem 3.2.3 to prove the following result, which implies Theorem 3.1.1.

Theorem 3.2.4. *Given a 3-OV instance, we can in $O(N^2d^2)$ time construct an unweighted, undirected graph with $O(N^2 + Nd^2)$ vertices and $O(N^2d^2)$ edges that satisfies the following two properties.*

1. *If the 3-OV instance has no solution, then for* all *pairs of vertices u and v we have $d(u, v) \leq 5$.*

2. *If the 3-OV instance has a solution, then there exists a pair of vertices u and v such that $d(u, v) \geq 8$.*

Construction of the graph. We construct a graph with the required properties by starting with the graph from Thereom 3.2.3 and adding more vertices and edges. Figure 3-1 illustrates the construction of the graph. We start by adding a set S' of N^2 vertices. S' consists of all tuples (a, b) of vertices $a \in A$ and $b \in B$. We connect every $(a, b) \in S'$ to its counterpart $(a, b) \in S$. Thus, there is a matching between the sets of vertices S and S'. We also add another set S'' of N vertices. S'' contains one vertex a for every $a \in A$. For every pair of vertices from S'' we add an edge between the vertices. Thus, the N vertices form a clique. Furthermore, for every vertex $a \in S''$ we add an edge to $(a, b) \in S$ for all $b \in B$. In total we added $N^2 + N = O(N^2)$ vertices and $\binom{N}{2} + 2N^2 = O(N^2)$ edges. We do a similar construction for the set T of vertices. We add a set T' of N^2 vertices - one vertex for every tuple (b, c) of vertices $b \in B$ and $c \in C$. We connect every $(b, c) \in T'$ to $(b, c) \in T$. Finally, we add a set T'' of N vertices. T'' contains one vertex for every vector $c \in C$. For every pair of vertices from T'' we add an edge between the vertices. We connect every $c \in T''$ to

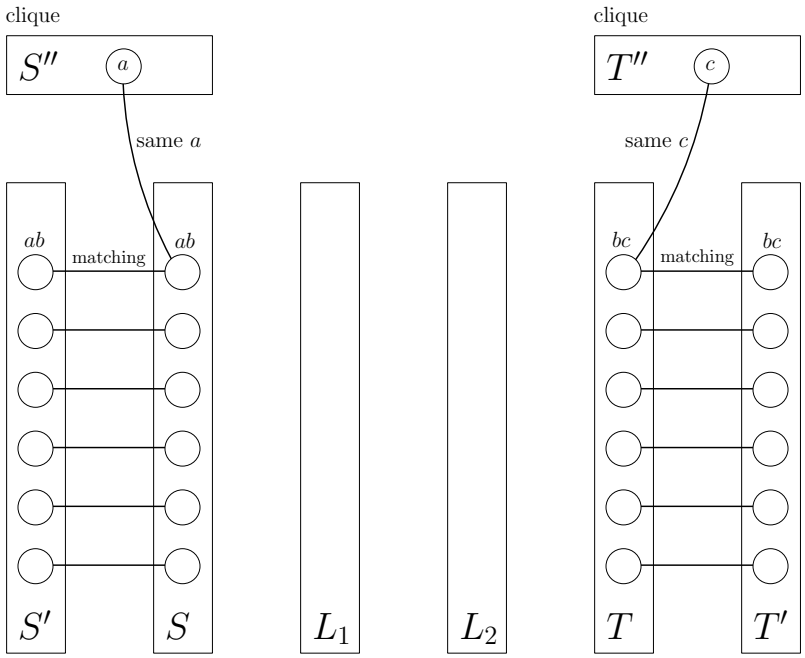

Figure 3-1: The illustration for the 5 vs 8 construction. The edges between sets S, L_1, L_2 and T are not depicted. The edges between vertices in S' and S (T and T') form a matching. Vertices in S'' (T'') form a clique.

$(b, c) \in T$ for all $b \in B$. This finishes the construction of the graph. In the rest of the section we show that the construction satisfies the promised two properties.

Correctness of the construction. We need to consider two cases.

Case 1: the 3-OV instance has no solution. In this case we want to show that for all pairs of vertices u and v we have $d(u, v) \leq 5$. We consider three subcases.

Case 1.1: $u \in S \cup S' \cup S'' \cup L_1$ and $v \in T \cup T' \cup T'' \cup L_2$. We observe that there exists $s \in S$ with $d(u, s) \leq 1$. Indeed, if $u \in S$, then $s = u$ works. If $u \in S' \cup S''$, then we are done by the construction. On the other hand, if $u \in L_1$, then there exists such an $s \in S$ by property 6 from Theorem 3.2.3. Similarly we can show that there exists $t \in T$ such that $d(v, t) \leq 1$. Finally, by property 3 we have that $d(s, t) = 3$. Thus, we can upper bound the distance between u and v by $d(u, v) \leq d(u, s) + d(s, t) + d(t, v) \leq 1 + 3 + 1 = 5$ as required.

Case 1.2: $u, v \in S \cup S' \cup S'' \cup L_1$. From the previous case we know that there are two vertices $s_1, s_2 \in S$ such that $d(u, s_1) \leq 1$ and $d(s_2, v) \leq 1$. To show that $d(u, v) \leq 5$ it is sufficient to show that $d(s_1, s_2) \leq 3$. This is indeed true since both vertices s_1 and s_2 are connected to some two vertices in S'' and every two vertices in S'' are at distance at most 1 from each other.

Case 1.3: $u, v \in T \cup T' \cup T'' \cup L_2$**.** The case is analogous to the previous case.

Case 2: the 3-OV instance has a solution. In this case we want to show that there is a pair of vertices u, v with $d(u, v) \geq 8$. Let $a \in A, b \in B, c \in C$ be a solution to the 3-OV instance. We claim that $d((a, b) \in S', (b, c) \in T') \geq 8$. Let P be an optimal path between $u = ((a, b) \in S')$ and $v = ((b, c) \in T')$ that achieves the smallest distance. We want to show that P uses at least 8 edges. Let $t \in T$ be the first vertex from the set T that is on path P. Let $s \in S$ be the last vertex on path P that belongs to S and precedes t in P. We can easily check that, if $s \neq ((a, b) \in S)$, then $d(u, s) \geq 3$ and, similarly, if $t \neq ((b, c) \in T)$, then $d(t, v) \geq 3$. We consider three subcases.

Case 2.1: $s \neq ((a, b) \in S)$ **and** $t \neq ((b, c) \in T)$**.** Since s and t are separated by two layers of vertices, we must have $d(s, t) \geq 3$. Thus we get lower bound $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq 3 + 3 + 3 = 9 > 8$ as required.

Case 2.2: $s = ((a, b) \in S)$ **and** $t = ((b, c) \in T)$**.** In this case we use property 4 and conclude $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) = 1 + d((a, b) \in S, (b, c) \in T) + 1 \geq 1 + 7 + 1 = 9 > 8$ as required.

Case 2.3: either $s = ((a, b) \in S)$ **or** $t = ((b, c) \in T)$ **holds but not both.** W.l.o.g. $s \neq ((a, b) \in S)$ and $t = ((b, c) \in T)$. If the path uses an edge in the clique on S'' before arriving at s, then $d(u, s) \geq 4$ and we get that $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq 4 + 3 + 1 = 8$. On the other hand, if the path does not use any edge of the clique, then $s = ((a, b') \in S)$ for some $b' \in B$. By property 5 we have $d(s, t) = d((a, b') \in S, (b, c) \in T) \geq 5$. We conclude that $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq 3 + 5 + 1 = 9 > 8$ as required.

3.2.3 6 vs 10 weighted undirected construction

In this section we will prove Theorem 3.1.2, which is implied by the following theorem.

Theorem 3.2.5. *Given a 3-OV instance, we can in $O(N^2d^2)$ time construct a weighted, undirected graph with $O(nN^2 + nNd^2)$ vertices and $O(nN^2d^2)$ edges that satisfies the following two properties.*

1. *If the 3-OV instance has no solution, then for* all *pairs of vertices u and v we have $d(u, v) \leq 6$.*

2. *If the 3-OV instance has a solution, then there exists a pair of vertices u and v such that $d(u, v) \geq 10$.*

Each edge of the graph has weight either 1 *or* 2*.*

Construction of the graph. The construction of the graph is the same as in Theorem 3.2.5 except all edges connecting vertices between sets L_1 and L_2 have weight 2 and all edges inside the cliques on vertices S'' and T'' have weight 2. All the remaining edges have weight 1.

Correctness of the construction. The correctness proof is essentially the same as for Theorem 3.2.4. As before we consider two cases.

Case 1: the 3-OV instance has no solution. In this case we want to show that for all pairs of vertices u and v we have $d(u, v) \leq 6$. In the analysis of Case 1 in Theorem 3.2.4 we show a path between u and v such that the path involves at most one edge from the cliques or between sets L_1 and L_2. Since we added weight 2 to the latter edges, the length of the path increased by at most 1 as a result. So we have upper bound $d(u, v) \leq 6$ for all pairs u and v of vertices.

Case 2: the 3-OV instance has a solution. In this case we want to show that there is a pair of vertices u, v with $d(u, v) \geq 10$. Similarly to Theorem 3.2.4 we will show that $d((a, b) \in S', (b, c) \in T') \geq 10$, where $a \in A, b \in B, c \in C$ is a solution to the 3-OV instance. The analysis of the subcases is essentially the same as in Theorem 3.2.4. For cases 2.1 and 2.2 in the proof of Theorem 3.2.4 we had $d((a, b) \in S', (b, c) \in T') \geq 9$. Since we increased edge weights between L_1 and L_2 to 2 and every path from $(a, b) \in S'$ to $(b, c) \in T'$ must cross the layer between L_1 and L_2, we also increased the lower bound of the length of the path from 9 to 10 for cases 2.1 and 2.2. It remains to consider Case 2.3. As in the proof of Theorem 3.2.4, w.l.o.g. $s \neq ((a, b) \in S)$ and $t = ((b, c) \in T)$. If the path uses an edge in the clique on S'' before arriving at s, then $d(u, s) \geq 5$ and we get lower bound $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq 5 + 4 + 1 = 10$. On the other hand, if the path does not use any edge of the clique, then $s = ((a, b') \in S)$ for some $b' \in B$. By property 5 and because we increased edge weights between L_1 and L_2 to 2, we have $d(s, t) = d((a, b') \in S, (b, c) \in T) \geq 6$. We conclude that $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq 3 + 6 + 1 = 10$ as required.

3.2.4 $3k - 4$ vs $5k - 7$ unweighted directed construction

In this section, we will prove Theorem 3.1.3.

In particular, we will use Theorem 3.2.2 to prove the following result, which implies Theorem 3.1.3.

Theorem 3.2.6. *Given a k-OV instance, we can in $O(kN^{k-1}d^{k-1})$ time construct an unweighted, directed graph with $O(kN^{k-1} + kN^{k-2}d^{k-1})$ vertices and $O(kN^{k-1}d^{k-1})$ edges that satisfies the following two properties.*

1. *If the k-OV instance has no solution, then for* all *pairs of vertices u and v we have $d(u, v) \leq 3k - 4$.*

2. *If the k-OV instance has a solution, then there exists a pair of vertices u and v such that* $d(u, v) \geq 5k - 7$.

Construction of the graph. We construct a graph with the required properties by starting with the graph from Thereom 3.2.2 and adding more vertices and edges. First we will construct a weighted graph and then we will make it unweighted. Figure 3-2 illustrates the construction of the graph for the special case $k = 4$.

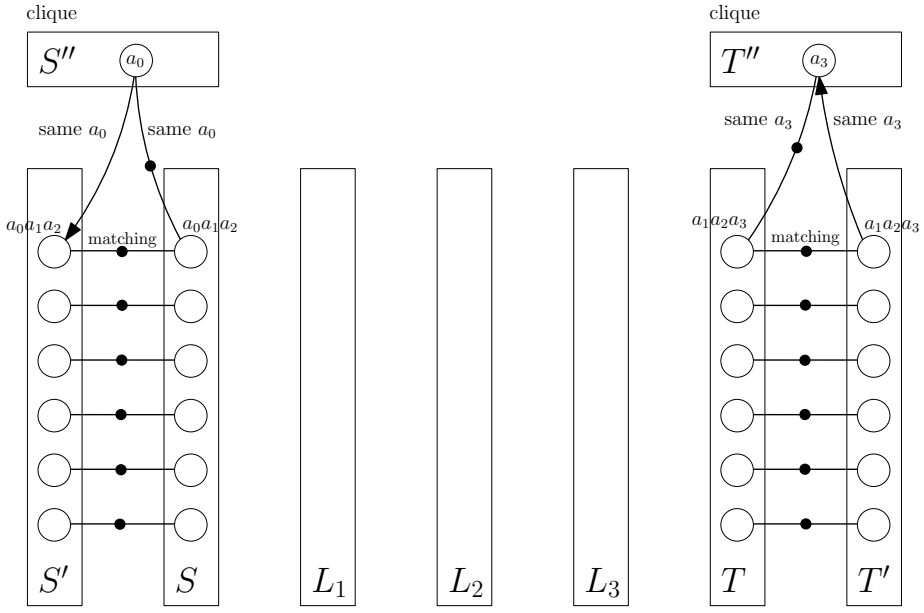

Figure 3-2: The $3k - 4$ vs $5k - 7$ construction for the special case $k = 4$. The edges between sets S, L_1, L_2, L_3 and T are not depicted. The matching between sets S and S' consists of unweighted paths of length $k - 2 = 2$. The edges between sets S and S'' consists of unweighted paths of length $k - 2 = 2$. Similarly for the right side.

We start by adding a set S' of N^{k-1} vertices. S' consists of all tuples $(a_0, a_1, \ldots, a_{k-2})$ where for each i, $a_i \in W_i$. We connect every $(a_0, a_1, \ldots, a_{k-2}) \in S'$ to its counterpart $(a_0, a_1, \ldots, a_{k-2}) \in S$ with an undirected edge of weight $k - 2$ to form a matching. We also add another set S'' of N vertices. S'' contains one vertex a_0 for every $a_0 \in W_0$. For every pair of vertices in S'' we add an undirected edge of weight 1 between the vertices. Thus, the N vertices form a clique. Furthermore, for every vertex $a_0 \in S''$ we add an undirected edge of weight $k - 2$ to $(a_0, b_1, \ldots, b_{k-2}) \in S$ for all $b_1, \ldots, b_{k-2}$. Finally for every vertex $a_0 \in S''$ we add a *directed* edge of weight 1 towards $(a_0, b_1, \ldots, b_{k-2}) \in S$ for all $b_1, \ldots, b_{k-2}$. Some of the edges that we added have weight $k - 2$. We make those unweighted by subdividing them into edges of weight 1. Let S''' be the set of newly added vertices. In total we added $O(kN^{k-1})$ vertices and $O(kN^{k-1})$ edges.

We do a similar construction for the set T of vertices. We add a set T' of N^{k-1} vertices — one vertex for every tuple $(a_1, \ldots, a_{k-1})$ where for each i, $a_i \in W_i$. We

connect every $(a_1, \ldots, a_{k-1}) \in T'$ to $(a_1, \ldots, a_{k-1}) \in T$ by an undirected edge of weight $k-2$. Finally, we add a set T'' of n vertices. T'' contains one vertex for every vector $a_{k-1} \in W_{k-1}$. We connect every pair of vertices in T'' by an undirected edge of weight 1. We connect every vertex $a_{k-1} \in T''$ to $(b_1, \ldots, b_{k-2}, a_{k-1}) \in T$ by an undirected edge of weight $k-2$ for all $b_1, \ldots, b_{k-2}$. Also, for every vertex $a_{k-1} \in T''$ we add a *directed* edge of weight 1 from $(b_1, \ldots, b_{k-2}, a_{k-1}) \in T'$ to a_{k-1} for all $b_1, \ldots, b_{k-2}$. Some of the edges that we just added have weight $k-2$. We make those unweighted by subdividing them into edges of weight 1. Let T''' be the set of newly added vertices. This finishes the construction of the graph. In the rest of the section we show that the construction satisfies the promised two properties stated in Theorem 3.2.6.

Correctness of the construction. We need to consider two cases.

Case 1: the k-OV instance has no solution. In this case we want to show that for all pairs of vertices u and v we have $d(u, v) \leq 3k-4$. We consider subcases.

Case 1.1: $u \in S \cup S' \cup S'' \cup S''' \cup L_i$ for $1 \leq i \leq k-2$ and $v \in T \cup T' \cup T'' \cup T''' \cup L_j$ for $2 \leq j \leq k-1$. We observe that there exists $s \in S$ that has $d(u, s) \leq k-2$. Similarly, there exists $t \in T$ with $d(t, v) \leq k-2$. By property 3 from Theorem 3.2.2 we have that $d(s, t) \leq k$. This gives us upper bound $d(u, v) \leq d(u, s) + d(s, t) + d(t, v) \leq (k-2) + k + (k-2) = 3k-4$ as required. The proof when the sets for u and v are swapped is identical since we only use paths on unweighted edges.

Case 1.2: $u, v \in S \cup S' \cup S'' \cup S''' \cup L_1$. We note that there is some vertex $s \in S''$ with $d(u, s) \leq 2(k-2)$ (via undirected edges). Also, there is some vertex $s' \in S''$ with $d(s', v) \leq k-1$ (possibly using directed edges). S'' is a clique so $d(s, s') \leq 1$. Thus, $d(u, v) \leq d(u, s) + d(s, s') + d(s', v) \leq 2(k-2) + 1 + (k-1) = 3k-4$.

Case 1.3: $u, v \in T \cup T' \cup T'' \cup T''' \cup L_{k-1}$. This case is similar to the previous case. We note that there is some vertex $t \in T''$ with $d(t, v) \leq 2(k-2)$ (via undirected edges). Also, there is some vertex $t' \in T''$ with $d(u, t') \leq k-1$ (possibly using directed edges). S'' is a clique so $d(t', t) \leq 1$. Thus, $d(u, v) \leq d(u, t') + d(t', t) + d(t, v) \leq (k-1) + 1 + 2(k-2) = 3k-4$.

Case 2: the k-OV instance has a solution. In this case we want to show that there is a pair of vertices u, v with $d(u, v) \geq 5k-7$. Let $(a_0, a_1, \ldots, a_{k-1})$ be a solution to the k-OV instance where for each i, $a_i \in W_i$. We claim that $d((a_0, \ldots, a_{k-2}) \in S', (a_1, \ldots, a_{k-1}) \in T') \geq 5k-7$. Let P be an shortest path between $u = ((a_0, \ldots, a_{k-2}) \in S')$ and $v = ((a_1, \ldots, a_{k-1}) \in T')$. We want to show that P uses at least $5k-7$ edges. Let $s \in S$ be the first vertex on path P that belongs to S and let $t \in T$ be the last vertex from the set T that is on path P. We observe that due to the directionality of the edges, s and t must be the counterparts of u and v respectively; that is, $s = ((a_0, \ldots, a_{k-2}) \in S)$ and $t = ((a_1, \ldots, a_{k-1}) \in T)$.

Note that these definitions of s and t differ from the definitions of s and t in previous proofs. We consider three subcases.

Case 2.1: A vertex in $S' \cup S'' \cup S'''$ appears after s on the path P. We observe that if $s_1, s_2 \in S$ is a pair of vertices on the path P such that no vertex in S appears between them on P, then the portion of P between s_1 and s_2 either contains only vertices in $S' \cup S'' \cup S'''$ or contains no vertices in $S' \cup S'' \cup S'''$. Let $s_1, s_2 \in S$ be such that the portion of P between them contains only vertices in $S' \cup S'' \cup S'''$. Such s_1, s_2 exist by the specification of this case. If $s_1 = s_2$ then P is not a shortest path. Otherwise, the portion of P between s_1 and s_2 must include a vertex in S''. Thus, $d(s_1, s_2) \geq 2(k-2)$. We consider three subcases.

- $s_1 \neq s$. The distance between any pair of vertices in S is at least 2 so $d(s, s_1) \geq 2$. Then, $d(u, v) \geq d(u, s) + d(s, s_1) + d(s_1, s_2) + d(s_2, t) + d(t, v) \geq (k-2) + 2 + 2(k-2) + k + (k-2) = 5k - 6$.

- $s_1 = s$ and $s_2 = ((a_0, b_1, \ldots, b_{k-2}) \in S)$ for some $b_1, \ldots, b_{k-2}$. In this case, by property 5 we have $d(s_2, t) \geq k + 2$. Thus, $d(u, v) \geq d(u, s_1) + d(s_1, s_2) + d(s_2, t) + d(t, v) \geq (k-2) + 2(k-2) + (k+2) + (k-2) = 5k - 6$.

- $s_1 = s$ and $s_2 = ((b_0, \ldots, b_{k-2} \in S)$ for some with $b_0 \neq a_0$. In this case, the path from s_1 to s_2 must include an edge in the clique S'' since these are the only edges among vertices in $S' \cup S'' \cup S'''$ for which adjacent tuples can differ with respect to their first element. Thus, $d(s_1, s_2) \geq 2(k-2)+1 \geq 2k-3$. Therefore, $d(u, v) \geq d(u, s_1)+d(s_1, s_2)+d(s_2, t)+d(t, v) \geq (k-2)+(2k-3)+k+(k-2) = 5k - 7$.

Case 2.2: A vertex in $T' \cup T'' \cup T'''$ appears before t on the path P. This case is analogous to the previous case.

Case 2.3: The portion of the path P between s and t contains no vertices in $S' \cup S'' \cup S''' \cup T' \cup T'' \cup T'''$. By property 4, $d(s, t) \geq 3k - 2$. Thus, $d(u, v) \geq d(u, s) + d(s, t) + d(t, v) \geq (k-2) + (3k-2) + (k-2) = 5k - 6$.

Chapter 4

Tightening the Time vs. Accuracy Trade-off for Diameter in Directed Graphs

4.1 Result

In this chapter, we prove the following result.

Theorem 4.1.1. *Let $k \geq 2$ be a fixed integer. Assuming SETH, for all $\delta > 0$, any $(\frac{2k-1}{k} - \delta)$-approximation algorithm for Diameter in an unweighted directed graph on m edges requires $m^{\frac{k}{k-1}-o(1)}$ time.*

Theorem 4.1.1 was previously known for $k = 2$ [RV13] and $k = 3$ [Li21], so we need to prove it for $k \geq 4$.

Theorem 4.1.1 was proved independently and concurrently by Ray Li [Li21] with a simpler proof.

4.2 Techniques

We use the graph of Theorem 3.2.2 as a starting point. This construction does not directly work for Diameter since in the NO case the diameter of the graph might be as big as $2k$, as two vertices in S can be far from each other. So, to get a construction for Diameter, the challenge is to add more vertices and edges to make these S-S distances smaller in the NO case, while not decreasing the S-T distances in the YES case.

To address this challenge for the case of $k = 4$, Bonnet [Bon21b] uses the following key idea. He copies one of the middle layers, where edges between this layer and its copy allow you to change the vectors of a node (v_1, v_2, x). These extra edges allow for shorter paths in the NO case exclusively.

A natural way to generalize Bonnet's construction to larger values of k is to make a copy of each of the internal layers. However, it is not clear which of the vectors

we should allow to change on an edge from a layer to its copy. If we allow certain vectors to change at the wrong layer, this shrinks the diameter too much in the YES case, while if we don't allow enough flexibility, this does not adequately decrease the distances in the NO case. A key insight for our construction is that changing different sets of vectors should have different costs.

These costs are encoded in our construction through an intricate system of "back edges". These back edges allow for paths that go from some layer to some previous layer while changing either some prefix or some suffix of the vector tuple. The size of the prefix or suffix that is permitted to change depends on which pair of layers these back edges are connecting. By carefully balancing the number of vectors we are permitted to change for which pairs of levels, we keep all the distances at most k in the NO case, and ensure that the diameter is at least $2k-1$ in the YES case.

4.3 Construction

In this section we suppose that $k \geq 5$ and we prove Theorem 4.1.1 by reduction from k-OV.

Although our construction is based on that of Theorem 3.2.2, it is different enough that we will describe it from scratch.

We are given a k-OV instance S where each vector in S is of length $d = c_k \log n$, where c_k is the constant defined in Hypothesis 1. We will create a graph $G = (V, E)$ with $O(n^{k-1} + n^{k-2}d^{k-1})$ vertices and $O(n^{k-1}d^{2k-2})$ edges, such that if the k-OV instance is a YES instance then G has diameter $2k-1$, and if it is a NO instance then G has diameter k.

Before presenting the construction of G, we note that the above conditions on G suffice to prove Theorem 4.1.1: suppose for contradiction that there exist $\delta > 0$ and $\varepsilon > 0$, such that there is a $(\frac{2k-1}{k} - \delta)$-approximation algorithm $\mathcal{A}$ for Diameter in directed graphs with M edges that runs in $O(M^{\frac{k}{k-1}-\varepsilon})$ time. That is, $\mathcal{A}$ can distinguish whether G has diameter k or $2k-1$ in $O(|E|^{\frac{k}{k-1}-\varepsilon}) = \tilde{O}(n^{k-(k-1)\varepsilon})$ time, where the last equality comes from the fact that $|E| = \tilde{O}(n^{k-1})$, since k is constant and $d = O(\log n)$. Then, the reduction from k-OV to Diameter on the graph G tells us that we can solve the k-OV instance in $\tilde{O}(n^{k-(k-1)\varepsilon})$ time, which contradicts the k-OV Hypothesis.

4.3.1 Vertex set

Given our k-OV instance, we first augment S with the all 1s vector. Note that this does not change the output of the k-OV instance. Then, we make k copies of S and call them $S_1, \ldots, S_k$. We let a *coordinate* be an element of $[d]$, that will represent a position in some vector in $S_1, \ldots, S_k$.

We start with $k+1$ layers of vertices $L_1, \ldots, L_{k+1}$. Vertices of L_1 are $k-1$ tuples $(a_1, \ldots, a_{k-1})$, with $a_i \in S_i$. Vertices of L_{k+1} are $k-1$ tuples $(b_2, \ldots, b_k)$, where $b_i \in S_i$. For $i = 2, \ldots, k$, the vertices of L_i are $(a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x)$, where

$a_j, b_j \in S_j$ for each j and $x = (x_1, \ldots, x_{k-1})$ is an array of coordinates that satisfies the following two conditions:

1. For each $1 \leq j \leq k-i$, $a_j[x_\ell] = 1$ for all $1 \leq \ell \leq k-j$, and
2. For each $k-i+3 \leq j \leq k$, $b_j[x_\ell] = 1$ for all $k-j+1 \leq \ell \leq k-1$.

See table 4.1.

	x_1	x_2	...	x_{i-2}	x_{i-1}	x_i	...	x_{k-2}	x_{k-1}
a_1	1	1	...	1	1	1	...	1	1
a_2	1	1	...	1	1	1	...	1	
...	...	...	...	...	...	...	...		
a_{k-i}	1	1	...	1	1	1			
b_{k-i+3}				1	1	1	...	1	1
...			...	...	...	...	...	...	...
b_{k-1}		1	...	1	1	1	...	1	1
b_k	1	1	...	1	1	1	...	1	1

Table 4.1: The relationship between the vector array $a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k$ and the coordinate array x of a node $(a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x)$ in layer L_i.

For each $i = 3, \ldots, k-1$, we add a set L'_i of vertices. Vertices of L'_i are $(a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x)$ where $a_j, b_j \in S_j$ and $x = (x_1, \ldots, x_{k-1})$ is any coordinate array.

For $i = 4, \ldots, k-1$, we have a set A_i, where each node $\alpha \in A_i$ is a $k-i$ tuple of vectors $(a_1, \ldots, a_{k-i})$, with $a_\ell \in S_\ell$ for each $\ell \in \{1, \ldots, k-i\}$. For $i = 3, \ldots, k-2$, we have a set B_i, where each node $\alpha \in B_i$ is an $i-2$-tuple of vectors $(b_{k-i+3}, \ldots, b_k)$, with $b_\ell \in S_\ell$ for each $\ell \in \{k-i+3, \ldots, k\}$. Note that these nodes do not have a coordinate array.

Let $A = \cup_i A_i$, let $B = \cup_i B_i$, let $L = \cup_i L_i$, and let $L' = \cup_i L'_i$. For all i, let *level* i denote $L_i \cup L'_i \cup A_i \cup B_i$ (or the union of these sets that exist for that i). In contrast, we use the word *layer* to refer to an individual set L_i or L'_i.

Finally we have 2 additional vertices, u and v. This completes the definition of the vertex set of G. See Figure 4-2. Figure 4-2 does not capture the case of $k = 5$ since in this case L_{k-2} comes before L_4, so we also include Figure 4-3 to depict the $k = 5$ case.

Number of nodes: The number of nodes of layers L_1 and L_{k+1} is n^{k-1}, and the number of nodes in each layer L_i and L'_i for $1 < i < k+1$ is at most $n^{k-2}d^{k-1}$. The number of nodes in each A_i and B_i is at most n^{k-2} and the number of fixed nodes is constant, so the graph has $O(n^{k-1} + n^{k-2}d^{k-1})$ nodes.

4.3.2 Edge set

All edges are undirected unless otherwise specified. We have five types of edges: *fixed edges*, *coordinate-change edges*, *vector-change edges*, *swap edges* and *back edges*.

- A *fixed edge* has u or v as one endpoint and is directed.
- A *coordinate-change edge* is between two nodes having the same sequence of vectors and different coordinate arrays and is undirected.
- A *vector-change edge* is between two nodes with the same coordinate array and the same vector array except for at most one entry, where a vector in some S_i is changed for another vector in S_i. A vector-change edge is undirected.
- A *swap edge* is between two nodes with the same coordinate array and the same vector array except for one entry, where a vector in S_i is changed for a vector in S_{i+2}, or vice versa. A swap edge can also be between L_1 and L_2 or between L_k and L_{k+1}, in which case a vector is changed for a coordinate array, or vice versa. A swap edge is undirected.
- A *back edge* is an edge with at least one endpoint in A or B, and is directed. A back edge incident to one vertex in B and one vertex in A is called a *ba*-type back edge. Otherwise, a back edge is called an a-type back edge if it is incident to a vertex in A, and a b-type back edge if it is incident to a vertex in B.

Now we specify each of these edges in the graph.

Swap edges: For each $i = 2, \ldots, k-1$, there are swap edges between $(a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x) \in L_i$ and $(a_1, \ldots, a_{k-i-1}, b_{k-i+2}, \ldots, b_k, x) \in L_{i+1}$. There are also swap edges between $(a_1, \ldots, a_{k-1}) \in L_1$ and $(a_1, \ldots, a_{k-2}, x) \in L_2$, as well as between $(b_2, \ldots, b_k) \in L_{k+1}$ and $(b_3, \ldots, b_k, x) \in L_k$.

