Finding Patterns, Short Cycles and Long Shortest Paths in Graphs

by

Mina Dalirrooyfard

B.S., Sharif University (2017)
M.S., Massachusetts Institute of Technology (2019)

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

May 2022

Author . . . . . .
Department of Electrical Engineering and Computer Science
April 20, 2022

Certified by . . . . . .
Virginia Vassilevska Williams
Steven and Renee Finn Career Development Associate Professor of
Electrical Engineering and Computer Science
Thesis Supervisor

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

Finding Patterns, Short Cycles and Long Shortest Paths in Graphs

by

Mina Dalirrooyfard

Submitted to the Department of Electrical Engineering and Computer Science
on April 20, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

This thesis is about finding useful structures in a graph using fast algorithms, or showing that no such fast algorithms exist using popular fine-grained hypotheses from the field of *Fine-Grained Complexity*. These structures can be any small fixed-sized pattern, or more specific bigger structures such as the longest shortest path in a graph, the size of which is represented by the *diameter* of a graph. Finding these structures has many applications, from protein-protein interactions in biology to anomaly detection in networks.

We start by the problem of finding fixed-sized patterns in graphs as a subgraph, known as *Graph Pattern Detection* or *Subgraph Isomorphism*. There are no fast algorithms for graph pattern detection for many patterns despite many efforts, and so we focus on finding the source of hardness of detecting different patterns. One of our results is that one such source is the appearance of cliques (complete graphs) in the pattern which can make the pattern hard to detect.

We then move to patterns that are not necessarily fixed sized but are either paths or cycles. The size of these patterns are often represented by popular parameters such as the diameter, *radius* and *girth* in the graph.

We focus on computing the diameter (longest shortest path) of a graph and more specifically on approximating the diameter since computing it exactly is known to be hard. There is a folklore 2-approximation algorithm for the diameter that works in linear time, and we show that this algorithm is optimal conditioned on the Strong Exponential Time Hypothesis (SETH). Our result shows that any better than 2-approximation algorithm for the diameter requires super linear time. Moreover, we give a series of time-accuracy trade-off lower bounds, completing a line of recent works.

The next pattern we discuss is a cycle, and more specifically it is the shortest cycle of a graph, the length of which is known to be the girth. We give the first 2-approximation algorithm for computing the girth in directed graphs in subquadratic time, improving the previous best approximation factor (in subquadratic time) which was 3.

Finally, we don't resort to the standard measures of these distance problems, as in many applications we need more specific notions of these problems. For example we might be only interested in the longest shortest path among specific pairs of vertices (a variant of the diameter). Hence we consider two variants: First we assume that we are given two subsets S and T of the vertex set of the graph, and we are asked to compute distance parameters such as diameter and radius by only

considering the pairs of nodes in $S \times T$. These problems are called ST-distance problems and when S and T are non-overlapping and cover all the vertex set, they are called *bichromatic* distance problems. We give a comprehensive study of approximation of ST and bichromatic distance parameters.

Second, we consider a "symmetric" distance measure in directed graphs called min-distance. We give big improvements in approximating min-diameter and min-radius in general graphs and in directed acyclic graphs.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Steven and Renee Finn Career Development Associate Professor of
Electrical Engineering and Computer Science

Acknowledgments

This thesis was made possible by the help and support of many people. First I want to thank my wonderful advisor, Virginia Vassilevska Williams, who supported me in every stage of my PhD. She introduced many exciting problems to me from the start and encouraged me to explore different fields and choose what I enjoy. I never felt pressured while working with her, and as a result I had a very smooth 4.5 years in my Ph.D. I had my first experience of working on theory problems in a group through Virginia's open problem session in my first year. This experience made research more fun for me and gave me a boost of confidence from the first papers I published through working in these sessions. Virginia was not just an advisor for me, she made sure that all her students are doing okay even outside of the academic life, and offered her genuine support.

I want to thank my thesis committee, Piotr Indyk and Ronitt Rubinfeld. They were great mentors throughout my Ph.D and they offered me endless support in my academic life, in my job search and in resolving the problems Covid made for me in the last two years of my studies. I also want to thank the bigger family of the theory group at MIT that made many collaborations and friendships possible.

I want to thank all my collaborators who contributed to this thesis and to my work in my Ph.D. I want to thank Andrea Lincoln, Thuy Duong Vuong, Nicole Wein, Nikhil Vyas, Yinzhan Xu, Yuancheng Yu, Bertie Ancona, Keren Censor-Hillel, Yuval Efron, Ray Li, Amir Abboud, Jenny Kaufmann, Surya Mathialagan and Alina Harbuzova.

I can't stress how wonderful my time was at MIT, and I want to thank MIT Rowing Club for introducing rowing to me and giving me a great support group of students from every department.

Getting into this Ph.D. program wouldn't have been possible without the support of my mentor Saeed Akbari during my undergraduate studies, where I had my first exposure to academic research.

Finally I want to thank my parents Zohre and Rasoul, My sister Mahsa, and my partner Dominic, for always supporting me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about graphs and extracting *information* from them using fast algorithms. The type of information we focus on is the existence of specific structures such as fixed-sized patterns, short cycles or long shortest paths. Some of these specific structures are represented by well-known parameters in a graph, such as the *girth* (the shortest cycle) or the *diameter* (the longest shortest path). As real world graphs get bigger every day, finding or detecting these structures (or computing these parameters) in a time close to the time needed to read the input is of large importance. However this doesn't seem possible for all the structures. There are *conditional lower bounds* for the running time of detecting some of these structures that suggest we should resort to *approximation* if we want fast algorithms.

In this thesis we obtain many algorithms and conditional lower bounds for detecting different important structures in a graphs. This in fact is what the field of *fine-grained complexity* (FGC) does: FGC seeks to find the exact complexity of polynomially solvable problems by designing fast algorithms and proving lower bounds conditioned on popular fine-grained hypotheses.

The remaining of this chapter is a high level introduction to the problems discussed in this thesis, and a summary of the results. In chapter 2 we define the most well-known FGC problems and hypotheses that our lower bound results are based on. Each of the subsequent chapters contain the results for a specific problem and can be read independently from the other chapters. Moreover, at the end of each chapter we mention the most recent improvements on the results of that chapter

that are made after its publication.

Finding Patterns The first type of information that we seek in a graph is the presence or absence of small fixed sized patterns. *Graph Pattern Detection* or *Subgraph Isomorphism* (SI) asks, given two graphs G and H, does G contain a subgraph isomorphic to H? While the general problem is NP-complete, many applications, such as biology, only need algorithms for the special case in which H is a small pattern, of constant size k, while the host graph G is large. For example, protein-protein interaction networks are modeled using graphs, and the appearance of some small sized patterns can reveal important characteristics about them [BGP+13]. Moreover, graph pattern detection can be used for database joins, anomaly detection in networks and in social networks where one looks for particular structures in communities.

There are two versions of SI: *induced* and not necessarily induced, *non-induced* for short. In the induced version, the copy of H in G must have both edges and non-edges preserved, whereas in the non-induced version only edges need to carry over, and the copy of H in G can be an arbitrary supergraph of H. It is well known that the induced version of H-pattern detection for any H of constant size is at least as hard as the non-induced version, and that often the non-induced version of SI has faster algorithms (e.g. the non-induced k-independent set problem is solvable in constant time). In both versions when k is a constant SI is easily in polynomial time: if G has n vertices, the brute-force algorithm solves the problem in $O(n^k)$ time, for any H.

It is well-known that the SI problem (induced and non-induced) for any k-node pattern H in n-node graphs for constant k, can be reduced to detecting a k-clique (complete graph of size k) in an $O(n)$ node graph. Thus the hardest pattern to detect is k-clique. Given this, a natural question is the following:

If a pattern contains a t-clique, is it at least as hard to detect as a t-clique?

And in general,

What makes some patterns harder to detect than others?

Finding short cycles and long shortest paths The other structures we look for in a graph are short cycles and long shortest paths. Finding these structures can be represented by distance parameters. The most important parameters are *diameter, radius, eccentricities* and *girth*. These

parameters have many applications. For instance, the diameter measures how fast information spreads in networks, which is central for paradigms such as distributed computing and sublinear algorithms.

The eccentricity of a node is the farthest shortest path distance between this node and any other node in the graph. The diameter of a graph is the largest shortest path distance between any two nodes of the graph. The *center* of a graph is the node with minimum eccentricity and the radius of a graph is the eccentricity of the center. Finally the girth of a graph is the length of the shortest cycle in the graph.

Computing these parameters exactly is too expensive, almost quadratic in the size of the graph, as it mostly requires computing the shortest path distance between *all* pairs of nodes. This is troublesome as in most applications the graphs are very big. Moreover, in most applications, having an *estimate* of these parameters is good enough. Hence many studies resort to *approximations* of these parameters. An α approximation algorithm of a parameter D, where $\alpha \geq 1$, outputs a number D' where $D \leq D' \leq \alpha D$. The goal is to obtain fast approximation algorithms for computing any of these parameters, and to prove that they are tight using popular fine-grained complexity conjectures.

Fine-grained complexity conjectures We are going to use some of the popular fine-grained complexity conjectures that we are going to define in Chapter 2, such as *Strong Exponential Time Hypothesis (SETH)* and *k-Orthogonal Vectors (k-OV)*.

1.1 Summary of Results

Here we present a high level summary of our contributions to the above problems.

1.1.1 Finding Fixed Sized Patterns

In Chapter 3 we study the algorithms and lower bounds for the graph pattern detection problem. We give “fine-grained" lower bounds for this problem, and show that for each k there is an "easy" pattern, where easy means that this pattern can be detected faster than k-clique detection time. More particularly, in [DVW21] Thuy Duong Vuong, Virginia Vassilevska Williams and I proved

the following results.

We prove that if a pattern H contains a k-clique subgraph, then detecting whether an n node host graph contains a *not necessarily induced* copy of H requires at least the time for detecting whether an n node graph contains a k-clique. The previous result of this nature required that H contains a k-clique which is disjoint from all other k-cliques of H.

We show that if the famous Hadwiger conjecture from graph theory is true, then detecting whether an n node host graph contains a *not necessarily induced* copy of a pattern with chromatic number t requires at least the time for detecting whether an n node graph contains a t-clique. This implies that: (1) under Hadwiger's conjecture for *every* k-node pattern H, finding an *induced* copy of H requires at least the time of $\sqrt{k}$-clique detection and size $\omega(n^{\sqrt{k}/4})$ for any constant depth circuit, and (2) unconditionally, detecting an *induced* copy of a random $G(k,p)$ pattern w.h.p. requires at least the time of $\Theta(k/\log k)$-clique detection, and hence also at least size $n^{\Omega(k/\log k)}$ for circuits of constant depth.

We show that for every k, there exists a k-node pattern that contains a $k-1$-clique and that can be detected as an *induced* subgraph in n node graphs in the best known running time for $k-1$-Clique detection. Previously such a result was only known for infinitely many k.

1.1.2 Finding Long Shortest Paths: The Diameter

The diameter is the length of the longest shortest path in the graph. Computing the diameter of the graph exactly is hard, it requires computing all pairwise distances of the graph, and it is shown that there is no substantially faster algorithm for it [RV13]. More precisely, computing the diameter in sparse graphs requires $m^{2-o(1)}$ time where m is the number of edges. On the other hand, there is a simple 2-approximation algorithm for computing the diameter that works in linear time. One question is what happens if we want an α approximation algorithm where $1 < \alpha < 2$? More importantly, is the 2-approximation algorithm optimal? Can we obtain a better approximation factor in the same running time? Figure 1-1 is a summary of the results about diameter, including the results that this thesis contributes to. We discuss these results in detail in Chapter 4. In the first glimpse, Figure 1-1 represents several algorithms (the dots) and lower bounds (horizental lines),

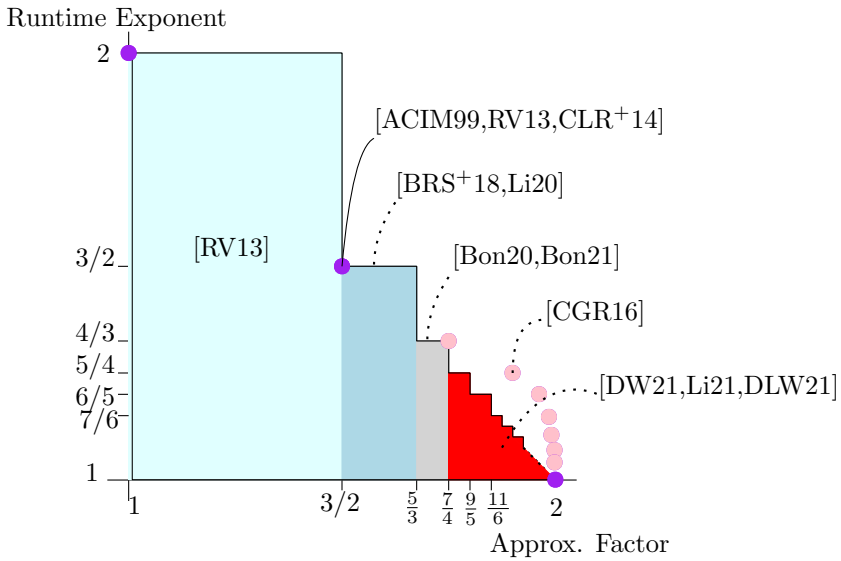

Figure 1-1: Hardness results for Diameter (in undirected graphs). The x-axis is the approximation factor and the y-axis is the runtime exponent, and the underlying graph is sparse, i.e. the number of edges m is $n \cdot poly(\log n)$ where n is the number of nodes. Black lines represent lower bounds. Purple dots represent algorithms [BRS$^+$18, RV13, CLR$^+$14], and pink dots represent algorithms that also lose an additive factor [CGR16]. The purple dot algorithms work for directed graphs as well, and they are the only algorithms for the directed diameter. In [BRS$^+$18], the labeled lower bound was proved for weighted graphs, and in unweighted graphs they proved a weaker lower bound which says a $1.6-\varepsilon$ approximation needs $m^{3/2-o(1)}$ time, and later [Li20] showed the stronger lower bound for unweighted graphs. The red region is due to our papers [DW21, DLW21]. Li [Li21] showed the same result as [DW21] for directed graphs in an independent and concurrent work.

which for the most part do not match. Our work, the red region, helps in making this gap smaller.

In [DW21] Nicole Wein and I show that for any integer $k \geq 2$, any better than $\frac{2k-1}{k}$ approximation algorithm for the diameter in *directed* graphs requires $n^{1+1/k-o(1)}$ time under SETH. Before our work, these lower bounds were known only for $k \leq 4$. In [DLW21], Ray Li, Virginia Vassilevska Williams and I prove the same result for *undirected* graphs. One main consequence of these results is that the simple 2-approximation algorithm is tight: If one wants an algorithm with an approximation factor smaller than 2, this algorithm would require super linear time, under SETH.

1.1.3 Finding Short Cycles: The Girth

The girth is the length of the shortest cycle in the graph. Similar to the diameter, the exact computation of the girth can be done by computing the shortest path distance between all pairs of nodes, i.e. solving *all pairs shortest paths (APSP)* problem in $\tilde{O}(mn)$ time[1], and one cannot do better up to $n^{o(1)}$ factors, both for sparse and dense weighted graphs, under popular hardness hypotheses from fine-grained complexity [VW10, LVW18].

Given this, one question is the following:
what is the lowest approximation factor an algorithm can have if it runs in truly subquadratic time, say $O(mn^{\epsilon})$ for $\epsilon < 1$?

To answer this question, we focus on directed graphs. Chechik et al [CLRS20] give a 3-approximation algorithm for computing the girth in $\tilde{O}(m\sqrt{n})$ time in directed graphs, and this was the best approximation factor obtained for algorithms with running time $\tilde{O}(mn^{\epsilon})$ for some positive $\epsilon < 1$. In [DVW20] Virginia Vassilevska Williams and I give a *tight* 2-approximation algorithm in $\tilde{O}(mn^{3/4})$ time for directed unweighted graphs. We show that any algorithm that gives a better than 2-approximation algorithm for the girth in directed graphs requires $m^{2-o(1)}$ under popular fine-grained complexity conjectures. We also give an almost 2-approximation algorithm in $\tilde{O}(m\sqrt{n})$ for directed weighted graphs.

1.1.4 Finding Long Shortest Paths and Short Cycles: Variants

In Chapter 6 we study variants of distance problems, where either the distance measure is not the standard measure used, or we are only interested about the shortest path distance between *some* pairs of node. These variants are natural and have many applications, but haven't been studied a lot. We present a thorough study of algorithms and conditional lower bounds for these variants.

Bichromatic and ST distance problems

Given two arbitrary subsets S and T of the nodes, we can define distance problems that only concern shortest path distances between the nodes in S and the nodes in T. This way Backurs et al [BRS$^+$18] define the ST-diameter, and analogously ST-eccentricities and ST-radius can be de-

[1] $\tilde{O}$ neglects poly logarithmic factors

fined. If S and T are non-overlapping and cover all the nodes, we call these problems bichromatic-eccentricities, radius and diameter.

There are related and well-studied problems to bichromatic and ST-distance problems such as point sets (commonly known as Bichromatic Farthest Pair), the subset version of spanners as well as the ST version of spanners (see Cection 6.1). Moreover, ST and bichromatic versions of distance problems are very related to the standard versions. For example, the techniques used in [BRS$^+$18] for obtaining lower bounds for approximating ST-diameter is later used to obtain lower bounds for the standard diameter.

In [DWVW19], Nicole Wein, Nikhil Vyas, Virginia Vassilevska Williams and I present a comprehensive study of the *approximability* of ST and Bichromatic Diameter, Radius, and Eccentricities, and variants, in graphs with and without directions and weights. We give the first nontrivial approximation algorithms for most of these problems, including time/accuracy trade-off upper and lower bounds. We show that nearly *all* of our obtained bounds are tight under the Strong Exponential Time Hypothesis (SETH), or the related Hitting Set Hypothesis.

For instance, for Bichromatic Diameter in undirected weighted graphs with m edges, we present an $\tilde{O}(m^{3/2})$ time $5/3$-approximation algorithm, and show that under SETH, neither the running time, nor the approximation factor can be significantly improved while keeping the other unchanged.

Min-distance Problems

Another set of variants of the standard distance problems use a different measure for computing the distance in directed graphs. In undirected graphs, the notion of distance is symmetrical but this is not true in directed graphs. There are natural ways to define a symmetric distance notion in directed graphs, such as *max, min* and *roundtrip* distances. In Sections 6.2 and 6.3 we focus on min-distance problems, where the min-distance between nodes v and u is defined as the minimum of the shortest path distance from v to u and from u to v. The main difficulty in working with min-distance problems is that min-distance doesn't obey triangle inequality, and hence the usual techniques for obtaining algorithms do not directly work for these problems.

The only known nontrivial algorithms are by Abboud, Vassilevska W. and Wang [AVW16].

For Min-Diameter [AVW16] gives a near-linear time 2-approximation algorithm if the input is a directed acyclic graph (DAG). For general graphs, the only nontrivial fast approximation algorithm is an $\tilde{O}(mn^{1-\epsilon})$ time n^{ϵ}-approximation algorithm for any constant $\epsilon > 0$ (No constant factor approximation algorithm is known that runs significantly faster than just computing APSP.) For Min-Radius, [AVW16] gives an $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for directed acyclic graphs. For general graphs, they only achieve a very weak n-approximation in near-linear time that checks if the Min-Radius is finite. There are no known approximation algorithms for Min-Eccentricities faster than just computing APSP.

In the [DWV+19] Nikhil Vyas, Yinzhan Xu, Yuancheng Yu, Nicole Wein, Virginia Vassilevska Williams and I greatly improve the bounds of [AVW16] by providing the first constant factor approximations for diameter, radius and eccentricities in general graphs. We provide an almost 4-approximation algorithm for min-diameter, an almost 5-approximation algorithm for min-eccentricities and an almost 3-approximation algorithm for min-radius all in $\tilde{O}(m\sqrt{n})$ running time. We also provide a series of time-accuracy trade-off algorithms for min-diameter

Later in [DK21], Jenny Kaufmann and I improve upon the results of [AVW16] for DAGs. For example we close the gap between upper and lower bounds for min-radius, obtaining a 2-approximation algorithm which runs in $\tilde{O}(m\sqrt{n})$ time. As the lower bound of [AVW16] for min-radius only works for sparse DAGs, we further show that our algorithm is conditionally tight for dense DAGs using a reduction from Boolean matrix multiplication.

Chapter 2

Preliminaries

In this chapter we explain some of the popular fine-grained complexity hypotheses. We operate mostly in the *world-RAM model* with $\log n$ bit words which is a model of computation that is a random-access machine able to do bitwise operations on a single word of $\log n$ bits, where n is often the size of the input to the problem at hand.

2.1 (Strong) Exponential Time Hypothesis (ETH/SETH)

The *k-CNF satisfiability (k-CNF SAT)* problem is one of the fundamental problems in computer science. It takes a formula Φ with n-variables and m clauses as input. This formula is in *Conjunctive Normal Form (CNF)*, which means that it is the AND of m clauses, where each clause is the OR of at most k literals (variables or negated variables). The k-CNF SAT problem asks if there is an assignment of variables to $\{0, 1\}$ such that the formula is satisfied. It is proved that 3-CNF SAT (and k-CNF SAT for any integer $k \geq 3$) is NP-complete [Coo71]. Using the Sparsification Lemma of Calabro, Impagliazzo and Paturi [CIP06], one can assume that $m = O(n)$, and hence there is a simple $O(2^n poly(n))$ algorithm for k-CNF SAT by considering every assignment of the variables.

Hypothesis 1 ((Strong) Exponential Time Hypothesis [IP01b])**.** *Let c_k be the smallest constant such that there is an algorithm in the world-RAM model with $\log n$ bit words for k-CNF SAT that*

runs in $O(2^{c_k n+o(n)})$ *time.* Exponential Time Hypothesis (ETH) *states that* $c_k > 0$ *for all* $k > 2$. Strong Exponential Time Hypothesis (SETH) *states that there is no constant* $\epsilon > 0$ *such that* $c_k \leq 1 - \epsilon$ *for all constant* k.