Vector-change edges: For each $i = 3, \ldots, k-1$, there are vector-change edges between L_i and L'_i. These edges are between $\alpha = (a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x) \in L_i$ and $\beta \in L'_i$, where β has coordinate array equal to x, and the same vectors as α, except for at most one of a_{k-i} or b_{k-i+3}.

Coordinate-change edges: For each $i = 3, \ldots, k-1$, there are coordinate-change edges within each L'_i, and between L'_i and L_i. We also have coordinate-change edges within L_2 and L_k.

Back edges: For $i = 4, \ldots, k-1$, and for every $\alpha = (a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x) \in L_i \cup L'_i$, we add an a-type back edge from α to $\beta = (a_1, \ldots, a_{k-i}) \in A_i$. For every node $\beta = (a_1, \ldots, a_{k-i}) \in A_i$, we add an a-type back edge from β to any vertex $(c_1, \ldots, c_{k-4}, c_{k-1}, c_k, x) \in L'_4$, if $c_j = a_j$ for all $j = 1, \ldots, k-i$. For $i = 3, \ldots, k-2$ and every node $\alpha = (a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x) \in L_i \cup L'_i$, we add a b-type back edge from $\beta = (b_{k-i+3}, \ldots, b_k) \in B_i$ to α. We add a b-type back edge from every node $(c_1, c_2, c_5, \ldots, c_k, x) \in L'_{k-2}$ to $\beta = (b_{k-i+3}, \ldots, b_k) \in B_i$ if $c_j = b_j$ for every $j = k-i+3, \ldots, k$. See Figure 4-1.

Additionally, for any $i = 3, \ldots, k-2$, we add a b-type back edge from $(a_{k-i+3}, \ldots, a_k) \in B_i$ to $(a_k) \in B_3$. For any $i = 4, \ldots, k-1$, we add an a-type back edge from

$(a_1) \in A_{k-1}$ to $(a_1, \ldots, a_{k-i}) \in A_i$. Also, for any $i = 4, \ldots, k-2$, we add a ba-type back edge from every node in B_i to every node in A_i. Note that for $k = 5$, we don't have any ba type back edges since $A = A_4$ and $B = B_3$. See Figure 4-1.

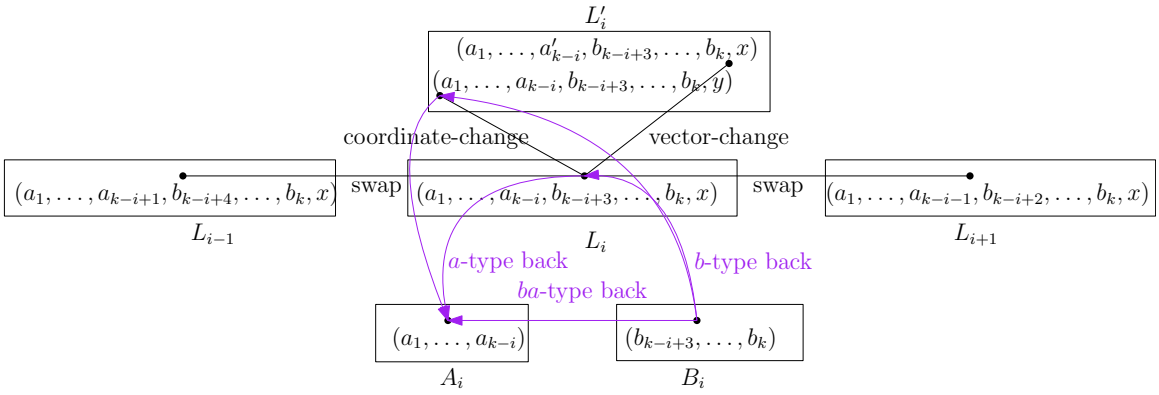

Figure 4-1: Edges attached to a node in L_i, for $i = 4, \ldots, k-2$. Purple edges are back edges.

Fixed edges: Now we specify fixed edges. There is a directed edge from each vertex of L'_{k-2} to v and a directed edge from v to each vertex of $L_1 \cup L_2$. There is a directed edge from each vertex of $L_k \cup L_{k+1}$ to u and a directed edge from u to each vertex of L'_4. See Figure 4-2.

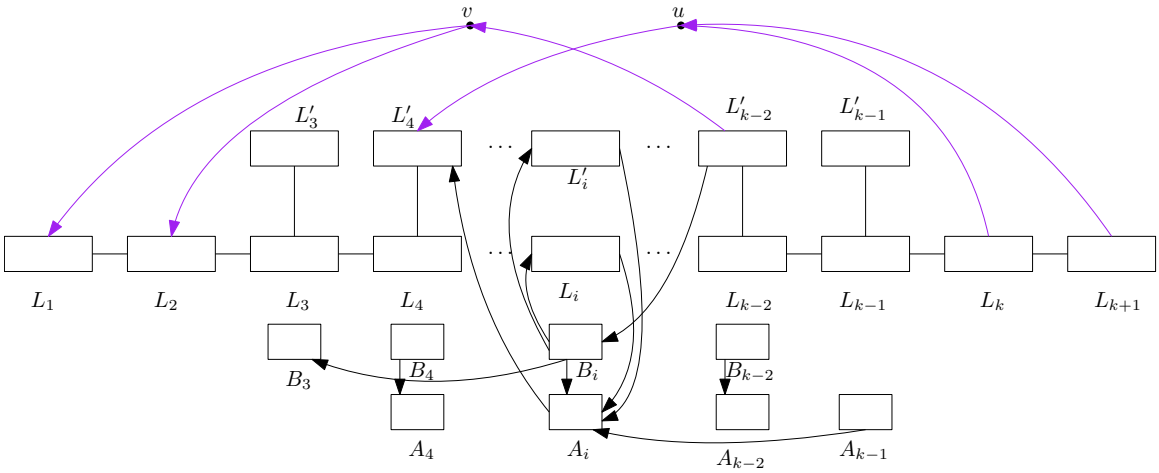

Figure 4-2: Vertex set of G and its directed edges. The purple edges show fixed edges and they are attached to all nodes in a set they are pointing to/from.

This finishes the definition of the graph G.

Number of edges: Coordinate-change edges and vector-change edges are only incident to vertices in $L' \cup L_2 \cup \cdots \cup L_k$, of which there are $O(n^{k-2}d^{k-1})$. Each such

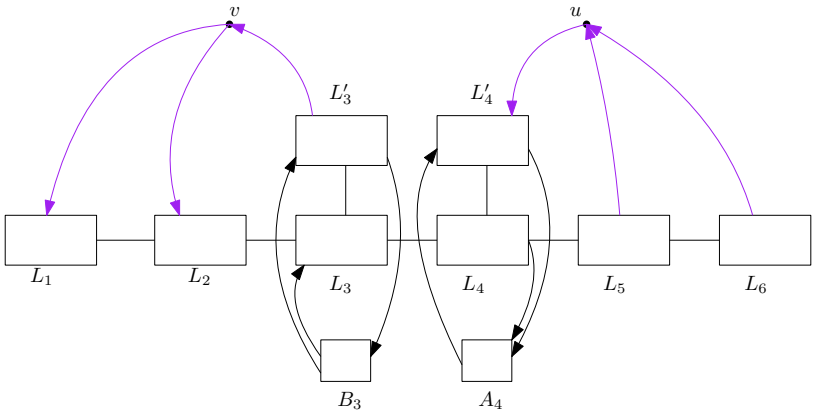

Figure 4-3: The construction when $k = 5$.

vertex has at most d^{k-1} incident coordinate-change edges, since this is the total number of possible coordinate arrays. Each such vertex has at most n incident vector-change edges, since each vector-change edge only changes one vector. Thus, there are $O(n^{k-1}d^{2k-2})$ coordinate-change and vector-change edges.

Swap edges are only incident to vertices in L. Each of the $O(n^{k-1})$ vertices in $L_1 \cup L_{k+1}$ is incident to at most d^{k-1} swap edges since swap edges from L_1 to L_2 and from L_{k+1} to L_k change a vector for a coordinate. Each of the $O(n^{k-2}d^{k-1})$ vertices in $L_2 \cup \cdots \cup L_k$ is incident to at most n swap edges since these edges change a vector or coordinate for a vector. Thus, there are $O(n^{k-1}d^{k-1})$ swap edges.

Each vertex is incident to $O(k)$ a-type or b-type back edges since each vertex has at most one edge to and from each A_i and B_i. The number of ba-type back edges between each pair B_i, A_i is $O(n^{k-2})$, since B_i has at most n^{k-i-2} nodes and A_i has at most n^i nodes.

Finally, each vertex is incident to at most two fixed edges.

Thus, we have shown that the total number of edges is $O(n^{k-1}d^{2k-2})$.

4.4 NO instance of k-OV implies diameter $\leq k$

In a NO instance, for every set F of at most k vectors, there exists a coordinate that is 1 for every vector in F. Given a set F of at most k vectors, we let $C(F)$ denote a coordinate that is 1 for every vector in F. For any vertex α, let $C(\alpha)$ be a coordinate that is 1 for every vector in α.

4.4.1 Fixed paths

First, we will calculate the distance between pairs of vertices whose distance does not depend on the answer to the k-OV instance. Note that regardless of the answer to the k-OV instance, we can assume that no set F of at most $k-1$ vectors is orthogonal. That is, $C(F)$ and $C(\alpha)$ are well-defined for any $|F| \leq k-1$ and any α.

Claim 1. *For each vertex $\alpha \in V$:*

1. *There exists a vertex* $\beta_1 \in L'_{k-2}$ *so that* $d(\alpha, \beta_1) \leq k-2$,

2. *There exists a vertex* $\beta_2 \in L'_4$ *so that* $d(\beta_2, \alpha) \leq k-2$,

3. *There exists a vertex* $\beta_3 \in L_k$ *so that* $d(\alpha, \beta_3) \leq k-1$, *and*

4. *There exists a vertex* $\beta_4 \in L_2$ *so that* $d(\beta_4, \alpha) \leq k-1$.

Proof. We prove 1 and 3. Then, 2 and 4 follow due to symmetry. Starting from α, we can proceed towards the appropriate layer (L'_{k-2} or L_k) as follows. First, by taking at most two edges we can go to a vertex in $L_2 \cup \cdots \cup L_k$ as follows. If $\alpha \in A \cup B$, the first edge is a back edge to a vertex in L'_j for some j; note that such a vertex exists because the new vectors can all be the all 1s vector, and the coordinate array can be $k-1$ copies of $C(\alpha)$. The second edge is a coordinate-change edge to L_j that does not actually change the coordinate. If α is in an L'_i set, we only take one edge which is a coordinate-change edge, where the new coordinate array is $k-1$ copies of $C(\alpha)$. If $\alpha = u$, we take a fixed edge to L'_4 and then an edge to L_4. If $\alpha = v$, we take a fixed edge to L_2. If $\alpha \in L_1 \cup L_{k+1}$, then we take an edge to L_2 or L_k (respectively) by adding the coordinate array that is $k-1$ copies of $C(\alpha)$.

Then we proceed to the L_{k-2} by taking swap edges that change some vector to the all 1s vector. Then we take a vector-change edge to L'_{k-2} or two swap edges to L_k. The vector-change edge need not actually change any vectors. It is straightforward to see that these paths are of the appropriate lengths. □

Due to the fixed edges in the graph, the consequences of Claim 1 are respectively that

1. For each vertex $\alpha \in V$ and for any vertex $\beta \in L_1 \cup L_2 \cup \{v\}$, $d(\alpha, \beta) \leq k$.

2. For each vertex $\alpha \in V$ and for any vertex $\beta \in L_k \cup L_{k+1} \cup \{u\}$, $d(\beta, \alpha) \leq k$.

3. For each vertex $\alpha \in V$, $d(\alpha, u) \leq k$.

4. For each vertex $\alpha \in V$, $d(v, \alpha) \leq k$.

4.4.2 Variable paths

The distances that we did not bound in Section 4.4.1 are those from a vertex α to a vertex β in the following cases. Cases 1 through 3 demonstrate paths with both endpoints in $L \cup L'$. Cases 4 through 7 demonstrate paths with one endpoint in A or B. For each case we show that the corresponding paths have length at most k.

1. $\alpha \in L_1 \cup L_2$ and $\beta \in L' \cup L \setminus (L_1 \cup L_2)$.

2. $\alpha \in L' \cup L \setminus (L_k \cup L_{k+1})$ and $\beta \in L_k \cup L_{k+1}$.

3. $\alpha, \beta \in L_3 \cup \cdots \cup L_{k-1} \cup L'_3 \cup \cdots \cup L'_{k-1}$.

4. $\alpha \in V \setminus \{u, v\}$ and $\beta \in B$.

5. $\alpha \in B$ and $\beta \in V \setminus \{u, v\}$.

6. $\alpha \in V \setminus \{u, v\}$ and $\beta \in A$.

7. $\alpha \in A$ and $\beta \in V \setminus \{u, v\}$.

Cases 1 and 2 are completely symmetric, as are cases 4 and 7 as well as cases 5 and 6. Hence we only analyze cases 1, 3, 4, and 5.

Case 1: $\alpha \in L_1 \cup L_2$ **and** $\beta \in L' \cup L \setminus (L_1 \cup L_2)$**.** If $\alpha \in L_1$ let $\alpha = (a_1, \ldots, a_{k-1})$ and if $\alpha \in L_2$ let $\alpha = (a_1, \ldots, a_{k-2}, x_\alpha)$ where we let a_{k-1} be the all 1s vector. We condition on which set β is in.

Case 1a: $\beta \in L_k \cup L_{k+1}$**.** If $\beta \in L_{k+1}$ let $\beta = (b_2, \ldots, b_k)$, and if $\beta \in L_k$ let $\beta = (b_3, \ldots, b_k, x_\beta)$ where we let b_2 be the all 1s vector.

We will use the coordinate array $x = (x_1, \ldots, x_{k-1})$ defined as follows: for all $1 \leq \ell \leq k-1$, $x_{k-\ell} = C(a_1, \ldots, a_\ell, b_{\ell+1}, \ldots, b_k)$. See Table 4.2(a).

(a) Cases 1a and 4

	x_1	$\ldots$	x_ℓ	$\ldots$	x_{k-1}
a_1	1	$\ldots$	1	$\ldots$	1
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	
$a_{k-\ell}$	1	$\ldots$	1		
$\ldots$	$\ldots$	$\ldots$			
a_{k-1}	1				
b_2					1
$\ldots$				$\ldots$	$\ldots$
$b_{k-\ell+1}$			1	$\ldots$	1
$\ldots$		$\ldots$	$\ldots$	$\ldots$	$\ldots$
b_k	1	$\ldots$	1	$\ldots$	1

(b) Cases 1b and 1c

	x_1	$\ldots$	x_ℓ	$\ldots$	x_{k-2}	x_{k-1}
a_1	1	$\ldots$	1	$\ldots$	1	1
a_2	1	$\ldots$	1	$\ldots$	1	1
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$		
$a_{k-\ell}$	1	$\ldots$	1			
$\ldots$	$\ldots$	$\ldots$				
a_{k-1}	1					
c_1					1	1
$\ldots$				$\ldots$	$\ldots$	$\ldots$
$c_{k-\ell-1}$			1	$\ldots$	1	1
$\ldots$		$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
c_{k-2}	1	$\ldots$	1	$\ldots$	1	1

Table 4.2: Coordinate arrays used in the $\alpha\beta$ path.

We now describe a path of length k from α to β. All of the internal vertices along the path use the coordinate x, and the existence of these vertices follows from the definition of x.

We begin by taking an edge from α to $(a_1, \ldots, a_{k-2}, x) \in L_2$ either by taking a swap edge or a coordinate-change edge (depending on whether α is in L_1 or L_2). We then use swap edges to go from L_2 to L_k, where to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b_{k-r+2}. After traversing these edges, we end at $(b_3, \ldots, b_k, x) \in L_k$. Finally, we take an edge to β (which is a swap edge or a coordinate-change edge depending on whether β is in L_{k+1} or L_k). This path is shown with black (solid and dashed lines) and blue edges in Figure 4-4.

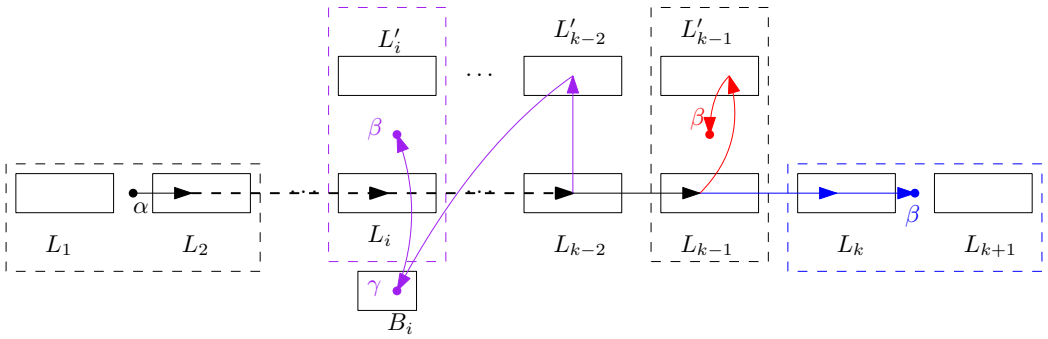

Figure 4-4: $\alpha\beta$ path in Case 1. Different cases of β are shown with different colors. The path between L_2 and L_{k-2} is shown with black dashed lines. Solid lines indicate edges. A node in a dashed box containing two sets means that the node is in either set.

Case 1b: $\beta \in L_{k-1} \cup L'_{k-1}$**.** Let $\beta = (a'_1, b_4, \ldots, b_k, x_\beta) = (c_1, \ldots, c_{k-2}, x_\beta) \in L_{k-1} \cup L'_{k-1}$. We will use the coordinate array $x = (x_1, \ldots, x_{k-1})$ defined as follows: for all $1 \le \ell \le k-2$, $x_\ell = C(a_1, \ldots, a_{k-\ell}, c_{k-\ell-1}, \ldots, c_{k-2})$, and we set $x_{k-1} = x_{k-2}$. See Table 4.2(b). We begin by taking an edge from α to $(a_1, \ldots, a_{k-2}, x) \in L_2$ either by taking a swap edge or a coordinate-change edge (depending on whether α is in L_1 or L_2). We then use swap edges to go from L_2 to L_{k-1}, where to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b_{k-r+2}. So we are at $(a_1, b_4, \ldots, b_k, x) \in L_{k-1}$. We then take a vector-change edge to $(a'_1, b_4, \ldots, b_k, x) \in L'_{k-1}$, and then take a coordinate-change edge to β. This path is shown with black (solid and dashed lines) and red edges in Figure 4-4.

Case 1c: $\beta \in L_3 \cup \cdots \cup L_{k-2} \cup L'_3 \cup \cdots \cup L'_{k-2}$**.** Suppose $\beta \in L_i \cup L'_i$ and let $\beta = (a'_1, \ldots, a'_{k-i}, b_{k-i+3}, \ldots, b_k, x_\beta) = (c_1, \ldots, c_{k-2}, x_\beta)$.

We will use the coordinate array $x = (x_1, \ldots, x_{k-1})$ defined as follows: for all $1 \le \ell \le k-2$, $x_\ell = C(a_1, \ldots, a_{k-\ell}, c_{k-\ell-1}, \ldots, c_{k-2})$, and we set $x_{k-1} = x_{k-2}$. See Table 4.2(b).

We begin by taking an edge from α to $(a_1, \ldots, a_{k-2}, x) \in L_2$ either by taking a swap edge or a coordinate-change edge (depending on whether α is in L_1 or L_2). We then use swap edges to go from L_2 to L_{k-2}, where to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b_{k-r+2}, where b_{k-r+2} has been defined as part of β if $r \le i-1$, and otherwise we define b_{k-r+2} as the all 1s vector. After traversing these edges, we end at $(a_1, a_2, b_5, \ldots, b_k, x) \in L_{k-2}$.

Then, we take a coordinate-change edge to arrive at $(a_1, a_2, b_5, \ldots, b_k, x) \in L'_{k-2}$ (without actually changing any coordinates). This vertex exists by option 2 of the specification of vertices in L'_{k-2}. So far, the path is of length $k-2$.

Then, we use a b-type back edge to go to $\gamma = (b_{k-i+3}, \ldots, b_k) \in B_i$, and then use another b-type back edge to go from γ to β. The full path is of length k. This path is shown with black (solid and dashed lines) and purple edges in Figure 4-4.

Case 3: α, $\beta \in L_3 \cup \cdots \cup L_{k-1} \cup L'_3 \cup \cdots \cup L'_{k-1}$. Suppose $\alpha \in L_i \cup L'_i$ and $\beta \in L_j \cup L'_j$. Let $\alpha = (a_1, \ldots, a_{k-i}, b_{k-i+3}, \ldots, b_k, x_\alpha) = (c_1, \ldots, c_{k-2}, x_\alpha)$ and let $\beta = (a'_1, \ldots, a'_{k-j}, b'_{k-j+3}, \ldots, b'_k, x_\beta) = (c'_1, \ldots, c'_{k-2}, x_\beta)$.

We will use the coordinate array $x = (x_1, \ldots, x_{k-1})$ defined as follows: for all $2 \leq \ell \leq k-2$, $x_\ell = C(c_1, \ldots, c_{k-\ell}, c'_{k-\ell-1}, \ldots, c'_{k-2})$, and we set $x_1 = x_2$ and $x_{k-1} = x_{k-2}$. See Table 4.3.

	x_1	x_2	$\ldots$	x_ℓ	$\ldots$	x_{k-2}	x_{k-1}
c_1	1	1	$\ldots$	1	$\ldots$	1	1
c_2	1	1	$\ldots$	1	$\ldots$	1	1
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$		
$c_{k-\ell}$	1	1	$\ldots$	1			
$\ldots$	$\ldots$	$\ldots$	$\ldots$				
c_{k-2}	1	1					
c'_1						1	1
$\ldots$					$\ldots$	$\ldots$	$\ldots$
$c'_{k-\ell-1}$				1	$\ldots$	1	1
$\ldots$			$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
c'_{k-3}	1	1	$\ldots$	1	$\ldots$	1	1
c'_{k-2}	1	1	$\ldots$	1	$\ldots$	1	1

Table 4.3: Case 2 coordinate array used in the $\alpha\beta$ path.

We now describe a path of length k from α to β. Follow Figure 4-5 for an illustration of the path. All of the internal vertices along the path use the coordinate x, and the existence of these vertices follows from the definition of x.

We first show a path of length 3 from α to $\gamma = (a_1, \ldots, a_{k-i}, a_{k-i+1}, \ldots, a_{k-4}, b'_{k-1}, b'_k, x) \in L_4$, where a_ℓ is the all 1 vector for all $\ell > k-i$ and if $j = 3$, b'_{k-1} is the all 1 vector (otherwise b'_{k-1} is defined for β).

If $i > 3$, using an a-type back edge we go from α to $(a_1, \ldots, a_{k-i}) \in A_i$, and using another a-type back edge we go to $(a_1, \ldots, a_{k-i}, a_{k-i+1}, \ldots, a_{k-4}, b'_{k-1}, b'_k, x) \in L'_4$. Now using a coordinate-change edge we go to $\gamma \in L_4$, without actually changing any coordinates. This corresponds to the green path in Figure 4-5.

If $i = 3$, we first take a coordinate-change edge to $(a_1, \ldots, a_{k-3}, b_k, x) \in L'_3$, then take a vector-change edge to change b_k to b'_k and arrive at $(a_1, \ldots, a_{k-3}, b'_k, x) \in L_3$, and then use a swap edge to go to $\gamma \in L_4$. This corresponds to the purple path in Figure 4-5.

Next, we show a path of length 3 from $\gamma' = (a_1, a_2, b'_5, \ldots, b'_k, x) \in L_{k-4}$ to β, where b'_ℓ is the all 1s vector for $\ell < k-j+3$ and a_2 is the all 1s vector if $j = k-1$. Then we show that we can go from γ to γ' in $k-6$ when $k \geq 6$. We handle $k = 5$ in the last paragraph of Case 2. For now, we assume that $k > 5$.

Now, we use swap edges to go from $\gamma \in L_4$ to L_{k-2} where for all $4 \leq r < k-2$ to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b'_{k-r+2}, where b'_{k-r+2} has already been defined as part of β if $r \leq j-1$, and otherwise we define b'_{k-r+2} as the all 1s vector. This path of swap edges is of length $k-6$, so the path is thus far of length

$k-3$. After traversing these edges, we end at $\gamma' = (a_1, a_2, b'_5, \ldots, b'_k, x) \in L_{k-2}$. The $\gamma\gamma'$ path is shown with black dashed lines in Figure 4-5. If $j < k-1$, then using a coordinate-change edge we go to $(a_1, a_2, b'_5, \ldots, b'_k, x) \in L'_{k-2}$ without actually changing any coordinates. Then we take a b-type back edge to go to $(b'_{k-j+3}, \ldots, b'_k) \in B_j$, and using another b-type back edge we go to β. This corresponds to the orange path in Figure 4-5.

If $j = k-1$, we use a swap edge to go to $(a_1, b'_4, \ldots, b'_k, x) \in L_{k-1}$, then use a vector-change edge to change a_1 to a'_1 and arrive at L'_{k-1}, and then use a coordinate-change edge to β. This corresponds to the red path in Figure 4-5. The total length of the path is hence k.

This concludes Case 2 when $k > 5$. Now suppose that $k = 5$. We have already shown that there is a path of length 3 from α to $\gamma = (a_1, b'_4, b'_5, x) \in L_4$. If $j = 4$, we get to β from γ using a vector-change edge to $(a'_1, b'_4, b'_5, x) \in L'_4$, and then coordinate-change edge. So suppose that $j = 3$. Then there is a path of length 3 from $\gamma'' = (a_1, a_2, b_5, x) \in L_3$ to β as follows: take a vector-change edge to $(a_1, a_2, b'_5, x) \in L'_3$, then a back edge to $(b'_5) \in B_3$, and then another back edge to β. Now to go from α to γ'' in at most 2, we do the following: If $i = 3$, we take a coordinate-change edge from α to γ''. If $i = 4$, we first take a coordinate-change edge to $(a_1, b_4, b_5, x) \in L_4$, and then a swap edge to γ''.

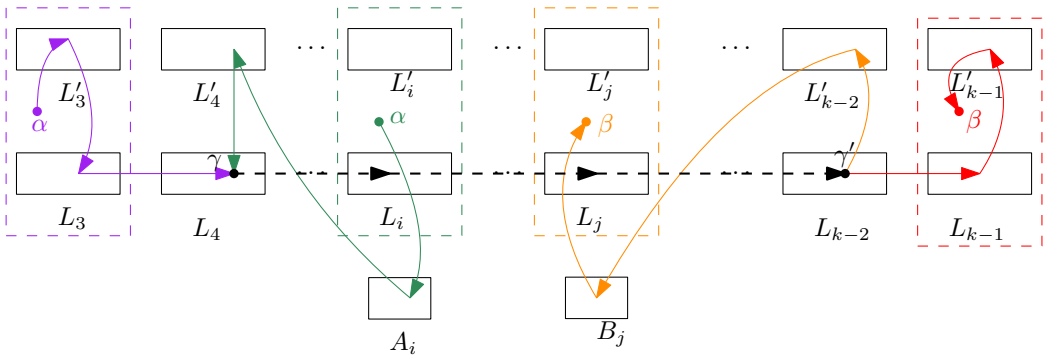

Figure 4-5: $\alpha\beta$ path in Case 3. Different cases of α and β are shown with different colors. The path between $\gamma \in L_4$ and $\gamma' \in L_{k-2}$ is shown with black dashed lines. Solid lines indicate edges. A node in a dashed box containing two sets means that the node is in either set.

Case 4: $\alpha \in V \setminus \{u, v\}$ and $\beta \in B$: For the majority of this case, we assume that $k > 5$, and then at the end of this case we include a paragraph to handle $k = 5$. Let $\beta \in B_j$ and $\beta = (b_{k-j+3}, \ldots, b_k)$.

An intermediate node on the path from α to β will be some $\gamma \in L_4$ of the form $(a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x)$, where b_k has already been defined, b_{k-1} has been defined unless $j = 3$ in which case b_{k-1} is the all 1s vector, each $a_1, \ldots, a_{k-4}$ is some vector from the appropriate set $S_1, \ldots, S_{k-4}$, and x is a coordinate array satisfying the following conditions: For $\ell = 4, \ldots, k-4$, $x_\ell = C(a_1, \ldots, a_{k-\ell}, b_{k-\ell+1}, \ldots, b_k)$, where for $r < k-j+3$, b_r is the all 1s vector, and for $r > k-4$, a_r is the all 1s vector. See

Table 4.2. Each vertex in the path from α to β that we will specify exists due to the definition of x.

First, we show that for *all* γ of the above form, there is a path of length at most $k-3$ from γ to β. Using swap edges we go from γ to $\gamma' = (a_1, a_2, b_5, \ldots, b_k, x) \in L_{k-2}$, where b_s has already been defined for $s \geq k-j+3$, and b_s is the all 1s vector for $s < k-j+3$. To construct this path from γ to γ', for all $r = 4, \ldots, k-2$, to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b_{k-r+2}. Then from γ' we take an edge to $(a_1, a_2, b_5, \ldots, b_k, x) \in L'_{k-2}$ and then take a b-type back edge to β. The $\gamma\beta$ path is specified with dashed black lines representing subpaths and black edges in Figure 4-6.

To complete the path from α to β, we will show a path of length at most 3 from α to *some* γ of the above form. We divide into cases based on where α is. Because $a_1, \ldots, a_{k-4}$ are unspecified in the definition of γ, we have the freedom to specify these vectors in the following cases.

Case 4a: $\alpha \in L_k \cup L_{k+1}$**.** From α we take a fixed edge to u and then from u we take a fixed edge to $(a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x) \in L'_4$, where we define a_i for $i = 1, \ldots, k-4$ as the all 1s vector. Using a coordinate-change edge we go to $\gamma = (a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x) \in L_4$ without actually changing any coordinates. This path is illustrated in purple in Figure 4-6.