2.2 k-Orthogonal Vectors (k-OV)

In the *k-OV problem*, we are given k unsorted lists $L_1, \ldots, L_k$ of n zero-one vectors of length d as input. If there are k vectors $v_1 \in L_1, \ldots, v_k \in L_k$ such that for each coordinate $i \in [1, d]$ there exist an index $j \in [1, k]$ such that v_j is zero in coordinate i (i.e. $v_j[i] = 0$) we call these k vectors an orthogonal k-tuple. One should return true if there is an orthogonal k-tuple in the input.

Hypothesis 2 (k-OV hypothesis [Vas18])**.** *The k-OV hypothesis states that the k-OV problem requires $n^{k-o(1)}$ time for randomized algorithms.*

The k-OV hypothesis is equivalent to saying no $O(n^{k-\epsilon})$ time algorithm exists for k-OV for constant $\epsilon > 0$. Moreover, there is a reduction from k-CNF SAT to ℓ-OV, for any integers $k \geq 3$ and $\ell \geq 2$ [Wil05] that shows that k-OV hypothesis is implied by SETH.

2.3 k-Clique Detection

k-clique detection is simply graph pattern detection when the pattern is a k-clique: We are given a host graph and we are asked to see if this graph has a k-clique as a subgraph.

Following Itai and Rodeh [IR78], Nešetril and Poljak [NP85] showed that a k-clique can be detected in an n node graph G asymptotically in time $C(n, k) := M(n^{\lfloor k/3 \rfloor}, n^{\lceil k/3 \rceil}, n^{\lceil (k-1)/3 \rceil})$, where $M(a, b, c)$ is the fastest known runtime for multiplying an $a \times b$ by a $b \times c$ matrix. A simple bound for $M(a, b, c)$ is $M(a, b, c) \leq abc/\min\{a, b, c\}^{3-\omega}$ where $\omega < 2.373$ is the exponent of square matrix multiplication [AV21], but faster algorithms are known (e.g. Le Gall and Urrutia [GU18]). In particular, $C(n, k) \leq O(n^{\omega k/3})$ when k is divisible by 3.

The $C(n, k)$ runtime for k-clique detection has had no improvements in more than 40 years. Because of this, several papers have hypothesized that the runtime might be optimal for k-cliques (and k-Independent Sets) (e.g. [ABW15, BW17, LWW18]).

Hypothesis 3 (k-clique Hypothesis)**.** *On a word-RAM with $O(\log n)$ bit words, for every constant $k \geq 3$, k-clique requires $n^{\omega(\lfloor k/3 \rfloor, \lceil k/3 \rceil, \lceil (k-1)/3 \rceil) - o(1)}$ time.*

Here $\omega(a, b, c)$ is the exponent of the best running time for multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix.

2.4 All Pairs Shortest Paths (APSP)

All Pair Shortest Paths problem takes as input a graph G with n nodes and m edges, with edge weights in $[-R, R]$ where $R = O(n^c)$ for some constant c. It asks for the length of the shortest path between any pair of nodes. The length of a path is the sum of the edge weights for all edges on that path.

APSP can be solved in $O(\min\{mn + n^2 \log\log n, n^3/exp(\sqrt{\log n})\})$ time [Pet04, Wil14]. For undirected unweighted graphs, APSP can be solved using matrix multiplication in time $\tilde{O}(n^\omega)$ [Sei95], where $2 \leq \omega < 2.373$ is the matrix multiplication exponent [AV21]. For directed unweighted graphs, APSP can be solved in time $\tilde{O}(n^{2.529})$ [Zwi02] (one can get slightly better bounds using rectangular matrix multiplication [LU18]). There is a conjecture that we cannot do better than n^3 time for APSP in weighted directed graphs.

Hypothesis 4 (APSP Hypothsis [WW13])**.** *APSP in weighted directed graphs requires $n^{3-o(1)}$ time.*

Chapter 3

Finding Fixed Sized Patterns

This chapter is written with authors Thuy Duong Vuong and Virginia Vassilevska Williams. In this chapter we consider the pattern detection problem in graphs: given a constant size pattern graph H and a host graph G, determine whether G contains a subgraph isomorphic to H. We present the following upper and lower bounds:

- We prove that if a pattern H contains a k-clique subgraph, then detecting whether an n node host graph contains a *not necessarily induced* copy of H requires at least the time for detecting whether an n node graph contains a k-clique. The previous result of this nature required that H contains a k-clique which is disjoint from all other k-cliques of H.

- We show that if the famous Hadwiger conjecture from graph theory is true, then detecting whether an n node host graph contains a *not necessarily induced* copy of a pattern with chromatic number t requires at least the time for detecting whether an n node graph contains a t-clique. This implies that: (1) under Hadwiger's conjecture for *every* k-node pattern H, finding an *induced* copy of H requires at least the time of $\sqrt{k}$-clique detection and size $\omega(n^{\sqrt{k}/4})$ for any constant depth circuit, and (2) unconditionally, detecting an *induced* copy of a random $G(k,p)$ pattern w.h.p. requires at least the time of $\Theta(k/\log k)$-clique detection, and hence also at least size $n^{\Omega(k/\log k)}$ for circuits of constant depth.

- We show that for every k, there exists a k-node pattern that contains a $k-1$-clique and that

can be detected as an *induced* subgraph in n node graphs in the best known running time for $k-1$-Clique detection. Previously such a result was only known for infinitely many k.

3.1 Introduction

One of the most fundamental graph algorithmic problems is Subgraph Isomorphism: given two graphs $G = (V, E)$ and $H = (V_H, E_H)$, determine whether G contains a subgraph isomorphic to H. While the general problem is NP-complete, many applications (e.g. from biology [ADH$^+$08, PCJ06]) only need algorithms for the special case in which H is a small graph pattern, of constant size k, while the host graph G is large. This graph pattern detection problem is easily in polynomial time: if G has n vertices, the brute-force algorithm solves the problem in $O(n^k)$ time, for any H.

Two versions of the Subgraph Isomorphism problems are typically considered. The first is the *induced* version in which one seeks an injective mapping $f : V_H \mapsto V$ so that $(u, v) \in E_H$ if and only if $(f(u), f(v)) \in E$. The second is the *not necessarily induced* version where one seeks an injective mapping $f : V_H \mapsto V$ so that if $(u, v) \in E_H$ then $(f(u), f(v)) \in E$ (however, if $(u, v) \notin E_H$, $(f(u), f(v))$ may or may not be an edge). It is not hard to show (e.g. via color-coding) that when k is a constant, any algorithm for the induced version can be used to solve the not necessarily induced one (for the same pattern) in asymptotically the same time, up to logarithmic factors. In this chapter we consider both of the settings.

3.1.1 Hardness

A standard generalization of a result of Nešetril and Poljak [NP85] shows that the induced subgraph isomorphism problem for any k-node pattern H in an n-node host graph can be reduced in $O(k^2n^2)$ time to the k-Clique (or induced k-Independent Set (IS)) detection problem in kn-node graphs. Thus, for constant k, k-Clique and k-IS are the hardest patterns to detect.

Following Itai and Rodeh [IR78], Nešetril and Poljak [NP85] showed that a k-Clique (and hence any induced or not-necessarily induced k-node pattern) can be detected in an n node graph G asymptotically in time $C(n, k) := M(n^{\lfloor k/3 \rfloor}, n^{\lceil k/3 \rceil}, n^{\lceil (k-1)/3 \rceil})$, where $M(a, b, c)$ is the fastest known runtime for multiplying an $a \times b$ by a $b \times c$ matrix. A simple bound for $M(a, b, c)$ is $M(a, b, c) \leq abc/\min\{a, b, c\}^{3-\omega}$ where $\omega < 2.373$ is the exponent of square matrix multiplication [Vas12, Le 14], but faster algorithms are known (e.g. Le Gall and Urrutia [GU18]). In

particular, $C(n,k) \leq O(n^{\omega k/3})$ when k is divisible by 3.

The $C(n,k)$ runtime for k-Clique detection has had no improvements in more than 40 years. Because of this, several papers have hypothesized that the runtime might be optimal for k-Cliques (and k-Independent Sets) (e.g. [ABW15, BW17, LWW18]).

Meanwhile, for some k-node patterns H that are not Cliques or Independent Sets, specialized algorithms have been developed that are faster than the $C(n,k)$ runtime for k-Clique. For instance, if H is a 3-node pattern that is not a triangle or an independent set, it can be detected in G in linear time, much faster than the $C(n,3) = O(n^{\omega})$ time for 3-Clique/triangle. Following work of [CPS85, Ola90, EG04, KKM00, KLL13], Vassilevska W. et al. [WWWY15] showed that every 4-node pattern except for the 4-Clique and 4-Independent Set can be detected in $C(n,3) = O(n^{\omega})$ time, much faster than the $C(n,4)$ runtime for 4-Clique. Bläser et al. [BKS18] recently showed that for $k \leq 8$ there are faster than $C(n,k)$ time algorithms for all non-clique non-independent set k-node patterns; for $k \leq 6$, their runtime is $C(n,k-1)$. Independently, we were able to show the same result, using an approach generalizing ideas from [WWWY15], see section 3.3.

A natural conjecture, consistent with the prior work so far is that for every k and every k-node pattern H that is not a clique or independent set, one can detect it in an n node graph in time $C(n,k-1)$. Blaeser et al. showed that for all k of the form $3 \cdot 2^{\ell}$ for integer ℓ, there is a k-node pattern that (1) is at least as hard to detect as $k-1$-Clique and (2) can be detected in $C(n,k-1)$ time. We show that such a pattern exists for *all* $k \geq 3$ (Theorem 3.3.4).

While there exist k-node patterns that can be detected faster than k-Clique, it seems unclear how hard k-node pattern detection actually is. For instance, it could be that for every k, there is *some* induced pattern on k-nodes that can be detected in say $n^{\log\log(k)}$ time, or even $f(k)n^c$ time, where c is independent of k. A Ramsey theoretic result tells us that every k-node H either contains an $\Omega(\log k)$ size clique or an $\Omega(\log k)$ size independent set. Hence intuitively, detecting any k-node H in an n node graph should be at least as hard as detecting an $\Omega(\log k)$ size clique in an n node graph. The widely believed Exponential Time Hypothesis (ETH) [IP01c] is known to imply that k-Clique cannot be solved in $n^{o(k)}$ time [CCF+05]. Coupled with the Ramsey result, ETH should intuitively imply that no matter which k-node H we pick, H-pattern detection cannot be solved in

$n^{o(\log k)}$ time.

Unfortunately, however, *it is still open whether every pattern that contains a t-clique is as hard to detect as a t-clique* (see e.g. [BKS18][1]). In general, it is not clear what makes patterns hard to detect[2].

One of the few results related to this is by Floderus et al. [FKLL15] who showed that if a pattern H contains a t-Clique that is disjoint from all other t-Cliques in H, then H is at least as hard to detect as a t-Clique. This implied strong clique-based hardness results for induced k-path and k-cycle. However, the reduction of [FKLL15] fails for patterns whose k-Cliques intersect non-trivially.

The main difficulty in reducing k-Clique to the detection problem for other graph patterns H can be seen in the following natural attempt used e.g. by [FKLL15]. Say H has a k-clique K and let H' be the graph induced by the vertices of H not in K. Let $G = (V, E)$ be an instance of k-Clique. We'll start by creating k copies of V, $V_1, \ldots, V_k$. For every edge (u, v) of G, add an edge between the copies of u and v in different parts (This is essentially the Kronecker/Tensor product of G and K_k). Every k-clique C of G appears in the new graph $k!$ times; we'll say that the main copy $\bar{C}$ of C has the ith vertex of C (in lexicographic order say) appearing in V_i. Now, add a copy $\bar{H}'$ of H', using fresh vertices, and for every edge (h, i) of H with $h \in H'$ and $i \in K$, add edges from $h \in \bar{H}'$ to all vertices in V_i. This forms the new graph G' and guarantees that if G has a k-clique C, G' contains a copy of H which is just $\bar{C}$ together with $\bar{H}'$.

The other direction of the reduction fails miserably however. If G' happens to have a copy of H, there is no guarantee that any of the k-cliques of H would have a node from each V_i and hence form a clique of G. As a simple counterexample (Figure 3-1) consider H as a 4-Cycle $(1, 2, 3, 4)$ together with a node 5 that has edges to all nodes of the 4-Cycle. Starting from a graph G, WLOG we would pick K to be $(1, 2, 5)$ and $H' = 3, 4$ and form G' as described. Let $\bar{H}'$ contain the nodes $\bar{3}, \bar{4}$ and let the parts of G be V_1, V_2, V_5. Now the reduction graph G' might contain a copy of H even if G has no 3-cliques, as $\bar{4}$ could represent 5, and $1, 3$ and $2, 4$ could be represented by two

[1] Bläser et al. [BKS18] show that for the particular types of algorithms that they use a pattern that contains a k-clique cannot be found faster than a k-clique, and they note that such a result is not known for arbitrary algorithms.

[2] In contrast, there are almost tight lower bounds for "partitioned subgraph isomorphism", See [Mar07].

Figure 3-1: An example of how a simple reduction attempt fails to reduce 3-Clique to H. The edges between the V_i are determined by the 3-Clique instance.

nodes each in V_1 and V_5 respectively; see Figure 3-1. Hence the copy of H wouldn't use V_2 at all and doesn't represent a triangle in G.

One could try to modify the reduction, say by representing the nodes of H' by copies of the vertices of G, as with K. However, the same issues arise, and they seem to persist in most natural reduction attempts.

With an intricate construction, we show how to overcome this difficulty. Our first main theorem is that patterns that contain t-cliques are indeed at least as hard as t-Clique, and in fact we prove it for the *not necessarily induced* case which automatically gives a lower bound for the induced case (Theorem 3.2.1 in the body):

Theorem 3.1.1. *Let $G = (V, E)$ be an n-node, m-edge graph and let H be a k-node pattern such that H has a t-clique as a subgraph. Then one can construct a new graph G^* of at most nk vertices in $O(k^2m + k^2n)$ time such that G^* has a not necessarily induced subgraph isomorphic to H if and only if G has a t-clique.*

Note that since the not necessarily induced pattern detection can be solved with the induced version, a lower bound for the not necessarily induced pattern detection gives a lower bound for the induced version. Since for every k-node graph H, either H or its complement contains a clique of size $\Omega(\log k)$, ETH implies that no matter which k-node H we pick, induced H-pattern detection cannot be solved in $n^{o(\log k)}$ time.

Our second theorem shows that some patterns are even harder, as in fact *the hardness of a pattern grows with its chromatic number!*

Our theorem relies on the widely believed Hadwiger conjecture [Had57] from graph theory which roughly states that every graph with chromatic number t contains a t-clique as a *minor*[3].

[3] H is called a minor of the graph G if H can be formed from G by deleting edges and vertices and by contracting edges.

The Hadwiger conjecture is known to hold for $t \leq 6$ [RST93] and to almost hold for $t = 7$ [KT05] (It is equivalent to the 4-Color Theorem for $t = 5, 6$ [RST93, Wag37, RSST97].). It also holds for almost all graphs [BCE80]. Our lower bound theorem, which also proved for the *not necessarily induced* case (Theorem 3.2.2 in the body) is:

Theorem 3.1.2. *Let $G = (V, E)$ be an n-node graph and let H be a k-node pattern with chromatic number t, for $t > 1$. Then assuming that Hadwiger conjecture is true, one can construct G^* on at most nk vertices in $O(n^2k^2)$ time such that G^* has a not necessarily induced subgraph isomorphic to H if and only if G has a t-clique.*

This is the first connection between the Hadwiger conjecture and Subgraph Isomorphism, to our knowledge. Let us see some exciting consequences of this theorem. First, we get that if t is the maximum of the chromatic numbers of H and its complement, then an induced H is at least as hard as t-Clique to detect. Now, it is a simple exercise that the maximum of the chromatic number of a k-node graph and its complement is at least $\sqrt{k}$. *Thus, every induced H on k-nodes is at least as hard as $\sqrt{k}$-Clique. There are no easy induced patterns.*

Corollary 3.1.1. *No matter what k-node H we take, under ETH and the Hadwiger Conjecture, the induced subgraph isomorphism problem for H in n-node graphs cannot be solved in $n^{o(\sqrt{k})}$ time.*

This is the first result of such generality.

A second consequence comes from circuit complexity. Rossman [Ros08] showed that for any constant integers k and d, any circuit of depth d requires size $\omega(n^{k/4})$ to detect a k-Clique. Because of the simplicity of our reduction (it can be implemented in constant depth), we also obtain a circuit lower bound for induced pattern detection for any H node subgraph:

Corollary 3.1.2. *Let d and k be any integer constants. No matter what k-node H we take, under the Hadwiger Conjecture, any depth d circuit for the induced subgraph isomorphism problem for H in n-node graphs requires size $\omega(n^{\sqrt{k}/4})$.*

A third consequence is that in fact almost all k-node induced patterns are very hard – at least as hard as $\Theta(k/\log k)$-Clique. Consider an Erdös-Renyi graph H from $G(k, p)$ for constant p. It

is known [BCE80] that the Hadwiger conjecture holds for H with high probability. Moreover, the chromatic number of such graphs (and their complements) is with high probability $\Theta(k/\log k)$ [Bol88]; meanwhile the clique and independent set size is only $O(\log k)$. Thus our chromatic number theorem significantly strengthens our first theorem.

Corollary 3.1.3. *For almost all k-node patterns H, under ETH, induced H detection in n node graphs cannot be done in $n^{o(k/\log k)}$ time.*

We also immediately obtain, via Rossman's lower bound, that for almost all k-node patterns H, any constant depth circuit that can detect an induced H requires size $n^{\Omega(k/\log k)}$.

Related Work. Vassilevska [Vas08b] showed that $K_k - e$ (a k-clique missing an edge) can be found in $O(n^{k-1})$ time without using fast matrix multiplication, whereas the fastest algorithms for k-Clique without fast matrix multiplication run in $O(n^k/\log^{k-1} n)$ time [Vas09]; this was recently improved by Bläser et al. [BKS18] who showed that every k node pattern except the k-Clique and k-Independent Set can be detected in time $O(n^{k-1})$. Before this, Floderus et al. [FKLL13] showed that 5 node patterns[4] can be found in $O(n^4)$ time, again without using fast matrix multiplication.

Some other related work includes improved algorithms for subgraph detection when G has special structure (e.g. [KL17] and [FLR$^+$12]). Other work counts the number of occurrences of a pattern in a host graph (e.g. [KLL13, WW13, CDM17]). Finally, there is some work on establishing conditional lower bounds. Floderus et al. [FKLL15] produced reductions from k-Clique (or k-Independent Set) to the detection problem of ℓ-patterns for $\ell > k$ (but still linear in k). They show for instance that finding an induced k-path is at least as hard as finding an induced $k/2$-independent set. Lincoln et al. [LWW18] give conditional lower bounds for not-necessarily induced directed k-cycle detection. For instance, they show that if k-Clique requires essentially $C(n,k)$ time, then finding a directed k-Cycle in an m edge graph requires $m^{2\omega k/(3(k+1))-o(1)}$ time. This lower bound is lower than the upper bounds in this chapter, but they do show that superlinear time is likely needed.

Detecting k-Cycles in undirected graphs is an easier problem, when k is an even constant.

[4] All patterns except for K_5, K_4+e, $(3,2)-$fan, gem, house, butterfly, bull, C_5, $K_{1,4}$, $K_{2,3}$ and their complements; for these subgraphs the fastest runtime remained $C(n,5) \leq O(n^{4.09})$.

Yuster and Zwick [YZ94] showed that a k-Cycle in an undirected graph can be detected (and found) in $O(n^2)$ time for all even constants k. Dahlgaard et al. [DKS17] extended this result showing that k-Cycles for even k in m-edge graphs can be found in time $\tilde{O}(m^{2k/(k+1)})$. Their result implies that of [YZ94], as by a result of Bondy and Simonovits [BS74], any n node graph with $\geq 100kn^{1+1/k}$ edges must contain a $2k$-Cycle. When k is an odd constant, the k-Cycle problems in undirected and directed graphs are equivalent (see e.g. [Vas08b]).

3.1.2 Organization of the chapter

We start by providing lower bounds for detecting small subgraphs in Section 3.2. In Section 3.3, we provide our algorithms for the induced pattern detection. We first introduce a technique for detecting any k-node pattern that is not a clique or independent set in time $C(n, k-1)$. Using this technique in Subsection 3.3.5, we show that there is a k-node pattern that can be detected in $C(n, k-1)$ time in an n-node graph for all k.

3.2 Lower bounds

In this section we consider the problem of detecting and finding a (not necessarily induced) copy of a given small pattern graph H in a host graph G (we assume G and H are simple graphs with no self-loops). This is the variant of subgraph isomorphism in which the pattern H is fixed, on a constant k number of vertices, and $G = (V, E)$ with $|V| = n$ is given as an input. We focus on the hardness of this problem: we show that any fixed pattern that has a t-clique as a subgraph, is not easier to detect as a subgraph than a t-clique, formally stated as Theorem 3.2.1. First, we start by an easier case of the theorem where the pattern is t-chromatic to depict the main idea of our proof and then we proceed with the proof of the theorem for all patterns. Recall that a *proper vertex coloring* of a graph is an assignment of colors to each of its vertices such that no edge connects two identically colored vertices. If the set of colors is of size c, we say that the graph is c-colorable. The *chromatic number* of a graph is the smallest number c for which the graph is c-colorable, and we call such graph *c-chromatic*. In the second part of this section, we prove a stronger lower bound using Hadwiger conjecture, showing that under this conjecture any t-chromatic pattern is not easier

to detect as a subgraph than a t-clique.

Theorem 3.2.1. *Let $G = (V, E)$ be an n-node m-edge graph and let H be a k-node pattern such that H has a t-clique as a subgraph. Then one can construct G^* on at most nk vertices in $O(k^2m + k^2n)$ time such that G^* has a not necessarily induced subgraph isomorphic to H if and only if G has a t-clique.*

More specifically, we show that if G has a t-clique then G^* has an "induced" subgraph isomorphic to H, and if H has a "not necessarily induced" subgraph isomorphic to H, then G has a t-clique.

3.2.1 Simple case: t-Chromatic patterns

We show Theorem 3.2.1 when H is t-chromatic in addition to having a t-clique as a subgraph. Construct the new graph G^* as follows: For each $v \in H$, let G_v be a copy of the vertices of G as an independent set. For any two vertices v and u in H where vu is an edge, add the following edges between vertex sets G_v and G_u: for each w_1 and w_2 in G, add an edge between the copy of w_1 in G_v and the copy of w_2 in G_u if and only if w_1w_2 is an edge in G. So G^* has nk vertices and since for each pair of vertices $u, v \in H$ we have at most m edges between G_u and G_v, the construction time is at most $O(k^2m + kn)$.

Now we show that G has a t-clique as a subgraph if and only if G^* has H as a subgraph. First suppose that G has a t-clique, say $T = v_1, \ldots, v_t$. Consider a t-coloring of the vertices of H, with colors $1, \ldots, t$. For each $w \in H$, pick v_i from G_w if w is of color i. Call the induced subgraph on these vertices H^*. We show that H^* is isomorphic to H: map each $w \in H$ to the vertex picked from G_w in G^*. If w and w' are adjacent in H, then their colors are different, so the vertices that are picked from G_w and $G_{w'}$ are different vertices of G, and they are part of the clique T, so they are adjacent. If w and w' are not adjacent, we don't have any edges between G_w and $G_{w'}$, so the vertices picked from them are not adjacent.

For the other direction, we show that if G^* has H as a subgraph then G has a t-clique. Since H has a t-clique as a subgraph, G^* also has a t-clique as a subgraph. Suppose the vertices of this clique are $W = \{w_1, \ldots, w_t\}$ where w_i is a copy of $v_i \in G$. Each pair of vertices of the clique are

in different copies of G, as these copies are independent sets. Moreover, for each $i, j \in \{1, \ldots, t\}$, since w_i and w_j are adjacent, they correspond to different vertices in G, so $v_i \neq v_j$. Since we connect two vertices in G^* if their corresponding vertices in G are connected, this means that v_i and v_j are connected in G. So $v_1, \ldots, v_t$ form a t-clique in G.

3.2.2 General case

Define a *t-clique covering* of a pattern H to be a collection $\mathcal{C}$ of sets of vertices of H, such that the induced subgraph on each set is t-colorable, and for any t-clique T of H, there is a set in $\mathcal{C}$ that contains all the vertices of T. For example, in Figure 3-2, the graph H_{ex} has the following 3-clique covering of size 2: $\{\{a_1, a_2, a_3, a_6\}, \{a_3, a_4, a_5, a_1, a_6\}\}$.

For each H we have at least one t-clique covering by considering the vertices of each t-clique of H as one set. However we are interested in the smallest collection $\mathcal{C}$. So for a fixed t, we define $p(H)$ to be the smallest integer $r \geq 1$, such that there is a t-clique covering of H of size r. We call a t-clique covering of size $p(H)$ a *minimum* t-clique covering. For example, if H is t-colorable, $p(H) = 1$ as the whole vertex set is the only set that the t-clique covering has. If H is not t-colorable but has a t-clique, then $p(H) > 1$. Note that when H has size k for a constant k, we can assume that finding a minimum t-clique covering for H takes constant time. One simple (and not very efficient) approach is to first list all the t-cliques of H, and then look at all the ways one can partition this list into subsets. For each partition, check whether the induced subgraph on these subsets is t-colarable. Call the partitions with this property valid, and take the valid partition with the least number of subsets.

Proof of Theorem 3.2.1. Let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be a minimum t-clique covering of H, where $r = p(H)$. The vertex set of the new graph G^* is the following: For each vertex $v \in C_1$, let G_v be a copy of the vertices of G as an independent set. For each vertex $v \in V(H) \setminus C_1$, let v^* be a copy of v in G^*. The edge set of G^* is as follows: For each two vertices $v, u \in C_1$ that uv is an edge in H, add the following edges between G_v and G_u: for each w_1 and w_2 in G, add an edge between the copy of w_1 in G_v and the copy of w_2 in G_u if and only if w_1w_2 is an edge in G. For each two vertices $u \in C_1$ and $v \in V(H) \setminus C_1$ that uv is an edge in H, connect v^* to all the vertices in G_u.

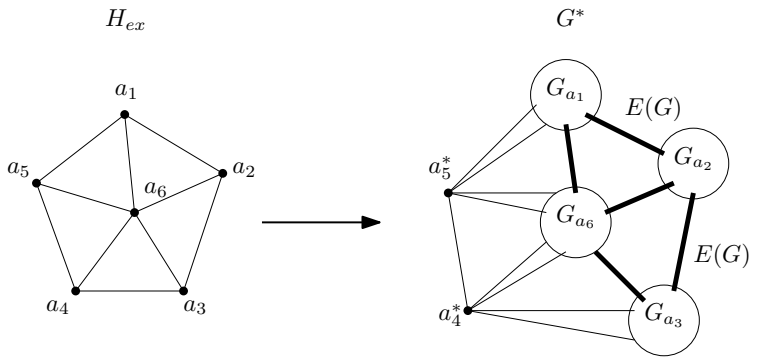

Figure 3-2: Graph H_{ex} on the left. The largest clique of this graph is a triangle. H_{ex} is 4-chromatic, so $p(H_{ex}) > 1$. We have $p(H_{ex}) = 2$, as a minimum 3-clique covering for it is $\{\{a_1, a_2, a_3, a_6\}, \{a_3, a_4, a_5, a_1, a_6\}\}$. The graph G^* is on the right, thick edges represent the way the edges are specified according to $E(G)$ between two copies of G.

For each two vertices $u, v \in V(H) \setminus C_1$ that uv is an edge in H, connect u^* and v^*. The way G^* is constructed is shown in Figure 3-2 for the particular pattern H_{ex} with maximum clique 3.

The number of nodes in G^* is $|C_1| \cdot |V(G)| + |V(H) \setminus C_1| \leq |V(H)| \cdot |V(G)| = nk$. The number of edges between G_v and G_u for some $u, v \in C_1$ is at most m, and the number of edges between any v^* and G_u for $u \in C_1$ and $v \notin C_1$ is at most n. The rest of the edges are at most k^2 many, so in total we have $O(k^2m+k^2n+k^2) = O(k^2m+k^2n)$ many edges. Note that since finding the minimum t-clique covering takes constant time (because k is a constant) the construction time is also $O(k^2m + k^2n)$.

Now we show that G has a t-clique as a subgraph if and only if G^* has H as a subgraph. First suppose that G has a t-clique, say $v_1, \ldots, v_t$. Consider a t-coloring of vertices of C_1, with colors $1, \ldots, t$. Let H^* be the subgraph on the following vertices in G^*: for each $w \in C_1$, pick v_i from G_w if w is of color i. For each $w \in V(H) \setminus C_1$, pick w^*. We show that H is isomorphic to H^*: for each $w \in C_1$, map w to the vertex picked from G_w, and for each $w \in V(H) \setminus C_1$, map w to w^*. If $w, u \in C_1$ such that $wu \in E(H)$, then their colors are different in the t-coloring of C_1, and so the vertices that are picked from G_w and G_u are different vertices of G and part of the t-clique of G, so they are adjacent. If wu is not an edge, then there is no edge between G_w and G_u. If $w \in C_1$ and $u \in V(H) \setminus C_1$ and wu is an edge in H, then u^* is adjacent to all vertices in G_w including the vertex that is picked from G_w for H^*. If wu is not an edge, then there is no edge between u^* and

G_w. If $w, u \in V(H) \setminus C_1$, then u^*, w^* are both picked in H^* and they are adjacent in G^* if and only if w and u are adjacent in H.

For the other direction, we show that if G^* has a subgraph H^* isomorphic to H, then G has a t-clique. Let $S_1 = \cup_{v \in C_1} G_v$. First suppose that H^* has a t-clique T using vertices in S_1. Since for each $v \in C_1$, G_v is an independent set, no two vertices of T are in the same G_v. So there are t vertices of H, $v_1, \ldots, v_t$ such that T has a vertex in each G_{v_i}. Let this vertex be a copy of $w_i \in G$. Since for each $i, j \in \{1, \ldots, t\}$, $i \neq j$, the copies of w_i and w_j are adjacent in G^*, we have that $w_i \neq w_j$ and they are adjacent in G. So $\{w_1, \ldots, w_t\}$ form a t-clique in G.

So assume that the induced subgraph on $V(H^*) \cap S_1$ in G^* has no t-clique. As S_1 has all the vertices in G^* that correspond to the vertices in C_1, we define similar sets for other C_is. For $i \in \{2, \ldots, p(H)\}$, let $S'_i = \cup_{v \in C_i \cap C_1} G_v$, $S''_i = \cup_{v \in C_i \setminus C_1} v^*$ and $S_i = S'_i \cup S''_i$. First note that the induced subgraph on S_i is t-colorable: Consider the t-coloring of C_i. For each $v \in C_i \setminus C_1$, color v^* the same as v. For each $v \in C_i \cap C_1$, color all vertices in G_v the same as v.

Now we show that any t-clique in H^* is in one of the sets $S_2, \ldots, S_{p(H)}$. This means that the collection $\{S_2 \cap V(H^*), \ldots, S_{p(H)} \cap V(H^*)\}$ is a t-clique covering for H^* (and thus for H) with size $p(H) - 1$, which is a contradiction. Consider a t-clique $T = v_1, \ldots, v_t$ in H^*. Each v_i is in one of the copies of G or is a copy of a vertex in H. So for each v_i, there is some vertex $w_i \in H$, such that $v_i \in G_{w_i}$ and $w_i \in C_1$ if $v_i \in S_1$, or $v_i = w_i^*$ and $w_i \notin C_1$ if $v_i \notin S_1$. Since for each i, j, v_i and v_j are adjacent in G^*, this means that w_i and w_j are different vertices in H and they are adjacent. So $W = \{w_1, \ldots, w_t\}$ form a clique in H. Since $T \not\subseteq S_1$, WLOG we can assume that $v_1 \notin S_1$. So $w_1 \notin C_1$. So the t-clique W is not in C_1, and so it is in C_i, for some $2 \leq i \leq p(H)$. Hence, $T \subseteq S_i$. □

Corollary 3.2.1. *Let H be a k-node pattern that has a t-clique or a t-independent set as a subgraph. Then the problem of finding H as an induced subgraph in an n-node graph is at least as hard as finding a t-clique in an $O(n)$-node graph.*

3.2.3 A Stronger Lower Bound

One of the oldest conjectures in graph theory is Hadwiger conjecture which introduces a certain structure for t-chromatic graphs. Assuming that this conjecture is true, we show that any fixed pattern with chromatic number t is not easier to detect as an induced subgraph than a t-clique. This strengthens the previous lower bound because the size of the maximum clique of a pattern is at most its chromatic number, and moreover there are graphs with maximum clique of size two but large chromatic number.

Conjecture 1 (Hadwiger's Conjecture)**.** *Let H be a graph with chromatic number t. Then one can find t disjoint connected subgraphs of H such that there is an edge between every pair of subgraphs.*

Contracting the edges within each of these subgraphs so that each subgraph collapses to a single vertex produces a t-clique as a minor of H. This is the property we are going to use to show that H is at least as hard to detect as a t-clique. Our main theorem is as follows.

Theorem 3.2.2. *Let $G = (V, E)$ be an n-node graph and let H be a k-node t-chromatic pattern, for $t > 1$. Then assuming that Hadwiger conjecture is true, one can construct G^* on at most nk vertices in $O(n^2k^2)$ time such that G^* has a (not necessarily induced) subgraph isomorphic to H if and only if G has a t-clique.*

To prove Theorem 3.2.2, we use a similar approach as Theorem 3.2.1. The approach of Theorem 3.2.1 is covering the maximum cliques of the pattern by a collection of subgraphs. However, since in Theorem 3.2.2 the pattern doesn't necessarily have a t-clique, we cover another particular subgraph of the pattern, and hence we introduce a similar notion as t-clique covering for this subgraph.

Let F be a graph with a vertex (not necessarily proper) coloring $C : V(F) \to \{1, \ldots, t\}$. We say that F has a K_t *minor with respect to the coloring* C if the vertices of each color induce a connected subgraph and for every color there is an edge from one of the vertices of that color to one of the vertices of every other color. For example, in Figure 3-2, consider the following coloring for

H_{ex}: $C_{ex} : \{a_1, \ldots, a_6\} \to \{1, \ldots, 4\}$, where $C_{ex}(a_1) = C_{ex}(a_2) = 1$, $C_{ex}(a_3) = C_{ex}(a_4) = 2$, $C_{ex}(a_5) = 3$ and $C_{ex}(a_6) = 4$. Clearly H_{ex} has a K_4 minor with respect to the coloring C_{ex}.

Let F and H be two fixed graphs, where F is t-chromatic. We say that H is (K_t, F) *minor colorable* if there is a (not necessarily proper) coloring $C : V(H) \to \{1, \ldots, t\}$ such that any induced copy of F in H has a K_t minor with respect to C. For example, in Figure 3-3, the graph H'_{ex} has graph H_{ex} (Figure 3-2) as a 4-chromatic subgraph, and it is (K_4, H_{ex}) minor colorable: There are exactly two copies of H_{ex} in H'_{ex}, one with vertex set $\{a_1, \ldots, a_6\}$ and one with vertex set $\{a_1, a_4, a_5, a_6, a_7, a_8\}$, and both have a K_4 minor with respect to the coloring given in Figure 3-3. Note that *minor colorability* is different from *colorability* and the *chromatic number* of a graph: recall that a graph is c-colarable for an integer c if the graph has a proper coloring using c colors and the graph is c-chromatic (its chromatic number is c) if c is the smallest integer such that the graph is c-colorable.

Let H be a pattern and let F be a t-chromatic subgraph of H. As a generalization to a t-clique covering of H, we define an *F-covering* of H to be a collection $\mathcal{C}$ of sets of vertices of H, such that the induced subgraph of each set is (K_t, F) minor colorable, and each (not necessarily induced) copy of F is completely inside one of the sets in $\mathcal{C}$.

For any graph H, we have at least one F-covering by considering the vertices of each (not necessarily induced) copy of F as one set where the (K_t, F) minor colorablitiy of each set comes from Conjecture 1. Similar to t-clique coverings we are interested in the smallest collection $\mathcal{C}$ among all F-coverings. So for a fixed number t and a t-chromatic subgraph F of H, we define $p_F(H)$ to be the smallest integer $r \geq 1$, such that there is an F-covering of H of size r. We call an F-covering of size $p_F(H)$ a *minimum* F-covering. Note that $p_{K_t}(H) = p(H)$. For example, in Figure 3-3, $p_{H_{ex}}(H'_{ex}) = 1$, according to the coloring given in the figure. Note that similar to the t-clique covering, we can find a minimum F-covering in constant time if the size of H is constant by a brute-force argument.

Now we are ready to prove Theorem 3.2.2.

Proof of Theorem 3.2.2. We are going to mimic the proof of Theorem 3.2.1, and so we are going to carefully choose a subgraph F and consider the minimum F-covering of it.

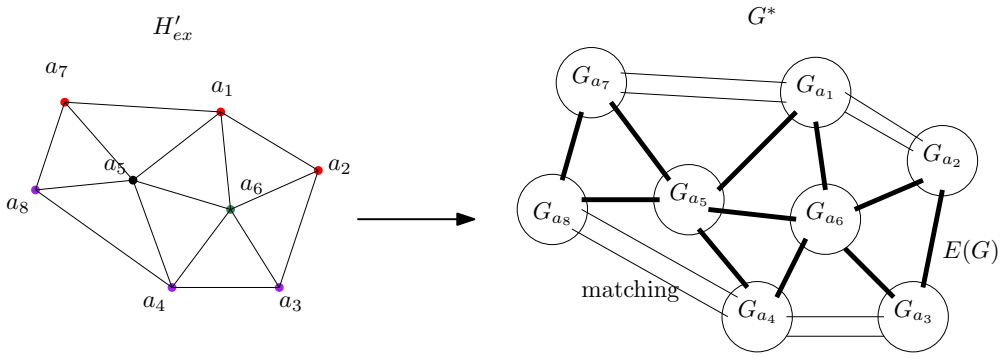

Figure 3-3: The 4-chromatic graph H'_{ex} on the left side has the coloring C'_{ex} which makes it (K_4, H_{ex}) minor colorable: $C'_{ex}(a_1) = C'_{ex}(a_2) = C'_{ex}(a_7) = 1$, $C'_{ex}(a_3) = C'_{ex}(a_4) = C'_{ex}(a_8) = 2$, $C'_{ex}(a_5) = 3$, $C'_{ex}(a_6) = 4$. On the right side we show how G^* is constructed as it is described in the proof of Theorem 3.2.2. The double edges indicate a matching where nodes that are copy of the same vertex in G are connected. The thick edges represent the way we add edges according to $E(G)$.

Let z be the largest integer such that every $(z-1)$-node subgraph of H is $t-1$ colorable. Let F be a t-chromatic subgraph of H on z nodes with maximum number of edges. Note that F is an induced subgraph of H. In Figure 3-3, $H = H'_{ex}$ is 4-chromatic and one can check that any subgraph on 5 vertices or less is 3 colorable. In this graph $z = 6$ and $F = H_{ex}$.

Now suppose that $\mathcal{C} = \{C_1, \ldots, C_r\}$ is a minimum F-covering of H, where $r = p_F(H)$. Let $f : C_1 \to \{1, \ldots, t\}$ be a (K_t, F) minor coloring of C_1. Define the vertex set of G^* as follows: For each vertex $v \in C_1$, let G_v be a copy of G as an independent set. For each vertex $v \in V(H) \backslash C_1$, let v^* be a copy of v in G^*. The edge set of G^* is as follows: For each pair of vertices $u, v \in C_1$, if uv is not an edge in H we don't add any edges between G_u and G_v. If uv is an edge and $f(u) = f(v)$, then add the following edges between G_u and G_v: For each $w \in G$, add an edge between the copy of w in G_u and the copy of w in G_v (So we have a complete matching between G_u and G_v). If uv is an edge and $f(u) \neq f(v)$, then add the following edges between G_u and G_v: for each w_1 and w_2 in G, add an edge between the copy of w_1 in G_u and the copy of w_2 in G_v if and only if $w_1 w_2$ is an edge in G. For each pair of vertices $u \in C_1$ and $v \in V(H) \setminus C_1$ such that uv is an edge in H, add an edge between v^* and all vertices in G_u. For each pair of vertices $u, v \in V(H) \setminus C_1$ such that uv is an edge in H, add an edge between u^* and v^* in G^*. In Figure 3-3, H'_{ex} has a H_{ex}-covering of size 1 which is the whole graph. On the right side of the figure we show how G^* is constructed.

The number of nodes in G^* is $|C_1|\cdot|V(G)|+|V(H)\setminus C_1| \leq |V(H)|\cdot|V(G)| = nk$. The number of edges between G_v and G_u for some $u, v \in C_1$ is at most $\max(m, n)$, where m is the number of edges of G. The number of edges between any v^* and G_u for $u \in C_1$ and $v \notin C_1$ is at most n. The rest of the edges are at most k^2 many, so in total we have $O(k^2m + k^2n + k^2) = O(k^2m + k^2n)$ many edges. Note that finding z and F takes constant time by a brute-force argument on all the subgraphs of H. Since finding the minimum F-covering takes constant time (because k is a constant) the construction time is also $O(k^2m + k^2n) \leq O(n^2k^2)$.

Now we show that G has a t-clique as a subgraph if and only if G^* has H as a subgraph. First, suppose that G has a t-clique, say $T = v_1, \ldots, v_t$. Let H^* be the induced subgrpah on the following vertices in G^*: for each $w \in C_1$, pick v_i from G_w if $f(w) = i$. For each $w \in V(H) \setminus C_1$, pick w^*. We show that H is isomorphic to H^*: for each $w \in C_1$, map w to the vertex picked from C_w, and for each $w \in V(H) \setminus C_1$, map w to w^*. If $u, w \in C_1$ and they are not adjacent, then there is no edge between G_u and G_w. If uw is an edge in H, then if $f(u) = f(w) = i$, we picked v_i from both G_u and G_w and hence there are adjacent (note that in this case the edges between G_u and G_w form a complete matching). If $f(u) \neq f(w)$, then the vertices that we picked from G_u and G_w are copies of different vertices of the clique T, and so they are adjacent in G^*. If $u \in C_1$ and $w \in V(H) \setminus C_1$ and uw is an edge in H, then w^* is adjacent to all vertices in G_u, so it is adjacent to the vertex chosen from G_u for H^*. If uw is not an edge, then there is no edge between w^* and G_u. If $u, w \in V(H) \setminus C_1$, then u^* and w^* are connected in H^* if and only if uw are connected in H.