Case 4b: $\alpha \in L_1 \cup L_2$**.** If $\alpha \in L_1$ then let $\alpha = (a_1, \ldots, a_{k-1})$, and if $\alpha \in L_2$ then let $\alpha = (a_1, \ldots, a_{k-2}x_\alpha)$, in which case a_{k-1} is the all 1s vector. We begin by taking an edge from α to $(a_1, \ldots, a_{k-2}, x) \in L_2$ either by taking a swap edge or a coordinate-change edge (depending on whether α is in L_1 or L_2). We then use two swap edges to go from L_2 to L_4, where to go from L_r to L_{r+1} we change the vector a_{k-r} to the vector b_{k-r+2}. So we arrive at $\gamma = (a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x)$. This path is illustrated in green in Figure 4-6.

Case 4c: $\alpha \in L_3 \cup L'_3 \cup B_3$**.** Let $\alpha = (a_1, \ldots, a_{k-3}, b'_k, x_\alpha) = (c_1, \ldots, c_{k-2}, x_\alpha)$ if $\alpha \in L_3 \cup L'_3$, and let $\alpha = (b'_k) = (c_{k-2})$ if $\alpha \in B_3$, in which case a_ℓ is the all 1s vector for $\ell = 1, \ldots, k-3$. From α, we take a back edge or a coordinate-change edge to $(a_1, \ldots, a_{k-3}, b'_k, x) \in L'_3$. Then take a vector-change edge to change b'_k to b_k and arrive at $(a_1, \ldots, a_{k-3}, b_k, x) \in L_3$. Then using a swap edge we proceed to $\gamma = (a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x) \in L_4$. This path is illustrated in red in Figure 4-6.

Case 4d: $\alpha \in L_i \cup L'_i \cup B_i \cup A_i$ **for** $i = 4, \ldots, k-1$**.** First if $\alpha \notin A_i$, we show how to get to a node in A_i. Then we show how to go to γ from any node in A_i using a path of length 2. Suppose that $\alpha = (a_1, \ldots, a_{k-i}, b'_{k-i+3}, \ldots, b'_k, x_\alpha)$ if $\alpha \in L_i \cup L'_i$, and suppose that $\alpha = (b'_{k-i+3}, \ldots, b'_k)$ if $\alpha \in B_i$, in which case we assume that a_ℓ is the all 1s vector for $\ell = 1, \ldots, k-i$. Take an a-type or ba-type back edge to $(a_1, \ldots, a_{k-i}) \in A_i$. Now we show how to proceed from this vertex to γ.

We take an a-type back edge to $(a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x) \in L'_4$, and then take a

coordinate-change edge to $\gamma = (a_1, \ldots, a_{k-4}, b_{k-1}, b_k, x) \in L_4$ without actually changing any coordinates. This path is illustrated in orange in Figure 4-6.

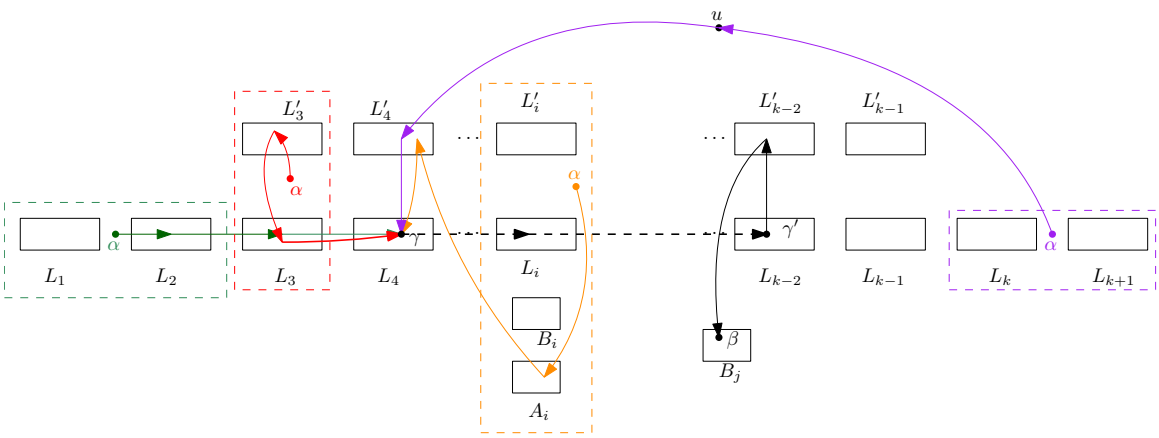

Figure 4-6: $\alpha\beta$ path in Case 4. Different cases of α are shown with different colors. The path between $\gamma \in L_4$ and $\gamma' \in L_{k-2}$ is shown with black dashed lines. Solid lines indicate edges. A node in a dashed box containing two sets means that the node is in either set.

This completes Case 4 when $k > 5$. Now we let $k = 5$ and specify a path of length at most 5 from α to β. Since $k = 5$, we have that $B = B_3$ so let $\beta = (b_k) \in B_3$. Regardless of its location in the graph, α has at most 4 vectors in its representation. Let a_1, a_2, a_3, a_4 be a set consisting of all of the vectors in the representation of α plus some all 1s vectors if α has fewer than 4 vectors in its representation. Let x be the coordinate array consisting of 4 copies of the coordinate $C(a_1, a_2, a_3, a_4, b_k)$. From α we can go to $\gamma = (a_1, a_2, b'_5, x) \in L_3$ using a path of length 3. To do this, we either take a coordinate-change edge or a back edge to a vertex in $L_2 \cup \ldots \cup L_4 \cup L'$, and then take swap or coordinate-change edges to γ. Then from γ, we take a vector-change edge to $(a_1, a_2, b_5, x) \in L'_3$, and then a back edge to β.

Case 5: $\alpha \in B$ and $\beta \in V \setminus \{u, v\}$. Suppose that $\alpha \in B_i$. First suppose that $i > 3$. We know that by Claim 1, there is a $\beta' \in L'_4$ such that $d(\beta', \beta) \leq k-2$. We show that there is a path of length 2 from α to β'. Suppose that $\beta' = (a_1, \ldots, a_{k-4}, b'_{k-1}, b'_k, x_{\beta'})$. From α, take a ba-type back edge to $(a_1, \ldots, a_{k-i}) \in A_i$, and then take an a-type back edge to β'.

Now suppose that $i = 3$ and $\alpha = (b_k)$. Suppose β is in the jth level. We represent β as follows: If $\beta \in L_j \cup L'_j$ for $j = 2, \ldots, k$, let $\beta = (a_1, \ldots, a_{k-j}, b'_{k-j+3}, \ldots, b'_k, x_\beta)$. If $\beta \in L_1$, let $\beta = (a_1, \ldots, a_{k-1})$, and if $\beta \in L_{k+1}$ let $\beta = (b'_2, \ldots, b'_k)$. Finally, if $\beta \in A_j$ for some $j = 4, \ldots, k-1$, let $\beta = (a_1, \ldots, a_{k-j})$ and if $\beta \in B_j$ for some $j = 3, \ldots, k-2$, let $\beta = (b_{k-j+3}, \ldots, b_k)$. Define the coordinate array x such that for all $\ell = 1, \ldots, k-1$, $x_\ell = C(a_1, \ldots, a_{k-j}, b'_{k-j+3}, \ldots, b'_k, b_k)$, where for $j = 1$, $x_\ell = C(a_1, \ldots, a_{k-1}, b_k)$ and for $j = k+1$, $x_\ell = C(b'_2, \ldots, b'_k, b_k)$. We exhibit a path of length at most k to β: First take a b-type back edge to $(a_1, \ldots, a_{k-3}, b_k, x) \in L'_3$, where a_ℓ is the all 1s vector for $\ell > k - j$. Then take a vector-change edge to change

b_k to b'_k and arrive at $(a_1, \ldots, a_{k-3}, b'_k, x) \in L_3$.

If $\beta \in L_j$ for $j = 1, 2, k, k+1$, take swap edges to β. This path is of length $2 + |j-3| \leq k$, and it is shown in Figure 4-7: For $j = 1, 2$ it is shown with black and green and for $j = k, k+1$ it is shown with black (solid and dashed) and blue.

If $\beta \in L_j \cup L'_j$ for some $j = 3, \ldots, k-1$, take swap edges to get to $(a_1, \ldots, a_{k-j}, b'_{k-j+3}, \ldots, b_{k-j}, x) \in L_j$. So far the path is of length $2 + |j-3| \leq k-2$. If $\beta \in L'_j$, take one coordinate-change edge to β. Note that the $\alpha\beta$ path in this case is of length $k-1$. If $\beta \in L_j$, take one coordinate-change edge to the copy of β in L'_j, and then another edge to $\beta \in L_j$. These paths are shown with black (dashed and solid) and purple in Figure 4-7.

If $\beta \in A \cup B$, then there is a $\beta' \in L'_j$ for some j, such that $d(\beta', \beta) = 1$. Above, we showed how to get to any $\beta' \in L'_j$ with a path of length at most $k-1$. For $\beta \in A_j$ the path is shown with black and orange in Figure 4-7, and for $\beta \in B_j$ it is shown with black and red.

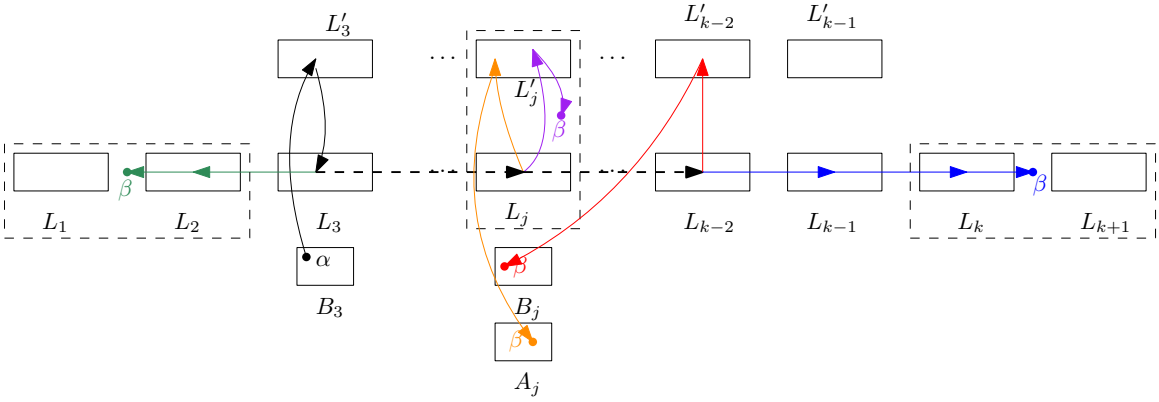

Figure 4-7: $\alpha\beta$ paths in Case 5 when $\alpha \in B_3$. Different cases of β are shown with different colors. The path between L_3 and L_{k-2} is shown with black dashed lines. Solid lines indicate edges. A node in a dashed box containing two sets means that the node is in either sets.

4.5 YES instance of k-OV implies diameter $\geq 2k-1$

Let $a_1, a_2, \ldots, a_k \in S$ be orthogonal. Let $\alpha = (a_1, \ldots, a_{k-1}) \in L_1$ and let $\beta = (a_2, \ldots, a_k) \in L_{k+1}$. We claim that $d(\alpha, \beta) \geq 2k-1$. Let P be a shortest path from α to β.

Recall that *level* i is defined as $L_i \cup L'_i \cup A_i \cup B_i$. Recall that a *layer* refers to an individual set L_i or L'_i.

We begin with two observations.

Observation 4.5.1. *The only edges that go from a vertex in some level i to a vertex in a level $j > i$ are swap edges between L_i and L_{i+1}.*

The next observation follows directly from Observation 4.5.1.

Observation 4.5.2. *For all $i < j$, any path from a vertex in level i to a vertex in level j uses a vertex in every layer $L_i, \ldots, L_j$.*

We claim that if P contains either u or v (or both) then the length of P is at least $2k-1$. If P contains v then P must go from L_1 to L_{k-2} to L'_{k-2} to v to L_2 to L_{k+1}, which costs at least $2k-1$ by Observation 4.5.2. The argument is symmetric for u: If P contains u then P must go from L_1 to L_k to u to L'_4 to L_4 to L_{k+1}, which costs at least $2k-1$ by Observation 4.5.2. From now on we assume that P does not contain u or v.

Next, we claim that if P contains a *ba*-type back edge then the length of P is at least $2k-1$. Since the only edges to B are from L'_{k-2}, and the only edges from A are to L'_4, if P contains a *ba*-type back edge then P must go from L_1 to to L_{k-2} to L'_{k-2} to B to A to L'_4 to L_4 to L_{k+1}. This costs at least $2k-1$ by Observation 4.5.2. From now on we assume that P does not contain any *ba*-type back edges.

We make one more observation, which follows from Observation 4.5.2 and the following fact: Ignoring *ba*-type edges, all edges from A go to L'_4, all edges to A are from a level that is at least 4, all edges to B are from L'_{k-2}, and all edges from B are to a level that is at most $k-2$.

Observation 4.5.3. *If P visits A, then P visits L_4 both before and after visiting A. If P visits B, then P visits L_{k-2} both before and after visiting B.*

Lemma 4.5.1. *Fix $i = 2, \ldots, k$ and let γ_1 and γ_2 be the first and last vertices in L_i that P visits, respectively. Let P_1 be the subpath of P from α to γ_1 and let P_2 be the subpath of P from γ_2 to β. If P_1 does not visit B, then γ_1 has the prefix $(a_1, \ldots, a_{k-i})$, and if P_2 does not visit A, then γ_2 has the suffix $(a_{k-i+3}, \ldots, a_k, x)$ for some coordinate array x.*

Proof. Suppose that P_1 does not visit B. By Observation 4.5.2, all vertices in P_1 except for γ_1 are in levels below i. Then by construction, all vertices on P_1 contain a prefix of the form $(a'_1, \ldots, a'_{k-i})$ where $a'_i \in S_i$. Since P_1 does not visit B, u, or v, all edges on P_1 do not change any of the vectors $(a'_1, \ldots, a'_{k-i})$. Thus, these vectors are the same for γ_1 and α; that is, γ_1 has the prefix $(a_1, \ldots, a_{k-i})$.

Now suppose P_2 does not visit A. By Observation 4.5.2, all vertices in P_2 except for γ_2 are in levels above i. Then by construction, all vertices on P_2 contain a suffix of the form $(a'_{k-i+3}, \ldots, a'_k)$ where $a'_i \in S_i$. Since P_2 does not visit A, u, or v, all edges on P_2 do not change any of the vectors $(a'_{k-i+3}, \ldots, a'_k)$. Thus, these vectors are the same for γ_2 and β; that is, γ_2 ends with $(a_{k-i+3}, \ldots, a_k, x)$ for some x. □

To analyze the length of P, we will condition on which L_i are in a *loop*, which is defined as follows.

Definition 4.5.1 (loop)**.** *For all indices i, we say that L_i* is in a loop *if P visits L_i at least twice.*

An outline of the remainder of the proof is as follows. First we will define the notion of a set L_i in a loop that *covers* a set L_j not in a loop. Then we will prove

that every L_i is either in a loop or covered. Then, we will partition L where each piece of the partition is composed of a set of consecutive layers that are all in loops as well as the layers covered by these layers. Then we will compute the length of the subpath of P in each of these pieces. Finally, we will take the sum over all of these subpaths and the edges between them.

Definition 4.5.2 (cover)**.** *For all $i = 3, \ldots, k-1$, we say that L_i* covers *L_{i-1} if the following conditions are satisfied:*

1. *L_{i-1} is not in a loop,*
2. *L_i is in a loop, and*
3. *If $\gamma_1 = (a'_1, \ldots, a'_{k-i}, a'_{k-i+3}, \ldots, a'_k, x_{\gamma_1})$ and $\gamma_2 = (a''_1, \ldots, a''_{k-i}, a''_{k-i+3}, \ldots, a''_k, x_{\gamma_2})$ are the first and last vertices from L_i that P visits, then $a'_{k-i+3} \neq a''_{k-i+3}$ and $x_{\gamma_1} \neq x_{\gamma_2}$.*

Symmetrically, for all $i = 3, \ldots, k-1$, we say that L_i covers *L_{i+1} if the following conditions are satisfied:*

1. *L_{i+1} is not in a loop,*
2. *L_i is in a loop, and*
3. *If $\gamma_1 = (a'_1, \ldots, a'_{k-i}, a'_{k-i+3}, \ldots, a'_k, x_{\gamma_1})$ and $\gamma_2 = (a''_1, \ldots, a''_{k-i}, a''_{k-i+3}, \ldots, a''_k, x_{\gamma_2})$ are the first and last vertices from L_i that P visits, then $a'_{k-i} \neq a''_{k-i}$ and $x_{\gamma_1} \neq x_{\gamma_2}$.*

Additionally, we say that L_4 covers both L_3 and L_2 if P visits A. Symmetrically, we say that L_{k-2} covers both L_{k-1} and L_k if P visits B.

Lemma 4.5.2. *For all $i = 2, \ldots, k$, L_i is either in a loop or covered.*

Proof. Suppose for contradiction that there exists i with $2 \leq i \leq k$ such that L_i is neither in a loop nor covered. Let γ be the single vertex in L_i that P visits. Let P_1 be the part of P from α to γ, and let P_2 be the part of P from γ to β. By Observation 4.5.1, except for γ, P_1 is entirely contained in levels below i, and except for γ, P_2 is entirely contained in levels above i.

We claim that P_1 does not visit B. The only edges to B from another set are from L'_{k-2}, so P_1 can only visit B if i is either $k-1$ or k. In this case L_i is covered, by the final part of the definition of cover. Thus P_1 does not visit B.

Similarly, P_2 does not visit A because the only edges from A to another set are to L'_4, so P_2 can only visit A if i is either 3 or 2, in which case L_i is covered, by the final part of the definition of cover. Since P_1 does not visit B and P_2 does not visit A, Lemma 4.5.1 implies that $\gamma = (a_1, \ldots, a_{k-i}, a_{k-i+3}, \ldots, a_k, x)$ for some coordinate array x.

Now consider the two edges incident to γ on the path P. Suppose that they are $\gamma_1\gamma$ and $\gamma\gamma_2$. Since L_i is not in a loop, Observation 4.5.2 implies that $\gamma_1\gamma$ is a swap edge from L_{i-1} to L_i. Hence, if $i \neq 2$, then $\gamma_1 = (a_1, \ldots, a_{k-i}, a'_{k-i+1}, a_{k-i+4}, \ldots, a_k, x)$

for some a'_{k-i+1}, and if $i = 2$, then $\gamma_1 = \alpha$. Symmetrically, Observation 4.5.2 implies that $\gamma\gamma_2$ is a swap edge from L_i to L_{i+1}. Hence, if $i \neq k$, then $\gamma_2 = (a_1, \ldots, a_{k-i-1}, a'_{k-i+2}, a_{k-i+3} \ldots, a_k, x)$ for some a'_{k-i+2}, and if $i = k$, then $\gamma_2 = \beta$.

By Observation 4.5.2, γ_1 is the last vertex in L_{i-1} that P visits and γ_2 is the first vertex in L_{i+1} that P visits. Suppose $i \neq 2$ and let γ'_1 be the first vertex in L_{i-1} that P visits. By condition 3 of the definition of cover, since L_{i-1} does not cover L_i, γ'_1 either contains a'_{k-i+1} or x in its representation. However, since P_1 does not visit B, Lemma 4.5.1 implies that γ'_1 has the prefix $(a_1, \ldots, a_{k-i+1})$. Thus, either $a_{k-i+1} = a'_{k-i+1}$ or γ'_1 contains x in its representation. If $a_{k-i+1} = a'_{k-i+1}$, then γ_1 has the prefix $(a_1, \ldots, a_{k-i+1})$, so for all $j = 1, \ldots, k-i+1$ we have $a_j[x_{i-1}] = 1$. If γ'_1 contains x in its representation, then since γ'_1 has the prefix $(a_1, \ldots, a_{k-i+1})$, we have the same conclusion that for all $j = 1, \ldots, k-i+1$, $a_j[x_{i-1}] = 1$.

If $i = 2$ then the edge between $\alpha = \gamma_1$ and γ implies that for all $j = 1, \ldots k-1$ we have $a_j[x_1] = 1$. So, regardless of i, we have that for all $j = 1, \ldots, k-i+1$, $a_j[x_{i-1}] = 1$.

Symmetrically, suppose $i \neq k$ and let γ'_2 be the last vertex in L_{i+1} that P visits. Since L_{i+1} does not cover L_i, γ'_2 either contains a'_{k-i+2} or x in its representation. However, since P_2 does not visit A, Lemma 4.5.1 implies that γ'_2 has the suffix $(a_{k-i+2}, \ldots, a_k, y)$ for some y. Thus, either $a_{k-i+2} = a'_{k-i+2}$ or γ'_2 contains x in its representation. If $a_{k-i+2} = a'_{k-i+2}$, then γ_2 has the suffix $(a_{k-i+2}, \ldots, a_k, x)$, so for all $j = k-i+2, \ldots, k$ we have $a_j[x_{i-1}] = 1$. If γ'_2 contains x in its representation, then since γ'_2 has the suffix $(a_{k-i+2}, \ldots, a_k, x)$, we have the same conclusion that for all $j = k-i+2, \ldots, k$ we have $a_j[x_{i-1}] = 1$.

If $i = k$ then the edge between γ and $\gamma_2 = \beta$ implies that for all $j = 2, \ldots k$ we have $a_j[x_{k-1}] = 1$. So, regardless of i, we have that for all $j = k-i+2, \ldots k$, $a_j[x_{i-1}] = 1$.

Thus, we have shown that for each $j = 1, \ldots k$, we have $a_j[x_{i-1}] = 1$. This contradicts the fact that $a_1, \ldots, a_k$ are orthogonal, completing the proof. □

In the next lemma, we bound the length of a subpath of P passing from L_i to L_j for some $i < j$ conditioned on the layers being covered or not.

Lemma 4.5.3. *Let i and j be such that $2 \leq i \leq j \leq k$, neither L_{i-1} nor L_{j+1} are in a loop, and for all $\ell = i, \ldots, j$, L_ℓ is in a loop. Let c be the total number of layers that are covered by $L_i, \ldots, L_j$. The subpath P' of P from the first time P visits L_i to the last time P visits L_j is of length at least $2(j-i)+c+1$.*

Proof. We will show the contrapositive: for any c', if P' is of length at most $2(j-i)+c'$, then $L_i, \ldots, L_j$ cover a total of at most $c'-1$ layers.

First, we note that since neither L_{i-1} nor L_{j+1} are in a loop, Observation 4.5.2 implies that P' is entirely contained in levels $i, \ldots, j$, and that no vertex on $P \setminus P'$ is in a level $i, \ldots, j$. Then, since L_ℓ is in a loop for all $\ell = i, \ldots, j$, P' must contain at least two vertices from each such L_ℓ. Let Z be a subset of the vertices of P' formed by taking exactly two arbitrary vertices on P' from each such layer L_ℓ. The size of Z is exactly $2(j-i)+2$.

By the definition of cover, any layer can only cover the layers at most two above and below itself, so the layers $L_i, \ldots, L_j$ can only cover a total of at most 4 layers. Thus, the lemma is trivially true for $c' \geq 5$. The lemma is also trivially true for $c' \leq 1$. We will condition on the length of P'; that is, whether $c' = 2$, 3, or 4.

Case 1: $c' = 2$. P' contains $2(j-i)+2$ edges and $2(j-i)+3$ vertices, so P' contains exactly one vertex γ in addition to Z. Since $Z \subseteq L$ and all paths from L to B as well as all paths from A to L go through L', $\gamma \notin A \cup B$. Thus, the only way that a layer in $L_i, \ldots, L_j$ can cover another layer is if there exist two vertices on P with different coordinate arrays (by condition 3 in the definition of cover). Every edge with both endpoints in L is a swap edge, which by definition connects two vertices with the same coordinate array. Thus, $\gamma \notin L$. We have shown that $\gamma \notin A \cup B \cup L$, so $\gamma \in L'$.

Let ℓ be such that $\gamma \in L'_\ell$. Let γ_1 and γ_2 be the vertices right before and right after γ on P, respectively. Since L_ℓ contains the only vertices in L that are adjacent to some vertex in L'_ℓ, γ_1 and γ_2 are both in L_ℓ. Let $\gamma_1 = (a'_1, \ldots, a'_{k-i}, a'_{k-i+3}, \ldots, a'_k, x_{\gamma_1})$ and let $\gamma_2 = (a''_1, \ldots, a''_{k-i}, a''_{k-i+3}, \ldots, a''_k, x_{\gamma_2})$. Since P' contains no other vertices in L_ℓ besides γ_1 and γ_2, and $P \setminus P'$ contains no vertices in L_ℓ, γ_1 and γ_2 are the first and last vertices from L_ℓ that P visits. Thus, by definition, if L_ℓ covers $L_{\ell-1}$ then $a'_{k-i+3} \neq a''_{k-i+3}$ and $x_{\gamma_1} \neq x_{\gamma_2}$. Similarly, if L_ℓ covers $L_{\ell+1}$ then $a'_{k-i} \neq a''_{k-i}$ and $x_{\gamma_1} \neq x_{\gamma_2}$. Thus, if L_ℓ covers both $L_{\ell-1}$ and $L_{\ell+1}$, then $a'_{k-i+3} \neq a''_{k-i+3}$, $a'_{k-i} \neq a''_{k-i}$, and $x_{\gamma_1} \neq x_{\gamma_2}$. However, there are only two edges on P' between γ_1 and γ_2 and each of them is either a vector-change edge or a coordinate-change edge. This means that only two out of the three above non-equalities can be true. Thus, L_ℓ can only cover at most one level. Since the edges on P' that are not incident to γ are swap edges and do not change the coordinate array, no other layer in $L_i, \ldots, L_j$ except for L_ℓ covers any layer. Thus, we have shown that $L_i, \ldots, L_j$ cover a total of at most one layer, as desired.

Case 2: $c' = 3$. P' contains $2(j-i)+3$ edges and $2(j-i)+4$ vertices, so P' contains exactly two vertices in addition to Z. We would like to show that $L_i, \ldots, L_j$ cover a total of at most two layers. The only way for $L_i, \ldots, L_j$ to cover more than two layers is to use the last part of the definition of cover, which allows L_4 or L_{k-2} to cover two layers. Thus, we will show that if $i = 4$ and L_4 covers L_3 and L_2, or if $j = k-2$ and L_{k-2} covers L_{k-1} and L_k, then no other layers can be covered by $L_i, \ldots, L_j$.

Suppose $i = 4$ and L_4 covers L_3 and L_2. By the definition of cover, P visits A. Then, since all vertices in $P \cap L_4$ are on P', Observation 4.5.3 implies that P' visits A. Therefore, the two vertices on P' in addition to Z are one vertex in A and one vertex in L'_4. Thus, the only edges that P' uses are swap edges with endpoints in $L_4 \cup \cdots \cup L_j$, an a-type back edge from $L_4 \cup \cdots \cup L_j$ to $A_4 \cup \cdots \cup A_j$, an a-type back edge from $A_4 \cup \cdots \cup A_j$ to L'_4, and either a vector-change or coordinate-change edge from L'_4 to L_4. It will be important to note that by construction, all of these edges go from a vertex that contains a vector $a'_{k-j} \in S_{k-j}$ to another vertex containing the same vector $a'_{k-j} \in S_{k-j}$. This is because the only edges that change a vector S_{k-j}

to a different vector in S_{k-j} are vector-change edges between L_j and L'_j or between L_{j+3} and L'_{j+3}, which P' does not use.

Our goal is to show that L_j does not cover L_{j+1}. Let $\gamma_1 = (a''_1, \ldots, a''_{k-j}, a''_{k-j+3}, \ldots, a''_k, x_{\gamma_1})$ and $\gamma_2 = (a'''_1, \ldots, a'''_{k-j}, a'''_{k-j+3}, \ldots, a'''_k, x_{\gamma_2})$ be the two vertices in L_j that P' visits. Since $P \setminus P'$ contains no vertices in L_j, γ_1 and γ_2 are the first and last vertices from L_j that P visits. Thus, if L_j covers L_{j+1}, then $a''_{k-j} \neq a'''_{k-j}$. But we have already shown that this cannot be the case since every edge on P' does not change the vector $a'_{k-j} \in S_{k-j}$.

The $j = k-2$ case is completely symmetric to the $i = 4$ case, but we include it for completeness. Suppose $j = k-2$ and L_{k-2} covers L_{k-1} and L_k. This means that P visits B. Then, since all vertices in $P \cap L_{k-2}$ are on P', Observation 4.5.3 implies that P' visits B. Therefore, the two vertices on P' in addition to Z are one vertex in L'_{k-2} and one vertex in B. Thus, the only edges that P' uses are swap edges with endpoints in $L_i \cup \cdots \cup L_{k-2}$, either a vector-change or coordinate-change edge from L_{k-2} to L'_{k-2}, a b-type back edge from L'_{k-2} to $B_i \cup \cdots \cup B_{k-2}$, and a b-type back edge from $B_i \cup \cdots \cup B_{k-2}$ to $L_i \cup \cdots \cup L_{k-2}$. It will be important to note that by construction, all of these edges go from a vertex that contains a vector $a'_{k-i+3} \in S_{k-i+3}$ to another vertex containing the same vector $a'_{k-i+3} \in S_{k-i+3}$.

Our goal is to show that L_i does not cover L_{i-1}. Let $\gamma_1 = (a''_1, \ldots, a''_{k-i}, a''_{k-i+3}, \ldots, a''_k, x_{\gamma_1})$ and $\gamma_2 = (a'''_1, \ldots, a'''_{k-i}, a'''_{k-i+3}, \ldots, a'''_k, x_{\gamma_2})$ be the two vertices in L_i that P' visits. Since $P \setminus P'$ contains no vertices in L_i, γ_1 and γ_2 are the first and last vertices from L_i that P visits. Thus, if L_i covers L_{i-1}, then $a''_{k-i+3} \neq a'''_{k-i+3}$. But we have already shown that this cannot be the case since every edge on P' does not change the vector $a'_{k-i+3} \in S_{k-i+3}$.