For the other direction, we show that if G^* has a (not necessarily induced) subgraph H^* isomorphic to H, then G has a t-clique. Let $S_1 = \cup_{v \in C_1} G_v$. First suppose that H^* has a copy of F in S_1. Let the vertices of this copy be $w_1, \ldots, w_z$. For each w_i there is a vertex $v_i \in H$ such that $w_i \in G_{v_i}$. Now if for some $i \neq j$, $v_i = v_j$, then the induced subgraph on $\{v_1, \ldots, v_z\}$ has less than z vertices, so it is $t-1$ colorable (using proper coloring). Now if we color w_i the same color as v_i, we get a proper coloring of this copy of F with $t-1$ colors, a contradiction to the chromatic number of F. So for each $i \neq j$, $v_i \neq v_j$. Now we show that the induced subgraph on $\{v_1, \ldots, v_z\}$ in H is isomorphic to F. Call this subgraph F'. We just showed that $|V(F')| = z$. Since there

is no edge between G_{v_i} and G_{v_j} if v_i and v_j are not connected, we have that F is a subgraph of F', and so F' is not $t-1$ colorable, and since it is a subgraph of H, it is t-chromatic. If F and F' are not isomorphic, then F' has more edges than F, which is a contradiction. So F and F' are isomorphic, and in particular w_i and w_j are adjacent if and only if v_i and v_j are adjacent. Suppose that $w_i \in G_{v_i}$ is the copy of w'_i in G. We show that $\{w'_1, \ldots, w'_z\}$ contains exactly t distinct vertices that induce a t-clique in G. Consider the coloring f on C_1. First note that if v_i and v_j are adjacent vertices such that $f(v_i) = f(v_j)$, then since w_i and w_j are adjacent, we have $w'_i = w'_j$. Since F' is a copy of F in C_1, it has a K_t minor with respect to the coloring f. So the subgraph that each color induces is connected, and so for each v_i and v_j with $f(v_i) = f(v_j) = a$ we have $w'_i = w'_j$. This means that all w_i's with $f(v_i) = a$ are copies of the same vertex, say u_a. Now take a pair of colors, $a, b \in \{1, \ldots, t\}$. There are vertices v_i and v_j such that $f(v_i) = a$, $f(v_j) = b$ and v_iv_j is an edge in H. So w_iw_j is an edge in G^*, and since $a \neq b$, $w'_i \neq w'_j$, and $w'_iw'_j$ is an edge in G. Since $w'_i = u_a$ and $w'_j = u_b$, we have that u_a and u_b are different vertices and they are adjacent in G. So $\{w'_1, \ldots, w'_z\} = \{u_1, \ldots, u_t\}$ induces a t-clique in G^*.

Now suppose that there is no copy of F in the induced subgraph on $V(H^*) \cap S_1$ in G^*. For $i \in \{2, \ldots, p_F(H)\}$, let $S'_i = \cup_{v \in C_i \cap C_1} G_v$, $S''_i = \cup_{v \in C_i \setminus C_1} \{v^*\}$ and $S_i = S'_i \cup S''_i$. We prove that the collection $\{S_2 \cap V(H^*), \ldots, S_{p_F(H)} \cap V(H^*)\}$ is an F-covering for H^*, which means that $p_F(H) = p_F(H^*) < r$, a contradiction.

First we show that any copy of F in H^* is in one of S_is. Let F^* with vertex set $\{w_1, \ldots, w_z\}$ be a copy of F in H^*. For each w_i, there is a $v_i \in H$ where $w_i \in G_{v_i}$ and $v_i \in C_1$ if $w_i \in S_1$, or $w_i = v_i^*$ and $v_i \notin C_1$ if $w_i \notin S_1$. If $v_i = v_j$ for some $i \neq j$, then F^* is $t-1$ colorable (with proper coloring): the induced graph on $\{v_1, \ldots, v_z\}$ has at most $z-1$ vertices and so it is $t-1$ colorable. Color w_i the same as v_i. From the way we construct G^* we know that if v_i and v_j are not connected, w_i and w_j are also not connected, and so this coloring of F^* is proper. Since F^* is t-chromatic, this is a contradiction. So if we call the induced graph on $\{v_1, \ldots, v_z\} \subseteq V(H)$ by F^H, then $|V(F^H)| = z$. We know that if w_i and w_j are connected, then v_i and v_j are connected. So F is a subgraph of F^H, and so F^H is t-chromatic. If F^H and F are not isomorphic, it means that F^H has more edges than F, which is a contradiction. So F^H and F are isomorphic. Now since F^*

is not in S_1, WLOG we can assume that $w_1 \notin S_1$, and so $w_1 = v_1^*$ and v_1 is not in C_1. So $F^H \not\subseteq C_1$ and there is some $i \geq 2$ such that $F^H \subseteq C_i$. So F^* is in S_i.

Now we show that for each $i \geq 2$, $S_i \cap V(H^*)$ is (K_t, F)-minor colorable. Since C_i is (K_t, F)-minor colorable, there is a coloring $f_i : C_i \to \{1, \ldots, t\}$ such that each induced copy of F in C_i has a K_t minor with respect to f_i. Let $f_i^* : S_i \cap V(H^*) \to \{1, \ldots, t\}$ be the following coloring: For each $v \in C_i \cap C_1$, let $f_i^*(u) = f_i(v)$ for all vertices $u \in S_i \cap V(H^*) \cap G_v$. For each $v \in C_i \setminus C_1$ where $v^* \in V(H^*)$, let $f_i^*(v^*) = f_i(v)$. Now if $F^* = \{w_1, \ldots, w_z\}$ is a copy of F in $S_i \cap V(H^*)$, we know that the set $F^H = \{v_1, \ldots, v_z\}$ is a copy of F in C_i, where $w_i \in G_{v_i}$ if $w_i \in S_1$ and $w_i = v_i^*$ if $w_i \notin S_1$. Note that $f(v_i) = f_i^*(w_i)$ and v_i and v_j are adjacent if and only if w_i and w_j are adjacent. So since the subgraph induced on vertices of any color in F^H is connected, the subgraph induced on any color in F^* is also connected. Moreover, since in f_i for any pair of colors there is an edge between one of the vertices of that color to one of the vertices of the other color, this property holds for f_i^*. So $S_i \cap V(H^*)$ is (K_t, F)-minor colorable, and so we have an F-covering for H of size less than $p_F(H)$. □

Corollary 3.2.2. *Let H be a pattern and let t be the maximum chromatic number of H and its complement. Then under Hadwiger conjecture, finding an induced copy of H in an n-node graph is at least as hard as finding a t-clique in an $O(n)$-node graph.*

3.3 Induced Pattern Detection: Algorithms

In this section we focus on the algorithmic part of the induced pattern detection problem, starting with some background on the problem. First, it is a simple and folklore exercise to show that if there is a $T(n)$ time algorithm that can *detect* whether G contains a copy of H, then one can also *find* such a copy in $O(T(n))$ time: Partition the vertices V of G into $k+1$ equal parts (WLOG n is divisible by $k+1$), for every k-tuple of parts, use the detection algorithm in $T(nk/(k+1))$ time to check whether the union of the parts contains a copy of H. The moment a k-tuple of parts is detected to contain a copy of H, stop looking at other k-tuples and recurse on the graph induced by the union of the k parts. (Stop the recursion when n is constant, and brute force then.) Since every k node subgraph is contained in some k-tuple of the parts, the algorithm is correct. The

runtime is

$$t(n) \leq \sum_{i=1}^{\log_{(1+1/k)} n} (k+1)T(n(k/(k+1))^i)$$
$$\leq (k+1)T(n)\sum_{i=1}^{\infty}((k/(k+1))^2)^i \leq O(T(n)).$$

The second inequality above follows since $T(n) \geq \Omega(n^2)$ as the algorithm needs to at least read the input and the input can be dense. Because of this, for some nondecreasing function $g(n)$, $T(n) = n^2 g(n)$. Hence for any $L \geq 1$, $T(n/L) = n^2/L^2 g(n/L) \leq n^2/L^2 g(n) = T(n)/L^2$. (Without this observation about $T(n)$, the analysis would incur at most a $\log n$ factor for finding from detection.) As finding and detection are equivalent, we will focus on the detection version of the problem.

Recall from the introduction, $C(n,k) := M(n^{\lfloor k/3 \rfloor}, n^{\lceil k/3 \rceil}, n^{\lceil (k-1)/3 \rceil})$. Nešetril and Poljak [IR78] showed that the pattern detection problem can be reduced to rectangular matrix multiplication. In particular, when $k \equiv q \mod 3$, detecting a k node pattern in an n node G can be reduced in $O(n^{(2k+q)/3})$ time to the product of an $n^{\lfloor k/3 \rfloor} \times n^{\lceil k/3 \rceil}$ matrix by an $n^{\lceil k/3 \rceil} \times n^{\lceil (k-1)/3 \rceil}$ matrix.

Here we first recall the approach from [WWWY15], and then generalize the ideas there to obtain an approach for all k to show that (1) for all $k \leq 6$ and for all k-node H that is not a Clique or Independent Set, H can be detected in $O(C(n, k-1))$ time, whp, and (2) for all $k \geq 3$, there is a pattern that can be detected in time $O(C(n, k-1))$, whp.

3.3.1 The approach from [WWWY15]

Vassilevska W. et al. [WWWY15] proposed the following approach for detecting a copy of H in G:

1. First obtain a random subgraph G' of G by removing each vertex of G independently and uniformly at random with probability $1/2$.

2. Compute a quantity Q that equals the number of induced H in G', modulo a particular integer

q.

3. If $Q \neq 0 \mod q$, return that G contains an induced H, and otherwise, return that G contains no induced H with high probability.

The following lemma from [WWWY15] implies that (regardless of q), if G contains a copy of H, after the first step, with constant probability, the number of copies of H in G' is not divisible by q.

Lemma 3.3.1 ([WWWY15])**.** *Let $q \geq 2$ be an integer, G, H be undirected graphs. Let G' be a random induced subgraph of G such that each vertex is taken with probability $\frac{1}{2}$, independently. If there is at least one induced-H in G, the number of induced-H in G' is not a multiple of q with probability at least $2^{-|H|}$.*

Now using Lemma 3.3.1, we can sample graph G' from G, and with probability 2^{-k} we have the number of induced H is not divisible by q. To obtain higher probability, we can simply repeat this procedure.

Hence, it suffices to provide an algorithm for counting the number of copies of H modulo some integer. The approach from [WWWY15] is to efficiently compute a quantity which is an integer linear combination $Q = \sum_{i=1}^{t} \alpha_i n_{H_i}$ of the number of copies n_{H_i} in G of several different patterns $H = H_1, H_2, \ldots, H_t$, so that some integer q divides the coefficients α_i in front of n_{H_i} for $i > 1$ but q does not divide α_1. Thus, $Q = \alpha_1 n_H \mod q$.

Suppose that d is the largest common divisor of α_1 and q. Suppose that $d \neq 1$. Since q divides every α_k with $k > 1$, d must divide all α_i. Hence, we could just consider Q/d in place of Q, and take everything mod q/d instead of q.Thus WLOG α_1 and q are coprime, and so α^{-1} exists in $\mathbb{Z}_q$. Hence, $Q\alpha^{-1} = n_H \mod q$, and we can use this quantity in step 2 of the approach above.

For instance, if H is $K_4 - e$ (the diamond), one can compute the square A^2 of the adjacency matrix A of G in $O(n^\omega)$ time, and compute

$$Q = \sum_{(u,v) \in E} \binom{A^2(u,v)}{2} = n_{K_4 - e} + 6n_{K_4},$$

so that $Q = n_{K_4-e} \mod 6$.

In prior work, the equations Q were obtained carefully for each particular 4 node pattern. In this section we provide a general and principled approach of obtaining such quantities that can be computed in $O(C(n, k-1))$ time for $k \leq 6$.

3.3.2 Setup

As mentioned earlier, two graphs H and H' are isomorphic if there is an injective mapping from the vertex set of H onto the vertex set of H' so that edges and non-edges are preserved. We will represent this mapping by presenting permutations of the vertices of H and H', i.e. for two graphs H and H' with vertex orders $H = (v_1, \ldots, v_t)$ and $H' = (w_1, \ldots, w_t)$, we say H maps to H' if for each i and j, $(v_i, v_j) \in E(H)$ if and only if $(w_i, w_j) \in E(H')$. Note that if H maps to H', H' maps to H as well.

Throughout this section, fix an integer k and let $k' = \lfloor \frac{k-1}{3} \rfloor$. We refer to k-node graphs as patterns, and we want to detect them in n-node graphs. We will assume that every graph we consider is given with a vertex ordering, unless otherwise specified. We call a pattern with an ordering *labeled*, and otherwise, the pattern is *unlabeled*. By the subgraph $(v_1, \ldots, v_h)$ in a graph G, we mean the subgraph induced by these vertices, with this specified order when considering isomorphisms.

We partition all k-node patterns with specified vertex orders (there are $2^{\binom{k}{2}}$ many of these) into classes and for each class we count the number of subgraphs in a given graph G which map to one of the graphs in this class. For a k-node pattern $H = (v_0, \ldots, v_{k-1})$, define the class of k-node patterns $C(H)$ as follows:

Let F be the set of the following pairs of vertices: $(v_0, v_1), \ldots (v_0, v_{k'})$ (We sometimes refer to these pairs as the first k' edges of H). Then the graph $H' = (w_0, \ldots, w_{k-1})$ is in class $C(H)$ if for all pairs of vertices $(v_i, v_j) \notin F$, we have $(v_i, v_j) \in E(H)$ if and only if $(w_i, w_j) \in E(H')$. In other words, all graphs in a class agree on the edge relation except possibly for the pairs in F.

Note that for any $H' \in C(H)$, we have $C(H') = C(H)$. So each k-node pattern is in exactly one class, which is obtained by changing its first k' edges. Figure 3-4 shows two classes of graphs

for $k = 4$ (and hence $k' = 1$). In this case the set F consists of only one edge ((v_0, v_1)) and hence the graph classes are of size two.

(a) class c_1 (b) class c_2

Figure 3-4: Two graph classes for $k = 4$. In both classes, the graphs in the class agree on all edges except the edge v_0v_1

3.3.3 General Approach

Our goal is to detect an unlabeled pattern by counting the number of (labeled) patterns in different classes of graphs, which can be done as fast as the fastest algorithm for detecting $k-1$-clique (i.e. $C(n, k-1)$). Theorem 3.3.1 states this result formally and we prove it at the end of this section. The graph classes possess some useful properties which we introduce in Theorem 3.3.2 and Lemma 3.3.2 and provide their proofs at the end of this subsection. Using these properties, we show how to use graph classes to detect unlabeled patterns.

Theorem 3.3.1. *Let G be an n-node graph and let c be one of the classes of k-node patterns. We can count the number of subgraphs in G which map to a pattern in c in $O(C(n, k-1)) = O(M(n^{\lfloor \frac{k-1}{3} \rfloor}, n^{\lceil \frac{k-1}{3} \rceil}, n^{\lceil \frac{k-2}{3} \rceil}))$ time, which is the runtime of the fastest algorithm for detecting K_{k-1}.*

Note that Theorem 3.3.1 counts the number of "labeled" patterns where the labeling comes from an arbitrary but fixed initial ordering on $V(G)$.

Now we need to relate unlabeled patterns to pattern classes. Each unlabeled k-node pattern has $k!$ possible vertex orderings. We say that an unlabeled pattern $\tilde{H}$ embeds in class c if there is an ordering of vertices of $\tilde{H}$ which is in c. Let $U(c)$ be the set of unlabeled patterns that embed in c. For example, for the classes c_1 and c_2 in Figure 3-4, $U(c_1)$ consists of the diamond (also called diam for abbreviation) and the paw (depicted in Figure 3-5), and $U(c_2)$ consists of the diamond and K_4. For each unlabeled pattern $\tilde{H}$, let $\alpha^c_{\tilde{H}}$ denote the number of ways $\tilde{H}$ can be embedded in c, i.e. the number of vertex orderings of $\tilde{H}$ that put $\tilde{H}$ into c. In the example of Figure 3-4,

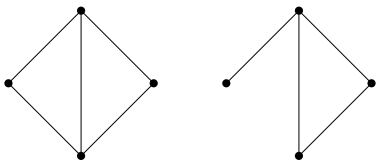

Figure 3-5: The diamond graph on the left and the paw graph on the right.

$\alpha^{c_1}_{diam} = 4 = \alpha^{c_2}_{diam}$, $\alpha^{c_1}_{paw} = 2$ and $\alpha^{c_2}_{K_4} = 24$. In this example, the $\alpha^c_{\tilde{H}}$ numbers are all equal to $|Aut(\tilde{H})|$, [5] each class contains at most one labeled copy of each H; in general, this need not be the case.

Let $n_{\tilde{H}}$ be the number of copies of $\tilde{H}$ in G. We have the following corollary:

Corollary 3.3.1. *The number of (labeled) subgraphs in G which map to a pattern in c is $\sum_{\tilde{H}\in U(c)} \alpha^c_{\tilde{H}} n_{\tilde{H}}$.*

The numbers $\alpha^c_{\tilde{H}}$ have some useful properties as shown in the next theorem.

Theorem 3.3.2. *For any unlabeled pattern $\tilde{H}$ we have $|Aut(\tilde{H})| \,|\, \alpha^c_{\tilde{H}}$. Moreover, for any class c, we have*

$$\sum_{\tilde{H}\in U(c)} \frac{\alpha^c_{\tilde{H}}}{|Aut(\tilde{H})|} = 2^{k'}.$$

First note that this theorem gives us upper and lower bounds on the size of $U(c)$. Each term in the above summation contributes at least 1, so $|U(c)| \leq 2^{k'}$. Moreover since c has at least $k'+1$ labeled patterns which have different numbers of edges, we have $|U(c)| \geq k'+1$. So we get the following corollary.

Corollary 3.3.2. *For any class c, we have $2^{k'} \geq |U(c)| \geq k'+1$.*

Define $b^c_{\tilde{H}} = \frac{\alpha^c_{\tilde{H}}}{|Aut(\tilde{H})|}$. By Corollary 3.3.1, the number of subgraphs in G that map to a pattern in c computed by Theorem 3.3.1 is of the following form:

$$\sum_{\tilde{H}\in U(c)} b^c_{\tilde{H}} |Aut(\tilde{H})| n_{\tilde{H}} \tag{3.1}$$

[5] $Aut(\tilde{H})$ is the automorphism group of $\tilde{H}$.

So far we showed how each pattern class relates to unlabeled patterns. Now we show how we can obtain different pattern classes from unlabeled patterns.

Lemma 3.3.2. *Let $\tilde{H}$ be an unlabeled k-node pattern. For an arbitrary vertex with degree at least k', consider k' of the edges adjacent to it; namely $e_1, \ldots, e_{k'}$. Let S be the set of all graphs obtained by removing any number of the edges in $\{e_1, \ldots, e_{k'}\}$. Then there is a class c, such that $U(c) = S$. Moreover, $b^c_{\tilde{H}} = 1$, and $\tilde{H}$ is the pattern with maximum number of edges in c.*

Applying Lemma 3.3.2 to our example, consider K_4 as the initial pattern and consider an arbitrary edge of it. Then the set S consists of the diamond and K_4, and so $U(c_2) = S$. Moreover, since $|Aut(K_4)| = 24 = \alpha^{c_2}_{K_4}$, we have $b^{c_2}_{K_4} = 1$. So by Theorem 3.3.2, $b^{c_2}_{diam} = 2 - 1 = 1$. Similarly if we consider the diamond as the initial pattern and take the edge between the degree three vertices, then the set S consists of the diamond and the paw, and so $U(c_1) = S$. Moreover, since $|Aut(diam)| = 4 = \alpha^{c_1}_{diam}$, we have $b^{c_1}_{diam} = 1$, and hence $b^{c_1}_{paw} = 1$.

Now we are ready to show how to detect unlabeled patterns using graph classes. First let B_r be the set of unlabeled patterns $\tilde{H}$ such that $r \mid |Aut(\tilde{H})|$. Note that we have $K_k, \bar{K_k} \in B_r$ for all r such that $r|k!$ (where K_k is the k-clique and $\bar{K_k}$ is the k-Independent set). For a fixed unlabeled pattern $\tilde{H}$ which is not the k-Independent Set or the k-Clique, the idea is to compute the sums of the form (3.1) for different pattern classes c, such that a linear combination of these sums gives us a sum consisting of only the terms from $\tilde{H}$ and patterns $\tilde{H'} \in B_r$ for some r such that $r \not| |Aut(\tilde{H})|$. More specifically, we want to compute a sum of the following form:

$$|Aut(\tilde{H})|n_{\tilde{H}} + \sum_{\tilde{H'} \in B_r} d_{\tilde{H'}}|Aut(\tilde{H'})|n_{\tilde{H'}} \tag{3.2}$$

where $d_{\tilde{H'}}$ are some integers. Then using the fact that this sum is equal to $|Aut(\tilde{H})|n_{\tilde{H}}$ modulo r, by the approach of Vassilevska W. et al. [WWWY15] we can assume with constant probability that $r \not| n_{\tilde{H}}$, and hence we can detect $\tilde{H}$ in G.

We first prove Theorem 3.3.2 and Lemma 3.3.2 below, and then we provide the proof of Theorem 3.3.1. Then we use our approach to show that for each k, there is a pattern that can be detected in time $O(C(n, k-1))$. Finally, we show how our approach is used to prove that any k-node

pattern except k-clique and k-independent set can be detected in $O(C(n, k-1))$ time, for $k \leq 6$.

Proof of Theorem 3.3.2. Let $H = (w_0, \ldots, w_{k-1})$ be an arbitrary pattern in c. Define $b^c_{\tilde{H}}$ to be the number of ways we can specify the edges $w_0w_1, \ldots, w_0w_{k'}$ so that the resulting vertex order maps to a vertex order of $\tilde{H}$. Note that this is independent of the choice of H, because all edges except the k' edges mentioned are the same for all $H \in c$. For each of these $b^c_{\tilde{H}}$ vertex orderings, we can apply $|Aut(\tilde{H})|$ automorphisms to get a different ordering that maps to it. So all these orderings make the $\alpha^c_{\tilde{H}}$ possible ways $\tilde{H}$ can be embedded in c; hence $\alpha^c_{\tilde{H}} = b^c_{\tilde{H}} \cdot |Aut(\tilde{H})|$. Now note that the total number of ways we can specify the k' edges $w_0w_1, \ldots, w_0w_{k'}$ is $2^{k'}$, so $\sum_{\tilde{H} \in U(c)} \frac{\alpha^c_{\tilde{H}}}{|Aut(\tilde{H})|} = \sum_{\tilde{H} \in U(c)} b^c_{\tilde{H}} = 2^{k'}$

Proof of Lemma 3.3.2. Let $H = (w_0, w_1, \ldots, w_{k-1})$ be an ordering of the vertices of $\tilde{H}$ such that $e_i = w_0w_i$ for each $i \in \{1, \ldots, k'\}$. Now each pattern $H' \in C(H)$ differs from H only in those k' edges, so the unlabeled version of H' is obtained from $\tilde{H}$ by removing some of e_i edges. So $C(H) \subseteq S$. Now consider $\tilde{H'} \in S$. Since $\tilde{H'}$ is obtained from $\tilde{H}$, we can consider the same ordering of vertices for it. Call this vertex order H'. So H' and H differ only in the k' first edges, so $H' \in C(H)$. Hence $S \subseteq U(c)$ which shows that $U(c) = S$.

Now since the number of ways we can embed H in class c is 1 (we have to put an edge between all the k' pairs of vertices), we have $b^c_{\tilde{H}} = 1$.

3.3.4 Proof of Theorem 3.3.1

The general idea is to remove one vertex, divide the rest of the vertices into three (almost) equal parts. Then form two matrices such that the first matrix captures the subgraphs isomorphic to the removed vertex plus the first part, and the second matrix captures the subgraphs isomorphic to the removed vertex plus the second and the third part, and then use matrix multiplication to count the number of subgraphs isomorphic to the whole pattern in the host graph. We show the approach more formally below.

Let $V(G) = \{v_1, \ldots, v_n\}$. Let $H = (w_0, \ldots, w_{k-1})$ be an arbitrary pattern in c (so $c = C(H)$). Recall that $k' = \lfloor \frac{k-1}{3} \rfloor$. Our algorithm consists of three steps. In step one, for each $t = k - k' - 1$ vertices $v_{i_1}, \ldots, v_{i_t}$, we count the number of vertices u in G such that the subgraph $(u, v_{i_1}, \ldots, v_{i_t})$

in G maps to the subgraph $(w_0, w_{k'+1}, w_{k'+2}, \ldots, w_{k-1})$ in H. In step two, we count the number of k'-tuples $(v_{j_1}, \ldots, v_{j_{k'}})$ such that the subgraph $(v_{j_1}, \ldots, v_{j_{k'}}, v_{i_1}, \ldots, v_{i_t})$ in G maps to the subgraph $(w_1, \ldots, w_{k-1})$ in H. In step three, we show how to combine the numbers obtained in the last two steps to get the resulting value.

Before we explain each step, here is some notation. Let $k_1 = \lceil \frac{k-1}{3} \rceil$ and $k_2 = \lceil \frac{k-2}{3} \rceil$. Note that $k_1, k_2 \in \{k', k'+1\}$ and $k' + k_1 + k_2 = k - 1$. Define the set S to be all t-tuples $p = (v_{i_1}, \ldots, v_{i_t})$ where the subgraph induced by p maps to the subgraph $(w_{k'+1}, \ldots, w_{k-1})$ in H. We can write each t-tuple p with a pair of k_1 and k_2 tuples, p' and p''; i.e. $p' = (v_{i_1}, \ldots, v_{i_{k_1}})$ and $p'' = (v_{i_{k_1+1}}, \ldots, v_{i_t})$

Step one: Construct two matrices B and C of sizes $n^{k_1} \times n$ and $n \times n^{k_2}$ as follows: For each k_1-tuple $p_1 = (v_{i_1}, \ldots, v_{i_{k_1}})$ and each vertex $v_h \in G$, let $B_{p_1, v_h} = 1$ if the subgraph (v_h, p_1) in G maps to the subgraph $(w_0, w_{k'+1}, \ldots, w_{k'+k_1})$ in H. Otherwise set it to 0. For each k_2-tuple $p_2 = (v_{j_1}, \ldots, v_{j_{k_2}})$ and each vertex $v_h \in G$, let $C_{v_h, p_2} = 1$ if the subgraph (v_h, p_2) in G maps to the subgraph $(w_0, w_{k'+k_1+1}, \ldots, w_{k-1})$ in H. Otherwise set it to 0. Compute $M = BC$. For any $p_1 = (v_{i_1}, \ldots, v_{i_{k_1}})$ and $p_2 = (v_{j_1}, \ldots, v_{j_{k_2}})$ such that the t-tuple $(p_1, p_2) \in S$, we have M_{p_1, p_2} is the number of vertices u such that the subgraph (u, p_1, p_2) in G maps to the subgraph $(w_0, w_{k'+1}, \ldots, w_{k-1})$ in H.

Step two: Construct two matrices B' and C' of sizes $n^{k_1} \times n^{k'}$ and $n^{k'} \times n^{k_2}$ as follows: For each k_1-tuple $p_2 = (v_{i_1}, \ldots, v_{i_{k_1}})$ and each k'-tuple $p_1 = (v_{j_1}, \ldots, v_{j_{k'}})$ in G, let $B'_{p_2, p_1} = 1$ if the subgraph (p_1, p_2) in G maps to the subgraph $(w_1, \ldots, w_{k'+k_1})$ in H. Otherwise set it to 0. For each k_2-tuple $p_3 = (v_{h_1}, \ldots, v_{h_{k_2}})$ and each k'-tuple $p_1 = (v_{j_1}, \ldots, v_{j_{k'}})$ in G, let $C'_{p_1, p_3} = 1$ if the subgraph (p_1, p_3) in G maps to the subgraph $(w_1, \ldots, w_{k'}, w_{k'+k_1+1}, \ldots, w_{k-1})$ in H. Otherwise set it to 0. Compute $M' = B'C'$. For any $p_2 = (v_{i_1}, \ldots, v_{i_{k_1}})$ and $p_3 = (v_{h_1}, \ldots, v_{h_{k_2}})$ such that the t-tuple $(p_2, p_3) \in S$, we have M'_{p_1, p_3} is the number of k'-tuples p_1 in G such that the subgraph (p_1, p_2, p_3) in G maps to the subgraph $(w_1, \ldots, w_{k-1})$ in H.

Step three: Let r be the number of vertices w_i in $\{w_1, \ldots, w_{k'}\}$, such that the subgraph $(w_i, w_{k'+1}, \ldots, w_{k-1})$ in H maps to the subgraph $(w_0, w_{k'+1}, \ldots, w_{k-1})$ in H. Compute the fol-

lowing sum using matrices M and M':

$$\sum_{p \in S} (M_{p',p''} - r) M'_{p',p''} \tag{3.3}$$

If $r = 0$, by the way we constructed M and M', each number $M_{p',p''}M'_{p',p''}$ is the number of $k'+1$ tuples $(v_{i_0}, \ldots, v_{i_{k'}})$ such that the subgraph (v_{i_0}, p', p'') in G maps to the subgraph $(w_0, w_{k'+1}, \ldots, w_{k-1})$ in H, and the subgraph $(v_{i_1}, \ldots, v_{i_{k'}}, p', p'')$ in G maps to the subgraph $(w_1, \ldots, w_{k-1})$ in H. So the number in equation (3.3) is the number of subgraphs in G which map to a pattern in c. Now if $r > 0$, then each k'-tuple that is counted in $M'_{p',p''}$ contains exactly r vertices that are also counted in $M_{p',p''}$ and cannot be used simultaneously. So in this case, the number $(M_{p',p''} - r)M'_{p',p''}$ counts the number of $k'+1$ tuples with the property mentioned above.

Now we analyze the running time. M and M' in step one and two can be computed in $O(M(n^{\lfloor \frac{k-1}{3} \rfloor}, n, n^{\lceil \frac{k-2}{3} \rceil}))$ and $O(M(n^{\lfloor \frac{k-1}{3} \rfloor}, n^{\lceil \frac{k-1}{3} \rceil}, n^{\lceil \frac{k-2}{3} \rceil}))$ time, respectively, using rectangular matrix multiplication. By checking all t-tuples of vertices in G in n^t time, we can identify the set S, and then the sum in step three can be computed in $O(|S|) \leq O(n^t)$ time. Note that $O(M(n^{\lfloor \frac{k-1}{3} \rfloor}, n^{\lceil \frac{k-1}{3} \rceil}, n^{\lceil \frac{k-2}{3} \rceil})) \geq n^{\max(\lfloor \frac{k-1}{3} \rfloor + \lceil \frac{k-1}{3} \rceil, \lceil \frac{k-1}{3} \rceil + \lceil \frac{k-2}{3} \rceil)}$ which is the size of the input in rectangular matrix multiplication, and also we have $t = k - 1 - k' \leq \max(\lfloor \frac{k-1}{3} \rfloor + \lceil \frac{k-1}{3} \rceil, \lceil \frac{k-1}{3} \rceil + \lceil \frac{k-2}{3} \rceil)$. So the total the running time is $O(M(n^{\lfloor \frac{k-1}{3} \rfloor}, n^{\lceil \frac{k-1}{3} \rceil}, n^{\lceil \frac{k-2}{3} \rceil}))$.

3.3.5 Patterns easier than cliques

Using the approach of Section 3.3, we show that for any k, there is a pattern that contains a $k-1$-clique and can be detected in $O(C(n, k-1))$ time in an n-node graph G. Since this pattern has a $k-1$-clique as a subgraph, it is at least as hard as $k-1$-clique to detect, which means that the runtime obtained for it is *tight*, if we assume that the best runtime for detecting $k-1$-clique is $O(C(n, k-1))$. Let H^k_s be the k-node pattern consisting of a $(k-1)$-clique and a vertex adjacent to s vertices of the $(k-1)$-clique. Assume that $s \geq \lceil \frac{k-1}{2} \rceil$. If $s \neq k-2$, then $|Aut(H^k_s)| = s!(k-s-1)!$. For $s = k-2$, $|Aut(H^k_{k-2})| = (k-2)!2!$. So in all cases $|Aut(H^k_s)|$ is divisible by $s!(k-s-1)!$.

Theorem 3.3.3. *Let k be any positive integer, and suppose that there exists s, $\lceil \frac{k-1}{2} \rceil \leq s \leq k-1-\lfloor \frac{k-1}{3} \rfloor$, such that $s+1$ is a prime number. Then H_s^k can be detected in $C(n, k-1)$ time with high probability.*

Proof. Let the vertex outside the $(k-1)$-clique in H_s^k be v_0. We know that if $k' = \lfloor \frac{k-1}{3} \rfloor$, there are at least k' vertices that are not adjacent to v_0 because $s \leq k-1-k'$. Let $v_1, \ldots, v_{k'}$ be k' of the vertices of the $(k-1)$-clique that v_0 is not adjacent to. Let $v_{k'+1}, \ldots, v_k$ be the rest of the vertices. Consider the ordering $H = (v_0, v_1, \ldots, v_k)$ of H_s^k, and let $c = C(H)$ be the class defined by H. Note that $U(c)$, which is the set of unlabeled graphs that can be embedded in c, is $\{H_s^k, H_{s+1}^k, \ldots, H_{s+k'}^k\}$. So if we want to detect H_s^k in an n-node graph G, using Theorem 3.3.1 we can count the number of subgraphs in G that map to a pattern in the class c in time $O(C(n, k-1))$. As proved in our set-up (see Equation (3.1)), this number is $Q = \sum_{i=0}^{k'} b_i |Aut(H_{s+i}^k)| n_{H_{s+i}^k}$, where b_i is some integer and $b_0 = 1$ (by an argument similar to Lemma 3.3.2). Since $s \geq (k-1)/2$, we have that $s+1 > k-s-1$, and so $|Aut(H_s^k)|$ is not divisible by $s+1$, which means that the coefficient of $n_{H_s^k}$ in the equation is not divisible by $s+1$. However, for all $i \geq 1$, we have that $|Aut(H_{s+i}^k)|$ is divisible by $s+1$. So Q is of the form (3.2) for $r = s+1$, and hence we can detect H_s^k in time $O(C(n, k-1))$ with high probability. □

Lemma 3.3.3. *For any positive integer $k \geq 3$, $k \neq 14$, there exists s such that $\lceil \frac{k-1}{2} \rceil \leq s \leq k-1-\lfloor \frac{k-1}{3} \rfloor$ and $s+1$ is prime.*

Proof. We are going to use two theorems about prime numbers in intervals. The first one is due to Loo [Loo11] that says for all $n > 1$, there is a prime number in $(3n, 4n)$. The second theorem is due to Nagura [Nag52] and says that for all $x \geq 25$, there is a prime number in $[x, 6x/5]$.

First suppose that $k = 6t + i$ for two nonnegative integers t and i where $0 \leq i \leq 5$ and $i \neq 2$. If $i < 2$, let $n = t$, and otherwise let $n = t+1$. We need a prime in the interval $I = (\lceil \frac{k-1}{2} \rceil, k - \lfloor \frac{k-1}{3} \rfloor + 1)$, and since $\lceil \frac{k-1}{2} \rceil \leq 3n$ and $4n \leq k - \lfloor \frac{k-1}{3} \rfloor + 1$, there exists such a prime by the first theorem. Now assume that $i = 2$. If $t \geq 8$, then $\lceil \frac{k-1}{2} \rceil + 1 \geq 25$, and so if $x = \lceil \frac{k-1}{2} \rceil + 1$, then $6x/5 \leq k - \lfloor \frac{k-1}{3} \rfloor$ and so there is a prime in the interval I by the second theorem. Now suppose that $t \leq 7$ and $i = 2$. For $t = 1, 3, 4, 5, 6, 7$, the prime numbers in the interval I associated to each k are $5, 11, 17, 17, 23, 23$ respectively. □

For $k = 14$, we show that we can detect H_7^k in $O(C(n, k-1))$ time. Note that $k' = 4$ in this case. The approach is the same as Theorem 3.3.3: we look at the class c where $U(c)$ consists of $H_7^k, \ldots, H_{11}^k$ and we consider the equation $Q = \sum_{i=0}^{4} b_i |Aut(H_{7+i}^k)| n_{H_{7+i}^k}$ which can be obtained in $O(C(n, k-1))$ time, where $b_0 = 1$ (by an argument similar to Lemma 3.3.2). Now note that $|Aut(H_{7+i}^k)|$ is divisible by 2^9 for all $0 < i \leq 4$, and $|Aut(H_7^k)|$ is not divisible by 2^9. So Q is of the form (3.2) for $r = 2^9$, and hence we can detect H_7^k in $O(C(n, k-1))$ time, and hence we have the following Theorem.

Theorem 3.3.4. *For all $k > 2$, there is some s where the k-node pattern H_s^k can be detected in $O(C(n, k-1))$ time.*

3.3.6 Induced pattern detection for $k \leq 6$

Note that the case of $k = 4$ is resolved by [WWWY15]. When $k \in \{5, 6\}$, we have $k' = 1$. Consider a class c. By Corollary 3.3.2, $U(c)$ has exactly two patterns which differ in only one edge e, namely $\tilde{H}$ and $\tilde{H} \setminus e$. By Theorem 3.3.2, $b_{\tilde{H}}^c + b_{\tilde{H}\setminus(e)}^c = 2$, so $b_{\tilde{H}}^c = b_{\tilde{H}\setminus(e)}^c = 1$. So for any class c and any unlabeled pattern $\tilde{H}$ that embeds in c, we have $\alpha_{\tilde{H}}^c = |Aut(\tilde{H})|$. Moreover by Lemma 3.3.2, there is some class c such that $U(c)$ consists of $\tilde{H}$ and $\tilde{H} \setminus \{e\}$, where e is an arbitrary edge in $\tilde{H}$.

Hence by Theorem 3.3.1 and Corollary 3.3.1, for any unlabeled pattern $\tilde{H}$ which has at least one edge, we can compute $n_{\tilde{H}}|Aut(\tilde{H})| + n_{\tilde{H}\setminus e}|Aut(\tilde{H} \setminus e)|$ in $O(M(n, n^2, n))$ time for $k = 5$ and $O(M(n, n^2, n^2))$ time for $k = 6$. Now we give an algorithm which detects any fixed pattern $\tilde{H}$ in a graph G, where $\tilde{H}$ is not the k-Clique or the k-Independent Set.

Let $e_1, \ldots, e_h$ be an arbitrary permutation of all the edges of $\tilde{H}$. Let $\tilde{H}_i = \tilde{H} \setminus \{e_1, \ldots, e_{i-1}\}$ where $\tilde{H}_1 = \tilde{H}$. Compute $q_i = n_{\tilde{H}_i}|Aut(\tilde{H}_i)| + n_{\tilde{H}_{i+1}}|Aut(\tilde{H}_{i+1})|$. Compute $Q = \sum_{i=1}^{h}(-1)^i q_i$. In fact, $Q = n_{\tilde{H}}|Aut(\tilde{H})| + (-1)^h n_{\tilde{H}_{h+1}}|Aut(\tilde{H}_{h+1})|$, which is of the form (3.2) for $r = k!$, since $\tilde{H}_{h+1}$ is the k-Independent Set. So we can detect all 5-node patterns in time $O(M(n, n^2, n)) \in O(n^{\omega+1})$, and all 6-node patterns in time $O(M(n, n^2, n^2)) \in O(n^{\omega+2})$.

3.4 Most recent results on this problem

Following our results in this chapter, Manurangsi, Rubinstein and Schramm [MRS21] formulated a brand new hypothesis on the hardness of planted clique. This new hypothesis implies many results that are not known to hold under standard hypotheses such as ETH or Strong ETH, including that for every k-node H, its induced pattern detection problem requires $n^{\Omega(k)}$ time.

Chapter 4

Finding Long Shortest Paths: The Diameter

This chapter was written with authors Ray Li and Virginia Vassilevska Williams, and focuses on computing the diameter of the graph. Approximating the graph diameter is a basic task of both theoretical and practical interest. A simple folklore algorithm can output a 2-approximation to the diameter in linear time by running BFS from an arbitrary vertex. It has been open whether a better approximation is possible in near-linear time. A series of papers on fine-grained complexity have led to strong hardness results for diameter in directed graphs, culminating in a recent tradeoff curve independently discovered by [Li, STOC'21] and [Dalirrooyfard and Wein, STOC'21], showing that under the Strong Exponential Time Hypothesis (SETH), for any integer $k \geq 2$ and $\delta > 0$, a $2 - \frac{1}{k} - \delta$ approximation for diameter in directed m-edge graphs requires $mn^{1+1/(k-1)-o(1)}$ time. In particular, the simple linear time 2-approximation algorithm is optimal for directed graphs.

In this chapter we prove that the same tradeoff lower bound curve is possible for undirected graphs as well, extending results of [Roditty and Vassilevska W., STOC'13], [Li'20] and [Bonnet, ICALP'21] who proved the first few cases of the curve, $k = 2, 3$ and 4, respectively. Our result shows in particular that the simple linear time 2-approximation algorithm is also optimal for undirected graphs. To obtain our result we develop new tools for fine-grained reductions that could be useful for proving SETH-based hardness for other problems in undirected graphs related to distance computation.

4.1 Introduction

One of the most basic graph parameters, the *diameter* is the largest of the shortest paths distances between pairs of vertices in the graph. Estimating the graph diameter is important in many applications (see e.g. [CGLM12, TK11, MLH09]. For instance, the diameter measures how fast information spreads in networks, which is central for paradigms such as distributed and sublinear algorithms.

The fastest known algorithms for computing the diameter of an n-node, m-edge graph with nonnegative edge weights solve All-Pairs Shortest Paths (APSP) and run in $O(\min\{mn + n^2 \log\log n, n^3/exp(\sqrt{\log n})\})$ time [Pet04, Wil14]. For unweighted graphs one can use fast matrix multiplication [Vas12, Le 14, AV21, Sei95, AGM97] and solve the problem in $O(n^{2.373})$ time.

Any algorithm that solves APSP naturally needs n^2 time, just to output the n^2 distances. Meanwhile, the diameter is a single number, and it is apriori unclear why one would need n^2 time, especially in sparse graphs, for which $m \leq n^{1+o(1)}$.

There is a linear time folklore algorithm that is guaranteed to return an estimate $\hat{D}$ for the diameter D so that $D/2 \leq \hat{D} \leq D$, a so called 2-approximation. The algorithm picks an arbitrary vertex and runs BFS from it, returning the largest distance found. The same idea achieves a near-linear time 2-approximation in directed and nonnegatively weighted graphs by replacing BFS with Dijkstra's algorithm to and from the vertex.

Roditty and Vassilevska W. [RV13], following Aingworth, Chekuri, Indyk and Motwani [ACIM99], designed a $3/2$-approximation algorithm running in $\tilde{O}(m\sqrt{n})$ time, for the case when the diameter is divisible by 3, and with an additional small additive error if it is not divisible by 3. Chechik, Larkin, Roditty, Schoenebeck, Tarjan and Vassilevska W. [CLR$^+$14] gave a variant of the algorithm that runs in $\tilde{O}(m^{3/2})$ time and always achieves a $3/2$-approximation (with no additive error). These algorithms work for directed or undirected graphs with nonnegative edge weights.

Cairo, Grossi and Rizzi [CGR16] extended the techniques of [RV13] and developed an approximation scheme that for every integer $k \geq 0$, achieves an "almost" $2 - 1/2^k$-approximation (i.e. it has an extra small additive error, similar to [RV13]) and runs in $\tilde{O}(mn^{1/(k+1)})$ time. The scheme

only works for undirected graphs, however.

These are all the known approximation algorithms for the diameter problem in arbitrary graphs: the scheme of [CGR16, RV13] for undirected graphs, and the three algorithms for directed graphs: the exact $\tilde{O}(mn)$ time algorithm using APSP, the $\tilde{O}(m)$ time 2-approximation and the $3/2$-approximation algorithms of [RV13, CLR$^+$14]. In Figure 1-1 the known algorithms are represented as purple and pink points.

A sequence of works [RV13, BRS$^+$18, Li20, Bon21b, Li21, DW21, Bon21a] provided lower bounds for diameter approximation, based on the Strong Exponential Time Hypothesis (SETH) [IP01a, CIP10] that CNF-SAT on n variables and $O(n)$ clauses requires $2^{n-o(n)}$ time. The first such lower bound by [RV13] showed that any $3/2-\varepsilon$ approximation to the diameter of a directed or undirected unweighted graph for $\varepsilon > 0$, running in $O(m^{2-\delta})$ time for $\delta > 0$, would refute SETH, and hence the [RV13] $3/2$-approximation algorithm has a (conditionally) optimal approximation ratio for a subquadratic time algorithm for diameter. Later, Backurs, Roditty, Segal, Vassilevska W. and Wein [BRS$^+$18] showed that under SETH, any $O(m^{3/2-\delta})$ time algorithm can at best achieve a 1.6-approximation to the diameter of an undirected unweighted graph. Thus, the [RV13] $3/2$-approximation algorithm has a (conditionally) optimal running time for a $(1.6-\varepsilon)$-approximation algorithm.