Case 3: $c' = 4$**.** P' contains $2(j-i)+4$ edges and $2(j-i)+5$ vertices, so P' contains exactly three vertices in addition to Z. We would like to show that $L_i, \ldots, L_j$ cover a total of at most three layers. Suppose for contradiction that $L_i, \ldots, L_j$ cover four layers. The only way for this to happen is if $L_4 = L_i$ covers L_3 and L_2, and $L_{k-2} = L_j$ covers L_{k-1} and L_k. This means that P visits both A and B. Then, since all vertices in $P \cap L_{k-2}$ and all vertices in $P \cap L_4$ are on P', Observation 4.5.3 implies that P' visits both A and B. Since the only edges to B from another set are from L'_{k-2} and the only edges from A to another set are to L'_4, P' must contain a vertex in each of A, B, L'_{k-2}, and L'_4. However, P' only contains three vertices in addition to Z, a contradiction. □

We will now take the sum over all of the subpaths defined by Lemma 4.5.3. Form a partition $\mathcal{P}$ of $2, \ldots, k$ where each piece of $\mathcal{P}$ contains a maximal interval $i, \ldots, j$ such that $L_i, \ldots, L_j$ are all in loops, as well as the layers that $L_i, \ldots, L_j$ cover. (The maximality condition means that either $i = 2$ or L_{i-1} is not in a loop, and either $j = k$ or L_{j+1} is not in a loop.) To make this a true partition of $2, \ldots, k$, for any ℓ that has been placed in two pieces of the partition due to being covered by multiple layers, we remove ℓ from an arbitrary one of these two pieces. Now, by Lemma 4.5.2, every value $2, \ldots, k$ is in exactly one piece of $\mathcal{P}$.

Consider a piece $i, \ldots, j$ of $\mathcal{P}$, and let $i' \geq i$ and $j' \leq j$ be such that $L_{i'}, \ldots, L_{j'}$ are each in a loop and the remaining layers in $L_i, \ldots, L_j$ are covered. Let c be the number of covered layers in $L_i, \ldots, L_j$. That is, $j' - i' + c = j - i$. By Observation 4.5.2 and the maximality condition of $\mathcal{P}$, any path with both endpoints in $L'_i, \ldots, L'_j$ is entirely contained within levels $i', \ldots, j'$.

Let $P_{i,j}$ and $P_{i',j'}$ be the subpaths of P in the graph induced by levels $i, \ldots, j$ and levels $i', \ldots, j'$, respectively. By Lemma 4.5.3, $P_{i',j'}$ is of length at least $2(j'-i')+c+1$. Adding an edge to each of the c covered levels, $P_{i,j}$ is of length at least $2(j'-i'+c)+1 = 2(j-i)+1$.

Let p be the number of pieces of $\mathcal{P}$. We first calculate the sum over the lengths of all $P_{i,j}$. Since $\mathcal{P}$ partitions $2, \ldots, k$, we have $\sum_{(i,\ldots,j)\in\mathcal{P}} 2(j-i)+1 = 2(k-1-p)+p = 2k-p-2$. In addition to the edges in each $P_{i,j}$, P also contains at least $p-1$ edges such that each endpoint is in a different piece of $\mathcal{P}$, as well as an edge from L_1 to L_2 and an edge from L_k to L_{k+1}. Thus, the length of P is at least $2k-p-2+(p-1)+2 = 2k-1$.

4.6 The $k = 4$ case

We mainly use the construction of [Bon21b] with a few alterations, to get a reduction from 4-OV to directed *unweighted* Diameter. Note that the lower bound of [Bon21b] is for directed weighted Diameter.

Given a 4-OV instance S where each vector in S is of length d, that is, there are d coordinates, we create a graph G such that if S is a NO case then the diameter of G is at most 4 and if S is a YES case the diameter of G is at least 7. See Figure 4-8.

We make 4 copies of S and call them $S_1, \ldots, S_4$. The vertex set of our graph is essentially the same as [Bon21b], and we redefine it for completeness. The graph G consists of layers L_i for $i = 1, \ldots, 5$. Vertices of L_1 are 3 tuples of the form (a_1, a_2, a_3), where $a_i \in S_i$. Vertices of L_5 are 3 tuples of the form (b_2, b_3, b_4), where $b_i \in S_i$. Vertices of L_2 are of the form (a_1, a_2, x), vertices of L_3 are of the form (a_1, b_4, x) and vertices of L_4 are of the form (b_3, b_4, x), where $a_i, b_i \in S_i$ and x is a coordinate array of length 3 satisfying the conditions of Table 4.1. We have an additional layer L'_3 with vertices of the form (a_1, b_4, x) for every coordinate array of length 3, where at least 5 out of the 6 following conditions hold: $a_1[x_\ell] = 1$ for each ℓ and $b_4[x_\ell] = 1$ for each ℓ.[1] Finally, we have two vertices v and u.

We have swap edges between L_i and L_{i+1} for $i = 1, \ldots, 4$. We have vector-change edges between L_3 and L'_3, between $(a_1, b_4, x) \in L_3$ and $(a'_1, b'_4, x) \in L'_3$ where either $a_1 = a'_1$ or $b_4 = b'_4$. We have coordinate-change edges between L_3 and L'_3, and within L'_3, L_2 and L_4. Finally we have fixed edges connected to u and v as follows. Every node in $L_4 \cup L_5$ has a directed edge to u, and every node in L_3 has a directed edge from u. Similarly, every node in $L_1 \cup L_2$ has an edge from v, and every node in L_3 has an edge to v.

[1] In fact, this last constraint is not necessary and we can instead define L'_3 to be all vertices of the form (a_1, b_4, x). We include this last constraint to be consistent with [Bon21b], so that we can use the correctness of their construction as a black box to argue the correctness of our construction.

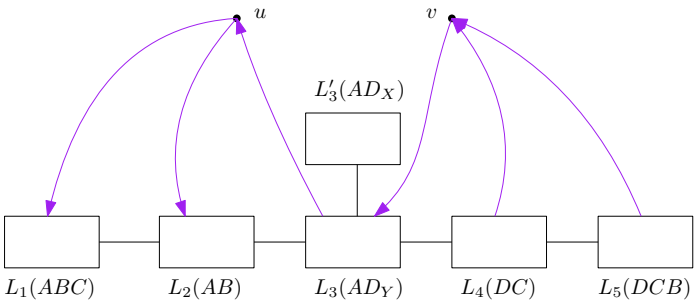

Figure 4-8: $k = 4$ construction. The names in parentheses are from the construction of [Bon21b] and are put here for ease of comparison. The purple edges are the fixed edges.

Note that the only difference between our construction and the construction of [Bon21b] is the coordinate-change edges inside L_3', and the fixed edges.

4.6.1 NO instance of 4-OV implies diameter at most 4

Using fixed edges

Case 1: v **to** $\beta \in V \setminus \{v\}$**.** If $\beta \in L_1 \cup L_2$, there is a direct edge from v to β. Now note that for any i, any node in L_i has a swap edge to a node in L_{i+1} and L_{i-1} changing one of its vectors to all 1s vector. So there is an undirected path of length at most 3 between $\beta \in L_3 \cup L_4 \cup L_5$ and some node β' in L_2. There is a direct edge from v to β', so $d(v, \beta) \leq 4$. If $\beta \in L_3'$, there is path of length 2 from a node $\beta' \in L_2$, using a swap edge and a coordinate-change edge (without changing the coordinate), and hence $d(v, \beta) \leq 3$. Finally if $\beta = u$, then from v we take a fixed edge to some node in L_2, then we take two swap edges to some node in L_4, and then we take a fixed edge to u.

Case 2: $\alpha \in V \setminus \{v\}$ **to** v**.** We need to show that there is a path of length at most 3 from α to some node $\alpha' \in L_3$. Then since $d(\alpha', v) = 1$, we have a path of length 4 from α to v. Again note that for any i, any node in L_i has a swap edge to a node in L_{i+1} and L_{i-1} changing one of its vectors to all 1s vector. So for $\alpha \in L_1, L_2, L_4, L_5$, there is a path of length 2 from α to some node in L_3. If $\alpha = (a_1, b_4, x) \in L_3'$, then we can take a coordinate change to $(a_1, b_4, x') \in L_3$ where $x'[i] = C(a_1, b_4)$ for $i = 1, 2, 3$. Finally if $\alpha = u$, there is a direct edge from u to all nodes in L_3.

Case 3: u **to** $\beta \in V \setminus \{u\}$**.** Symmetric to case 2.

Case 4: $\alpha \in V \setminus \{u\}$ **to** u**.** Symmetric to case 1.

Case 5: $\alpha \in V$ **to** $\beta \in L_1 \cup L_2$**.** From α take at most two edges to some node in L_3. The we take a fixed edge to v, and finally we take another fixed edge to β.

Case 6: $\alpha \in L_4 \cup L_5$ **to** $\beta \in V$**.** Symmetric to case 5.

Using variable edges

In a NO instance, for every set F of at most 4 vectors, there exists a coordinate that is 1 for every vector in F. Given a set F of at most 4 vectors, recall that $C(F)$ denotes a coordinate that is 1 for every vector in F.

The only remaining cases that are not covered by fixed edges are the following.

Case 1: $\alpha \in L_1 \cup L_2$ **to** $\beta \in L_4 \cup L_5$**.** Let $\alpha = (a_1, a_2, a_3)$ if $\alpha \in L_1$ and let $\alpha = (a_1, a_2, x_\alpha)$ if $\alpha \in L_2$, in which case a_3 is the all 1s vector. Let $\beta = (b_2, b_3, b_4)$ if $\beta \in L_5$ and let $\beta = (b_3, b_4, x_\beta)$ if $\beta \in L_4$, in which case b_2 is defined to be the all 1s vector. We will use the coordinate array $x = (x_1, x_2, x_3)$ where for each $i = 1, 2, 3$, $x_i = C(a_1, \ldots, a_{4-i}, b_{5-i}, \ldots, b_4)$. From α, take a swap edge or a coordinate-change edge to $(a_1, a_2, x) \in L_2$, a swap edge to $(a_1, b_4, x) \in L_3$, a swap edge to $(b_3, b_4, x) \in L_4$, and then a swap edge or a coordinate-change edge to β.

Case 2: $\alpha \in L_1 \cup L_2$ **to** $\beta \in L_3 \cup L'_3$**.** Let $\alpha = (a_1, a_2, a_3)$ if $\alpha \in L_1$ and let $\alpha = (a_1, a_2, x_\alpha)$ if $\alpha \in L_2$, in which case a_3 is the all 1s vector. Let $\beta = (a'_1, b_4, x_\beta)$. Consider the following coordinate array x: $x_1 = C(a_1, a_2, a_3, b_4)$, and $x_i = C(a_1, a_2, a'_1, b_4)$ for $i = 2, 3$. We take the following path: From α, take a coordinate-change edge or a swap edge to $(a_1, a_2, x) \in L_2$. Then take a swap edge to $(a_1, b_4, x) \in L_3$, a vector-change edge to $(a'_1, b_4, x) \in L'_3$, and finally a coordinate-change edge to β.

Case 3: $\alpha \in L_3 \cup L'_3$ **to** $\beta \in L_4 \cup L_5$**.** Symmetric to Case 3.

Case 4: $\alpha \in L_3 \cup L'_3$ **to** $\beta \in L_3 \cup L'_3$**.** Let $\alpha = (a_1, b_4, x_\alpha)$ and $\beta = (a'_1, b'_4, x_\beta)$. Consider the coordinate array x where $x_i = C(a_1, a'_1, b_4, b'_4)$ for $i = 1, 2, 3$. We use the following path: From α take a coordinate-change edge to $(a_1, b_4, x) \in L'_3$. Then go to $(a'_1, b_4, x) \in L_4$ and then to $(a'_1, b'_4, x) \in L'_4$ using two vector-change edges, and finally take a coordinate-change edge to β.

4.6.2 YES instance of 4-OV implies diameter at least 7

Suppose that $a_1, \ldots, a_4$ are orthogonal. We will show that the distance from $\alpha = (a_1, a_2, a_3)$ to $\beta = (a_2, a_3, a_4)$ is at least 7 if it uses one of the following edges: a coordinate-change edge in L'_3 or a fixed edge. This is because the rest of the construction is included in the construction of [Bon21b], and if the path does not use any of these edges, it is included in the construction of [Bon21b] and thus it is of length at least 7.

First suppose that the path uses a coordinate-change edge inside L'_3. Then the path has at least two nodes in L'_3 and L_3, and at least one node in each L_i for $i = 1, 2, 4, 5$. So the path is of length at least 7.

Next suppose that the path uses a fixed edge. So the path passes through u or v. This means that the path has a subpath passing through L_j, then v or u, then L_i,

where $j > i$. So L_i and L_j have at least two nodes in the path, the path has at least one node in each L_r for $r = 1, \ldots, 5$, $r \neq i, j$ and it has u or v. So it has at least 8 nodes and hence it is of length at least 7.

Part II

Approximating the Diameter of a Graph in Various Settings

Chapter 5

ST-Diameter, Bichromatic Diameter, and Subset Diameter

5.1 Results

In this chapter, we provide a comprehensive study of the approximability of ST-Diameter, Bichromatic Diameter, and Subset Diameter in directed/undirected weighted/unweighted graphs. Our results are summarized in Table 5.1. A more detailed statement of each result is provided in the appropriate section. We note that the ST-Diameter lower bounds in Table 5.1 are from [BRS$^+$18] and the theorem proving them was stated in Chapter 3.

5.2 Techniques

Our conditional lower bounds against subquadratic algorithms are modifications of the reduction from OV to Diameter of Roditty and Vassilevska W. [RV13] described in Chapter 2. Our time vs. accuraacy trade-off lower bound for undirected Bichromatic Diameter is a modification of the reduction from k-OV to ST-Diameter from [BRS$^+$18]. The near-linear time algorithms are simple modifications of the folklore 2-approximation algorithm for Diameter, which is to simply return the largest distance from an arbitrary vertex.

The more technically interesting part of this chapter are the algorithms that run in time $\tilde{O}(m\sqrt{n})$ (or $\tilde{O}(m^{3/2})$). These algorithms use as a starting point the 3/2-approximation for Diameter of Roditty and Vassilevska W. [RV13]. At a very high level, this algorithm cleverly chooses a set of $\tilde{O}(\sqrt{n})$ vertices to run Dijkstra's algorithm from, using a combination of random sampling and deterministic choice, and returns the largest distance found. In the analysis, shows that the returned distance is large by showing that the chosen set of vertices hits close to one of the true diameter end points, and applying the triangle inequality.

We provide an overview of our techniques for ST-Diameter and then Bichromatic Diameter. Let $s^* \in S$ and $t^* \in T$ be end-points of an ST-diameter path. Our goal is to run Dijkstra's algorithm from some $s \in S$ which is close to s^*, and hence far from

	Upper Bounds			Lower Bounds	
Problem	Runtime	Approx.	Comments	Runtime	Approx.
Undirected ST-Diameter	$O(m+n\log n)$	3	weighted, **tight**	$m^{1+o(1)}$	$3-\delta$
	$\tilde{O}(m\sqrt{n})$	almost 2	unweighted, nearly **tight**	$m^{\frac{k}{k-1}-o(1)}$	$3-2/k-\delta$
	$\tilde{O}(m^{3/2})$	2	weighted, **tight**	„	„
Directed ST-Diameter	N/A	N/A	**tight**	$m^{2-o(1)}$	any finite
Undirected Bichromatic Diameter	$O(m+n\log n)$	almost 2	unweighted, **tight**	$m^{1+o(1)}$	$2-\delta$
	$\tilde{O}(m\sqrt{n})$	almost 5/3	unweighted, nearly **tight**	$m^{\frac{k}{k-1}-o(1)}$	$2-\frac{1}{2k-1}-\delta$
	$\tilde{O}(m^{3/2})$	5/3	weighted, **tight**	„	„
	$O(m\|B\|)$	almost 3/2	unweighted, **tight***	$m^{2-o(1)}$	$3/2-\delta$
Directed Bichromatic Diameter	$\tilde{O}(m^{3/2})$	2	weighted, **tight***	$m^{2-o(1)}$	$2-\delta$
	$O(m\|B'\|)$	almost 3/2	unweighted, **tight***	$m^{2-o(1)}$	$3/2-\delta$
Subset Diameter	$\tilde{O}(m)$	2	weighted, directed, **tight**	$m^{2-o(1)}$	$2-\delta$

Table 5.1: Results. The ST-Diameter lower bounds are from [BRS+18]. All of our near-linear time algorithms and parameterized algorithms. The rest are randomized and work with high probability. Each upper bound is tight with the lower bound in its row. Our lower bounds are under SETH. All of our lower bounds hold even for unweighted graphs. The trade-off lower bounds in terms of k hold for any fixed integer $k \geq 2$. δ is any constant > 0. B and B' are parameters defined in our parameterized algorithms that measure the size of the *boundary* between S and T. The lower bound constructions for the parameterized algorithms have $|B| = \tilde{O}(1)$
* Multiplicative approximation factor is tight, but not runtime.

t^*, or from some $t \in T$ which is close to t^* and hence far from s^* (by the triangle inequality). The known algorithm for the standard Diameter problem finds a vertex v that is close to either s^* or t^*, but v could be in either S or T (or neither) so v is not helpful for approximating the ST-diameter. To get around this issue where v is in the "wrong" set, whenever we run Dijkstra's algorithm from a vertex u, we also run Dijkstra's algorithm from the closest vertex to u that is in the "right" set. Repeated applications of the triangle inequality show that this gives a 2-approximation for

ST-Diameter.

For undirected Bichromatic Diameter, we need additional ideas to get the approximation factor down to 5/3. To obtain our improvements for Bichromatic Diameter over the known ST-Diameter algorithms, we crucially exploit the basic fact that as S, T partition V any path that starts from a vertex $s \in S$ and ends in a vertex $t \in T$ must cross a (u, v) edge such that $u \in S, v \in T$. While this fact is clear, it not at all obvious how one might try to exploit it.

Our 5/3-approximation algorithms are a combination of two themes:

1. Randomly sample nodes in S and nodes in T – similarly to prior works, the sampling works well if there are many nodes of S that are close to s^*, or if there are many nodes of T that are close to t^*.

2. If 1 is not good enough, we show that we can find a node $w \in S$ close to t^* for which we can "catch" an $S \times T$ edge (s, t) on the shortest $w \to t^*$ path, such that t is close to t^*.

Theme 2 is our new contribution. Theme 2 makes our algorithm more delicate than the algorithm for ST-Diameter because we are trying to identify a particular $S \times T$ edge. Because of theme 2, our algorithms are more complicated than the ST-Diameter algorithms, but run in asymptotically the same time, and achieve a better approximation guarantee.

For directed Bichromatic Diameter, the previously known techniques for approximating Diameter in directed graphs fail. The main issue is that the prior techniques were general enough that they also gave algorithms for the related problem of Radius as a byproduct. In the bichromatic case, however, there is a genuine difference between Diameter and Radius. It turns out that for Radius there is a conditional lower bound precluding any finite approximation. Hence, any algorithm for Bichromatic Diameter should not extend to Bichromatic Radius. Again, theme 2 from our undirected Bichromatic Diameter algorithm, combined with edge sampling instead of vertex sampling, helps us obtain such an algorithm.

5.3 Preliminaries

The following standard lemmas will be useful for our algorithms.

Lemma 5.3.1. *Let $G = (V, E)$ be a (possibly directed and weighted graph) and let $W \subseteq V$. Let $g \geq \Omega(\ln n)$ be an integer. Let $S \subseteq W$ be a random subset of $c(|W|/g) \ln n$ vertices for some constant $c > 1$. For every $v \in V$, let $W(v)$ be the set of vertices $x \in W$ for which $d(v, x) < d(v, S)$. Then with probability at least $1 - 1/n^{c-1}$, for every $v \in V$, $|W(v)| \leq g$, and moreover, if one takes the closest g vertices of W to v, they will contain $W(v)$.*

Proof. For each $v \in V$, imagine sorting the nodes $x \in W$ according to $d(v, x)$. Define Q_v to be the first g nodes in this sorted order - those are the nodes of W closest to v (in the $v \to x$ direction).

We pick S randomly by selecting each vertex of W with probability $(c \ln n)/g$. The probability that a particular $q \in Q_v$ is not in S is $1 - (c \ln n)/g$, and the probability that no $q \in Q_v$ is in S is $(1-(c \ln n)/g)^g \leq 1/n^c$. By a union bound, with probability at least $1 - 1/n^{c-1}$, for every $v \in V$, we have that $Q_v \cap S \neq \emptyset$.

Now, for each particular v, say that $w(v)$ is a node in $Q_v \cap S$. Since all nodes $x \in W$ with $d(v,x) < d(v,w(v))$ must be in Q_v, and since $d(v,w(v)) \geq d(v,S)$, we must have that $W(v) \subseteq Q_v$. Hence, with probability at least $1 - 1/n^{c-1}$, for every $v \in V$, $|W(v)| \leq g$ and $W(v) \subseteq Q_v$. □

Lemma 5.3.2. *Let $G = (V,E)$ be a (possibly directed and weighted) graph. Let $M, W \subseteq V$ and let $S \subseteq W$ be a random subset of $c(n/g)\ln n$ vertices for some large enough constant c and some integer $g \geq 1$.*

Then, for any $D > 0$ and for any $w \in M$ with $d(w,S) > D$, if one takes the closest g vertices of W to w, they will contain all nodes of W at distance $< D$ from w, with high probability.

Proof. Let Q be the closest g vertices of W to w. By Lemma 5.3.1, with high probability Q contains all nodes of W at distance $< d(w,S)$ from w, and hence Q contains all nodes of W at distance $< D$ from w, with high probability. □

We sometimes sample edges instead of vertices, so analogous lemmas to Lemmas 5.3.1 and 5.3.2 hold when the sample is from a set of edges. Here is the analogue of Lemma 5.3.2. The other lemma is similar.

Lemma 5.3.3. *Let $G = (V,E)$ be a (possibly directed and weighted graph) and let $M, W \subseteq V$. Let $E' \subseteq E$ be a random subset of $c(|E|/g)\ln n$ edges for some large enough constant c and some integer $g \geq 1$. Let Q be the endpoints of edges in E' that are in W.*

Then, for any $D > 0$, and for any w with $d(w,S) > D$, if one takes the closest g edges of E' to w wrt the distance from their W endpoints, they will contain all edges of E' whose W endpoints are at distance $< D$ from w, with high probability.

5.4 Algorithms

5.4.1 Undirected ST-Diameter

Proposition 1. *There is an $O(m+n)$ time deterministic algorithm that for any n vertex m edge unweighted graph $G = (V,E)$ and $S \subseteq V, T \subseteq V$, computes an estimate D' such that $D_{S,T}/3 \leq D' \leq D_{S,T}$ and two vertices $s \in S$, $t \in T$ such that $d(s,t) = D'$. In graphs with nonnegative weights, the same estimate can be achieved in $O(m + n\log n)$ time.*

Proof. The algorithm is extremely simple: pick arbitrary vertices $s \in S$ and $t \in T$, compute BFS(s) and BFS(t) and return $\max\{\max_{t' \in T} d(s,t'), \max_{s' \in S} d(s',t)\}$ (also returning the two vertices achieving the maximum). For weighted graphs, run Dijkstra's algorithm instead of BFS.

Let's see why this algorithm provides the promised guarantee. Suppose that for every $t' \in T$, $d(s,t') < D_{S,T}/3$ (otherwise we are done). Then for every $t', t'' \in T$, $d(t',t'') \leq d(t',s) + d(s,t'') < 2D_{S,T}/3$. In particular, for all $t' \in T$, $d(t,t') < 2D_{S,T}/3$. If we also had that for every $s' \in S$, $d(t,s') < D_{S,T}/3$, then we'd get that for all $s' \in S, t' \in T$, $d(s',t') \leq d(s',t) + d(t,t') < D_{S,T}$, contradicting the definition of $D_{S,T}$. Thus, $\max\{\max_{t' \in T} d(s,t'), \max_{s' \in S} d(s',t)\} \geq D_{S,T}/3$. □

We will now show an analogue to the $\tilde{O}(m\sqrt{n})$ time almost-3/2-approximation for Diameter of Roditty and Vassilevska W. [RV13] for ST-Diameter giving a 2-approximation. Using a trick from Chechik et al. [CLR+14] we also obtain a true 2 approximation algorithm running in $\tilde{O}(m^{3/2})$.

Algorithm 1 2-Approximation for ST-Diameter

procedure 2-Approx
  X - random sample of vertices, $|X| = \Theta(\sqrt{n} \log n)$
  $D_1 := 0$
  for every $x \in X$ **do**
    Run BFS(x)
    Let t_x be the closest vertex to x in T
    Run BFS(t_x)
    $D_1 = \max\{D_1, \max_{s \in S} d(s, t_x)\}$
  Let $\bar{t}$ be the furthest vertex of T from X (computed above)
  Run BFS($\bar{t}$)
  $D_2 = \max_{s \in S} d(s, \bar{t})$.
  Let Y be the closest $\sqrt{n}$ vertices to $\bar{t}$.
  for every $y \in Y$ **do**
    Run BFS(y)
    Let s_y be the closest vertex to y in S
    Run BFS(s_y)
    $D_2 = \max\{D_2, \max_{t \in T} d(s_y, t)\}$
  return $\max\{D_1, D_2\}$

We use Algorithm 1 to prove:

Theorem 5.4.1. *There is an $\tilde{O}(m\sqrt{n})$ time randomized algorithm that with high probability outputs an estimate D' for the ST-diameter D of an m edge n vertex unweighted undirected graph such that $2\lfloor D/4 \rfloor \leq D' \leq D$.*

In $\tilde{O}(m^{3/2})$ time one can obtain an estimate D'' such that $D/2 \leq D'' \leq D$.

Proof. First we analyze Algorithm 1. Let $s^* \in S$ and $t^* \in T$ be a pair of vertices with $d(s^*, t^*) = D$. Let $d = \lfloor D/4 \rfloor$.

Suppose first that for some $x \in X$, $d(x, t^*) \leq d$. Then, $d(x, t_x) \leq d(x, t^*) \leq d$ and hence $d(t_x, t^*) \leq d(t_x, x) + d(x, t^*) \leq 2d$. However, then $d(t_x, s^*) \geq d(t^*, s^*) - d(t^*, t_x) \geq D - 2d \geq D/2$. In this case, $D_1 \geq D/2$ and we are done.

Thus, if $D_1 < D/2$, it must be that for every $x \in X$, $d(x, t^*) \geq d + 1$. Hence, for every $x \in X$, $d(x, \bar{t}) \geq d(x, t^*) \geq d + 1$ by the definition of $\bar{t}$. If $d(\bar{t}, s^*) \geq D/2$, then $D_2 \geq D/2$ and we are done, so let us assume that $d(\bar{t}, s^*) \leq D/2$.

Now, as X is random of size $c\sqrt{n}\log n$ for large enough c, with high probability, X hits the $\sqrt{n}$-neighborhoods of all vertices. In particular, $X \cap Y \neq \emptyset$. However, since $d(x, \bar{t}) \geq d + 1$ for every $x \in X$, it must be that Y contains all vertices at distance d from $\bar{t}$ as it contains all vertices closer to $\bar{t}$ than $x \in Y \cap X$.

If $s^* \in Y$, then we would have run BFS from s^* and returned D. Hence $d(\bar{t}, s^*) > d$. Let a be the vertex on the shortest path between $\bar{t}$ and s^* with $d(\bar{t}, a) = d$. We thus have that $a \in Y$. Also, since $d(\bar{t}, s^*) \leq D/2$, $d(a, s^*) \leq D/2 - d$ and hence $d(a, s_a) \leq D/2 - d$, so that $d(s_a, t^*) \geq D - 2(D/2 - d) \geq 2d$. This finishes the argument that 2-Approx returns an estimate D' with $2\lfloor D/4 \rfloor \leq D' \leq D$.

It is not too hard to see that the only time that we might get an estimate that is less than $D/2$ is in the last part of the argument and only if the diameter is of the form $4d + 3$. (We will prove the algorithm guarantees formally soon.) The analysis fails to work in that case because Y is guaranteed to contain only the vertices at distance d from $\bar{t}$.

In particular, if Y contains all vertices at distance $d + 1$ from $\bar{t}$ instead of just those at distance at most d, we could consider a to be the vertex on the shortest path between $\bar{t}$ and s^* with $d(\bar{t}, a) = d + 1$, and $a \in Y$. Now since $d(\bar{t}, s^*) \leq 2d + 1$ (as otherwise we'd be done), $d(a, s_a) \leq d(a, s^*) \leq 2d + 1 - d - 1 = d$, so that $d(s_a, t^*) \geq 2d + 3$. Hence everything would work out.

We handle this issue with a trick from Chechik et al. [CLR+14]. First, we make graph have constant degree by blowing up the number of vertices and adding 0 weight edges as follows. Let v be an original vertex and suppose it has degree $d(v)$. Replace v with a $d(v)$-cycle of 0 weight edges so that each of the cycle vertices is connected to a one of the neighbors of v, where each neighbor has a cycle vertex corresponding to it. This makes every vertex have degree 3 and increases the number of vertices to $O(m)$.

Now, we run algorithm 2-Approx with two changes. The first is that instead of BFS we use Dijkstra's algorithm[1] because the edges now have weights. The second change is that we redefine Y as follows. Let Z be the closest $\sqrt{m}$ vertices to $\bar{t}$. Define Y to be Z, together with all vertices that have a non-zero weight edge to some vertex of Z.

Since every vertex has degree 3, the number of vertices in Y is $\leq 4|Z| \leq O(\sqrt{m})$ and hence we can afford to run Dijkstra from each of them and complete the algorithm in $\tilde{O}(m^{3/2})$ time.

Let us now formally analyze the guarantees of the algorithm. If some vertex $x \in X$ has $d(x, t^*) \leq D/4$, we get that $d(t_x, s^*) \geq D - 2(D/4) = D/2$. If we are not done, all vertices of X have $d(x, \bar{t}) \geq d(x, t^*) > D/4$ and Z contains all vertices at distance $\leq D/4$ from $\bar{t}$. If $s^* \in Z$, we are done so we must have $d(s^*, \bar{t}) > D/4$. Consider the last vertex a' on the $\bar{t}$ to s^* shortest path (in the direction towards s^*) for which

[1] We can also use Thorup's algorithm [Tho99], which runs in linear time and is stated for positive weight edges but can also handle zero weight edges.

$d(\bar{t}, a') \leq D/4$. We have that $a' \in Z$. Also, the vertex a after a' on the $\bar{t}$ to s^* shortest path must be in Y by definition.

If $d(\bar{t}, s^*) \geq D/2$, we are done. If we are not done, then we get that $d(a, s^*) < D/4$ since $d(\bar{t}, a) > D/4$. Hence, $d(a, s_a) < D/4$, so $d(s_a, t^*) > D - 2(D/4) = D/2$. □

It is quite straightforward to extend the the ST-Diameter algorithms to work for weighted undirected graphs as well:

Theorem 5.4.2. *In $\tilde{O}(m\sqrt{n})$ time one can obtain an estimate D' for the ST-diameter D of an m edge n vertex undirected graph with nonnegative edge weights such that $D/2 - 2w(a, a') \leq D' \leq D$ for some edge (a, a').*

In $\tilde{O}(m^{3/2})$ time one can obtain an estimate D'' such that $D/2 \leq D'' \leq D$.

Proof. The $\tilde{O}(m\sqrt{n})$ time algorithm is identical to Algorithm 1, but with BFS replaced by Dijkstra's algorithm. The proof is very similar to that of Theorem 5.4.1. The main difference concerns the definition of the vertex a, which is the vertex on the shortest path between $\bar{t}$ and s^* with $d(\bar{t}, a) = d$. Such a vertex a may not exist here since the graph is weighted. Instead, we let a' be the last vertex on the $\bar{t}$ to s^* shortest path that is at distance $\leq D/4$ from $\bar{t}$, and let a be the vertex after a'.

We include the full analysis of correctness here for completeness. Let $s^* \in S$ and $t^* \in T$ be the endpoints of the ST-diameter path so that $d(s^*, t^*) = D$.

If some vertex $x \in X$ has $d(x, t^*) \leq D/4$, we get that $d(t_x, s^*) \geq D - 2(D/4) = D/2$. If we are not done, all vertices of X have $d(x, \bar{t}) \geq d(x, t^*) > D/4$ and Y contains all vertices at distance $\leq D/4$ from $\bar{t}$. If $s^* \in Y$, we are done so we must have $d(s^*, \bar{t}) > D/4$.

Recall that a' is the last vertex on the $\bar{t}$ to s^* shortest path that is at distance $\leq D/4$ from $\bar{t}$, and that a is the vertex after a'. We have that $a' \in Y$. If $d(\bar{t}, s^*) \geq D/2$, we are done. If we are not done, then we get that $d(a, s^*) < D/4$ since $d(\bar{t}, a) > D/4$. Thus, $d(a', s^*) < D/4 + w(a, a')$. Therefore, $d(a', s_{a'}) < D/4 + w(a, a')$, so $d(s_{a'}, t^*) < D - 2(D/4 + w(a, a')) = D/2 - 2w(a, a')$. This completes the analysis of the $\tilde{O}(m\sqrt{n})$ time algorithm.

For the $\tilde{O}(m^{3/2})$ time algorithm, we apply precisely the same trick from [CLR+14] as the proof of Theorem 5.4.1, with identical analysis. □

5.4.2 Undirected Bichromatic Diameter

We begin with a simple near-linear time algorithm.

Proposition 2. *There is an $O(m + n \log n)$ time algorithm, that given an undirected graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, can output an estimate D' such that $D_{ST}(G)/2 - W/2 \leq D' \leq D_{ST}$, where W is the minimum weight of an edge in $S \times T$.*

Proof. Let (s, t) be a minimum weight edge of G with $s \in S$ and $t \in T$. Run Dijkstra's algorithm from s and from t. Let $D' = \max\{\max_{t' \in T} d(s, t'), \max_{s' \in S} d(s', t)\}$. Let $s^* \in S, t^* \in T$ be endpoints of an ST-diameter path, i.e. $d(s^*, t^*) = D_{ST}$. Then, suppose that $\max_{t' \in T} d(s, t') < D_{ST}/2 - W/2$. In particular, $d(s, t^*) < D_{ST}/2 - W/2$,

and hence $d(s, s^*) > D_{ST}/2 + W/2$ by the triangle inequality. Also by the triangle inequality,

$$D_{ST}/2 + W/2 < d(s,t) + d(t,s^*) \leq w(s,t) + \max_{s' \in S} d(s',t).$$

Hence, $D' > D_{ST}/2 - W/2$, where W is the minimum weight of an edge in $S \times T$. □

Now we turn to our 5/3-approximation algorithms. Our first theorem is for unweighted graphs. Later on, we modify the algorithm in this theorem to obtain an algorithm for weighted graphs as well, and at the same time remove the small additive error that appears in the theorem below.

Theorem 5.4.3. *There is an $\tilde{O}(m\sqrt{n})$ time algorithm, that given an unweighted undirected graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, can output an estimate D' such that $3D_{ST}(G)/5 \leq D' \leq D_{ST}(G)$ if $D_{ST}(G)$ is divisible by 5, and otherwise $3D_{ST}(G)/5 - 6/5 \leq D' \leq D_{ST}(G)$.*

Proof. Let $D = D_{ST}(G)$ and let us assume that D is divisible by 5. If D is not divisible by 5, the estimate we return will have a small additive error. For clarity of presentation, we omit the analysis of the case where D is not divisible by 5.

Suppose the (bichromatic) ST-diameter endpoints are $s^* \in S$ and $t^* \in T$ and that the ST-diameter is D. The algorithm does not know D, but we will use it in the analysis.

Algorithm Step 1 The algorithm first samples $Z \subseteq S$ of size $c\sqrt{n} \ln n$ uniformly at random. For every $z \in Z$, run BFS, and let $D_1 = \max_{z \in Z, t \in T} d(z,t)$.

Analysis Step 1 If for some $s' \in Z$ we have that $d(s^*, s') \leq 2D/5$, then $D_1 \geq d(s', t^*) \geq D - d(s^*, s') \geq 3D/5$.

Algorithm Step 2 Now, sample a set X from T of size $C\sqrt{n} \ln n$ uniformly at random for large enough constant C. For every $t \in X$, run BFS and find the closest node $s(t)$ of S to t. Run BFS from every $s(t)$. Let $D_2 = \max_{t \in X, t' \in T} d(s(t), t')$.

Analysis Step 2 If s^* is at distance $\leq D/5$ from some node t of X, then $d(s^*, s(t)) \leq 2D/5$ (since $s(t)$ is closer to t than s^*), and so $D_2 \geq d(s(t), t^*) \geq 3D/5$.

If neither D_1, nor D_2 are good approximations, it must be that $d(s^*, X) > D/5$ and $d(s^*, Z) > 2D/5$. Consider the nodes M of S that are at distance $> 2D/5$ from Z, then the node $w \in M$ that is furthest from X among all nodes of M. If neither D_1, nor D_2 was a good approximation, $s^* \in M$ and since $d(s^*, X) > D/5$, we must have that $d(w, X) > D/5$ (and also $d(w, Z) > 2D/5$). In the next step we will look for such a w.

Algorithm Step 3 For each $s \in S$ define D_s to be the biggest integer which satisfies $d(s, X) > D_s/5$ and $d(s, Z) > 2D_s/5$. Let $w = \arg\max D_s$ and $D' = \max D_s$.

Analysis Step 3 By Lemma 5.3.2 we have that whp, the number of nodes of T at distance $\leq D'/5$ from w and the number of nodes of S at distance $\leq 2D'/5$ from w are both $\leq \sqrt{n}$. Also if neither D_1, nor D_2 are good approximations, it must be that $d(s^*, X) > D/5$ and $d(s^*, Z) > 2D/5$ and hence $D' \geq D$.

Algorithm Step 4 Run BFS from w. Take all nodes of S at distance $\leq 2D'/5$ from w, call these S_w, and run BFS from them. Whp, $|S_w| \leq \sqrt{n}$, so that this BFS run takes $O(m\sqrt{n})$ time. Let $D_3 := \max_{s \in S_w, t \in T} d(s,t)$.

For every $s \in S_w$, let $t(s)$ be the closest node of T to s (breaking ties arbitrarily). Run BFS from each $t(s)$. Let $D_4 := \max_{s \in S_w, s' \in S} d(s', t(s))$.

Analysis Step 4 If $D_3 \geq 3D/5$ or $D_4 \geq 3D/5$, we are done, so let us assume that $D_3, D_4 < 3D/5$. Since $D_3 < 3D/5$, and since $D_3 \geq d(w, t^*)$, it must be that $d(w, t^*) < 3D/5$. Let P_{wt^*} be the shortest w to t^* path. Consider the node b on P_{wt^*} for which $d(w, b) = 2D/5$. If $b \in S$, then since $D' \geq D$, $b \in S_w$ and hence we ran BFS from $t(b)$. But since $d(b, t^*) = d(w, t^*) - 2D/5 < D/5$, and $d(b, t(b)) \leq d(b, t^*)$ we have that $d(t(b), t^*) \leq 2D/5$ and hence $D_4 \geq d(s^*, t(b)) \geq D - d(t(b), t^*) \geq 3D/5$. Thus, if $D_4 < 3D/5$, it must be that $b \in T$.

Algorithm Step 5 Take all nodes of T at distance $\leq D'/5$ from w, call these T_w and run BFS from them. Since $d(w, X) > D'/5$, whp $|T_w| \leq \sqrt{n}$, so this step runs in $O(m\sqrt{n})$ time. Let $D_5 = \max_{t \in T_w, s \in S} d(t, s)$.

Analysis Step 5 If $D_5 \geq 3D/5$, we would be done, so assume that $D_5 < 3D/5$. Let a be the node on the shortest w to t^* path P_{wt^*} with $d(w, a) = D/5$. Suppose that $a \in T$. Since $D' \geq D$, $a \in T_w$ and we ran BFS from it. However, also $d(a, t^*) = d(w, t^*) - d(w, a) < 3D/5 - D/5 = 2D/5$, and hence $D_5 \geq d(a, s^*) \geq d(t^*, s^*) - d(t^*, a) \geq D - 2D/5 = 3D/5$. Since $D_5 < 3D/5$, it must be that $a \in S$.

Now, since $a \in S$ and $b \in T$, somewhere on the a to b shortest path P_{ab}, there must be an edge (s', t') with $s' \in S, t' \in T$. Since s' is before b, $d(w, s') \leq 2D/5 \leq 2D'/5$, and hence $s' \in S_w$. Thus we ran BFS from $t(s')$. Since s' has an edge to $t' \in T$, $d(s', t(s')) \leq d(s', t') = 1$. Also, since $d(w, s') \geq d(w, a) = D/5$ and $d(w, t^*) \leq 3D/5 - 1$, $d(s', t^*) \leq 2D/5 - 1$. Thus,

$$\begin{aligned} D_4 \geq d(t(s'), s^*) &\geq d(s^*, t^*) - d(t(s'), t^*) \\ &\geq D - d(t(s'), s') - d(s', t^*) \\ &\geq D - 1 - 2D/5 + 1 \\ &= 3D/5. \end{aligned}$$

Hence if we set $D'' = \max\{D_1, D_2, D_3, D_4, D_5\}$, we get that $3D/5 \leq D'' \leq D$. $\square$

We now modify the algorithm for unweighted graphs, both making the algorithm work for weighted graphs and removing the additive error, at the expense of increasing the runtime to $\tilde{O}(m^{3/2})$.

Theorem 5.4.4. *There is an $\tilde{O}(m^{3/2})$ time algorithm, that given an undirected graph $G = (V, E)$ with nonnegative integer edge weights and $S \subseteq V, T = V \setminus S$, can output an estimate D' such that $3D_{ST}(G)/5 \leq D' \leq D_{ST}$.*

Proof. Suppose as before the (bichromatic) ST-diameter endpoints are $s^* \in S$ and $t^* \in T$ and that the ST-diameter is D.

Algorithm Modified Step 1 The algorithm here samples $E' \subseteq E$ of size $c\sqrt{m}\ln n$ uniformly at random, for large enough c. Let Z be the endpoints of edges in E' that are in S. For every $z \in Z$, run Dijkstra's algorithm, and let $D_1 = \max_{z \in Z, t \in T} d(z, t)$.

Analysis Step 1 If for some $s' \in Z$ we have that $d(s^*, s') \leq 2D/5$, then $D_1 \geq d(s', t^*) \geq D - d(s^*, s') \geq 3D/5$. Let us then assume that $d(s^*, Z) > 2D/5$.

Algorithm Modified Step 2 Let X be the endpoints of edges in E' that are in T. For every $t \in X$, run Dijkstra's algorithm and find the closest node $s(t)$ of S to t. Run Dijkstra's algorithm from every $s(t)$. Let $D_2 = \max_{t \in X, t' \in T} d(s(t), t')$.

Analysis Step 2 If s^* is at distance $\leq D/5$ from some node t of X, then $d(s^*, s(t)) \leq 2D/5$ (since $s(t)$ is closer to t than s^*), and so $D_2 \geq d(s(t), t^*) \geq 3D/5$. Let us then assume that $d(s^*, X) > D/5$.

As before, if we consider the nodes M of S that are at distance $> 2D/5$ from Z, then the node $w \in M$ that is furthest from X among all nodes of M, would have both $d(w, Z) > 2D/5$ and $d(w, X) > D/5$, as s^* is in M and satisfies $d(s^*, X) > D/5$. We will find a node w with these properties in the next step.

Algorithm Unmodified Step 3 Perform exactly the same Step 3 as before, finding the largest integer D' such that there is some node $w \in S$ with $d(w, Z) > 2D'/5$ and $d(w, X) > D'/5$.

Analysis Step 3 Let $w \in S$ be the node we found such that $d(w, X) > D'/5, d(w, Z) > 2D'/5$. By Lemma 5.3.3 we have that whp, the number of edges (s, g) where $s \in S, g \in V$ and $d(w, s) \leq 2D'/5$ and the number of edges (t, g') where $t \in T, g' \in V$ and $d(w, t) \leq D'/5$ is at most $\sqrt{m}$. Also, if $D_1, D_2 < 3D/5$, then $D' \geq D$, so that we also have that the number of edges (s, b) where $s \in S$ and $d(w, s) \leq 2D/5$ and the number of edges (t, b') where $t \in T$ and $d(w, t) \leq D/5$ is at most $\sqrt{m}$, whp.

Algorithm Modified Step 4 Run Dijkstra's algorithm from w. Take all edges incident to nodes of S at dist $\leq 2D'/5$ from w. Call these edges E_S and their endpoints S_w. Run Dijkstra's algorithm from both of their endpoints. Whp, $|E_S| \leq \sqrt{m}$ and so $|S_w| \leq 2\sqrt{m}$, so that this Dijkstra run takes $\tilde{O}(m^{3/2})$ time. Let $D_3 := \max_{t \in S_w \cap T, s \in S} d(s, t)$.

For every $s \in S_w \cap S$, determine a closest node $t(s) \in T$ to s, and run Dijkstra's algorithm from $t(s)$ as well. This search also takes $O(m^{3/2})$ time. Let $D_4 := \max_{s \in S_w \cap S, s' \in S} d(s', t(s))$.

Analysis Step 4 If $d(w, t^*) \geq 3D/5$, or $D_3 \geq 3D/5$ or $D_4 \geq 3D/5$, we are done, so let us assume that $d(w, t^*), D_3, D_4 < 3D/5$.

Now consider the node b on the shortest w to t^* path P_{wt^*} for which $d(w, b) \leq 2D/5$, but such that the node b' after it on P_{wt^*} has $d(w, b') > 2D/5$.

Suppose that $b \in S$. Then since $D' \geq D$, we have $d(w, b) \leq 2D'/5$ and hence $(b, b') \in E_S$. Let us consider $d(b', t^*) = d(w, t^*) - d(b', w)$. Since $d(w, t^*) < 3D/5$ and $d(b', w) > 2D/5$, $d(b', t^*) < D/5$. If $b' \in T$, then since we ran Dijkstra's algorithm from b', we got $D_3 \geq D - D/5 = 4D/5$. If $b' \in S$, then we ran Dijkstra's algorithm from $t(b')$ and $d(t(b'), t^*) \leq d(t(b'), b') + d(b', t^*) \leq 2d(b', t^*) < 2D/5$, and hence $D_4 \geq d(t(b), s^*) \geq D - 2D/5 = 3D/5$. Thus if neither $d(w, t^*)$, D_3, nor D_4 are good approximations, then $b \in T$.

Algorithm Modified Step 5 Take all edges incident to nodes of T at dist $\leq D'/5$ from w. Call these edges E_T and their endpoints that are in T, T_w. Run Dijkstra's algorithm from all nodes in T_w.

Since $d(w, X) > D'/5$, whp $|T_w| \leq 2\sqrt{m}$, so this step runs in $O(m^{3/2})$ time. Let $D_5 = \max_{t \in T_w, s \in S} d(t, s)$.

Analysis Step 5 If $D_5 \geq 3D/5$, we would be done, so assume that $D_5 < 3D/5$. Let a be the node on P_{wt^*} with $d(w, a) \leq D/5$ but so that the node a' after a on P_{wt^*} has $d(w, a') > D/5$. Suppose that $a' \in T$. Since $D' \geq D$, $(a, a') \in E_T$, $a' \in T_w$ and we ran Dijkstra's algorithm from a'. However, also $d(a', t^*) = d(w, t^*) - d(w, a') < 3D/5 - D/5 = 2D/5$, and hence $D_5 \geq d(a, s^*) \geq d(t^*, s^*) - d(t^*, a') \geq D - 2D/5 = 3D/5$. Since $D_5 < 3D/5$, it must be that $a' \in S$.

Now, since $a' \in S$ and $b \in T$, somewhere on the a' to b shortest path P_{ab}, there must be an edge (s', t') with $s' \in S, t' \in T$. However, since s' is before b, we have that $d(w, s') \leq d(w, b) \leq 2D/5 \leq 2D'/5$. Thus, $(s', t') \in E_S$ and we ran Dijkstra's algorithm from t'. However, $d(t', t^*) = d(w, t^*) - d(w, t') \leq d(w, t^*) - d(w, a') < 3D/5 - D/5 = 2D/5$, and hence $D_3 \geq d(t', s^*) \geq d(s^*, t^*) - d(t', t^*) > 3D/5$.

Hence if we set $D'' = \max\{d(w, t^*), D_1, D_2, D_3, D_4, D_5\}$, we get that $3D/5 \leq D'' \leq D$. □

Undirected parameterized Bichromatic Diameter

Theorem 5.4.5. *There is an $O(m|B|)$ time algorithm, that given an unweighted undirected graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, outputs an estimate D' such that $2D_{ST}(G)/3 - 1 \leq D' \leq D_{ST}(G)$.*

We note that our conditional lower bounds for Bichromatic Diameter rule out a better-than-5/3-approximation algorithm in subquadratic time for Bichromatic Diameter using a construction with boundary size $|B| = \tilde{O}(1)$. The algorithm of Theo-

rem 5.4.5 achieves the *better* multiplicative approximation factor of 3/2, however this does not contradict the lower bound due to the additive error of 1 in the algorithm.

Proof of Theorem 5.4.5.

Algorithm For all $v \in T$, we let $\varepsilon_{ST}(v) = \max_{s \in S} d(s, v)$ ($\varepsilon_{ST}(v)$ is already defined for $v \in S$). Suppose without loss of generality that $B \subseteq S$ (a symmetric argument works for $B \subseteq T$). For every vertex $v \in B$, run BFS from v, let v_T be an arbitrary neighbor of v such that $v_T \in T$, and run BFS from v_T. Let D_1 be the largest $S-T$ distance found. That is, $D_1 = \max_{v \in B} \max\{\varepsilon_{ST}(v), \varepsilon_{ST}(v_T)\}$. Let $s \in S$ be the farthest vertex from B. That is, s is the vertex in S that maximizes $d(s, B)$. Then, we run BFS from s and let $D_2 = \varepsilon_{ST}(s)$. Return $D' = \max\{D_1, D_2\}$.

Analysis Let $s^* \in S, t^* \in T$ be the true endpoints of the bichromatic diameter and let D denote $D_{ST}(G)$. If s^* is of distance at most $D/3 + 1$ from some vertex $v \in B$ then by the triangle inequality $d(v, t^*) \geq 2D/3 - 1$ so $D_1 \geq 2D/3 - 1$ and we are done. If t^* is of distance at most $D/3$ from some vertex $v \in B$ then by the triangle inequality $d(v_T, s^*) \geq 2D/3 - 1$ so $D_1 \geq 2D/3 - 1$ and we are done.

Now, if we are not already done, s^* is of distance at least $D/3+1$ from every vertex in B, so s is also of distance at least $D/3 + 1$ from every vertex in B. Additionally, t^* is of distance at least $D/3$ from every vertex in B. We observe that the shortest path between s and t^* must contain a vertex in B. Thus, $d(s, t^*) = \min_{v \in B} d(s, v) + d(v, t^*) \leq (D/3+1)+(D/3) = 2D/3+1$. Thus, $D_2 \geq 2D/3+1$ and we are done. □

5.4.3 Directed Bichromatic Diameter

Theorem 5.4.6. *There is an $\tilde{O}(m^{3/2})$ time algorithm, that given a directed graph $G = (V, E)$ with nonnegative integer weights and $S \subseteq V, T = V \setminus S$, can output an estimate D' such that $D_{ST}(G)/2 \leq D' \leq D_{ST}(G)$.*

Proof. Suppose the (bichromatic) ST-diameter endpoints are $s^* \in S$ and $t^* \in T$ and that the ST-diameter is D. The algorithm does not know D, but we will use it in the analysis.

Algorithm Step 1 The algorithm first samples $E' \subseteq E$ of size $c\sqrt{m} \ln m$ for large enough c uniformly at random from the edges which go from S to T. Let R be the set of S nodes incident to these edges. Define $D_1 = \max_{u \in R, t \in T} d(u, t)$.

Analysis Step 1 If there exists an $s \in R$ with $d(s^*, s) \leq D/2$ then we are done as by triangle inequality $D_1 \geq d(s, t^*) \geq d(s^*, t^*) - d(s^*, s) \geq D/2$.

Algorithm Step 2 Let w be the vertex in S which maximizes $d(w, R)$. Defining the distance to an edge (u, v) to be distance to u we find the $\sqrt{m}$ closest edges to w which cross from S to T. Let P be the set of T nodes incident to these edges. Let $D_2 = \max_{s \in S, v \in P} d(s, v)$ and $D_3 = \max_{t \in T} d(w, t)$. Our estimate is $D' = \max(D_1, D_2, D_3)$.

Analysis Step 2 Note that all 3 estimates are underestimates so we will just bound D' from below. Suppose $D_3 \geq D/2$ then we are already done. So we can assume that $d(w, t^*) < D/2$. Let (s, t) be the first edge going from S to T in the shortest path from w to t^*. If $D_1 < D/2$ then by Lemma 5.3.3, this edge is among the $\sqrt{m}$ closest edges to w. Hence $D_2 \geq d(s^*, t) \geq d(s^*, t^*) - d(t, t^*) \geq D - d(t, t^*) \geq D - d(w, t^*) \geq D/2$ □

Directed parameterized Bichromatic Diameter

For Bichromatic Diameter in undirected graphs, we assumed that only one of S' or T' was small (i.e. we set B to be the smaller of the two); however for directed graphs we impose that both S' and T' are small, by defining a new parameter $B' = S' \cup T'$.

Theorem 5.4.7. *There is an $O(m|B'|)$ time algorithm that, given an unweighted directed graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, returns an estimate D' such that $2D_{ST}(G)/3 \leq D' \leq D_{ST}(G)$.*

Proof.

Algorithm For all $v \in T$, we let $\varepsilon_{ST}(v)$ denote $\max_{s \in S} d(s, v)$ ($\varepsilon_{ST}(v)$ is already defined for $v \in S$). Run forward BFS from every vertex in S' and run backward BFS from every vertex in T'. Let D_1 be the largest $S \to T$ distance found. That is, $D_1 = \max_{v \in B'} \varepsilon_{ST}(v)$. Let $s \in S$ be the farthest vertex from B'. That is, s is the vertex in S that maximizes $d(s, B')$. Then, we run BFS from s and let $D_2 = \varepsilon_{ST}(s)$. Return $\max\{D_1, D_2\}$.

Analysis Let $s^* \in S$ and $t^* \in T$ be the true bichromatic diameter endpoints and let D denote $D_{ST}(G)$. If there exists a vertex $s' \in S'$ such that $d(s^*, s') \leq D/3$, then by the triangle inequality, $d(s', t^*) \geq 2D/3$ so $D_1 \geq 2D/3$ and we are done. Similarly, if there exists a vertex $t' \in T'$ such that $d(t', t^*) \leq D/3$, then by the triangle inequality, $d(s^*, t') \geq 2D/3$ so $D_1 \geq 2D/3$ and we are done.

Suppose we are not done. Then, for every vertex $s' \in S'$, $d(s^*, s') > D/3$ and for every vertex $t' \in T'$, $d(t', t^*) > D/3$. By choice of s, for all $s' \in S'$, $d(s, s') > D/3$. We observe that every path from s to t^* must contain an edge from a vertex in S' to a vertex in T'. Let $(s'' \in S', t'' \in T')$ be an edge on the shortest path from s to t^*. Then, $d(s, t^*) = d(s, s'') + d(s'', t'') + d(t'', t^*) > D/3 + 1 + D/3 = 2D/3 + 1$, so $D_2 \geq 2D/3 + 1$. □

5.4.4 Directed Subset Diameter

Proposition 3. *There is an $\tilde{O}(m)$ time algorithm, that given a directed graph $G = (V, E)$ with nonnegative integer weights and $S \subseteq V$, outputs an estimate D' such that $D_S/2 \leq D' \leq D_S$.*

Proof. Run Dijkstra's algorithm both "forward" and "backward" from s to obtain $D_1 = \max_{s' \in S} d(s, s')$ and $D_2 = \max_{s' \in S} d(s', s)$. Return $D' = \max\{D_1, D_2\}$.

Let $s^*, t^* \in S$ be the true endpoints of the subset diameter. Then, by the triangle inequality $D_S \leq d(s^*, s) + d(s, t^*)$. Then since $d(s^*, s) \leq D_2$ and $d(s, t^*) \leq D_1$, $D_S \leq D_1 + D_2$. Thus, $D_S/2 \leq \max\{D_1, D_2\} \leq D_S$. □

5.5 Conditional lower bounds

5.5.1 Directed ST-Diameter

Proposition 4. *Under SETH, any algorithm for ST-Diameter that achieves a finite approximation factor in m-edge directed graphs requires $m^{2-o(1)}$ time.*

Proof. Given an instance $U, V \subseteq \{0,1\}^d$ of OV, let $G(U, V)$ be its OV-graph. Now, direct the edges from U to C and from C to V and set $S = U$, $T = V$.

Now, for every $u \in U, v \in V$, if $u \cdot v \neq 0$, $d(u, v) = 2$ and if $u \cdot v = 0$, $d(u, v) = \infty$ as there is no path from u to v. Thus, if there is an OV pair, then the ST-diameter is ∞, and otherwise it is 2. Any finite approximation for ST-Diameter can distinguish between ∞ and 2, and thus can solve OV. Thus, there can be no $m^{2-\varepsilon}$ time algorithm for $\varepsilon > 0$ that achieves a finite approximation factor if SETH holds. □

5.5.2 Undirected Bichromatic Diameter

The following theorem implies that our algorithms for undirected Bichromatic Diameter from Theorem 5.4.4 and Proposition 2 are tight under SETH.

Theorem 5.5.1. *Under SETH, for every $k \geq 2$, every algorithm that can distinguish between bichromatic diameter $2k - 1$ and $4k - 3$ in undirected unweighted graphs requires $m^{1+1/(k-1)-o(1)}$ time.*

In particular setting $k = 2$ and 3 in Theorem 5.5.1 implies that our $m^{3/2}$ time 5/3-approximation algorithm from Theorem 5.4.4 is tight in approximation factor and runtime, respectively. Furthermore, setting k to be arbitrarily large implies that our $\tilde{O}(m)$ time almost 2-approximation algorithm from Proposition 2 is tight under SETH.

As a starting point, we use the construction for ST-Diameter from Chapter 3. In particular we start with the *k-OV graph* from the following theorem, which is a less detailed version of Theorem 3.2.2.

Theorem 5.5.2. *Let $k \geq 2$ be a fixed integer. Given a k-OV instance consisting of sets $W_0, W_1, \ldots, W_{k-1} \subseteq \{0,1\}^d$, each of size n, we can in $O(kn^{k-1}d^{k-1})$ time construct an unweighted, undirected graph with $O(n^{k-1} + kn^{k-2}d^{k-1})$ vertices and $O(kn^{k-1}d^{k-1})$ edges that satisfies the following properties.*

1. *The graph consists of $k+1$ layers of vertices $L_0, L_1, L_2, \ldots, L_k$. The number of nodes in the sets is $|L_0| = |L_k| = n^{k-1}$ and $|L_1|, |L_2|, \ldots, |L_{k-1}| \leq n^{k-2}d^{k-1}$.*
2. *L_0 consists of all tuples $(a_0, a_1, \ldots, a_{k-2})$ where for each i, $a_i \in W_i$. Similarly, L_k consists of all tuples $(b_1, b_2, \ldots, b_{k-1})$ where for each i, $b_i \in W_i$.*

3. *If the k-OV instance has no solution, then $d(u, v) = k$ for all $u \in L_0$ and $v \in L_k$.*

4. *If the k-OV instance has a solution $a_0, a_1, \ldots, a_{k-1}$ where for each i, $a_i \in W_i$ then if $\alpha = (a_0, \ldots a_{k-2}) \in L_0$ and $\beta = (a_1, \ldots, a_{k-1}) \in L_k$, then $d(\alpha, \beta) \geq 3k - 2$.*

5. *For all i from 1 to $k-1$, for all $v \in L_i$ there exists a vertex in L_{i-1} adjacent to v and a vertex in L_{i+1} adjacent to v.*

We will prove the following lemma, which immediately implies Theorem 5.5.1.

Lemma 5.5.1. *Let $k \geq 2$ be any integer. Given a k-OV instance, we can in $O(kn^{k-1}d^{k-1})$ time construct an unweighted, undirected graph with $O(kn^{k-1}+kn^{k-2}d^{k-1})$ vertices and $O(kn^{k-1}d^{k-1})$ edges that satisfies the following two properties.*

1. *If the k-OV instance has no solution, then for* all *pairs of vertices $u \in S$ and $v \in T$ we have $d(u, v) \leq 2k - 1$.*

2. *If the k-OV instance has a solution, then there exists a pair of vertices $u \in S$ and $v \in T$ such that $d(u, v) \geq 4k - 3$.*

Proof. We first describe the construction and then argue correctness.

Construction. We begin with the k-OV-graph from Theorem 5.5.2. Additionally, we add $k-1$ new layers of vertices $L_{k+1}, \ldots, L_{2k-1}$, where each new layer contains n^{k-1} vertices and is connected to the previous layer by a matching. That is, each new layer contains one vertex for every tuple $(a_1, \ldots, a_{k-1})$ where $a_i \in W_i$ for all i, and each $(a_1, \ldots, a_{k-1}) \in L_j$ is connected to its counterpart $(a_1, \ldots, a_{k-1}) \in L_{j-1}$ by an edge, for all j.