Following work of Li [Li20] and Bonnet [Bon21b], Li [Li21] and independently Dalirrooyfard and Wein [DW21], provided a scheme of tradeoff lower bounds for diameter in *directed* graphs. They showed that under SETH, for every integer $k \geq 2$, a $(2-1/k-\varepsilon)$-approximation algorithm for $\varepsilon > 0$ for the diameter in m-edge directed graphs, requires at least $m^{1+1/(k-1)-o(1)}$ time. Thus in particular, under SETH, the linear time 2-approximation algorithm for diameter is optimal for directed graphs.

For undirected graphs, however, only three conditional lower bounds are known: the $m^{2-o(1)}$ [RV13] lower bound for $(3/2-\varepsilon)$-approximation, the $m^{3/2-o(1)}$ [Li20] lower bound for $(5/3-\varepsilon)$-approximation, and the $m^{4/3-o(1)}$ [Bon21a] lower bound for $(7/4-\varepsilon)$-approximation (see Figure 1-1 in Chapter 1).

The tradeoff lower bounds for directed diameter of [DW21] and [Li21] crucially exploited

the directions of the edges. One might think that one can simply replace the directed edges with undirected gadgets. However, this does not seem possible. A very high level reason is that the triangle inequality in undirected graphs can be used in both directions. The directed edges in the prior constructions were used to make sure that some pairs of vertices have short paths between them, while leaving the possibility of having large distances between other pairs. If undirected edges (or even gadgets) are used instead however, the triangle inequality implies short paths for pairs of vertices that the construction wants to avoid. A short path from u to v and a short path from x to v does imply a short path from u to x in undirected graphs, but not in directed graphs. This simple reason is basically why no simple extensions of the results of [DW21] and [Li21] to undirected graphs seem to work. (See Section 4.4 for more about this.)

The fact that the triangle inequality can be used in both directions in undirected graphs, makes it difficult to extend the lower bound constructions to undirected graphs, but it also seems to make more algorithmic tradeoffs possible for undirected than for directed graphs, as evident from the Cairo, Grossi, Rizzi [CGR16] algorithms. It thus seems possible that a better than 2 approximation algorithm running in linear time could be possible for undirected graphs.

The main result of this chapter is a delicate construction that achieves the same tradeoff lower bounds for diameter in undirected graphs as the ones in directed graphs, thus showing that undirected diameter is just as hard. Namely:

Theorem 4.1.1. *Assuming SETH, for all integers $k \geq 2$, for all $\varepsilon > 0$, a $(2 - \frac{1}{k} - \varepsilon)$-approximation of Diameter in* unweighted, undirected *graphs on m edges requires $m^{1+1/(k-1)-o(1)}$ time.*

Theorem 4.1.1 was proved previously for $k = 2$ [RV13], $k = 3$ [BRS$^+$18, Li20], and $k = 4$ [Bon21a]. The theorem is stated in terms of the number of edges m; our lower bound constructions are for the special case when $m = n^{1+o(1)}$ (i.e. very sparse graphs).

The main consequence of our theorem is that under SETH, there can be no better near-linear time approximation algorithm for undirected unweighted diameter than the simple 2-approximation algorithm that runs BFS from an arbitrary vertex.

Outline In Section 4.2, we give some preliminaries for our construction. In Section 4.3 we show how to prove Theorem 4.1.1 for small cases $k = 4$ and $k = 5$ to illustrate some of our ideas. We

(re)prove Theorem 4.1.1 for $k = 4$, giving a simplified proof of Bonnet's result, and show how the proof can be modified to give a proof for $k = 5$. The full proof for $k = 5$ is deferred to section 4.6. In Section 4.4, we highlight some of the ideas in the construction. Afterwards, we prove our formal results. In Section 4.5, we prove Theorem 4.1.1 in full generality.

4.2 Preliminaries

For a positive integer a, let $[a] = \{1, 2, \ldots, a\}$.

k-OV. A k-OV instance Φ is a set $A \subseteq \{0,1\}^{\mathbb{d}}$ of n binary vectors of dimension $\mathbb{d} = \theta(\log n)$ and the k-OV problem asks if we can find k vectors $a_1, \ldots, a_k \in A$ such that they are orthogonal, i.e. $a_1 \cdot \ldots \cdot a_k = 0$. The *$k$-OV Hypothesis* says that solving k-OV requires $n^{k-o(1)}$ time, and it is implied by SETH [Wil04, Wil05].

Now we give the definitions that we use for our construction. At a very high level, we are going to start from a k-OV instance and create a diameter instance. To do so, we are going to make a graph where each node is a "configuration", which we are going to define later. Each configuration consists of a number of "stacks", where each stack has some of the vectors of the k-OV instance. There are relationships between different stacks in a configuration, and we define those relationships using "coordinate arrays". Below we define these notions more formally.

Stacks. Given a k-OV instance $A \subset \{0,1\}^{\mathbb{d}}$, we make the following definitions. A *stack* $S = (a_1, \ldots, a_{|S|})$ is a vector of elements of A whose *length* $|S|$ is a nonnegative integer. Denote a_1 as the *bottom* element of the stack and $a_{|S|}$ as the *top* element of the stack. We let $()$ denote the *empty stack*, i.e., a stack with 0 vectors. Given a stack $S = (a_1, \ldots, a_\ell)$, a *substack* $S_{\leq \ell'} = (a_1, \ldots, a_{\ell'})$ is given by the bottom ℓ' vectors of S, where $\ell' \leq \ell$. We call these tuples *stacks*, because of the following operations. The stack $popped(S)$ is the stack $(a_1, \ldots, a_{\ell-1})$, i.e., the stack S with the top element removed. For a vector $b \in A$ and a stack $S = (a_1, \ldots, a_\ell)$, the stack $S + b$ is the stack $S + b = (a_1, \ldots, a_\ell, b)$. The use of stacks as a primitive in our construction is motivated in Section 4.4.

Coordinate arrays.

	$x[1]$	$x[2]$	$x[3]$		$x[1]$	$x[2]$	$x[3]$		$x[1]$	$x[2]$	$x[3]$		$x[1]$	$x[2]$	$x[3]$
a_1	1	1	1	a_1	1	1	1	a_1	1	1	1	a_1	1	1	1
a_2	1	1		a_2		1	1	a_2	1	1		a_2		1	1
a_3	1			a_3		1									

Table 4.1: In each of the above, $x = (x[1], x[2], x[3])$ is a 4-coordinate array. The left two tables depict that stack (a_1, a_2, a_3) satisfies x, and the right two tables depict that stack (a_1, a_2) satisfies x.

Definition 4.2.1. *A* k-coordinate-array x *is an element of* $[\mathbb{d}]^{k-1}$.

In the reduction from k-OV, we only consider k-coordinate arrays, so we omit k when it is understood. For a k-coordinate array $x \in [\mathbb{d}]^{k-1}$ and an integer $\ell \in [k-1]$, let $x[\ell]$ denote the ℓth coordinate in the coordinate array x. Also for a coordinate c and a vector $a \in A$, $a[c]$ is the cth coordinate of a. We index coordinate arrays by $x[\ell]$ and vectors in A by $a[c]$, rather than x_ℓ and a_c (respectively), for clarity. For a set of non-orthogonal vectors $\{a_1, \ldots, a_s\}$ for $s \leq k$, let $ind(a_1, \ldots, a_s)$ return a coordinate c such that $a_i[c] = 1$ for all $i = 1, \ldots, s$.

Definition 4.2.2 (Stacks satisfying coordinate arrays)**.** *Let* $S = (a_1, \ldots, a_s)$ *be a stack where* $|S| \leq k-1$*, and let* $x \in [\mathbb{d}]^{k-1}$ *be a* k*-coordinate array. We say that* S satisfies x *if there exists sets* $[k-1] = I_1 \supset \cdots \supset I_s$ *such that, for all* $h = 1, \ldots, s$*, we have* $|I_h| = k - h$ *and* $a_h[x[i]] = 1$ *for all* $i \in I_h$*.*

Lemma 4.2.1. *If stack* S *satisfies a coordinate array* x*, then any substack of* S *satisfies* x *as well.*

Proof. This follows from the definition of satisfiability. □

Lemma 4.2.2. *Let* $S = (a_1, \ldots, a_{|S|})$ *and* $S' = (b_1, \ldots, b_{|S'|})$ *be stacks, each with at most* $k-1$ *vectors from* A*, the* k*-OV instance, such that any* k *vectors from among* $a_1, \ldots, a_{|S|}, b_1, \ldots, b_{|S'|}$ *are not orthogonal. Then there exists a coordinate array* x *such that* S *and* S' *both satisfy* x*.*

Proof. By Lemma 4.2.1, it suffices to prove this in the case that $|S| = |S'| = k-1$. Then $S = (a_1, \ldots, a_{k-1})$ and $S' = (b_1, \ldots, b_{k-1})$. Let $x[\ell] = ind(a_1, \ldots, a_{k-\ell}, b_1, \ldots, b_\ell)$. Then for all

$h = 1, \ldots, k-1$, we have $a_h[x[\ell]] = 1$ for $\ell \leq k-h$, so S satisfies x with sets $I_h = \{1, \ldots, k-h\}$. Additionally, for all $h = 1, \ldots, k-1$, we have $b_h[x[\ell]] = 1$ for $\ell = h, \ldots, k-1$ so S' satisfies x with sets $I_h = \{h, \ldots, k-1\}$. Hence, both S and S' satisfy x. □

Lemma 4.2.3. *Let $a_1, \ldots, a_k$ be k orthogonal vectors. Suppose that j is an index, x is a coordinate array and $S = (a_1, \ldots, a_j)$ and $S' = (a_k, \ldots, a_{j+1})$ are two stacks. Then stacks S and S' cannot satisfy x simultaneously.*

Proof. Suppose for contradiction that S and S' both satisfy x. Let $[k-1] = I_1 \supset I_2 \supset \cdots \supset I_j$ be the sets showing that stack S satisfies coordinate array x, and let $[k-1] = I_k \supset \cdots \supset I_{j+1}$ be the sets showing that stack S' satisfies coordinate array x. Here, I_j has size $k-j$ and I_{j+1} has size j. We have $|I_j \cap I_{j+1}| = |I_j| + |I_{j+1}| - |I_j \cup I_{j+1}| = k - |I_j \cup I_{j+1}| > 0$. Then $I_1 \cap I_2 \cap \cdots \cap I_k = I_j \cap I_{j+1} \neq \emptyset$, so there exists some $i \in I_1 \cap \cdots \cap I_k$. For this i, we have $a_1[x[i]] = a_2[x[i]] = \cdots = a_k[x[i]] = 1$, so $a_1, \ldots, a_k$ are not orthogonal, a contradiction. Thus, stacks S and S' cannot satisfy x simultaneously. □

4.3 Main theorem for $k = 4$

In this section, we prove Theorem 4.1.1 for $k = 4$. Theorem 4.1.1 for $k = 4$ was previously proven by Bonnet [Bon21a]. Here we present a simpler proof that also illustrates some ideas in our general construction. Furthermore, our construction for $k = 4$ can be easily modified to give a hardness construction that proves Theorem 4.1.1 for $k = 5$. We point out how this can be done in the $k = 4$ construction below. Since the modification is simple, and the proof of correctness is similar but more involved, we defer the full proof of the $k = 5$ construction to section 4.6, which can be read for more intuition for the main construction. For two stacks $S = (a_1, \ldots, a_s)$ and $T = (b_1, \ldots, b_t)$, let $S \circ T$ denote the stack $(a_1, \ldots, a_s, b_1, \ldots, b_t)$.

Theorem 4.3.1. *Assuming SETH, for all $\varepsilon > 0$, a $(\frac{7}{4} - \varepsilon)$-approximation of Diameter in* unweighted, undirected *graphs on m edges needs $m^{4/3-o(1)}$ time.*

Proof. Start with a 4-OV instance Φ given by a set $A \subset \{0,1\}^{\mathtt{d}}$ with $|A| = n_{OV}$ and $\mathtt{d} = \theta(\log n_{OV})$. We show how to solve Φ using an algorithm for Diameter. First check in time $O(n_{OV}^3)$

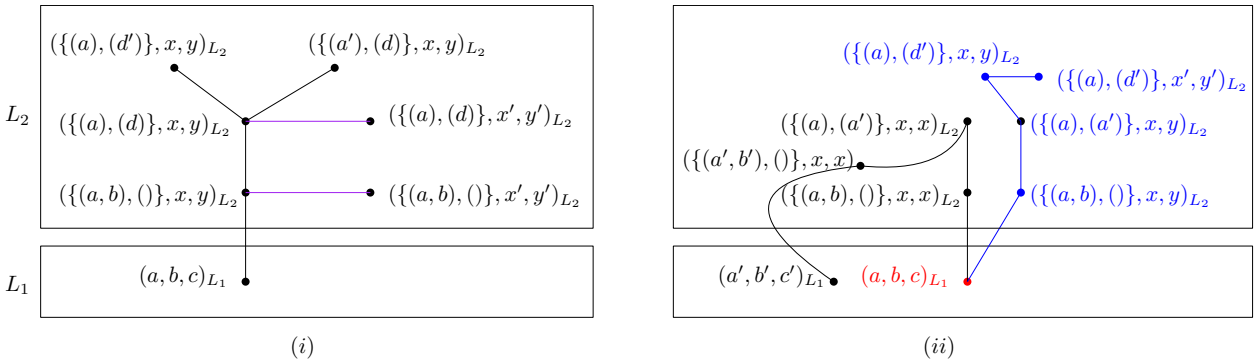

Figure 4-1: (i) 4-OV reduction graph. The purple edges are coordinate change edges. (ii) Paths in the first two cases of the NO case. Black path is for the case where both vertices are in L_1, blue path is for the case where one vertex is in L_1 and the other is in L_2 with two stacks of size 1.

whether there are three orthogonal vectors in A. If so, we know that Φ also has 4 orthogonal vectors, as we can add an arbitrary fourth vector to the triple and obtain a 4-OV solution.

Thus, let us assume that there are no three orthogonal vectors. We construct a graph with $\tilde{O}(n_{OV}^3)$ vertices and edges from the 4-OV instance such that (1) if Φ has no solution, any two vertices are at distance 4, and (2) if Φ has a solution, then there exists two vertices at distance 7. Any $(7/4-\varepsilon)$-approximation for Diameter distinguishes between graphs of diameter 4 and 7. Since solving Φ needs $n_{OV}^{4-o(1)}$ time under SETH, a $7/4-\varepsilon$ approximation of diameter needs $n^{4/3-o(1)}$ time under SETH.

Construction of the graph The graph G is illustrated in Figure 4-1(i) and constructed as follows.

The vertex set $L_1 \cup L_2$ is defined on

$$
\begin{aligned}
L_1 &= \{S : S \text{ is a stack with } |S| = 3\},\\
L_2 &= \big\{(\{S_1,S_2\},x,y) : S_1,S_2 \text{ are stacks with } |S_1|+|S_2| = 2,\\
&\qquad\qquad x,y \in [\mathbb{d}]^3 \text{ are coordinate arrays such that}\\
&\qquad\qquad S_1 \circ S_2 \text{ satisfies } x \text{ and } S_2 \circ S_1 \text{ satisfies } y, \text{ OR}\\
&\qquad\qquad S_1 \circ S_2 \text{ satisfies } y \text{ and } S_2 \circ S_1 \text{ satisfies } x\big\}. \qquad (4.1)
\end{aligned}
$$

In vertex subset L_2, the notation $\{S_1,S_2\}$ denote an *unordered* set of two stacks. As shown in Figure 4-1, the vertices in L_2 are of two types: $(\{(a),(b)\},x,y)_{L_2}$ and $(\{(a,b),()\},x,y)_{L_2}$ for

$a, b \in A$, $x, y \in [d]$.

Throughout, we identify tuples (a, b, c) and $(\{S_1, S_2\}, x, y)$ with vertices of G, and we denote vertices in L_1 and L_2 by $(a, b, c)_{L_1}$ and $(\{S_1, S_2\}, x, y)_{L_2}$ respectively. The (undirected unweighted) edges are the following.

- (L_1 to L_2) Edge between $(S)_{L_1}$ and $(\{\text{popped}(S), ()\}, x, y)_{L_2}$ if stack S satisfies both x and y.

- (vector change in L_2, type 1) For some vector $a \in A$ and stacks S_1, S_2 with $|S_1| \geq 1$, an edge between $(\{S_1, S_2\}, x, y)_{L_2}$ and $(\{\text{popped}(S_1), S_2 + a\}, x, y)_{L_2}$ if both vertices exist.

  In particular, as Figure 4-1 shows, $(\{(a, b), ()\}, x, y)_{L_2}$ has an edge to $(\{(a), (b')\}, x, y)_{L_2}$ if both vertices exist. These are the only type 1 edges.

- (vector change in L_2, type 2) For some vector $a \in A$ and stacks S_1, S_2 with $|S_1| \geq 1$, an edge between $(\{S_1, S_2\}, x, y)_{L_2}$ and $(\{\text{popped}(S_1) + a, S_2\}, x, y)_{L_2}$ if both vertices exist.

  In particular, as Figure 4-1 shows, $(\{(a), (b)\}, x, y)_{L_2}$ has edges to $(\{(a'), (b)\}, x, y)_{L_2}$ and $(\{(a), (b')\}, x, y)_{L_2}$ if the vertices exist. These are the only type 2 edges.

- (coordinate change in L_2) Edge between $(\{S_1, S_2\}, x, y)_{L_2}$ and $(\{S_1, S_2\}, x', y')_{L_2}$ if both vertices exist.

There are at most n_{OV}^3 vertices in L_1 and at most $n_{OV}^2 \mathbb{d}^6$ vertices in L_2. Note that each vertex of L_1 has $O(\mathbb{d}^2)$ neighbors, each vertex of L_2 has $O(n_{OV} + \mathbb{d}^2)$ neighbors. The total number of edges and vertices is thus $O(n_{OV}^3 \mathbb{d}^2) = \tilde{O}(n_{OV}^3)$. We first show below how to change this construction for $k = 5$, and then we show that the construction for $k = 4$ has diameter 4 when Φ has no solution and diameter at least 7 when Φ has a solution.

Modifications for $k = 5$. The construction of the Diameter instance G when $k = 5$ is very similar. We instead start with a 5-OV (rather than 4-OV) instance $A \subset \{0, 1\}^d$, and use the exact same graph, except the nodes in L_1 have a stack of size 4 (rather than 3), and the total sizes of the stacks in L_2 is 3 (rather than 2). The descriptions of the edges are exactly the same. We defer

the proof of correctness of this construction for $k = 5$ to section 4.6. It is similar to the proof for $k = 4$, but is more involved.

4-OV no solution Assume that the 4-OV instance $A \subset \{0,1\}^{\mathrm{d}}$ has no solution, so that no four (or three or two) vectors are orthogonal. We show that any pair of vertices have distance at most 4, by casework:

- **Both vertices are in L_1:** Let the vertices be $(a,b,c)_{L_1}$ and $(a',b',c')_{L_1}$. By Lemma 4.2.2 there exists coordinate array x satisfied by both stacks (a,b,c) and (a',b',c'). We claim the following is a valid path (see Figure 4-1ii):

$$(a,b,c)_{L_1} - (\{(a,b),()\},x,x)_{L_2} - (\{(a),(a')\},x,x)_{L_2} - (\{(a',b'),()\},x,x)_{L_2} - (a',b',c')_{L_1} \tag{4.2}$$

  The first edge and second vertex are valid because (a,b,c) satisfies x (and thus, by Lemma 4.2.1, stack (a,b) satisfies x). By the same reasoning the last edge and fourth vertex are valid. The third vertex is valid because each of a and a' have a 1 in all coordinates of x, so both (a,a') and (a',a) satisfy x.

- **One vertex is in L_1 and the other vertex is in L_2 with two stacks of size 1:** Let the vertices be $(a,b,c)_{L_1}$ and $(\{(a'),(d')\},x',y')_{L_2}$. By Lemma 4.2.2, there exists a coordinate array x satisfied by both stacks (a,b,c) and (a',d'), and there exists a coordinate array y satisfied by both stacks (a,b,c) and (d',a'). We claim the following is a valid path (see Figure 4-1ii):

$$\begin{aligned}(a,b,c)_{L_1} &- (\{(a,b),()\},x,y)_{L_2} \\ &- (\{(a),(a')\},x,y)_{L_2} \\ &- (\{(d'),(a')\},x,y)_{L_2} - (\{(a'),(d')\},x',y')_{L_2}.\end{aligned} \tag{4.3}$$

  The first edge and second vertex are valid because (a,b,c) satisfies x and y. Vector a has a one in each coordinate of x and y, and stack (a',d') satisfies x and stack (d',a') satisfies y, so stack (a',a) satisfies x and stack (a,a') satisfies y, so the third vertex $(\{(a),(a')\},x,y)_{L_2}$

is valid, and thus the second edge is also valid. The fourth vertex is valid because (a', d') satisfies x and (d', a') satisfies y by construction of coordinate arrays x and y, and thus the third and fourth edges are valid. Hence, this is a valid path.

- **Both vertices are in L_2 with two stacks of size 1:** Let the vertices be $(\{(a), (d)\}, x, y)_{L_2}$ and $(\{(a'), (d')\}, x', y')_{L_2}$. Let $z_1 \in [\mathbb{d}]$ be a coordinate where a, d, a', d' are all 1, and let $z = (z_1, z_1, z_1)$ be a coordinate array. Then the following is a valid path:

$$\begin{aligned}(\{(a), (d)\}, x, y)_{L_2} &- (\{(a), (d)\}, z, z)_{L_2} \\ &- (\{(a'), (d)\}, z, z)_{L_2} \\ &- (\{(a'), (d')\}, z, z)_{L_2} - (\{(a'), (d')\}, x', y')_{L_2}. \end{aligned} \tag{4.4}$$

  Indeed it's easy to check that any stack of two of a, d, a', d' satisfies z, so all the vertices are valid and thus all the edges are valid, so this is a valid path.