We let $S = L_0$ and we let T contain the rest of the vertices in the graph.

Correctness. *Case 1: The k-OV instance has no solution.* By property 3 of Theorem 5.5.2 for all $u \in S$ and $v \in L_k$, $d(u, v) = k$. Then, since $L_k, \ldots, L_{2k-1}$ form a series of matchings, for all $u \in S$ and $v \in L_{k+1} \cup \cdots \cup L_{2k-1}$, $d(u, v) \leq 2k-1$. Furthermore, property 5 of Theorem 5.5.2 implies that for all $u \in S$ and $v \in L_1 \cup \cdots \cup L_{k-1}$, $d(u, v) \leq 2k - 1$. Thus, we have shown that for all $u \in S$ and $v \in T$ we have $d(u, v) \leq 2k - 1$.

Case 2: The k-OV instance has a solution. Let $(a_0, a_1, \ldots, a_{k-1})$ be a solution to the k-OV instance where $a_i \in W_i$ for all i. We claim that $d((a_0, \ldots, a_{k-2}) \in S, (a_1, \ldots, a_{k-1}) \in L_{2k-1})) \geq 4k - 3$. Since $L_k, \ldots, L_{2k-1}$ form a series of matchings, every path from $(a_0, \ldots, a_{k-2}) \in S$ to $(a_1, \ldots, a_{k-1}) \in L_{2k-1}$ contains the vertex $(a_1, \ldots, a_{k-1}) \in L_k$. By property 4 of Theorem 5.5.2, $d((a_0, \ldots, a_{k-2}) \in S, (a_1, \ldots, a_{k-1}) \in L_k) \geq 3k-2$. Thus, $d((a_0, \ldots, a_{k-2}) \in S, (a_1, \ldots, a_{k-1}) \in L_{2k-1})) \geq 4k - 3$. □

Undirected parameterized Bichromatic Diameter

The following theorem implies that the multiplicative factor in our $\tilde{O}(m|B|)$ time almost 3/2-approximation algorithm for undirected Bichromatic Diameter from Theorem 5.4.5 is tight under SETH for $|B| = \omega(\log n)$.

Theorem 5.5.3. *For any integer $\ell > 0$, under SETH any algorithm for Bichromatic Diameter in undirected unweighted graphs that distinguishes between bichromatic diameter 4ℓ and 6ℓ requires $m^{2-o(1)}$ time, even for graphs with $|B| = d = \tilde{O}(1)$.*

Proof. We first describe the construction and then argue correctness.

Construction Given an instance $U, V \in \{0,1\}^d$ of OV, we begin with the OV-graph U, C, V defined on this instance. We add a new set U' of n vertices, one vertex for each vector in U, and connect each vertex in U to its corresponding vertex in U' to form a matching. Symmetrically, we add a new set V' of n vertices, one vertex for each vector in V, and connect each vertex in V to its corresponding vertex in V' to form a matching. Then we subdivide each of the edges in the graph into a path of length ℓ. Let T contain $C \cup V \cup V'$ as well as the vertices on the subdivision paths from C to V and from V to V'. Let S be the remaining vertices, that is, S contains U, U', the vertices that subdivide the edges between U and U', and the vertices that subdivide the edges between U and C.

Analysis We note that $T' = C$ and $|C| = d$ so $|B| = d = \tilde{O}(1)$.

If the OV instance has no solution then for every pair of vertices $u \in U$, $v \in V$, $d(u, v) = 2\ell$. Every vertex in S is at most distance ℓ from some vertex in U and every vertex in T is at most distance ℓ from some vertex in V so the bichromatic diameter is at most 4ℓ.

Suppose the OV instance has a solution $u \in U$, $v \in V$. We know that $d(u, v) \geq 4\ell$. Let u' be the vertex in U' that is matched to u and let v' be the vertex in V' that is matched to v. We claim that $d(u', v') \geq 6\ell$. Since U, U' and V, V' form matchings the only paths between u' and v' contain u and v. Thus, $d(u', v') = d(u', u) + d(u, v) + d(v, v') \geq 6\ell$. □

5.5.3 Directed Bichromatic Diameter

The following theorem implies that our $\tilde{O}(m^{3/2})$ 2-approximation algorithm for directed Bichromatic Diameter from Theorem 5.4.6 has a tight approximation factor under SETH.

Theorem 5.5.4. *Under SETH, any algorithm for directed Bichromatic Diameter that achieves a $(2-\delta)$-approximation factor for $\delta > 0$ in m-edge graphs requires $m^{2-o(1)}$ time.*

Proof. We will show that under SETH, for any positive integer ℓ, distinguishing between bichromatic diameter $\ell + 1$ and $2\ell + 1$ requires $m^{2-o(1)}$ time.

Given an instance $U, V \subseteq \{0,1\}^d$ of OV, let $G(U,V)$ be its OV-graph. Create G' which has the same vertex set as $G(U,V)$ except instead of having one vertex for every $v \in V$ it has ℓ copies $v_i \in V_i$ for $1 \leq i \leq \ell$. It also has $\ell - 2$ additional vertices: $P = \{p_1, p_2, \ldots, p_{\ell-2}\}$.

The edges of G' are: for $u \in U, c \in C$, we add (u,c) as an edge iff $u[c] = 1$, and for $c \in C, v \in V$, we add (c, v_1) as an edge iff $v[c] = 1$. We add a matching going from V_i to V_{i+1} where edges join the nodes which are copies of each other. For each $c \in C$, we add an edge (c, p_1). We add a path from p_1 to $p_{\ell-2}$. For each $u \in U$, we add an edge $(p_{\ell-2}, u)$. Set $S = U, T = C \cup P \cup V_1 \cup V_2 \ldots V_\ell$. The number of edges in the graph is $O(nd)$.

Consider any $u \in U$. By construction, $d(u,z) \leq \ell + 1$ for $z \in C \cup P$. Suppose that there is no OV solution, then for all $u \in U, v \in V$, $u \cdot v \neq 0$ and hence $d(u, v_i) \leq \ell + 1$. If there is an OV solution $u \in U, v \in V$, then, $d(u, v_\ell) \geq 2\ell + 1$ as the only path is through P. □

Directed parameterized Bichromatic Diameter

Recall that for directed graphs, S' is the set of vertices in S with an outgoing edge to a vertex in T, T' is the set of vertices in T with an incoming edge from a vertex in S, and $B' = S' \cup T'$. We will show that the construction from Theorem 5.5.3 can be made to have small B' (i.e. small S' and T'), with a slight additive cost to the diameter values. The construction will remain undirected.

The following proposition implies that the multiplicative factor in our $\tilde{O}(m|B'|)$ time almost 3/2-approximation algorithm for directed Bichromatic Diameter from Theorem 5.4.7 is tight under SETH for $|B'| = \omega(\log n)$.

Proposition 5. *For any integer $\ell > 0$, under SETH any algorithm for Bichromatic Diameter in directed unweighted graphs that distinguishes between bichromatic diameter $4\ell + 1$ and $6\ell + 1$ requires $m^{2-o(1)}$ time, even for graphs with $|B| = d = \tilde{O}(1)$.*

Proof. We first describe the construction and then argue correctness.

Construction We begin with the construction from Theorem 5.5.3. We replace each vertex $c \in C$ by a pair of vertices c_1, c_2 and let (c_1, c_2) be an edge. Let C_1 and C_2 be the set of all c_1's and c_2's respectively. That is, C_1 and C_2 form a matching. For every edge originally between $u \in U$ and $c \in C$, we replace it with the undirected edge (u, c_1) and for every edge originally between $c \in C$ and $v \in V$, we replace it with the undirected edge (c_2, v).

Analysis The correctness follows from the analysis of Theorem 5.5.3. Here, we get $4\ell + 1$ and $6\ell + 1$ instead of 4ℓ and 6ℓ due to the addition of the matching between C_1 and C_2. □

5.5.4 Undirected Subset Diameter

The following proposition implies that our $\tilde{O}(m)$ time 2-approximation algorithm for Subset Diameter from Proposition 3 is tight under SETH.

Proposition 6. *Under SETH, any algorithm for Subset Diameter that achieves a $(2-\delta)$-approximation for $\delta > 0$ in m-edge directed graphs requires $m^{2-o(1)}$ time.*

Proof. Given an instance $U, V \in \{0,1\}^d$ of OV, we begin with the OV-graph defined on this instance. We add a vertex u adjacent to every vertex in U and a vertex v adjacent to every vertex in V. Let $S = U \cup V$.

If there is no OV solution, every pair of vertices $s \in U$, $s' \in V$ $d(s, s') = 2$. Also, every pair of vertices $s, s' \in U$ or $s, s' \in V$ has $d(s, s') = 2$ due to the addition of the vertices u and v.

On the other hand, if there is an OV solution, in the original OV-graph there exists $s \in U$, $s' \in V$ such that $d(s, s') = 4$. We note that the addition of the vertices u and v does not change this fact. □

Chapter 6

Min-Diameter

6.1 Result

In this chapter, we obtain a subquadratic-time constant-factor approximation algorithm for Min-Diameter. Specifically, we prove the following theorem.

Theorem 6.1.1. *For any integer $0 < \ell \leq O(\log n)$, there is an $\tilde{O}(mn^{1/(\ell+1)})$ time randomized algorithm that, given a directed weighted graph G with edge weights non-negative and polynomial in n, can output an estimate $\tilde{D}$ such that $D/(4\ell-1) \leq \tilde{D} \leq D$ with high probability, where D is the min-diameter of G.*

When we set $\ell = 1$, we obtain an $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm, and when we set $\ell = \lceil \log n \rceil$, we get an $\tilde{O}(m)$ time $O(\log n)$-approximation.

Our trade-off achieves the first constant factor approximation algorithms for Min-Diameter in general graphs that run in $O(mn^{1-\varepsilon})$ time for constant $\varepsilon > 0$. Such a result was only known for directed acyclic graphs [AVW16], whereas for general graphs the only known efficient algorithm could achieve an n^{ε}-approximation.

Throughout this chapter, let D be the min-diameter, let s^*, t^* the endpoints of the min-diameter, and for a vertex v let the *min-eccentricity* $\epsilon(v)$ be $\max_{u \in V} d_{min}(u, v)$.

6.2 Techniques

The main difference between Min-Diameter and the other variants of Diameter in this dissertation is that min-distances do not obey the *triangle inequality*. All known algorithms for Diameter and related problems rely heavily on the triangle inequality, so completely new techniques are necessary for approximating Min-Diameter. The following outlines several techniques that we use.

Partial search graphs. The idea of partial search graphs is used in the algorithms of [AVW16] for Min-Diameter on DAGs. This algorithms uses the following high-level framework: perform Dijkstra's algorithm from some vertices and then perform a *partial* Dijkstra's algorithm from *every* vertex. The partial search from a vertex v is with respect to a carefully defined partial search graph $G_v \subset G$. The

crux of the analysis for the algorithms on DAGs is to argue that if the executions of Dijkstra's algorithm on the full graph did not find a good estimate for the min-diameter, then the partial search from some vertex v returns a good estimate of the largest min-distance from v, which in turn is a good estimate of the min-diameter. In DAGs it is natural to define the partial search graphs G_v by considering a topological ordering of the vertices and letting each G_v be a particular carefully chosen interval of the ordering containing v. For general graphs it is completely unclear how to even define such intervals since there is no natural notion of an ordering of the vertices, and thus figuring out what the G_v's should be is nontrivial. Our approach to overcoming this hurdle is to define a DAG-like structure in general graphs. Such a structure may be of independent interest.

Defining a DAG-like structure in general graphs. As a first step, we use the following idea. Suppose we have performed Dijkstra's algorithm from a vertex v. We let $S_v = \{u : d(u,v) < d(v,u)\}$ and we let $T_v = \{u : d(u,v) > d(v,u)\}$[1]. Then, we partially order the vertices so that the vertices in S_v appear before v and those in T_v appear after v. We note that this partial ordering is "DAG-like" because it is consistent with the topological ordering of a DAG; that is, if we apply this partition into S_v and T_v to a DAG then there trivially exists a topological ordering such that every vertex in S_v appears before v and every vertex in T_v appears after v. After partitioning into S_v and T_v, we recursively partition each set to create a more precise partial ordering. Importantly, we show that by recursively sampling vertices randomly, we can guarantee that our partitioning is approximately balanced which is crucial for the running time analysis. The obtained partial ordering is the starting point for our algorithm.

Graph augmentation. The Min-Diameter algorithm on DAGs from [AVW16] relies heavily on the following key property of DAGs. Consider a topological ordering and the graphs induced by the first and second halves of the ordering; which are defined with respect to the middle vertex in the ordering. For all pairs of vertices in the same half of the ordering, their min-distance in the graph induced by this half is the same as their min-distance in the full graph. As previously mentioned, if we sample a vertex v, we can make sure that S_v and T_v are approximately balanced, so that we can think of S_v and T_v as corresponding to the first and second half of a DAG topological ordering, respectively. However it is unclear how to obtain a property of S_v and T_v analogous to the above key property of DAGs. In particular, the min-distance between a pair of vertices in the graph induced by S_v could be wildly different from their min-distance in the full graph, since paths whose endpoints are in S_v can contain vertices outside of S_v. To overcome this hurdle, we *augment* the graph induced by S_v and the graph induced by T_v by carefully adding edges so that distances within these augmented graphs approximate the large min-distances in the original graph (we do not care about preserving the small min-distances in the graph since we only care about approximating the min-diameter).

[1] u's with $d(u,v) = d(v,u)$ are added to either S_v or T_v as specified in the formal definition later

6.3 Overview of algorithm

Algorithm for DAGs

We begin by outlining the $\tilde{O}(n+m)$ time 2-approximation algorithm for Min-Diameter on DAGs from [AVW16]. Consider a topological ordering of the vertices and perform Dijkstra's algorithm from the middle vertex v. Then recurse on the graphs induced by the vertices in the first half (before v) and in the second half (after v). A key observation in the analysis is that if the true endpoints s^* and t^* of the min-diameter fall on opposite sides of v in the ordering, then the min-eccentricity $\varepsilon(v)$ of v is a 2-approximation for the min-diameter D. This is because if $\varepsilon(v) < D/2$ and s^* and t^* fall on opposite sides of v in the ordering, then $d(s^*, v) < D/2$ and $d(v, t^*) < D/2$ so $d(s^*, t^*) < D$, a contradiction. So, suppose (without loss of generality) that s^* and t^* both fall before v in the ordering. Since the graph is a DAG, every path between s^* and t^* only uses vertices before v in the ordering. Thus, the min-distance between s^* and t^* in the graph induced by the first half of the graph is still D.

Algorithm for general graphs

We now outline a precursor to our Min-Diameter algorithm for general graphs that mimics the algorithm for DAGs. This $\tilde{O}(n + m)$ time algorithm does not achieve a constant approximation factor, however it provides intuition for our constant-factor approximation algorithms. We begin by performing Dijkstra's algorithm from a vertex v and constructing S_v and T_v as defined in the previous section. Analogously to the DAG algorithm if the true min-diameter endpoints s^* and t^* fall into different sets S_v, T_v then the min-eccentricity $\varepsilon(v)$ is a 2-approximation. This is because if $\varepsilon(v) < D/2$, $s^* \in S_v$, and $t^* \in T_v$ then $d(s^*, v) < D/2$ and $d(v, t^*) < D/2$ so $d(s^*, t^*) < D$, a contradiction. However, unlike the DAG algorithm, we cannot simply recurse independently on the graphs induced by S_v and T_v since the shortest path between a pair of vertices in S_v may not be completely contained in S_v (and analogously for T_v).

To overcome this hurdle, before recursing we first augment the graphs induced by S_v and T_v by carefully adding edges so that distances within these augmented graphs approximate distances in the original graph. Specifically, for every vertex $u \in S_v$, we add the directed edge (u, v) with weight 0 and the directed edge (v, u) with weight $\max\{0, d(v, u) - \varepsilon(v)\}$. This choice of edges allows us to argue that the distances within the augmented graphs are approximations of the distances in G up to an additive error of $2\varepsilon(v)$. Then, by returning the maximum of $\varepsilon(v)$ and the min-diameter estimates from recursing on the augmented graphs, we get an approximation guarantee, which turns out to be a logarithmic factor. Intuitively, the approximation factor is not constant because the recursion causes the distance distortion to compound at each level of recursion.

To reduce the approximation factor to a constant, we would like to decrease the number of recursion levels. To achieve this, we initially partition the graph into more than just two parts S_v and T_v, by sampling more vertices. For our $\tilde{O}(m\sqrt{n})$ time 3-approximation, we perform a full Dijkstra's algorithm from $\tilde{O}(\sqrt{n})$ vertices to define

an ordered partition of the vertices into $\tilde{O}(\sqrt{n})$ parts of $\tilde{O}(\sqrt{n})$ vertices each. Then we apply the above idea of adding weighted edges within each part, however we must refine the definition of the graph augmentation to take into account *all* of the $\tilde{O}(\sqrt{n})$ vertices we initially perform Dijkstra's algorithm from, instead of just v. Finally we use brute force (without recursion) on each part in the partition by running an exact all-pairs shortest paths algorithm.

To achieve our time-accuracy trade-off algorithm, we carefully combine ideas from the logarithmic factor approximation and the 3-approximation algorithms. Specifically, we initially perform Dijkstra's algorithm from fewer than $\sqrt{n}$ vertices to define an ordered partition with larger parts than in the 3-approximation. Then we augment the graph induced by each part and carry out a constant number of recursion levels to further partition the graph before applying brute-force.

6.4 Algorithm

6.4.1 Preliminary graph partitioning

In this section we describe a graph partitioning procedure that we use as a first step in our algorithm. The goal of this partitioning is to define a DAG-like structure in general directed graphs.

Definition 6.4.1. *Assign each vertex a unique ID from $[n]$. For each vertex v, let $S_v = \{u \in V : d(u,v) < d(v,u) \vee [d(u,v) = d(v,u) \wedge ID(u) < ID(v)]\}$. Let $T_v = V \setminus (S_v \cup \{v\})$.*

The running time of our algorithms relies on whether the partition into S_v and T_v is *balanced*. Using the observation that if $u \in S_v$, then $v \in T_u$, the following lemma shows that for most vertices, the partition is indeed approximately balanced.

Lemma 6.4.1. *For any graph on n vertices there are more than $\frac{n}{2}$ vertices v such that $\frac{|S_v|}{8} \leq |T_v| \leq 8|S_v|$.*

More generally, for any $U \subseteq V$, there are more than $\frac{|U|}{2}$ vertices $v \in U$ such that $\frac{|S_v \cap U|}{8} \leq |T_v \cap U| \leq 8|S_v \cap U|$.

Proof. Since the first statement is a special case of the second statement with $U = V$, we prove the more general statement. Let $|U| = k$. Let M be a $k \times k$ matrix indexed by the vertices in U where $M_{u,v} = -1$ if $u \in S_v \cap U$, $M_{u,v} = 1$ if $u \in T_v \cap U$, and $M_{u,u} = 0$ for $u \in U$. Note that M is skew-symmetric, i.e., $M_{u,v} = -M_{v,u}$ for all u, v. For any $A, B \subseteq U$, let M_B be the $k \times |B|$ submatrix consisting of the columns indexed by B, and let $M_{A,B}$ the $|A| \times |B|$ submatrix of M_B consisting of its rows indexed by A.

Suppose for contradiction there is a set $C \subset U$ of $\frac{k}{4}$ vertices v such that $|T_v \cap U| > 8|S_v \cap U|$. Then M_C contains at least $\frac{8}{9}k \cdot \frac{k}{4} = \frac{2}{9}k^2$ ones.

The $\frac{k}{4} \times \frac{k}{4}$ submatrix $M_{C,C}$ is also skew-symmetric, so at most half of its entries are ones, i.e., $M_{C,C}$ contains at most $\frac{k^2}{32}$ ones. Letting $\bar{C} = U \setminus C$, we see that $M_{\bar{C},C}$

has $\frac{3}{4}k \times \frac{k}{4} = \frac{3}{16}k^2$ entries, and hence at most $\frac{3}{16}k^2$ ones. In total, M_C contains at most $\frac{7}{32}k^2 < \frac{2}{9}k^2$ ones, contradiction.

Therefore the number of vertices $v \in U$ such that $|T_v \cap U| > 8|S_v \cap U|$ is less than $\frac{k}{4}$, and symmetrically the number of vertices $v \in U$ such that $|T_v \cap U| < \frac{|S_v \cap U|}{8}$ is less than $\frac{k}{4}$. Hence more than half of the vertices $v \in U$ have that

$$\frac{|S_v \cap U|}{8} < |T_v \cap U| < 8|S_v \cap U|.$$

□

Next, we describe how we use Lemma 6.4.1 to recursively construct a balanced partition of the vertices into a given number of of sets.

Lemma 6.4.2. *Given a graph G with n vertices and a constant $c > 0$, in $\tilde{O}(mn^{1-c})$ time we can partition V into disjoint sets $W, V_1, V_2, \ldots, V_{q+1}$, where $q = |W| = n^{1-c}$, such that with high probability:*

1. *for all i, $|V_i| = \Theta(\frac{n}{q})$;*
2. *for all $i \neq j$, there exists a vertex $w \in W$ such that either $V_i \subseteq S_w$,$V_j \subseteq T_w$, or $V_i \subseteq T_w$,$V_j \subseteq S_w$;*
3. *for all $U \subseteq W$, let $V_U = \left(\bigcap_{w \in U} S_w\right) \bigcap \left(\bigcap_{w \in W \setminus U} T_w\right)$, then $V_U \subseteq V_i$ for some $i \in [q+1]$.*

Proof. We begin with $W = \emptyset$ and we will iteratively populate W with vertices. We let $\mathcal{V}_0 = \{V\}$ and for all $i \in [q]$ when we add the i^{th} vertex to W, we will construct $\mathcal{V}_i$ from $\mathcal{V}_{i-1}$ by partitioning the largest set in $\mathcal{V}_{i-1}$ into two parts. After adding q vertices to W we will have constructed $\mathcal{V}_q = \{V_1 \ldots V_{q+1}\}$.

For all $i \in [q]$, let A_i, B_i be the largest and smallest sets in $\mathcal{V}_i$, respectively.

We describe how to construct W and $\mathcal{V}_q$ inductively. Suppose $|W| = r - 1$ and we have constructed $\mathcal{V}_{r-1}$. By Lemma 6.4.1, if we randomly sample $O(\log^2 n)$ vertices from A_{r-1}, with probability at least $1 - 2^{-\log^2 n} = 1 - n^{-\log n}$ we will sample a vertex w_r such that $A_S = A_{r-1} \cap S_{w_r}$ and $A_T = A_{r-1} \cap T_{w_r}$ differ by a factor of at most 8. We add w_r to W and let $\mathcal{V}_r = \mathcal{V}_{r-1} \cup \{A_S, A_T\} \setminus \{A_{r-1}\}$.

By union bound over the $q = n^{1-c}$ partitionings, we have that with probability at least $1 - n^{1-c-\log n}$, every partitioning produces two sets that differ in size by a factor of at most 8.

We prove property 1 by induction on $|W| = r$. Specifically, we will show that for all $r \in [q]$, $|A_r| \leq 9|B_r|$. This implies that $|A_q| = O(|B_q|)$, and property 1 follows. Lemma 6.4.1 implies that $|A_1| \leq 9|B_1|$. Assume inductively that $|A_{r-1}| \leq 9|B_{r-1}|$. Since no subset grows in size, $|A_r| \leq |A_{r-1}|$ and $|B_r| \leq |B_{r-1}|$. If $|B_r| = |B_{r-1}|$, then $|A_r| \leq |A_{r-1}| \leq 9|B_{r-1}| = 9|B_r|$. Otherwise, $|B_r| < |B_{r-1}|$, which implies that B_r is one of the two sets obtained by partitioning A_{r-1}. In this case $|A_{r-1}| \leq 9|B_r|$ by Lemma 6.4.1. Hence $|A_r| \leq |A_{r-1}| \leq 9|B_r|$, completing the induction.

Property 2 follows from the partitioning procedure: for any $i \neq j$, if for all $w \in W$, $V_i, V_j \subseteq S_w$ or $V_i, V_j \subseteq T_w$ then $V_i \cup V_j$ would never have been partitioned.

Property 3 also follows from the partitioning procedure: observe that for all $w \in W$ and all $U \subseteq W$, $V_U \subseteq S_w$ or $V_U \subseteq T_w$, so V_U is never partitioned and thus $V_U \subseteq V_i$ for some $i \in [q+1]$.

Since we sample $n^{1-c}\log^2 n$ vertices and for all v finding S_v, T_v takes $O(m)$ time, the running time is $\tilde{O}(mn^{1-c})$. □

6.4.2 An $\tilde{O}(m\sqrt{n})$ time 3-approximation

Theorem 6.4.1. (Theorem 6.1.1 *with $\ell = 1$) There is an $\tilde{O}(m\sqrt{n})$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with edge weights non-negative and polynomial in n, can output an estimate $\tilde{D}$ such that $D/3 \leq \tilde{D} \leq D$ with high probability, where D is the min-diameter of G.*

Algorithm Description

Applying Lemma 6.4.2 with $q = \sqrt{n}$ we obtain a partition of the vertices into $W, V_1, V_2, \ldots, V_{\sqrt{n}+1}$.

We perform Dijkstra's algorithm from every vertex in W and let $D' = \max_{w \in W} \varepsilon(w)$. We will later show that D' is a good approximation of the min-diameter when s^* and t^* are not in the same vertex set V_i.

For every $i \in [\sqrt{n}+1]$, define $W_i^S = \{w \in W : V_i \subseteq S_w\}$, and $W_i^T = \{w \in W : V_i \subseteq T_w\}$. Then, for every i, we construct two graphs G_i^S and G_i^T. The first graph G_i^S contains all vertices of V_i and an additional node w_i^S. It has the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i^S.
2. Add a directed edge from w_i^S to every $v \in V_i$, with weight
$$\max\left\{\min_{w \in W_i^S} d(w, v) - D', 0\right\}$$
and a directed edge from every $v \in V_i$ to w_i^S with weight 0.

The second graph G_i^T is symmetric to G_i^S. It contains all vertices in V_i and an additional node w_i^T. It has the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i^T.
2. Add a directed edge from every $v \in V_i$ to w_i^T, with weight
$$\max\left\{\min_{w \in W_i^T} d(v, w) - D', 0\right\}$$
and add a directed edge from w_i^T to every $v \in V_i$ with weight 0.

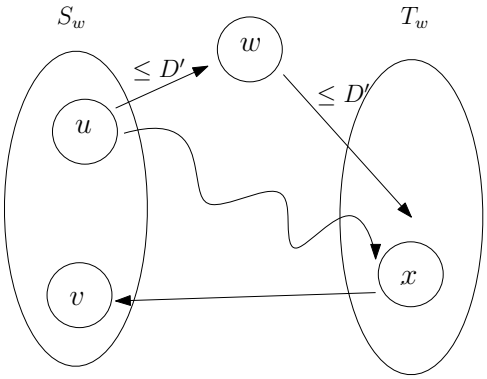

Figure 6-1: The case where $u, v \in S_w$ and the shortest path from u to v contains a node $x \in T_w \cup \{w\}$.

For all i, we run an exact all-pairs shortest paths algorithm on G_i^S and G_i^T. This allows us to compute for all i and all $u, v \in V_i$ the quantity $\min\{d_{G_i^S}(u,v), d_{G_i^T}(u,v)\}$, which we denote by $d'_i(u,v)$.

We choose the larger between D' and $\max_{i\in[\sqrt{n}+1],u,v\in V_i} \min\{d'_i(u,v), d'_i(v,u)\}$ as our final estimate for the min-diameter.

Analysis

The following lemma will be used to show that D' is a good estimate for the min-diameter if s^* and t^* happen to fall into different sets V_i

Lemma 6.4.3. *For all vertices v, if either $s^* \in S_v$, $t^* \in T_v$, or $t^* \in S_v$, $s^* \in T_v$, then $\varepsilon(v) \geq D/2$.*

Proof. We only consider the case when $s^* \in S_v$ and $t^* \in T_v$ as the other case is symmetric. By way of contradiction, assume that $\varepsilon(v) < D/2$, then we have $d_{\min}(s^*, v) < D/2$ and $d_{\min}(t^*, v) < D/2$. Since $s^* \in S_v$, $d(s^*, v) = d_{\min}(s^*, v) < D/2$; similarly, since $t^* \in T_v$, $d(v, t^*) = d_{min}(t^*, v) < D/2$. Therefore, by the triangle inequality, $d(s^*, t^*) < D$, a contradiction. □

The next two lemmas are used for the case where s^* and t^* fall into the same set V_i.

Lemma 6.4.4. *For every i, and every pair of vertices $u, v \in V_i$, $d'_i(u,v) \leq d(u,v)$; that is,*
$\min\{d_{G_i^S}(u,v), d_{G_i^T}(u,v)\} \leq d(u,v)$.

Proof. Take any shortest path in the original graph G from u to v. If this path does not leave V_i, then this path also exists in G_i^S and G_i^T, and thus the inequality is true.

It remains to prove for the case when the shortest u, v path in the original graph leaves V_i. Let $x \notin V_i$ be any vertex on a shortest u, v path. By Lemma 6.4.2, property 2, there exists $w \in W$ such that $x \in S_w \cup \{w\}$ and $V_i \subseteq T_w$, or $x \in T_w \cup \{w\}$ and $V_i \subseteq S_w$. We first assume $x \in T_w \cup \{w\}$ and $V_i \subseteq S_w$ as shown in Figure 6-1, and the other case is symmetric.

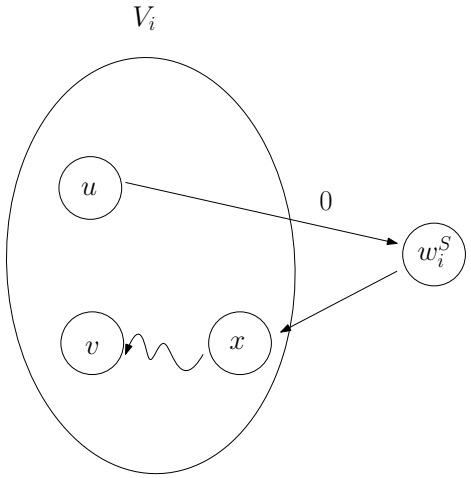

Figure 6-2: A shortest u, v path in G_i^S that contains w_i^S. The path goes from u, directly to w_i^S using a weight 0 edge, then directly to a vertex x, and finally reaches v.

Since x is on the shortest path from u to v, we have $d(u,v) \geq d(x,v)$. Also, we have $d(w,x) \leq D'$, by definition of D'. Therefore,

$$\begin{aligned} d(u,v) &\geq d(x,v) \\ &\geq d(x,v) + (d(w,x) - D') \\ &\geq d(w,v) - D' \end{aligned} \tag{6.1}$$

Now consider the path $u \to w_i^S \to v$ in G_i^S. The first part $u \to w_i^S$ costs 0, because there is an edge from u to w_i^S with weight 0; the second part $w_i^S \to v$ costs at most $\max\{0, d(w,v) - D'\}$. If $d(w,v) < D'$, then $d'_i(u,v) \leq d_{G_i^S}(u,v) = 0 \leq d(u,v)$; otherwise, $d'_i(u,v) \leq d_{G_i^S}(u,v) \leq d(w,v) - D' \leq d(u,v)$, where the last step is Equation 6.1.

When $x \in S_w \cup \{w\}$, and $V_i \subseteq T_w$, we have a symmetric argument: $d(u,v) \geq d(u,x) \geq d(u,x) + (d(x,w) - D') \geq d(u,w) - D'$. Consider the path $u \to w_i^T \to v$ in G_i^T. The second part $w_i^T \to v$ costs 0, because there is an edge from w_i^T to v with weight 0; the first part $u \to w_i^T$ costs at most $\max\{0, d(u,w) - D'\}$. If $d(u,w) < D'$, then $d'_i(u,v) \leq d_{G_i^T}(u,v) = 0 \leq d(u,v)$; otherwise, $d'_i(u,v) \leq d_{G_i^T}(u,v) \leq d(u,w) - D' \leq d(u,v)$. □

Lemma 6.4.5. *For every i, and every pair of vertices $u, v \in V_i$, it holds that $d'_i(u,v) \geq d(u,v) - 2D'$; that is, it holds that $d_{G_i^S}(u,v) \geq d(u,v) - 2D'$ and $d_{G_i^T}(u,v) \geq d(u,v) - 2D'$.*

Proof. We only provide full proof for $d_{G_i^S}(u,v) \geq d(u,v) - 2D'$. The inequality for G_i^T can be proved by a symmetrical argument. If the shortest path from u to v in G_i^S does not contain w_i^S, then this path also exists in the original graph G, and thus the inequality is true.

Otherwise, the shortest path from u to v in G_i^S contains w_i^S, as shown in Figure 6-2. All edges on the shortest path from w_i^S to v exist in the original graph G except for the first edge from w_i^S to some node x, since a shortest path cannot use the vertex w_i^S more than once. That is, $d_{G_i^S}(x, v) = d(x, v)$.

By the definition of w_i^S and the edges incident to it, there exists a $w \in W_i^S$ such that $d(w, x) \leq d_{G_i^S}(w_i^S, x) + D'$. Thus, we have

$$\begin{aligned}
d_{G_i^S}(u, v) &= d_{G_i^S}(u, w_i^S) + d_{G_i^S}(w_i^S, x) + d_{G_i^S}(x, v) \\
&= d_{G_i^S}(w_i^S, x) + d_{G_i^S}(x, v) && \text{since } d_{G_i^S}(u, w_i^S) = 0 \text{ by construction} \\
&= d_{G_i^S}(w_i^S, x) + d(x, v) && \text{from argument above} \\
&\geq d(w, x) - D' + d(x, v) && \text{by the definition of } w \\
&\geq d(w, v) - D' && \text{by the triangle inequality} \\
&\geq (d(w, v) - D') + (d(u, w) - D') && \text{since } d(u, w) \leq D' \text{ by definition} \\
&\geq d(u, v) - 2D' && \text{by the triangle inequality}
\end{aligned}$$

□

We are now ready to prove our approximation ratio guarantee: $D/3 \leq \tilde{D} \leq D$. Clearly $D' \leq D$ because D' is the min-eccentricity of a vertex. By Lemma 6.4.4 $\max_{i,u \in V_i, v \in V_i} \min\{d_i'(u, v), d_i'(v, u)\} \leq \max_{i,u \in V_i, v \in V_i} d_{min}(u, v) \leq D$. Therefore, we never over estimate the min-diameter.

If $s^* \in W$ or $t^* \in W$, then since we run Dijkstra from all vertices in W we have $D' = D$. So assuming that $s^*, t^* \notin W$, we have two cases.

Case 1: s^* and t^* are not in the same vertex set V_i. By Lemma 6.4.2, property 2, there exists $w \in W$ such that one of s^* and t^* is in S_w and the other is in T_w, so by Lemma 6.4.3, $\varepsilon(w) \geq D/2$. Since $D' \geq \varepsilon(w)$, we have $D' \geq D/2$.

Case 2: s^* and t^* are in the same vertex set V_i for some i. By Lemma 6.4.5, $\min(d_i'(s^*, t^*), d_i'(t^*, s^*)) \geq d_{min}(s^*, t^*) - 2D' = D - 2D'$. Since $\max\{D - 2D', D'\} \geq D/3$, we get a 3-approximation.

Running time analysis. It takes $\tilde{O}(m\sqrt{n})$ time to perform the partitioning from Lemma 6.4.2 and to perform Dijkstra's algorithm from all $w \in W$ since $|W| = O(\sqrt{n})$.

For all i, the number of vertices in G_i^S is $|V_i| + 1 = O(\sqrt{n})$ with high probability by property 1 of Lemma 6.4.2 and the number of edges is $m_i + O(\sqrt{n})$ where m_i is the number of edges in the graph induced by V_i. Hence we can run an All-Pairs Shortest Paths algorithm on G_i^S in time $\tilde{O}((m_i + \sqrt{n})\sqrt{n})$. Summing over all i gives us $\tilde{O}(m\sqrt{n})$. The same analysis also works for G_T^i.

6.4.3 Time/accuracy trade-off algorithm

Algorithm Description

We begin by briefly outlining the differences between our trade-off algorithm and our $O(m\sqrt{n})$ time algorithm. For our trade-off algorithm, instead of applying Lemma 6.4.2 to sample $q = \sqrt{n}$ vertices, we will apply Lemma 6.4.2 with a smaller value of q to save time. This results in a smaller set W and larger sets V_i. In our $O(m\sqrt{n})$ time algorithm, we had time to apply brute force (i.e. run all-pairs shortest paths) on the graphs G_i^S and G_i^T, however in our trade-off algorithm we do not. Instead, we apply recursion. Simply constructing G_i^S and G_i^T and recursing on both of them does not suffice because each recursive call only returns the min-diameter, whereas we require knowing all distances. To overcome this issue, instead of constructing G_i^S and G_i^T separately, we construct a graph G_i that combines these two graphs. Then, we show that it suffices to recurse on G_i to compute only its min-diameter rather than all distances.

The algorithm is as follows. We apply Lemma 6.4.2 with $q = O(n^{1/(\ell+1)})$ to partition the vertices into $W, V_1, V_2, \ldots, V_{q+1}$. We perform Dijkstra's algorithm from every vertex in W and define $D' = \max_{w \in W} \varepsilon(w)$. For every $i \in [\sqrt{n} + 1]$, we define $W_i^S = \{w \in W : V_i \subseteq S_w\}$, and $W_i^T = \{w \in W : V_i \subseteq T_w\}$. For every $i \in [q + 1]$, we construct the graph G_i as follows. The vertex set of G_i is all vertices V_i and two additional vertices w_i^S and w_i^T. It contains the following edges:

1. For every directed edge $(u, v) \in E$ such that $u, v \in V_i$, add this edge to G_i.
2. Add a directed edge from w_i^S to every $v \in V_i$, with weight

$$\max\{\min_{w \in W_i^S} d(w, v) - D', 0\}$$

   and add a directed edge from every v to w_i^S with weight 0.
3. Add a directed edge from every $v \in V_i$ to w_i^T, with weight

$$\max\{\min_{w \in W_i^T} d(v, w) - D', 0\}$$

   and add a directed edge from w_i^T to every $v \in V_i$ with weight 0.

For all i, we recursively compute a $(4\ell - 5)$-approximation for Min-Diameter of G_i by calling the algorithm for $\ell - 1$. We use the $\ell = 1$ algorithm from the previous section as the base case.

We choose the larger between D' and the maximum approximated min-diameter over all G_i as our final estimate.

Analysis

Before proving the main theorem for Min-Diameter, we need to prove two lemmas for G_i, which are analogous to Lemma 6.4.4 and Lemma 6.4.5.

Lemma 6.4.6. *For every i, and every pair of vertices $u, v \in V_i$, $d(u,v) \geq d_{G_i}(u,v)$.*

Proof. Since $G_i^S \subseteq G_i$ and $G_i^T \subseteq G_i$, we have $d_{G_i}(u,v) \leq d_{G_i^S}(u,v)$ and $d_{G_i}(u,v) \leq d_{G_i^T}(u,v)$. Then by Lemma 6.4.4, we have $d(u,v) \geq \min\{d_{G_i^S}(u,v), d_{G_i^T}(u,v)\} \geq d_{G_i}(u,v)$. □

Lemma 6.4.7. *For every i, and every pair of vertices $u, v \in V_i$, it holds that $d_{G_i}(u,v) \geq d(u,v) - 4D'$.*

Proof. Consider the shortest path from u to v in G_i. If this path does not contain both w_i^S and w_i^T, then this path exists in G_i^S or G_i^T, and thus we can directly apply Lemma 6.4.5 to get $d_{G_i}(u,v) \geq d_{G_i^S}(u,v) \geq d(u,v) - 2D'$, or $d_{G_i}(u,v) \geq d_{G_i^T}(u,v) \geq d(u,v) - 2D'$.

Otherwise, the shortest path from u to v contain both w_i^S and w_i^T. Such path can only be one of the following two forms:

- $u \to w_i^S \to x \to w_i^T \to v$ for some vertex $x \in V_i$. The first half $u \to w_i^S \to x$ is contained in G_i^S, so we can apply Lemma 6.4.5 to get $d_{G_i}(u,x) = d_{G_i^S}(u,x) \geq d(u,x) - 2D'$; similarly, the second half $x \to w_i^T \to v$ is contained in G_i^T so $d_{G_i}(x,v) \geq d(x,v) - 2D'$. In total, $d_{G_i}(u,v) = d_{G_i}(u,x) + d_{G_i}(x,v) \geq (d(u,x) - 2D') + (d(x,v) - 2D') \geq d(u,v) - 4D'$.

- $u \to w_i^T \to x \to w_i^S \to v$ for some vertex $x \in V_i$. We can similarly split this path to two halves, and apply the same analysis as the previous case to get $d_{G_i}(u,v) \geq d(u,v) - 4D'$.

□

We are now ready to prove our approximation ratio: $D/(4\ell - 1) \leq \tilde{D} \leq D$. We prove the result inductively. When $\ell = 1$, it is exactly Theorem 6.4.1. Now assume it is true for $\ell - 1$, and we will prove it for ℓ.

Clearly $D' \leq D$ because D' is the min-eccentricity of a vertex. By induction, the $(4\ell - 5)$-approximation for the min-diameter of G_i never exceeds the true min-diameter of G_i. Then by Lemma 6.4.6, the min-diameter of G_i does not exceed the min-diameter of G. Therefore, we never over estimate the min-diameter.

If $s^* \in W$ or $t^* \in W$, then since we run Dijkstra from all vertices in W we have $D' = D$. So assuming that $s^*, t^* \notin W$, we have two cases.

Case 1: s^* and t^* are not in the same vertex set V_i. By Lemma 6.4.2, property 2, there exists $w \in W$ such that one of s^* and t^* is in S_w and the other is in T_w, so by Lemma 6.4.3, $\varepsilon(w) \geq D/2$. Since $D' \geq \varepsilon(w)$, we have $D' \geq D/2$.

Case 2: s^* and t^* are in the same vertex set V_i for some i. If $D' \geq D/(4\ell - 1)$, D' is already a good approximation. So assume $D' < D/(4\ell - 1)$. By Lemma 6.4.7, $\min\{d_{G_i}(s^*, t^*), d_{G_i}(t^*, s^*)\} \geq d_{min}(s^*, t^*) - 4D' = D - 4D'$. Since we calculate a $(4\ell - 5)$-approximation of G_i's min-diameter, our estimate is at least

$$(D - 4D')/(4\ell - 5) \geq (D - 4(D/(4\ell - 1)))/(4\ell - 5) = D/(4\ell - 1)$$

Running time analysis. It takes $\tilde{O}(mn^{1/(\ell+1)})$ time to perform the partitioning from Lemma 6.4.2 and to perform Dijkstra's algorithm from all $w \in W$ since $|W| = O(n^{1/(\ell+1)})$. For all i, the number of vertices in G_i is $|V_i| + 2 = O(n^{\ell/(\ell+1)})$ with high probability by Lemma 6.4.2, property 1, and the number of edges is $m_i + O(n^{\ell/(\ell+1)})$ where m_i is the number of edges in the graph induced by V_i. By induction, it takes $\tilde{O}\left((m_i + n^{\ell/(\ell+1)})\left(n^{\ell/(\ell+1)}\right)^{1/\ell}\right)$ time to compute a $(4\ell-5)$-approximation of the min-diameter of G_i for each i. Summing over all i gives us $\tilde{O}(mn^{1/(\ell+1)})$.

Note that we apply Lemma 6.4.2 at most $poly(n)$ times in the recursion and this the only randomization so the whole algorithm works with high probability.

Chapter 7

Dynamic Diameter

7.1 Results

The best known dynamic algorithms either just use the best known dynamic algorithms for All-Pairs Shortest Paths (APSP), or recompute the parameter estimate from scratch after each edge update. This leads to the following bounds:

1. Demetrescu and Italiano [DI04] obtained a fully dynamic exact APSP algorithm with an amortized update time of $\tilde{O}(n^2)$ and $O(1)$ query time; this is the best exact fully dynamic algorithm for Diameter. Abboud and Vassilevska W. [AV14] showed that under SETH, any $(4/3-\varepsilon)$-approximation fully dynamic algorithm for Diameter (for $\varepsilon > 0$) requires $n^{2-o(1)}$ amortized update or query time even in sparse graphs. Thus the APSP approach is conditionally optimal for fully dynamic $(4/3 - \varepsilon)$-approximate Diameter algorithms. It is unclear however whether a 4/3-approximation with better update time is possible.

2. By recomputing an approximation for Diameter after every update, one can obtain the various time vs. accuracy trade-off algorithms outlined in Chapter 2. The only related lower bounds here are (a) by Henzinger et al. [HKNS15] which showed that under the Online Matrix Vector hypothesis (OMv), any fully dynamic Diameter algorithm that achieves a $(2 - \varepsilon)$-approximation for undirected *weighted* graphs, or any finite approximation in directed graphs needs $n^{0.5-o(1)}$ amortized update time and (b) by Henzinger et al. [HLNW17] which proved under the combinatorial Boolean Matrix Multiplication conjecture any fully dynamic Diameter algorithm that achieves a $(4/3 - \varepsilon)$-approximation in undirected *unweighted* graphs with $n^{3-o(1)}$ preprocessing time requires $n^{2-o(1)}$ update or query time (and the same result for undirected *weighted* graph using the APSP conjecture). While these results give some limitation, they are far from tight.

We obtain a conditional lower bound for fully dynamic Diameter approximation. We strengthen the conditional lower bound of [AV14] by increasing the approximation ratio from $(4/3 - \varepsilon)$ to $(3/2 - \varepsilon)$.

Theorem 7.1.1. *Under SETH, every fully dynamic $(3/2-\varepsilon)$-approximation algorithm for Diameter with polynomial preprocessing time requires $n^{2-o(1)}$ amortized update or query time in the word-RAM model of computation with $O(\log n)$ bit words, even for dynamic undirected unweighted graphs that are always sparse.*

These conditional lower bounds imply that the $\tilde{O}(m^{3/2})$ time static approximation algorithm that recomputes the diameter from scratch are optimal in the sense that any improvement of the approximation factor causes the update time to grow to n^2, and Demetrescu and Italiano's algorithm achieves $\tilde{O}(n^2)$ update time even for the exact maintenance of APSP.

Our conditional lower bound for the fully dynamic setting also applies to partially dynamic algorithms that have *worst case* update and query time guarantees. This is due to the nature of our reductions: they all produce an initial graph on which we perform update stages that only insert or only delete (we can choose which) a small batch of edges, ask a query and undo the changes just made, returning to the initial graph. An incremental/decremental algorithm can be used to implement such reductions by performing the deletions/insertions by rolling back the data structure. Because of this, we have very strong worst case lower bounds, and it makes sense to focus on amortized algorithms for the partially dynamic setting.

We develop a new partially dynamic algorithm for approximate Diameter. Our algorithm is actually a very efficient reduction to partially dynamic single source shortest paths (SSSP) in unweighted directed/undirected graphs, so that any improvement over dynamic SSSP would improve our algorithm. We extend the static 3/2-approximation of Roditty and Vassilevska W. [RV13] to the partially dynamic setting with essentially no loss to the approximation factor and running time guarantees, for undirected graphs. Because the static 3/2-approximation is conditionally tight in terms of both running time and approximation factor (as shown in Chapter 3 and [RV13] respectively), this immediately implies that our dynamic algorithm is essentially tight. For directed graphs, if partially dynamic algorithms for SSSP become as efficient as those for undirected graphs, this would imply the same lossless algorithm for Diameter as we have for undirected graphs.

Let D_0 and D_f be the initial and final values of the diameter, respectively. Let $T_{inc}(n,m,k,\epsilon)$ (resp., $T_{dec}(n,m,k,\epsilon)$) be the total time of an incremental (decremental) approximate SSSP algorithm from source u that maintains an estimate $d'(u,v)$ for all v such that if $d(u,v) \leq k$ then $(1-\epsilon)d(u,v) \leq d'(u,v) \leq d(u,v)$. For directed graphs we assume that the approximate SSSP algorithm works in directed graphs, and for undirected graphs, the SSSP algorithm only needs to work in undirected graphs. Our black-box reductions can be summarized in the theorem below.

The running times of our algorithm is written in terms of n and m, which refer to an upper bound on the number of vertices and edges, respectively, over the entire sequence of updates. That is, for incremental algorithms, the running time is written in terms of the final values of n and m and for decremental algorithms the running time written is in terms of the initial values of n and m.

Theorem 7.1.2. *There is a Las Vegas randomized algorithm for incremental (resp., decremental) Diameter in unweighted, directed graphs against an oblivious (resp.,*

adaptive) adversary that given $\varepsilon > 0$, runs in total time

$$\tilde{O}\Big(\max_{D_f \le D' \le D_0} \Big\{T_{inc}(n, m, D', \varepsilon)\frac{\sqrt{n/D'}}{\varepsilon^2}\Big\}\Big)$$

$$\Big(\textit{resp.}, \tilde{O}\Big(\max_{D_0 \le D' \le D_f} \Big\{T_{dec}(n, m, D', \varepsilon)\frac{\sqrt{n/D'}}{\varepsilon^2}\Big\}\Big)\Big)$$

with high probability, and maintains an estimate $\hat{D}$ such that $\frac{2(1-\varepsilon)}{3}D - \frac{2}{3} \le \hat{D} \le D$ where D is the diameter of the current graph.

There is a long line of work on partially dynamic $(1+\epsilon)$-approximate SSSP. Concluding this line of work for undirected graphs in the decremental setting, Bernstein, Probst G., and Saranurak [BGS21] obtained a deterministic $(1+\varepsilon)$-approximate decremental algorithm for SSSP in undirected graphs with total expected update time $m^{1+o(1/\varepsilon)}$. As an immediate corollary we obtain:

Corollary 7.1.1. *There is a Las Vegas randomized algorithm for decremental Diameter in unweighted, undirected graphs against an adaptive adversary that given $\varepsilon > 0$, runs in total time $m^{1+o(1/\varepsilon)}\sqrt{n}/\varepsilon^2$ in expectation, and maintains an estimate $\frac{2(1-\varepsilon)}{3}D - \frac{2}{3} \le \hat{D} \le D$, where D is the diameter of the current graph.*

A precursor to the above SSSP algorithm of [BGS21], was an algorithm by Henzinger, Krinninger, and Nanongkai [HKN14] with the same guarantees but randomized against an oblivious adversary [1]. Although it is not published, it is likely (based on personal communication) that this algorithm extends to the incremental setting as well, in which case Corollary 7.1.1 also holds for the incremental setting but against an oblivious adversary.

For directed graphs, the best known algorithm for us to apply is the Even Shiloach Tree data structure [ES81], which deterministically maintains exact SSSP up to any distance k in total time $O(mk)$. The original result was for undirected graphs, but Henzinger and King recognized that it can be extended to directed graphs [HK95]. As an immediate corollary we obtain:

Corollary 7.1.2. *There is a Las Vegas randomized algorithm for partially dynamic Diameter in unweighted, directed graphs that given $\varepsilon > 0$, runs in total time $\tilde{O}(m\sqrt{nD_{\max}}/\varepsilon^2)$ with high probability where $D_{\max}$ is the maximum diameter throughout the algorithm, and maintains an estimate $\hat{D}$ such that $\frac{2(1-\varepsilon)}{3}D - 1 \le \hat{D} \le D$, where D is the diameter of the current graph. The incremental algorithm works against an oblivious adversary; the decremental algorithm works against an adaptive adversary.*

7.2 Techniques

To obtain our conditional lower bound we use the reduction from 3-OV to dynamic Diameter of Abboud and Vassilevska W. [AV14] as a starting point. We augment their

[1] The exact expected update time of this algorithm is $m^{1+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)}$.

construction by adding some extra vertices and edges that improve the approximation factor.

We describe our algorithmic techniques in more detail. Our partially dynamic nearly 3/2-approximation algorithm for Diameter is based on known algorithms in the static setting [RV13, CLR+14]. This static algorithms work by carefully choosing a set U of vertices, performing SSSP from every vertex in U, and showing that at least one of these SSSP instantiations yields a good estimate for the parameter of interest. The set U is chosen as follows. We pick a random sample S of $\tilde{\Theta}(\sqrt{n})$ vertices and let w^* be the vertex that is farthest from S; that is, w^* is the vertex that maximizes $\min_{s \in S} d(w^*, s)$. Then, letting $N(w, \sqrt{n})$ be the closest $\sqrt{n}$ vertices to w, we set $U = S \cup \{w^*\} \cup N(w^*, \sqrt{n})$.

Adapting these static algorithms to the dynamic setting presents two main challenges:

Firstly, given a set S of vertices, the farthest vertex w^* from S can change over time. We wish to minimize the total number of vertices that we ever run dynamic SSSP from, as reinitializing dynamic SSSP from a new vertex is expensive. Suppose we run dynamic SSSP from every vertex in $N(w^*, \sqrt{n})$ at all times. Then, every time w^* changes, we must reinitialize the dynamic SSSP data structure from $\sqrt{n}$ new vertices. If w^* changes frequently, this is prohibitively slow. To overcome this issue, we show that it suffices to choose a vertex w that *approximates* w^* (for a careful notion of approximation); and furthermore, by doing so we can limit the number of times we choose a new w.

Due to inherent differences between the incremental and decremental settings, we choose w in different ways in the different settings. In the decremental setting, distances can only increase, so our current choice of w can only become a poor approximation for w^* if $d(w^*, S)$ increases. Then, we use the fact that $d(w^*, S)$ is monotonically increasing to bound the number of times we need to choose a new w.

The incremental setting is more involved. Since distances can only decrease, our current choice of w becomes a poor approximation of w^* if $d(w, S)$ decreases. A challenge arises because unlike $d(w^*, S)$, the distance $d(w, S)$ does not change monotonically. One can imagine a scenario in which whenever we choose a new w, an edge is added causing $d(w, S)$ to immediately decrease to 1, which mandates that we choose a new w. We address this challenge by carefully employing randomness against an oblivious adversary. We argue that by randomly sampling w from a specifically chosen set of vertices, in expectation it will take a long time for w to become a poor approximation for w^*.

The second main challenge is that we wish to apply a partially dynamic SSSP algorithm as a subroutine, however the state of such algorithms is much better for undirected graphs than directed graphs. For instance, for *undirected* decremental graphs, there is a randomized $(1+\epsilon)$-approximate SSSP algorithm that runs amortized $m^{o(1)}$ time [HKN14] (and it is believed, but not published, than a similar result is possible for incremental graphs), while for incremental/decremental *directed* graphs the best known algorithms for SSSP up to distance k run in amortized time $O(k)$ [ES81]. To address this discrepancy, we carefully exploit the fact that longer paths are easier to hit by randomly sampling: we augment the algorithm with an additional

subsampling routine that quadratically decreases the dependence of the running time on the diameter D.

7.3 Preliminaries

Our algorithm is written as a reduction to a black-box incremental or decremental approximation algorithm for *truncated SSSP*; that is, SSSP which provides a distance estimate for all nodes whose distance from the source is at most a given value k. For generality, our algorithms are written for directed graphs and use directed SSSP algorithms, however if the graph is undirected one can simply run an undirected SSSP algorithm instead.

Let *out*-$\mathcal{A}_{inc}(u, k, \delta)$ (resp., *out*-$\mathcal{A}_{dec}(u, k, \delta)$) be an incremental (resp., decremental) algorithm that maintains for all v an estimate $d'(u, v)$ such that if $d(u, v) \leq k$ then $(1 - \delta)d(u, v) \leq d'(u, v) \leq d(u, v)$. Analogously, let *in*-$\mathcal{A}_{inc}(u, k, \delta)$ (resp., *in*-$\mathcal{A}_{dec}(u, k, \delta)$) be an incremental (decremental) algorithm that maintains an estimate $d'(v, u)$ for all v such that if $d(u, v) \leq k$ then $(1 - \delta)d(v, u) \leq d'(v, u) \leq d(v, u)$. We assume that after every update, these algorithms output all nodes whose distance estimate has changed. Let $T_{inc}(n, m, k, \delta)$ (resp., $T_{dec}(n, m, k, \delta)$) be the total time of out-$\mathcal{A}_{inc}(u, k, \delta)$ and in-$\mathcal{A}_{inc}(u, k, \delta)$ (resp., out-$\mathcal{A}_{inc}(u, k, \delta)$ and in-$\mathcal{A}_{inc}(u, k, \delta)$) (or the corresponding undirected algorithms, depending on the setting).

The running times of our algorithms are written as the maximum of an expression over all values of the diameter D throughout the entire sequence of updates. Although the maximum value of D in a partially dynamic graph either occurs at the beginning or end of the update sequence, the maximum value of the running time expression could occur for any value of D or R.

Suppose we run in-$\mathcal{A}_{inc}$, in-$\mathcal{A}_{dec}$, out-$\mathcal{A}_{inc}$, or out-$\mathcal{A}_{dec}$ from a vertex v. Then, let $B_{out}(v, r)$ be the set of vertices u with $d'(v, u) \leq r$.

For a subset $S \subseteq V$ of vertices and a vertex $v \in V$ we define $d(S, v) := \min_{s \in S} d(s, v)$. Similarly, $d(v, S) := \min_{s \in S} d(v, s)$. When the algorithms call for an approximation $d'(S, v)$ of $d(S, v)$, we add a dummy vertex x with an edge to every vertex in S and run out-$\mathcal{A}_{inc}$ (or out-$\mathcal{A}_{dec}$) from x; let $d'(S, v) = d'(x, v) - 1$. We define and maintain $d'(v, S)$ analogously by adding a dummy vertex with an edge from every vertex in S.

Claim 2. *For all $u \notin S$, $(1 - 2\delta)d(u, S) \leq d'(u, S) \leq d(u, S)$.*

Proof. $(1-2\delta)d(u, S) = (1-2\delta)(d(u, x)-1) = d(u, x)-\delta d(u, x)-1+\delta(2-d(u, x)) \leq d(u, x) - \delta d(u, x) - 1 = (1 - \delta)d(u, x) - 1 \leq d'(u, x) - 1 = d'(u, S)$ and $d'(u, S) = d'(u, x) - 1 \leq d(u, x) - 1 = d(u, S)$. □

7.4 Partially dynamic algorithm

In this section we will prove Theorem 7.1.2.

To prove Theorem 7.1.2, we prove the following lemma, which gives an algorithm with a similar guarantee but takes as input a parameter D' that acts as a proxy for the true diameter D.

Lemma 7.4.1. *There is a Las Vegas randomized algorithm against an oblivious adversary for incremental Diameter in unweighted, directed graphs that given $D', \varepsilon > 0$, runs in total time $\tilde{O}\left(T_{inc}(n, m, D')\frac{\sqrt{n/D'}}{\varepsilon}\right)$ with high probability, and maintains an estimate $\hat{D} \leq D$ such that if $D' \leq D$ then $\frac{2(1-\varepsilon)}{3}D' - \frac{2}{3} \leq \hat{D}$ where D is the diameter of the current graph.*

Proof of Theorem 7.1.2 given Lemma 7.4.1. Let $\varepsilon' = \varepsilon/2$. We run the algorithm from Lemma 7.4.1 with parameters D' and ε', where D' is defined as follows. Initially, we set D' so that $D' \geq D_0$ and $D' = O(D_0)$. One simple way to do this is to run BFS from an arbitrary vertex in the graph, which (by the triangle inequality) provides a 2-approximation for D_0. Throughout the sequence of updates, whenever the algorithm returns $\hat{D} < D'(\frac{2(1-\varepsilon')}{3}) - \frac{2}{3}$, we decrease D' by multiplying it by $\frac{1-\varepsilon}{1-\varepsilon'}$ and reinitialize the algorithm from Lemma 7.4.1. Otherwise, the algorithm returns $\hat{D} \geq D'(\frac{2(1-\varepsilon')}{3}) - \frac{2}{3}$ and we return $\hat{D}$ as the estimate for Theorem 7.1.2.

We claim that this rule for updating D' implies that at all times the following inequality holds:

$$D\left(\frac{1-\varepsilon}{1-\varepsilon'}\right) \leq D'. \tag{7.1}$$

We note that inequality 7.1 does not imply that D', itself, is a good estimate for D, since D' could be larger than D. However, inequality 7.1 does imply the guarantee of Theorem 7.1.2 as follows. Combining inequality 7.1 with the guarantee of Lemma 7.4.1, we have $\hat{D} \geq D'(\frac{2(1-\varepsilon')}{3}) - \frac{2}{3} \geq D(\frac{2(1-\varepsilon)}{3}) - \frac{2}{3}$. Also, Lemma 7.4.1 guarantees that $\hat{D} \leq D$. Together, these bounds prove the bound in Theorem 7.1.2.