- **One vertex is in L_2 with two stacks of size 2 and 0:** For every vertex $u = (\{(a, b), ()\}, x, y)_{L_2}$ in L_2 with two stacks of size 2 and 0, any vertex of the form $v = (a, b, c)_{L_1}$ in L_1 has the property that the neighborhood of u is a superset of the neighborhood of v (by considering coordinate change edges from u). Thus, any vertex that v can reach in 4 edges can also be reached by u in 4 edges. In particular, since any two vertices in L_1 are at distance at most 4, any vertex in L_1 is distance at most 4 from any vertex in L_2 with two stacks of size 2 and 0. Applying a similar reasoning, any vertex in L_2 with two stacks of size 2 and 0 is distance at most 4 from any vertex in L_2 with two stacks of size 2 and 0, and any vertex in L_2 with two stacks of size 1.

We have thus shown that any two vertices are at distance at most 4, proving the diameter is at most 4.

4-OV has solution. Now assume that the 4-OV instance has a solution. That is, assume there exists $a_1, a_2, a_3, a_4 \in A$ such that $a_1[i] \cdot a_2[i] \cdot a_3[i] \cdot a_4[i] = 0$ for all i. Since there are no 3 orthogonal vectors, vectors a_1, a_2, a_3, a_4 are pairwise distinct.

Suppose for contradiction there exists a path of length at most 6 from $u_0 = (a_1, a_2, a_3)_{L_1}$ to $u_6 = (a_4, a_3, a_2)_{L_1}$.

Since all vertices in L_2 have self-loops with trivial coordinate-change edges, we may assume the path has length exactly 6. Let the path be $u_0 = (a_1, a_2, a_3)_{L_1}, u_1, \ldots, u_6 = (a_4, a_3, a_2)_{L_1}$. We may assume the path never visits L_1 except at the ends: if $u_i = (S) \in L_1$, then $u_{i-1} = (\{\text{popped}(S), ()\}, x, y)$ and $u_{i+1} = (\{\text{popped}(S), ()\}, x', y')$ are in L_2, and in particular u_{i-1} and u_{i+1} are adjacent by a coordinate change edge, so we can replace the path $u_{i-1} - u_i - u_{i+1}$ with $u_{i-1} - u_{i+1} - u_{i+1}$, where the last edge is a self-loop.

For $i = 1, 2, 3$, let p_i denote the largest index such that vertices $u_0, u_1, \ldots, u_{p_i}$ all contain a stack that has $(a_1, \ldots, a_i)$ as a substack. Because we never revisit L_1, we have $p_3 = 0$. For $i = 1, 2, 3$, let q_i be the smallest index such that vertices $u_{q_i}, \ldots, u_6$ all contain a stack with $(a_4, \ldots, a_{5-i})$ as a substack. Because we never revisit L_1, we have $q_3 = 6$. We show that,

Claim 1. *For $i = 1, 2, 3$, between vertices u_{p_i} and $u_{q_{4-i}}$, there must be a coordinate change edge.*

Proof. Suppose for contradiction there is no coordinate change edge between u_{p_i} and $u_{q_{4-i}}$. We show a contradiction for each of $i = 1, 2, 3$.

First, consider $i = 3$. Here, $u_{p_i} = u_0 = (a_1, a_2, a_3)_{L_1}$. By minimality of q_1, vertex u_{q_1} is of the form $(\{(e), (a_4)\}, x, y)_{L_2}$ for some vector e. Then stack (a_4), satisfies one of x and y. Since there is no coordinate change edge between u_0 and u_{q_1}, we must have $u_1 = (\{(a_1, a_2), ()\}, x, y)$ for the same coordinate arrays x and y, so stack (a_1, a_2, a_3) satisfies both x and y. Hence, there is some coordinate array satisfied by both (a_1, a_2, a_3) and (a_4), which is a contradiction of Lemma 4.2.3. By a similar argument, we obtain a contradiction if $i = 1$.

Now suppose $i = 2$. By maximality of p_2, vertex u_{p_2} is of the form $(\{(a_1, a_2), ()\}, x, y)_{L_2}$. By minimality of q_2, vertex u_{q_2} is of the form $(\{(a_4, a_3), ()\}, x, y)_{L_2}$. The coordinate arrays x and y are the same between the two vertices because there is no coordinate change edge between them by assumption. Then stacks (a_1, a_2) and (a_4, a_3) satisfy both coordinate arrays x and y, which contradicts Lemma 4.2.3. □

Since coordinate change edges do not change any vectors, by maximality of p_i, the edge $u_{p_i}u_{p_i+1}$ cannot be a coordinate change edge for all $i = 1, 2, 3$. Similarly, by minimality of q_i,

the edge $u_{q_i-1}u_{q_i}$ cannot be a coordinate change edge for all $i = 1, 2, 3$. Consider the set of edges

$$u_{p_3}u_{p_3+1}, u_{p_2}u_{p_2+1}, u_{p_1}u_{p_1+1}, u_{q_1-1}u_{q_1}, u_{q_2-1}u_{q_2}, u_{q_3-1}u_{q_3}. \tag{4.5}$$

By the above, none of these edges are coordinate change edges. These edges are among the 6 edges $u_0u_1, \ldots, u_5u_6$. Additionally, the edges $u_{p_i}u_{p_i+1}$ for $i = 1, 2, 3$ are pairwise distinct, and the edges $u_{q_i-1}u_{q_i}$ for $i = 1, 2, 3$ are pairwise distinct. Edge $u_{p_3}u_{p_3+1}$ cannot be any of $u_{q_i-1}u_{q_i}$ for $i = 1, 2, 3$, because we assume our orthogonal vectors a_1, a_2, a_3, a_4 are pairwise distinct and $u_{p_3-1} = u_1$ does not have any stack containing vector a_4. Similarly, $u_{q_3-1}u_{q_3}$ cannot be any of $u_{p_i}u_{p_i+1}$ for $i = 1, 2, 3$. Thus, the edges in (4.10) have at least 4 distinct edges, so our path has at most 2 coordinate change edges. By Claim 1, there must be at least one coordinate change edge. We now do casework on the number of coordinate change edges.

Case 1: the path $u_0, \ldots, u_6$ has one coordinate change edge. By Claim 1, since vertex $u_{p_3} = u_0$ is before the coordinate change edge, edge $u_{q_1-1}u_{q_1}$ must be after the coordinate change edge, and similarly edge $u_{p_1}u_{p_1+1}$ must be before the coordinate change edge. Then all of the edges in (4.5) are pairwise distinct, so then the path has 6 edges from (4.5) plus a coordinate change edge, for a total of 7 edges, a contradiction.

Case 2: the path has two coordinate change edges. Again, by Claim 1, for $i = 1, 2, 3$, edges $u_{q_i-1}u_{q_i}$ must be after the first coordinate change edge, and edge $u_{p_i}u_{p_i+1}$ must be before the second coordinate change edge. Since we have 6 edges total, we have at most 4 distinct edges from (4.5), so there must be at least two pairs (i, j) such that the edges $u_{p_i}u_{p_i+1}$ and $u_{q_j-1}u_{q_j}$ are equal. By above this edge must be between the two coordinate change edges, so edges $u_{p_2}u_{p_2+1}, u_{p_1}u_{p_1+1}, u_{q_2-1}u_{q_2}, u_{q_1-1}u_{q_1}$ are all between the two coordinate change edges. However, this means that vertices u_{p_2} and u_{q_2} are between the two coordinate change edges, contradicting Claim 1.

This shows that $(a_1, a_2, a_3)_{L_1}$ and $(a_4, a_3, a_2)_{L_1}$ are at distance at least 7, completing the proof. □

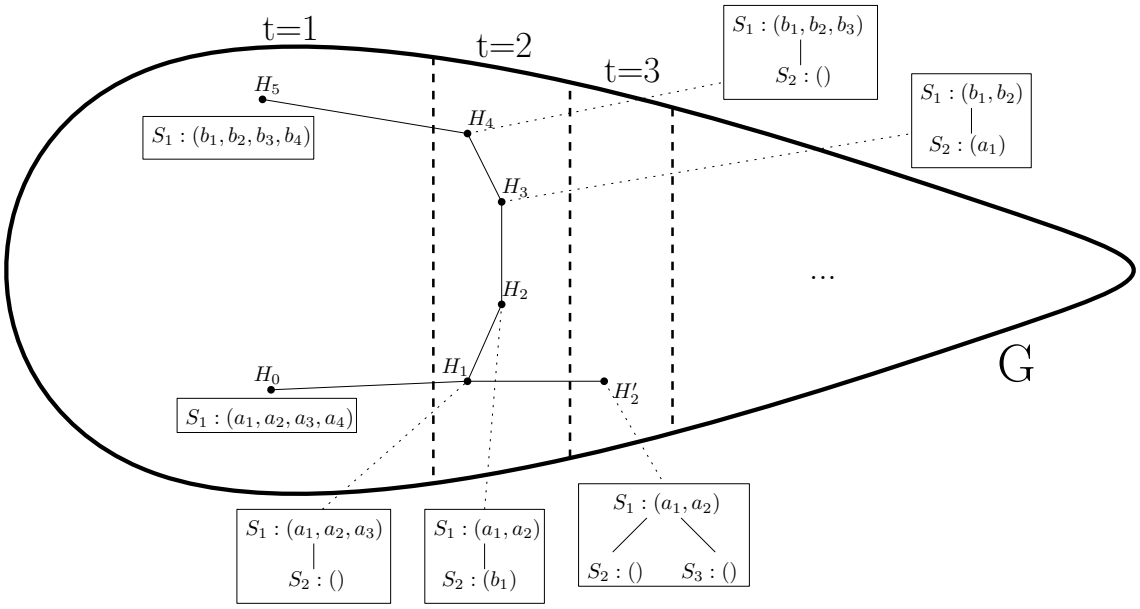

Figure 4-2: Our Diameter instance G, illustrated for $k = 5$. Vertices are *configurations* and edges are *operations* on configurations. Edges within configurations hold coordinate arrays (suppressed in the figure).

4.4 Overview of the general k reduction

4.4.1 The basic setup

To prove Theorem 4.1.1 in general, we start with a k-OV instance $A \subset \{0,1\}^{\mathtt{d}}$ with size $|A| = n_{OV}$ and dimension $\mathtt{d} \leq O(\log n_{OV})$, and construct a graph G on $\tilde{O}(n_{OV}^{k-1})$ vertices and edges such that, if the set A has k orthogonal vectors (Yes case), the diameter of G is at least $2k-1$, and otherwise (No case) the diameter of G is at most k. Throughout, we refer to elements of A as *vectors* and elements of $[\mathtt{d}]$ as *coordinates*. Each vertex of G is identified by a *configuration* H, which contains vectors (in A) and coordinates (in $[\mathtt{d}]$), along with some meta-data. Vertices must be *valid* configurations H, meaning vectors of H have 1s in specified coordinates of H. Edges between configurations of G change up to one vector and/or some coordinates, and we think of edges as performing *operations* on configurations. We ensure the graph is undirected by choosing operations that are invertible.

4.4.2 The Diameter instance construction

We now sketch the definitions of configurations and operations, which define the vertices and edges, respectively, of the Diameter instance G. Figure 4-2 illustrates our graph G and some vertices and edges.

Configurations. A *configuration* H is identified by the following:

1. A positive integer t and a sequence of t lists of vectors $S_1, \ldots, S_t$, which we call *stacks*. Stack S_1 is special and is called the *root*, and we require S_1 to have at least $(k-2)/2$ vectors.

2. A collection of $O(k^2)$ elements of $[\mathbb{d}]^{k-1}$, which we call *coordinate arrays*, which are each tagged with one or two of the stacks $S_1, \ldots, S_t$ (here, we omit the details of this tagging).

The *size* of a configuration is $t + \sum_{i=1}^{t} |S_i|$, the number of stacks plus the number of vectors. The vertices of our Diameter instance G correspond to the valid (defined below) size-k configurations (see Figure 4-2).[1]

Valid configurations. A configuration is *valid* if every coordinate array is *satisfied* (defined in Definition 4.2.2) by its one or two tagged stacks. This technical notion of "stacks satisfying coordinate arrays", implicit in prior constructions [BRS$^+$18, Bon21b, DW21, Li21, Bon21a], has two key properties.

1. (Lemma 4.2.2, used in No case) If every k vectors among the vectors of stacks S and S' are not orthogonal, S and S' satisfy a common coordinate array.

2. (Lemma 4.2.3, used in Yes case) If stacks $(a_1, \ldots, a_j)$ and $(a_k, \ldots, a_{j+1})$ satisfy a common coordinate array, then $a_1, \ldots, a_k$ are not orthogonal.

Operations. Operations (Figure 4-3) are composed of half-operations, which are one of the following.

[1] Prior lower bounds [BRS$^+$18, Bon21b, DW21, Li21, Bon21a] resemble this construction but with only $t \leq 2$ stacks. Handling more than two stacks is nontrivial but seems necessary for our undirected, general-k result.

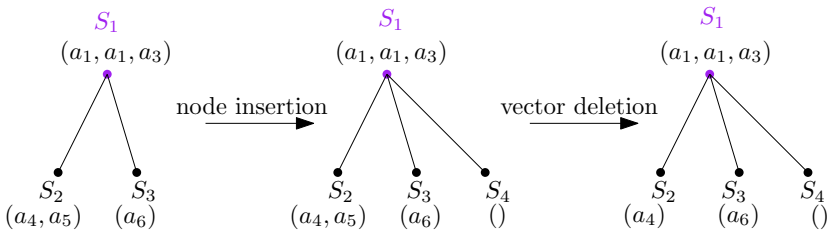

Figure 4-3: (Full) Operation on configuration of size $k = 7$. Root stack S_1 is in purple. Coordinate arrays (suppressed in figure) are attached to edges.

1. (Vector insertion) Insert a vector at the end of a stack.

2. (Vector deletion) Delete the last vector of a stack.

3. (Node[2] insertion) Insert an empty non-root stack.

4. (Node deletion) Delete an empty non-root stack.[3]

5. (Flip) If $t = 2$, switch the two stacks S_1 and S_2, making S_2 the new root.

Note, vectors are inserted and deleted "First In Last Out", hence the term "stack" (see **Why stacks?**).

During node insertion and deletions, we also insert and delete, respectively, coordinate arrays from the configuration. Specifying how to do this is a significant challenge. At a high level, we associate with each configuration a star graph[4] having vertices $S_1, \ldots, S_t$ and edges S_1S_i for $i = 2, \ldots, t$ (hence S_1 is called the root, see Figures 4-2 and 4-3). We attach each coordinate array to an edge (the edge's endpoints may be different from the coordinate array's tagged stack(s)), and insert and delete coordinate arrays when their associated edge is inserted or deleted.

A *(full) operation* consists of two half-operations: a vector insertion or node insertion followed by a vector deletion or node deletion. We also allow operations to include a flip operation after

[2]We say "Node insertion" instead of "stack insertion" because in the actual construction, we place the stacks at nodes of a graph.

[3]In the formal construction, we require that the deleted stack is either S_{t-1} or S_t, and require an analogous condition for node insertions. The proof holds without this requirement, but it is a notational convenience in the proof of Lemma 4.5.3.

[4]We emphasize there are now two types of graphs: the Diameter instance, and the star graphs of each configuration.

the half-operations. To ensure at most $\tilde{O}(n_{OV}^{k-1})$ edges, we do not allow operations between two configurations with one stack ($t = 1$). An operation is *valid* if the starting and ending configuration are valid.[5] The Diameter instance G has the edge (H, H') if applying a valid operation to H gives H'.

Basic properties. We check a few basic properties of the construction.

- Operations leave the size $t + \sum_{i=1}^{t} |S_i|$ of a configuration invariant, so the edges are well-defined. (this is why we defined size as $t + \sum_{i=1}^{t} |S_i|$.)
- Since the Diameter instance deals with size k configurations, each configuration has at most $k - 1$ vectors, so there are at most $\tilde{O}(n_{OV}^{k-1})$ vertices. Similarly, one can check that there are $\tilde{O}(n_{OV}^{k-1})$ edges, and that the graph can be constructed in $\tilde{O}(n_{OV}^{k-1})$ time.
- Operations are invertible, so the graph is undirected. For example, a vector insertion/node deletion can be inverted by a node insertion/vector deletion.

Why stacks? That is, why are vectors inserted "First In Last Out" from stacks? Crucially, stacks ensure that $k - 1$ operations are needed to delete the bottom vector of a configuration with one stack. As in prior constructions, the Yes case shows that if $a_1, \ldots, a_k$ are orthogonal, the one-stack configurations H and H' with stacks $(a_1, \ldots, a_{k-1})$ and $(a_k, \ldots, a_2)$ are at distance $2k - 1$. If we could delete a_1 from H in less than $k - 1$ operations, we could arrive in $k - 2$ operations at a configuration H'' such that any k vectors among H'' and H' are not orthogonal. Then H'' and H' are distance k by the No case, so $d(H, H') \leq d(H, H'') + d(H'', H') \leq 2k - 2$ by the triangle inequality, a contradiction.

4.4.3 Correctness

We now sketch why G has diameter at most k in the No case and at least $2k-1$ in the Yes case.

No case. Suppose any k vectors are not orthogonal. We want to show we can reach any configuration H' from any other configuration H with k valid operations. If the operations do not need to

[5] We also require validity of intermediate configurations after one of the two half-operations. In the Yes case, this gives an extra +2, proving the diameter is $2k - 1$, rather than $2k - 3$.

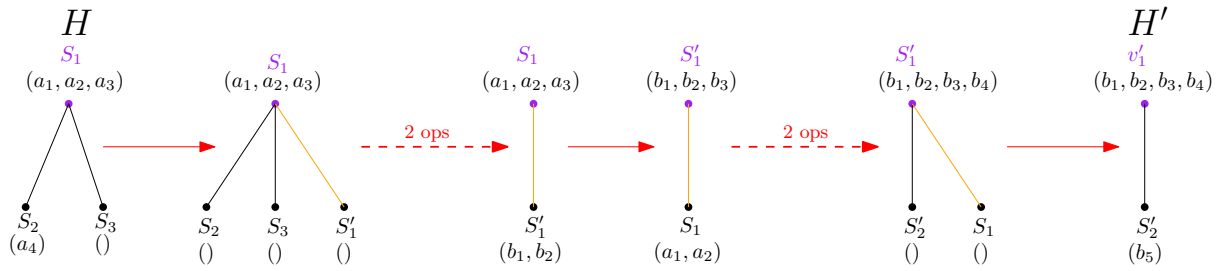

Figure 4-4: No-case path between configurations H and H' for $k = 7$. We delete all non-root stacks of H before inserting any non-root stacks of H'. Orange edges hold auxiliary coordinate arrays not belonging to H or H'.

be valid, this is easy: insert the nodes and vectors of H' while deleting the vectors and nodes from H. We need k deletions to remove H (because it has size k), and k insertions to build H', so we pair the insertions and deletions to get from H to H' in k full operations.

Since these operations may not all be valid, we must carefully choose the order of the insertions and deletions. The root stack is key in choosing the path. Let S_1 and S'_1 be the root stacks of H and H'. Because S_1 and S'_1 each have at least $(k-2)/2$ vectors (by definition, and crucially), we can choose a path from H to H' that first deletes all the non-root stacks of H while only adding stack S'_1 and its vectors (see Figure 4-4). Then when S'_1 has at least $(k-2)/2$ vectors, we apply a flip operation, so that S'_1 is the new root, and we build the remainder of H' while deleting stack S_1.[6]

Roughly, this path works because all coordinate arrays tagged with both a stack in H and a stack in H' are "auxiliary", belonging to neither H nor H'; they are attached to $S_1 S'_1$, the orange edges in Figure 4-4. This requirement is necessary, as H and H' are generic configurations, so stacks of H may not satisfy any coordinate array of H' and vice-versa. Furthermore, Lemma 4.2.2 and non-orthogonality let us choose these auxiliary coordinate arrays to always be satisfied by their tagged stacks, making the path valid.

[6] By viewing a path $H_1, H_2, \ldots$ as a sequence of operations on H_1, we can naturally identify stacks and coordinates across different configurations in the path, talking about, for example, a stack S_1 of H_1 being in H_i. For this overview, this informality suffices. To avoid ambiguity in the formal proof, we give stacks a label that does not change between operations (and contract pairs of configurations that are equivalent up to permuting labels).

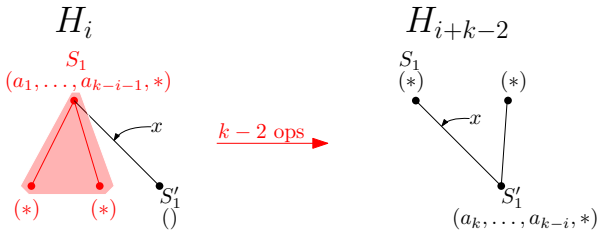

Figure 4-5: The Yes case. We find a coordinate array x satisfied by stack $(a_1, \ldots, a_{k-i-1})$ in some configuration and satisfied by stack $(a_k, \ldots, a_{k-i})$ in another configuration, contradicting Lemma 4.2.3. Here, coordinate array x is both attached to edge S_1S_1' (so it is inserted and deleted with the edge) and tagged with stacks S_1 and S_1' (so stacks S_1 and S_1' need to satisfy x). The *s represents some (possibly zero) vectors.

Yes case. Suppose that there are k orthogonal vectors $a_1, \ldots, a_k$. We sketch why our graph G has diameter at least $2k-3$. The formal proof shows the diameter is at least $2k-1$ (see footnote 5).