Now, we prove inequality 7.1. Initially it holds since $D' \geq D_0$. Because the graph is incremental, D cannot increase so inequality 7.1 can only be violated due to an update of D'. Lemma 7.4.1 implies that right before D' is updated, $D' > D$. Thus, right after D' is updated, inequality 7.1 holds.

Now, we consider the running time. The total time is $\tilde{O}\left(\sum_{D'} T_{inc}(n, m, D')\frac{\sqrt{n/D'}}{\varepsilon}\right)$ with high probability. All values of D' are $O(D_0)$ and $\Omega(D_f)$. We update D' at most $\log_{(1-\varepsilon')/(1-\varepsilon)} n = \tilde{O}(1/\varepsilon)$ times. Thus, with high probability the running time is $\tilde{O}(\max_{D_f \leq D' \leq D_0}\{T_{inc}(n, m, D')\frac{\sqrt{n/D'}}{\varepsilon^2}\})$. □

Proof of Lemma 7.4.1. First we describe the algorithm and then the analysis.

Algorithm. Let $\delta = 2\varepsilon/11$. Throughout the incremental (resp., decremental) algorithm we will run in-$\mathcal{A}_{inc}$ (in-$\mathcal{A}_{dec}$) and out-$\mathcal{A}_{inc}$ (out-$\mathcal{A}_{dec}$) from carefully chosen sets of vertices. For ease of notation, we let in-$\mathcal{A}$ denote either in-$\mathcal{A}_{inc}$ or in-$\mathcal{A}_{dec}$, depending on the setting, and similarly we let out-$\mathcal{A}$ denote either out-$\mathcal{A}_{inc}$ or out-$\mathcal{A}_{dec}$.

Initialization. Let α be such that $D' = \Theta(n^{1-2\alpha})$. We randomly sample a set S of size $\Theta(n^\alpha \log^2 n)$ so that with high probability, for every vertex v, S hits $N_{out}(v, n^{1-\alpha})$. For the incremental algorithm, since the adversary is oblivious it is also true that with high probability, for every vertex v, *after every update*, S hits $N_{out}(v, n^{1-\alpha})$.

Throughout the entire execution of the algorithm, for all $s \in S$ we run in-$\mathcal{A}(s, D', \delta)$. Additionally, we maintain the approximate distance $d'(v, S)$ from every vertex v to S as described in the preliminaries. Let W be the dynamically changing set of vertices v that satisfy $d'(v, S) > D'/3$.

Phases. The algorithm runs phases. The first phase begins right after initialization. At the beginning of each phase, we choose a vertex $w \in W$ if $W \neq \emptyset$. The decremental algorithm only has one phase and we let w be an arbitrary vertex in W. Note that in the decremental setting distances can only increase so w never leaves W.

In the incremental setting on the other hand, distances can decrease so vertices can leave W. The incremental algorithm may have many phases, and at the beginning of each phase, we choose $w \in W$ uniformly at random. The beginning of a new phase is triggered when w leaves the set W.

Throughout the phase, we run out-$\mathcal{A}(w, D', \delta)$. Also, we will define a subset $S' \subseteq B_{out}(w, \frac{D'}{3})$ and for all $s' \in S'$, we run in-$\mathcal{A}(s', D', \delta)$. S' is initially empty and we independently add each vertex in $B_{out}(w, \frac{D'}{3})$ to S' with probability $\min\{1, \frac{\log^2 n}{\delta D'}\}$. In the incremental setting (but not the decremental setting), $B_{out}(w, \frac{D'}{3})$ can grow, and whenever a vertex u joins $B_{out}(w, \frac{D'}{3})$ we add u to S' with probability $\min\{1, \frac{\log^2 n}{\delta D'}\}$.

Reinitialization. If at any point during the execution of the algorithm, any of the following events occur, we reinitialize the entire algorithm:

- $|B_{out}(w, \frac{D'}{3})| > n^{1-\alpha}$, or
- $|S'| > \frac{n^{1-\alpha} \log^4 n}{\delta D'}$, or
- there is a vertex $v \in B_{out}(w, \frac{D'}{3})$ such that $d'(v, S') > \delta D'$.

We will show in the analysis that with high probability we never reinitialize the algorithm.

Query. Following each update, the return value $\hat{D}$ is the maximum distance estimate found over all instantiations of out-$\mathcal{A}$ and in-$\mathcal{A}$. That is,

$$\hat{D} = \max\Big\{ \max_{v \in V} d'(w, v), \max_{v \in V, s \in S \cup S'} d'(v, s) \Big\}.$$

To maintain this value, we maintain the following heaps. For every vertex v that we run out-$\mathcal{A}$ (resp., out-$\mathcal{A}$) from, we keep a max-heap $\mathcal{H}(v)$ that stores for each other vertex u the estimate $d'(v, u)$ (resp., $d'(u, v)$). Let $\hat{d}_{out}(v)$ be the value that $\mathcal{H}(v)$ outputs. Additionally we keep a max-heap $\mathcal{H}$ which stores each $\hat{d}_{out}(v)$.

Analysis:

Correctness: The return value $\hat{D}$ is $d'(u,v)$ for some u, v, so $\hat{D} \leq D$. It remains to show that if $D' \leq D$, then $\frac{2(1-\varepsilon)}{3}D' - \frac{2}{3} \leq \hat{D}$.

Let s^* and t^* be the true diameter endpoints.

Case 1: $s^* \notin W$**.** Let $s \in S$ be such that $d'(s^*, s) \leq \frac{D'}{3}$. Then $d(s^*, s) \leq D'(\frac{1}{3(1-\delta)})$. Then by the triangle inequality, $d(s, t^*) \geq D - D'(\frac{1}{3(1-\delta)}) \geq D'(1 - \frac{1}{3(1-\delta)})$. Thus, $\hat{D} \geq d'(s, t^*) \geq D'(1-\delta)(1 - \frac{1}{3(1-\delta)}) = D'(\frac{2}{3} - \delta)$.

Case 2: $s^* \in W$**.** We don't explicitly use the fact that $s^* \in W$; we just use the fact that w exists. If $d'(w, t^*) \geq \frac{2D'}{3}$, then we are done. So suppose otherwise; that is, suppose $d'(w, t^*) < \frac{2D'}{3}$ so $d(w, t^*) < D'(\frac{2}{3(1-\delta)})$.

Consider the shortest path from w to t^*. Let q be the vertex on this path at distance $\lfloor \frac{D'}{3} \rfloor$ from w. So $d(q, t^*) < D'(\frac{2}{3(1-\delta)}) - \lfloor \frac{D'}{3} \rfloor \leq D'(\frac{2}{3(1-\delta)} - \frac{1}{3}) + \frac{2}{3}$. We know that $q \in B_{out}(w, \frac{D'}{3})$, so $d'(S', q) \leq \delta D'$ or else we would have reinitialized the algorithm. Thus, by Claim 7.3 from the preliminaries, $d(S', q) \leq \frac{\delta D'}{1-2\delta}$. Let $q' \in S'$ be a vertex with $d(q', q) \leq \frac{\delta D'}{1-2\delta}$.

By the triangle inequality, $d(q', t^*) \leq d(q', q) + d(q, t^*) \leq D'(\frac{\delta}{1-2\delta} + \frac{2}{3(1-\delta)} - \frac{1}{3}) + \frac{2}{3}$. By the triangle inequality, $d(s^*, q') \geq D - D'(\frac{\delta}{1-2\delta} + \frac{2}{3(1-\delta)} - \frac{1}{3}) - \frac{2}{3} \geq D'(\frac{4}{3} - \frac{\delta}{1-2\delta} - \frac{2}{3(1-\delta)}) - \frac{2}{3}$. Thus, $\hat{D} \geq d'(s^*, q') \geq D'(1-\delta)(\frac{4}{3} - \frac{\delta}{1-2\delta} - \frac{2}{3(1-\delta)}) - \frac{2}{3}) = D'(\frac{4(1-\delta)}{3} - \frac{\delta(1-\delta)}{1-2\delta} - \frac{2}{3}) - \frac{2}{3}$.

Setting $\delta = 2\varepsilon/11$ completes the proof of correctness.

Running time:

Reinitialization. We will argue that with high probability, we never reinitialize the algorithm. One event that triggers algorithm reinitialization is if $|B_{out}(w, \frac{D'}{3})| > n^{1-\alpha}$. Recall that with high probability, for every vertex v, S hits $N_{out}(v, n^{1-\alpha})$; this is true for the decremental algorithm only initially, and for the incremental after every update. By the definition of W, $D'/3 < d'(w, S) \leq d(w, S)$. Thus, $B_{out}(w, \frac{D'}{3})$ contains no vertices in S. So, with high probability $B_{out}(w, \frac{D'}{3})$ does not contain $N_{out}(v, n^{1-\alpha})$. That is, with high probability, $|B_{out}(w, \frac{D'}{3})| < n^{1-\alpha}$. For the decremental algorithm we have shown that this inequality holds only for the initial graph, however it also holds after every update since $|B_{out}(w, \frac{D'}{3})|$ can only decrease over time. Thus, for both the incremental and decremental algorithms, with high probability we never reinitialize the algorithm due to $|B_{out}(w, \frac{D'}{3})| > n^{1-\alpha}$.

Another event that triggers reinitialization is if $|S'| > \frac{n^{1-\alpha}\log^2 n}{\delta D'}$. $|S'|$ is a random variable drawn from a binomial distribution with $p = \frac{\log^2 n}{\delta D'}$ and expected value $\frac{|B_{out}(w, \frac{D'}{3})|\log^2 n}{\delta D'}$. We know that $|B_{out}(w, \frac{D'}{3})| \leq n^{1-\alpha}$ or else we would have reinitialized the algorithm due to the above event. Thus, $|S'| \leq \frac{n^{1-\alpha}\log^4 n}{\delta D'}$ with high probability.

The last event that triggers reinitialization is if there is a vertex $v \in B_{out}(w, \frac{D'}{3})$

such that $d'(v, S') > \delta D'$. The expected size of S' is $\frac{|B_{out}(w, \frac{D'}{3})| \log^2 n}{\delta D'}$. Thus, with high probability, for all vertices $v \in B_{out}(w, \frac{D'}{3})$, after every update, S' hits a vertex of distance at most $\delta D'$ to v.

We have shown that each of the three events that trigger reinitialization do not occur with high probability (probability at least $1 - 1/n^c$ for all constants c). Thus, with high probability we never reinitialize the algorithm.

Running in-$\mathcal{A}$ and out-$\mathcal{A}$. We will calculate the total number a of vertices that we ever run in-$\mathcal{A}$ or out-$\mathcal{A}$ from. Then the total time is $\tilde{O}(aT_{inc}(n, m, D', \varepsilon))$; maintaining the heaps $\mathcal{H}(v)$ and $\mathcal{H}$ increases the running time only by a factor of $O(\log n)$.

In the initialization step, we initialize in-$\mathcal{A}$ from all $\tilde{O}(n^\alpha)$ vertices in S as well as a dummy vertex. Throughout each phase, we run out-$\mathcal{A}$ from w and we run in-$\mathcal{A}$ from all $\tilde{O}(\frac{n^{1-\alpha}}{\varepsilon D'})$ vertices that are added to S' during the phase, as well as a dummy vertex.

The decremental algorithm has only one phase. We calculate the number of phases in the incremental algorithm. The beginning of a new phase is triggered when w leaves the set W. We note that since we are in the incremental setting, no vertices are added to W during a phase. The sequence of updates dictates if and when each vertex is removed from W. Each update may trigger any number of vertices to leave W.

Fix a choice of w and let W_0 be the set W at the point in time that the algorithm chooses w. We say that w is a *success* if at most half of the vertices in W_0 leave W after w leaves W. Since w is chosen randomly and the adversary is oblivious, the probability that w is a success is at least $1/2$. Once $\log_2 n$ choices of w are successful, then W is empty. Let Y be a random variable defined as the number of times the algorithm chooses a new w until W is empty. Y is a negative binomial random variable, which implies the following concentration bound for any constant c: $P[Y > 2c \log_2 n] \leq \exp(\frac{-c(1-1/c)^2}{2} \log_2 n)$. Thus, $Y = \tilde{O}(1)$ with high probability.

Putting everything together, with high probability, $a = \tilde{O}(n^\alpha + \frac{n^{1-\alpha}}{\varepsilon D'}) = \tilde{O}\left(\sqrt{n/D'}\right)$, so the total time is $\tilde{O}\left(T_{inc}(n, m, D', \varepsilon)\frac{\sqrt{n/D'}}{\varepsilon}\right)$. □

7.5 Fully dynamic conditional lower bound

In this section we will prove the following theorem, which implies Theorem 7.1.1.

Theorem 7.5.1. *Let t, ε, and ε' be positive constants. SETH implies that there exists no fully dynamic algorithm for $(3/2 - \varepsilon)$-approximate Diameter on undirected, unweighted graphs with n vertices and $\tilde{O}(n)$ edges, which has preprocessing time $p(n) = O(n^t)$, amortized update time $u(n) = O(n^{2-\varepsilon'})$, and amortized query time $q(n) = O(n^{2-\varepsilon'})$.*

The same result holds for the incremental and decremental settings but for worst-case update and query times.

Proof of Theorem 7.5.1. Suppose for contradiction that a dynamic $(3/2-\varepsilon)$-approximation algorithm exists with preprocessing time n^t, amortized update time $n^{2-\varepsilon'}$ and query time $n^{2-\varepsilon'}$, for positive numbers $\varepsilon, \varepsilon'$, and t. We define $\delta = \frac{1-\varepsilon'}{t}$.

Construction: The construction is shown in Figure 7-1 and described as follows.

Initialization. Let $a = \lceil \frac{1-2\varepsilon}{8\varepsilon} \rceil + 1$.

We begin with an instance of 3-OV with vector sets U, V, and W such that $|U| = |V| = N^\delta$ and $|W| = N^{1-2\delta}$. We first discard degenerate vectors and coordinates: any coordinates that are 0 for every vector in U, every vector in V, or every vector in W, and any vectors that are all zeroes. Note that removing degenerate coordinates does not change the correct output value. If there is a degenerate vector in a 3-OV instance, the correct output value is always "yes".

For each coordinate c, create two nodes, and denote one by c_U and the other by c_V. We denote the two sets of coordinate nodes by C_U and C_V respectively. Next, create a path of length a for each vector $u \in U$; denote the start by u^0 and the end by u^a, and each node at distance i from u^0 by u^i. We make a similar path for each $v \in V$. Then, we encode each vector $u \in U$ in the graph by adding a path of length a between u^a and c_U if $u[c] = 1$, and encode each $v \in V$ by adding a path of length a between c_V and v^0 if $v[c] = 1$. Finally, we add two nodes x and y. For each node u^a, add a path of length a between x and u^a, and for each node v^0, add a path of length a between y and v^0.

Stages. We proceed in $N^{2-\delta}$ stages, one for each element $w \in W$. For the current w, for each coordinate c where $w[c] = 1$, we add an edge between c_U and c_V. We then query the diameter of G. We will show that if there is an orthogonal triple that includes w, then the diameter is least $6a+1$, and otherwise, the diameter is at most $4a+1$. A $(3/2-\varepsilon)$-approximation algorithm for Diameter distinguishes between these two cases and can thus detect an orthogonal triple that includes w if one exists. If such an orthogonal triple is not detected, we undo the edge additions for the stage and continue to the next w.

Analysis:

Approximation factor. If for the current stage, for all $u \in U$, $v \in V$, $u \cdot v \cdot w \neq 0$, then for each u, v there exists a coordinate c such that $u[c] = v[c] = w[c] = 1$. Thus, for all u and v, $d(u^a, v^0) = 2a+1$. Also, for all $u, u' \in U$, $d(u^a, u'^a) \leq 2a$. We note that every vertex in the graph is of distance at most a from *some* vertex u^a or v^0. Thus, the diameter is at most $4a+1$.

Suppose for the current stage there exist $u \in U$, $v \in V$ such that $u \cdot v \cdot w = 0$. Fix u and v. We claim that $d(u^0, v^a) \geq 6a+1$. The only paths between u^0 and v^a go through u^a and v^0. There does not exist a coordinate such that $u[c] = v[c] = w[c] = 1$, so every path between u^a and v^0 must visit a vertex u'^a or v'^0 (for $u \in U$, $u' \neq u$, $v' \in V$,

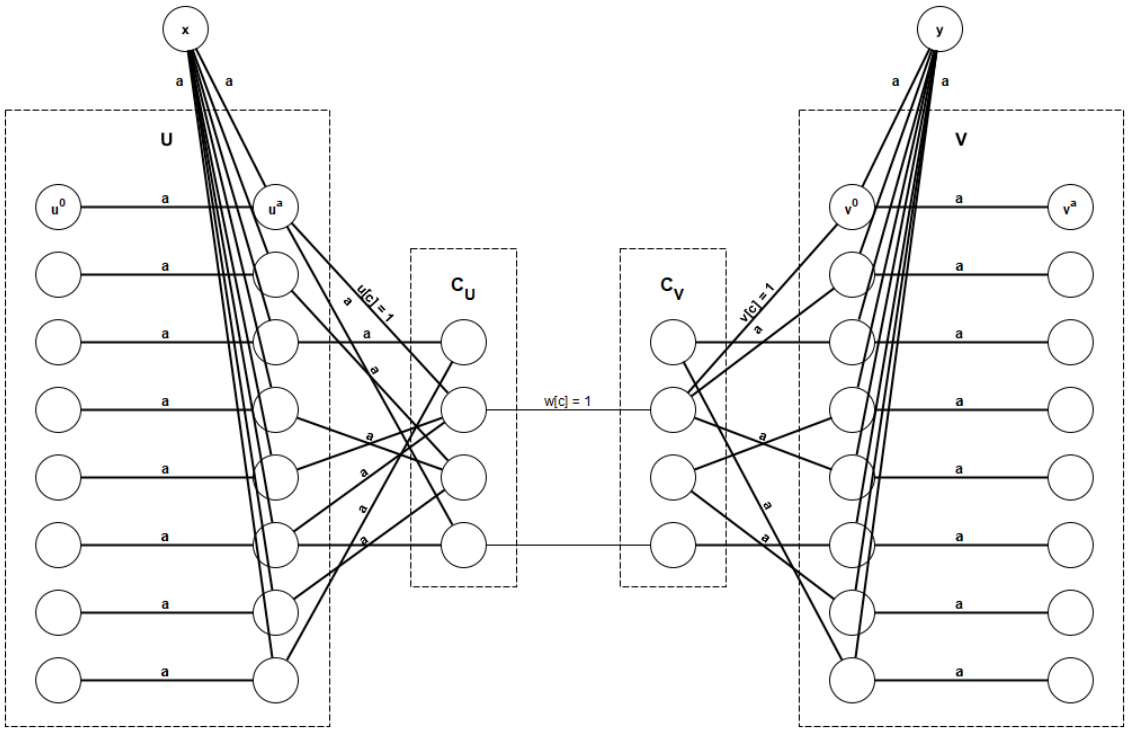

Figure 7-1: Theorem 7.5.1 Construction. Bold edges represent paths, whose labels denote their length.

$v' \neq v$), which are of distance $2a$ from u^a and v^0, respectively. Thus, $d(u^a, v^0) \geq 4a+1$ so $d(u^0, v^a) \geq 6a + 1$.

Running time. The preprocessing time of the algorithm for a graph of size N^δ is $(N^\delta)^t = N^{1-\varepsilon'}$. Each update or query takes amortized $O((N^\delta)^{2-\varepsilon'})$ time, so after $N^{1-2\delta}$ stages wherein we make $\tilde{O}(1)$ updates and queries, the total amortized time is $\tilde{O}(N^{1-\delta\varepsilon'})$. This gives an algorithm for 3-OV in $\tilde{O}(N^{1-\varepsilon'} + N^{1-\delta\varepsilon'}) = O((|U||V||W|)^{1-\varepsilon''})$ for a positive constant ε'', refuting SETH. □

Chapter 8

Open Problems

The following are 8 open problems related to the content of this dissertation.

Open Problem 1. (Exact Diameter in dense graphs) The most notorious open problem about Diameter is whether there exists an algorithm that computes the diameter of an n-vertex weighted graph exactly that runs in time $O(n^{3-\epsilon})$ for some constant $\epsilon > 0$, or evidence to suggest that such an algorithm does not exist. (Note that for unweighted graphs, $\tilde{O}(n^\omega)$ time algorithms exist, where ω is the matrix multiplication exponent.)

Open Problem 2. (Lower bounds for other problems) To the best of our knowledge, Diameter and its variants are the only problems in fine-grained complexity for which there are known time vs. accuracy trade-off conditional lower bounds. Do such trade-off lower bounds exist for other problems?

Open Problem 3. (Closing the gap for approximate Diameter) As shown in Figure 2-1, there is a gap between algorithms and conditional lower bounds for some parts of the time vs. accuracy trade-off for approximate Diameter. From the lower bounds side, for any fixed integer $k' \geq 2$, SETH implies that any $(\frac{2k'-1}{k'} - \varepsilon)$-approximation algorithm requires $m^{\frac{k'}{k'-1}-o(1)}$ time, for undirected unweighted graphs (Chapter 3 and [DW21, Li21, DLV21]). From the algorithms side, there is an $\tilde{O}(mn^{1/(k+1)})$ time algorithm that achieves an almost-$(2 - 1/2^k)$-approximation for any fixed constant $k \geq 0$ for undirected weighted graphs [CGR16]. This algorithm is tight with the above conditional lower bound for $k = 0, 1$ and $k \to \infty$ (and also works for directed graphs for these values of k [RV13, CLR$^+$14]).

What about intermediate values of k? Can the algorithms for intermediate values of k be improved to match the conditional lower bounds? Can we get *any* algorithms for directed graphs for intermediate values of k?

There is some evidence from [Li21] under nondeterministic versions of SETH, that we should expect the improvement to be on the algorithms side rather than the lower bounds side (although this result only holds for deterministic reductions).

Open Problem 4. (Radius) We proved a hierarchy of conditional lower bounds for Diameter that trade off time and accuracy Chapters 2 and 3. Such a hierarchy

does not exist, however, for the very related problem of *Radius*. Many techniques for Diameter extend to Radius and vice versa, so we might expect that this hierarchy of conditional lower bounds extends from Diameter to Radius, but it is unknown whether or not it does.

There is, however, one known conditional lower bound for Radius [AVW16]. Analogous to the reduction from OV to Diameter, there is a reduction from a problem called *Hitting Set* to Radius. The Hitting Set problem is similar to OV but it has different quantifiers. To prove the hierarchy of conditional lower bounds for Diameter, we reduce from k-OV to Diameter, which raises the question of whether there is an appropriate definition of "k-Hitting Set" that reduces to Radius.

Open Problem 5. (Roundtrip Diameter) Although we have made progress on a number of variants of Diameter in this dissertation, there is one curious variant, *Roundtrip Diameter*, that we did not make progress on. Given a directed graphs, the *roundtrip distance* between a pair of vertices u and v is defined as $d(u, v) + d(v, u)$. The roundtrip diameter is the largest roundtrip distance in the graph.

The only known approximation algorithm for Roundtrip Diameter is a simple 2-approximation algorithm in near-linear time, by simply running an SSSP algorithm from an arbitrary vertex and returning the largest roundtrip distance found. From the conditional lower bounds side, any lower bound for Diameter immediately applies to Roundtrip Diameter, but we do not know any better lower bounds.

Can we improve the bounds for Roundtrip Diameter, either from the upper or lower bounds side? Specifically, is there a subqudratic time algorithm that achieves an approximation factor better than 2, or a conditional lower bound to rule out such an algorithm?

More information about this problem is provided in Open Problem 2.4 of [RV19].

Open Problem 6. (Directed Bichromatic Diameter) The running time of our algorithm for Bichromatic Diameter in directed graphs from Theorem 5.4.6 is not tight. We give a 2-approximation in $\tilde{O}(m^{3/2})$ time and a conditional lower bound that any better than 2-approximation requires quadratic time under SETH. Is there a faster 2-approximation algorithm or a conditional lower bound ruling out such an algorithm?

Open Problem 7. (Min-Diameter) Our algorithms for Min-Diameter from Theorem 6.1.1 are not tight with the known conditional lower bounds. The best approximation factor that our algorithms achieve is 3 (in $\tilde{O}(m\sqrt{n})$ time), whereas there is a conditional lower bound that one cannot achieve a truly subquadratic time algorithm with an approximation factor better than 2 for weighted graphs, or better than 5/3 for unweighted graphs [AVW16]. Can we get a better algorithm or conditional lower bound?

Open Problem 8. (Directed partially dynamic approximate SSSP) Is there a faster algorithm for $(1 + \epsilon)$-approximate partially dynamic SSSP in directed (unweighted) sparse graphs? The best known decremental algorithm runs in time $\tilde{O}(mn^{2/3})$ in expectation (against an oblivious adversary) [BGWN20] and the best known incremental algorithm runs in time $\tilde{O}(mn^{1/2} + m^{7/5})$ in expectation (against an adaptive

adversary) [CZ21]. This is an important problem in the area of dynamic shortest paths, and improving these bounds would also immediately imply a better algorithm for partially dynamic Diameter in directed graphs via Theorem 7.1.2.

Bibliography

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

[AHR+19] Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[AJB99] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[AV14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.

[AVW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

[AWY15] Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230, 2015.

[BCH+15] Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A Kosters, Andrea Marino, and Frank W Takes. Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theoretical Computer Science*, 586:59–80, 2015.

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. *arXiv preprint arXiv:2101.07149*, 2021.

[BGWN20] Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. *arXiv preprint arXiv:2004.04496*, 2020.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 51–58, 2015.

[Bon21a] Édouard Bonnet. 4 vs 7 sparse undirected unweighted diameter is SETH-hard at time $n^{4/3}$. In *Proceedings of the 48rd International Colloquium on Automata, Languages, and Programming, ICALP 2021*, 2021.

[Bon21b] Édouard Bonnet. Inapproximability of diameter in super-linear time: Beyond the 5/3 ratio. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[BRS+18] Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 267–280, 2018.

[CGLM12] Pierluigi Crescenzi, Roberto Grossi, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world directed (weighted) graphs. In Ralf Klasing, editor, *Experimental Algorithms: 11th International Symposium, SEA 2012, Bordeaux, France, June 7-9, 2012. Proceedings*, pages 99–110, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CGR16] Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

[CGS15] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28:1–28:30, September 2015.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

[CLR+14] Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

[CM09] Monojit Choudhury and Animesh Mukherjee. The structure and dynamics of linguistic networks. In *Dynamics on and of Complex Networks*, pages 145–166. Springer, 2009.

[CPPU16] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. A practical parallel algorithm for diameter approximation of massive weighted graphs. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016*, pages 12–21. IEEE, 2016.

[CPPU20] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. Distributed graph diameter approximation. *Algorithms*, 13(9):216, 2020.

[CW16] Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255, 2016.

[CZ21] Shiri Chechik and Tianyi Zhang. Incremental single source shortest paths in sparse digraphs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2463–2477. SIAM, 2021.

[DI04] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. Announced at STOC'03.

[DLV21] M. Dalirrooyfard, R. Li, and V. Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. *arXiv preprint arXiv:2106.06026*, 2021.

[DVV+19] Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, Nicole Wein, Yinzhan Xu, and Yuancheng Yu. Approximation algorithms for min-distance problems. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[DVVW19] Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, and Nicole Wein. Tight approximation algorithms for bichromatic graph diameter and related problems. In *Proceedings of the 46th International*

Colloquium on Automata, Languages, and Programming, ICALP 2019. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[DW21] Mina Dalirrooyfard and Nicole Wein. Tight conditional lower bounds for approximating diameter in directed graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[EN+20] Michael Elkin, Ofer Neiman, et al. Near-additive spanners and near-exact hopsets, a unified view. *Bulletin of EATCS*, 1(130), 2020.

[ES81] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, 1981.

[GO95] Anka Gajentaan and Mark H Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[HHZ21] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 356–369, 2021.

[Hir98] Edward A Hirsch. Two new upper bounds for SAT. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, SODA 1998*, pages 521–530, 1998.

[HK95] Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 664–672. IEEE, 1995.

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Proceedings of the IEEE 55th Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 146–155. IEEE, 2014.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Symposium on Theory of Computing, STOC 2015*, pages 21–30, 2015.

[HLNW17] Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional Hardness for Sensitivity Problems. In Christos H. Papadimitriou, editor, *Proceedings of the 8th Innovations in Theoretical Computer Science Conference, ITCS 2017*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:31, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[KMX+04] Abhishek Kumar, Shashidhar Merugu, Jun Xu, Ellen W Zegura, and Xingxing Yu. Ulysses: a robust, low-diameter, low-latency peer-to-peer network. *European transactions on telecommunications*, 15(6):571–587, 2004.

[Li21] Ray Li. Settling SETH vs. approximate sparse directed unweighted diameter (up to (NU)NSETH). In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[LWCW16] T. C. Lin, M. J. Wu, W. J. Chen, and B. Y. Wu. Computing the diameters of huge social networks. In *Proceedings of the 2016 International Computer Symposium, ICS 2016*, pages 6–11, 2016.

[MC11] Paulo Morgado and Nuno Costa. Graph-based model to transport networks analysis through gis. In *Proceedings of European Colloquium on Quantitative and Theoretical Geography*, pages 2–5, 2011.

[MLH09] Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics (JEA)*, 13:1–10, 2009.

[Pet04] S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

[PPSZ05] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.

[PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, 34(6):1398–1431, 2005.

[RV13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 515–524, New York, NY, USA, 2013. ACM.

[RV19] Aviad Rubinstein and Virginia Vassilevska Williams. SETH vs approximation. *ACM SIGACT News*, 50(4):57–76, 2019.

[Sch99] T Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 410–414. IEEE, 1999.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

[TK11] Frank W Takes and Walter A Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM international conference on Information and Knowledge Management*, pages 1191–1196, 2011.

[VSPB19] Alexander Veremyev, Alexander Semenov, Eduardo L Pasiliao, and Vladimir Boginski. Graph-based exploration and clustering analysis of semantic spaces. *Applied Network Science*, 4(1):1–26, 2019.

[Wil05] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2–3):357–365, 2005.

[Wil18] R Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. Announced at FOCS'98.