Let H_0 be the configuration with one stack $S_1 = (a_1, \ldots, a_{k-1})$, and let H_{2k-4} be the configuration with one stack $S_1' = (a_k, \ldots, a_2)$. Suppose for contradiction there is a path $H_0, H_1, \ldots, H_{2k-4}$. At the highest level, we find two stacks $(a_k, \ldots, a_{j+1})$ and $(a_1, \ldots, a_j)$ from intermediate configurations satisfying a common coordinate array, contradicting Lemma 4.2.3.

Let i be the smallest index such that configurations $H_i, H_{i+1}, \ldots, H_{2k-4}$ all contain stack S_1'. It is easy to check that $i \leq k-3$ so H_i also contains stack S_1. For this sketch, assume that stacks S_1 and S_1' are adjacent in the configuration H_i's star graph.[7] Since this star graph is always a tree, and valid operations can only delete leaf nodes, stack S_1 can only be deleted by deleting all of H_i minus stack S_1' (The red stacks/vectors in Figure 4-5), which takes $k-1$ operations (Lemma 4.5.4). Thus, configurations $H_i, \ldots, H_{i+k-2}$ all have stacks S_1 and S_1' and the edge between them.

Our construction guarantees a coordinate array x attached to edge S_1S_1' that is satisfied by S_1 and S_1'. Hence, x is satisfied by S_1 and S_1' in each of $H_i, \ldots, H_{i+k-2}$. In H_i, stack S_1 must have a prefix of $(a_1, \ldots, a_{(k-1)-i})$, which thus satisfies x. [8] In H_{i+k-2}, stack S_1' must have a prefix of $(a_k, \ldots, a_{(2k-4)-(i+k-2)+2})$, which also satisfies x. Hence stacks $(a_1, \ldots, a_{k-i-1})$ and

[7] There are two other cases: S_1 and S_1' are the same stack, and S_1 and S_1' are nonadjacent stacks in the star graph. The first case is easy, and the nonadjacent case is similar but more technical, depending on the details of tagging coordinate arrays with stacks.

[8] If a stack satisfies coordinate array, its prefixes (substacks) also satisfy that coordinate array (Lemma 4.2.1).

$(a_k, \ldots, a_{k-i})$ satisfy a common coordinate array, contradicting Lemma 4.2.3.

4.5 The main theorem for general k

We describe below a reduction from k-OV to $2k-1$ vs. k Diameter with time $O(n^{k/(k-1)})$ on graphs with edges of weight 1 or 0. This immediately gives a reduction from k-OV to $2k-1$ vs. k Diameter with time $O(n^{k/(k-1)})$ on unweighted graphs, by contracting the edges of weight 0. For clarity of exposition, we describe the reduction to the 0/1-weighted graph.

Throughout the construction, fix $k' = \lfloor k/2 \rfloor + 1$. Throughout the construction, all coordinate arrays are k-coordinate arrays. Let Φ be a k-OV instance given by a set A of n vectors of length $O(\log n)$. We create a graph G using this instance. First we need a few definitions.

4.5.1 Configurations

Edge constraints. The vertices of our construction are "configurations" which we are going to define formally later. Each configuration is a small graph, in which each vertex is assigned a stack and each edge puts constraints between those stacks. These *edge-constraints* on edges are of the following form. Recall that a coordinate array is an element of $[\mathbb{d}]^{k-1}$.

Definition 4.5.1 (Edge-constraint)**.** *In a graph with an edge connecting vertices v and v', a (v, v')-*edge-constraint *X (or* edge-constraint *when (v, v') is implicit) is a tuple of $2k'+1$ coordinate-arrays: $X_{v,i}$ and $X_{v',i}$ for $i \in [k']$, and X_*.*

We later define how these $2k'+1$ coordinate arrays of a (v, v')-edge-constraint relate with the stacks assigned to v and v', as well as the stacks of other vertices.

Configurations. With these edge-constraints defined, we can now define a configuration.

Definition 4.5.2 (Configuration)**.** *A* configuration *H is an undirected star*[9] *graph H with nodes $V(H)$ labeled by distinct elements of $[2k']$, such that*

1. *The center node, denoted $\rho(H)$, of the star graph H is called the* root *(if the graph has two nodes, either one could be the root),*

[9]Recall a star graph is a tree with a *center* vertex adjacent to all other vertices.

2. *H is equipped with a total order $\prec_H$ on the vertices of H such that the root is the smallest node of $\prec_H$,*

3. *Each node v of H is assigned a stack $S_v(H)$, and*

4. *Each edge (v, v') of H is labeled with an (v, v')-edge constraint $X^{v,v'}$. As graph H is undirected, we equivalently denote $X^{v,v'}$ by $X^{v',v}$.*

Again, we emphasize that there are now two types of graphs, the configuration graph, and the Diameter instance, whose vertices are identified by configuration graphs. We say configuration H is a *t-stack* configuration if H has t vertices. The vertices of our Diameter instance are identified with configurations. We use the following definition to specify how many nodes and vectors are in these configuration. As we specify later, the vertices of our Diameter instance are identified by configurations of size k.

Definition 4.5.3 (Size of a configuration)**.** *The* size *of a configuration H is the integer $\sum_{v \in V(H)}(1 + |S_v(H)|)$.*

Note that the size of a configuration is the number of stacks plus the total number of vectors in all the stacks.

Equivalent configurations. The node labels of a configuration H in $[2k']$ are irrelevant except so that we can reason about operations on configurations (defined later) in a well defined way (see footnote 6). Accordingly, we say two configurations are *equivalent* if, informally, one can be obtained by permuting the node labels of the other. Formally, we have the following definition.

Definition 4.5.4 (Equivalence of configurations)**.** *We say two configurations H and H' are* equivalent *if there is some permutation $\pi : [2k'] \to [2k']$ such that,*

- *Configuration H' contains node $\pi(v)$ for each node v of H, and an edge $(\pi(v), \pi(v'))$ with $(\pi(v), \pi(v'))$-edge constraint $Y^{\pi(v),\pi(v')}$ for each edge (v, v') of H with (v, v')-edge-constraint $X^{v,v'}$, such that $Y^{\pi(v),\pi(v')}_{\pi(v),j} = X^{v,v'}_{v,j}$ and $Y^{\pi(v),\pi(v')}_{\pi(v'),j} = X^{v,v'}_{v',j}$ for all $j \in [k']$, and $Y^{\pi(v),\pi(v')}_{*} = X^{v,v'}_{*}$.*

- *The stacks satisfy $S_{\pi(v)}(\pi(H)) = S_v(H)$ for every node v of H.*
- *The ordering $\prec_{H'}$ on H' has $\pi(v) \prec_{H'} \pi(v')$ if and only if $v \prec_H v'$.*
- *The root $\rho(\pi(H))$ of $\pi(H)$ satisfies $\rho(\pi(H)) = \pi(\rho(H))$.*

In this case, we write $H' = \pi(H)$.

It is easy to check the following fact from Definition 4.5.4. Taking $\pi' = \pi^{-1}$ below verifies that the equivalence in Definition 4.5.4 is indeed an equivalence relation.

Lemma 4.5.1. *For two permutations π and π', we have $\pi(\pi'(H)) = (\pi \circ \pi')(H)$*

Edge-satisfying and valid configurations. For a configuration to be a valid vertex of our diameter instance, the stacks of a configuration need to satisfy particular coordinate arrays in the configuration. We now make precise how we want the coordinate arrays to constrain the stacks. This is the most technical definition in the construction.

Definition 4.5.5 (Edge-satisfying and $\mathcal{X}_v(H)$)**.** *A configuration H with $s \geq 1$ vertices $v_1 \prec_H \cdots \prec_H v_s$ is* edge-satisfying *if and only if for every $i \in [s]$, the stack $S_{v_i}(H)$ satisfies each coordinate array in the following set $\mathcal{X}_{v_i}(H)$ of coordinate arrays.*

1. *For every neighbor v' of v_i, and every index $j \in [k']$, set $\mathcal{X}_{v_i}(H)$ includes the coordinate array $X^{v_i,v'}_{v_i,j}$ and $X^{v_i,v'}_{*}$. Note that either v' or v_i is the root.*
2. *For every $i' > i$, set $\mathcal{X}_{v_i}(H)$ includes the coordinate array $X^{v_{i'},v_1}_{v_{i'},i}$, where recall that v_1 is the root $\rho(H)$. See Figure 4-6.*

We highlight the subtle detail that the edge constraint X^{v_1,v_i} belonging to the edge v_1v_i where $v_1 = \rho(H)$ might hold coordinate arrays constraining the stacks of the nodes other than its endpoints v_1 and v_i. To get more intuition, the coordinate arrays a given stack S_{v_i} needs to satisfy are illustrated in Figure 4-6, and for an edge v_1v_i in configuration H the stacks that must satisfy each coordinate array in X^{v_1,v_i} are illustrated in Table 4.2. Table 4.2 shows that every coordinate array in the edge-constraint X constrains at most two stacks.

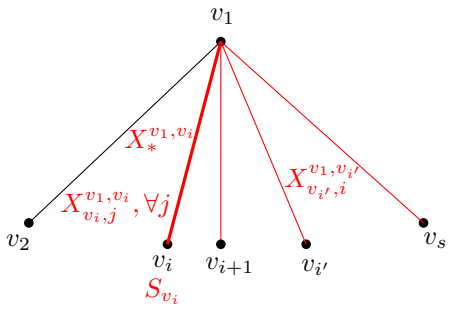

Figure 4-6: The coordinate arrays $\mathcal{X}_{v_i}(H)$ that stack $S_{v_i}(H)$ satisfies, for $i \geq 2$. The relevant edges are colored red, and the coordinate array that is satisfied by $S_{v_i}(H)$ is written on them. the edge v_1v_i is shown in bold since many coordinate arrays on this edge-constraint are satisfied by $S_{v_i}(H)$.

	$*$	1	$\ldots$	j	$\ldots$	$i-1$	i	$\ldots$	k'
$*$	S_{v_1}, S_{v_i}	–	$\ldots$	–	$\ldots$	–	–	$\ldots$	–
v_1	–	S_{v_1}	$\ldots$	S_{v_1}	$\ldots$	S_{v_1}	S_{v_1}	$\ldots$	S_{v_1}
v_i	–	S_{v_1}, S_{v_i}	$\ldots$	S_{v_j}, S_{v_i}	$\ldots$	$S_{v_{i-1}}, S_{v_i}$	S_{v_i}	$\ldots$	S_{v_i}

Table 4.2: Edge satisfying constraints for X^{v_1,v_i} in a configuration H. The entry in row u and column t represent the stacks satisfying $X_{u,t}^{v_1,v_i}$. The entry in row $*$ and column $*$ represent the stacks satisfying $X_*^{v_1,v_i}$. We drop H in $S_u(H)$ for space constraints.

Definition 4.5.6 (Valid configuration)**.** *The configuration H is* valid *if it is edge-satisfying and the stack of the root node satisfies $|S_{\rho(H)}(H)| \geq (k-2)/2$. Here, k is the parameter of our reduction. We use this definition even when the size of configuration H is not k.*

The choice of our global constant k' is motivated by this definition: Since all valid configurations have a stack with at least $(k-2)/2$ vectors, all valid size-k configurations, and hence all configurations at vertices of our Diameter instance, have at most $k - \lceil (k-2)/2 \rceil = k'$ nodes.

Operations on configurations. As mentioned earlier, our final construction consists of configurations. To relate different configurations to each other, we define operations as follows.

Definition 4.5.7 (Operations on configurations)**.** *We define the following* half-operations *on configurations H, that produce a resulting configuration H'.*

1. ***Vector insertion**. H' has the same nodes, root node, edges, stacks, and ordering as H, except that $S_v(H') = S_v(H) + b$ for some vector $b \in A$ and some node v.*

2. ***Vector deletion**. H' has the same nodes, root node, edges, stacks, and ordering as H, except that $S_v(H') = popped(S_v(H))$ for some node v.*

3. ***Node insertion**. H' has the same nodes, root node, edges, stacks as H, except that H' also contains a node v labeled in $[2k'] \setminus V(H)$, assigned with an empty stack $S_v(H') = \emptyset$, and an edge from node v to the root node $\rho(H') = \rho(H)$ with a $(v, \rho(H'))$-edge constraint X, and such that the total order $\prec_{H'}$ is a total order consistent with $\prec_H$ on the nodes of H and the new node v as either the largest or second largest node of $\prec_H$.*[10]

4. ***Node deletion**. H' has the same nodes, root node, edges, stacks as H, except that for some non-root (leaf) node v with $S_v(H) = \emptyset$ that is either the second-largest or largest node of $\prec_H$, H' does not contain node v or the edge incident to it, and the order $\prec_{H'}$ is the order $\prec_H$ restricted to the nodes of H'*

[10]This requirement that the new node v is either the largest or second largest node of $\prec_H$ is not necessary, but makes the rest of the proof, especially Lemma 4.5.3, easier to write. Similarly, for node deletions, the deleted node does not need to be the largest or second-largest node of $\prec_H$.

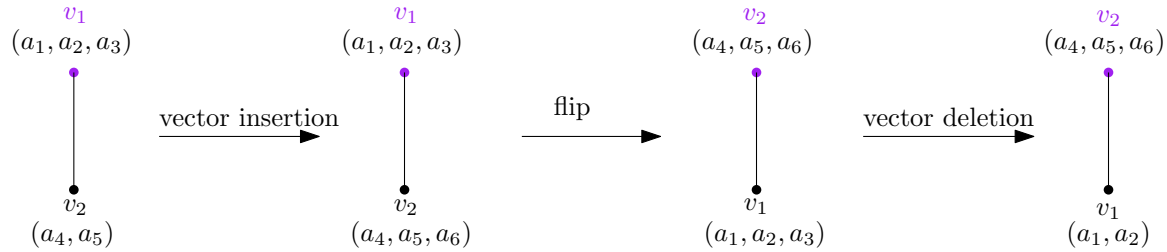

Figure 4-7: Example of a full operation consisting of a vector insertion (in v_2), a flip and a vector deletion (from v_1). We assume that $k = 7$ in this example. Note that when the flip operation happens, the two nodes have the same number of vectors in their stacks. The root in all four configurations is colored purple.

5. ***Flip***. *This half-operation is "only" defined when H has* exactly two *nodes v and v' with $v = \rho(H)$ as the root and $|S_v(H)| = |S_{v'}(H)|$. Then H' has the same nodes, edges, and stacks as H, but $v' = \rho(H')$ is the root of H' and the ordering $\prec_{H'}$ of the nodes of H' is switched accordingly, so that $v' \prec_{H'} v$.*

Call such a half-operation valid *if configurations H and H' are both valid.*

A full operation *is obtained by applying a vector or node insertion, possibly applying a flip (if applicable), and then applying a vector or node deletion. We say a full operation from H to H' is* valid *if each of the two (or three, if there is a flip) participating half-operations is valid, and if at least one of H or H' has at least two nodes.*

By combining a "delete" (node or vector) half operation to an insert (node or vector) half operation, we make sure that the endpoints of a full-operation have the same size. For examples of full operations, see Figure 4-3 and Figure 4-7. Full operations have the following useful properties.

Lemma 4.5.2 (Properties of half and full operations)**.** *Let H and H' be configurations.*

- *If applying a vector insertion to H gives H', it is possible to apply a vector deletion to H' to get H.*
- *If applying a vector deletion to H gives H', it is possible to apply a vector insertion to H' to get H.*
- *If applying a node insertion to H gives H', it is possible to apply node deletion to H' to get H.*

- *If applying a node deletion to H gives H', it is possible to apply node insertion to H' to get H.*

- *Applying two flip operations to a 2-stack configuration gives the same configuration.*

- *If applying a valid full operation to H gives H', it is possible to apply a valid full operation to H' to get H.*

Proof. For the first item, if H' is obtained from a vector insertion at node v in H, then H is obtained by a vector deletion at node v in H. The second, third, and fourth items are similar. For the fifth item, flip operations do not change the 2-node graph, and two flips preserve the root node and the ordering of the two nodes.

For the sixth item, note that the first five items imply that every half-operation has an inverse. If H' is obtained by applying two half-operations to H that give H'' then H', and both half operations are valid, then configurations H, H'', H' are all valid configurations. Then the full operation $H' \to H'' \to H$ is a valid full operation. Similarly if H' is obtained with a valid full operation including a flip, having intermediate configurations $H \to H'' \to H''' \to H'$, then all the intermediate configurations are valid, and $H' \to H''' \to H'' \to H$ is a valid full operation. □

4.5.2 Defining the Diameter graph G

We are now ready to define our graph G. The vertex set of G is the set of valid size-k configurations. Recall that for all size-k configurations, the number of stacks plus the total number of vectors in all stacks is k, and a configuration is valid if it is edge-satisfying (Definition 4.5.5) and the root stack has at least $(k-2)/2$ vectors in it. The edge-set of G includes the following types of edges:

- edges (H, H') such that configuration H can be obtained from configuration H' by a valid full operation. We call these edges *operation edges*. By the last part of Lemma 4.5.2, (H, H') are connected by an operation edge *if and only if* H can be obtained from H' by a valid full operation, so these edges can indeed by undirected.

- weight-0 edges $(H, \pi(H))$ for all valid size-k configurations H and all permutations $\pi : [2k'] \to [2k']$ (recall $\pi(H)$ is defined in Definition 4.5.4). We call these edges *permutation edges*.

- (if k is even) weight-0 edges (H, H') if H' can be obtained by applying a flip to H. We call these edges *flip edges*.

For disambiguation, we always refer to vertices of configurations as *nodes*, and vertices of the Diameter instance G as *vertices* or *configurations*.

Runtime analysis. We first show that the graph G can be constructed in time $O_k(n_{OV}^{k-1}\mathbb{d}^{O(k^2)})$. One can construct the vertices of G by enumerating over all possible star graphs labeled by $[2k']$, of which there are at most $O_k(1)$, and then enumerating over all possible orderings $\prec$ of the nodes of star graphs, of which there are at most $O_k(1)$, and then enumerating over all possible stacks for each star graph, of which there are at most $O_k(n_{OV}^{k-1})$ (each configuration is size-k, meaning the total number of nodes (stacks) plus the total number of vectors equals k, and since there is always at least one node (stack), the total number of vectors is at most $k-1$), and enumerating over all possible edge-constraints, of which there are at most $O_k(\mathbb{d}^{(k'-1)\cdot(2k'+1)}) \leq O_k(\mathbb{d}^{2k^2})$. Hence, there are at most $O_k(n_{OV}^{k-1}\mathbb{d}^{2k^2})$ vertices of G. Furthermore, for $t \geq 2$, there are at most $O_k(n_{OV}^{k-2}\mathbb{d}^{2k^2})$ many t-stack configurations of G.

For any configuration, there are $O_k(n_{OV})$ vector insertions possible, $O_k(1)$ vector deletions possible, $O_k(\mathbb{d}^{2k'+1})$ node insertions possible, and $O_k(1)$ node deletions possible. Hence, each configuration of G has at most $O_k(n_{OV} + \mathbb{d}^{2k'+1})$ neighbors. Every edge of G has at least one endpoint that has $t \geq 2$ stacks (by definition of valid full operation), so the total number of edges of G is at most $O_k(n_{OV} + \mathbb{d}^{2k'+1}) \cdot O_k(n_{OV}^{k-2}\mathbb{d}^{2k^2}) \leq O_k(n_{OV}^{k-1}\mathbb{d}^{4k^2})$.

Hence, G has $\tilde{O}(n_{OV}^{k-1})$ vertices (configurations) and edges. Checking whether any half-operation is valid takes time $O_k(\mathbb{d}) = \tilde{O}_k(1)$. Hence enumeration of vertices (configurations) and edges of the Diameter graph G is standard and can be done in time near-linear in the number of vertices and edges, so the construction of G takes time $\tilde{O}(n_{OV}^{k-1})$.

4.5.3 Some useful properties of configurations

We now move on to proving the correctness of our configurations, showing that the Diameter is at least $2k-1$ when the k-OV instance Φ has a solution (Yes case), and the Diameter is at most k when Φ has no solution (No case). We begin with some useful lemmas about configurations.

Lemma for the No case. In the No case, we need to construct length k paths between every pair of nodes and verify that those paths are valid paths in the Diameter instance. The following natural lemma facilitates these verifications. Call H' a *subconfiguration* of H if H' can be obtained from H by vector deletions and node deletions.

Lemma 4.5.3. *If H' is a subconfiguration of H and H is edge-satisfying, then H' is also edge-satisfying.*

Proof. It suffices to prove that if H' is obtained by applying a single vector deletion or node deletion to H, and H is valid, then H' is valid. The full lemma follows from induction of the number of deletions needed to obtain H' from H. Let H have vertices $v_1 \prec_H \cdots \prec_H v_s$.

Suppose H' is obtained from H by a vector deletion. Then H and H' have the same node set and edge set. Let $i \in [s]$. In the Definition 4.5.5, the set of coordinate arrays $\mathcal{X}_{v_i}(H')$ is the same as the set of coordinate arrays $\mathcal{X}_{v_i}(H)$, because H and H' are the same graph with the same edge-constraints. Since we assume H is edge-satisfying, we have that $S_{v_i}(H)$ satisfies all the coordinate arrays in $\mathcal{X}_{v_i}(H) = \mathcal{X}_{v_i}(H')$, so $S_{v_i}(H')$ does as well, by Lemma 4.2.1. This holds for all $i \in [s]$, so we have that H' is edge-satisfying.

Now suppose H' is obtained from H by a node deletion, so that the graph H' is a subgraph of the graph H with a leaf node deleted. We claim that, for all i such that node v_i is in H', we have $\mathcal{X}_{v_i}(H') \subseteq \mathcal{X}_{v_i}(H)$. First, suppose v_s is deleted from H to give H'. Then, for each $i = 1, \ldots, s-1$, by Definition 4.5.5, the set $\mathcal{X}_{v_i}(H')$ is the same as the set of coordinate arrays $\mathcal{X}_{v_i}(H)$, except with $X_{v_s,i}^{v_s,v_1}$ deleted, and, if v_i is a neighbor of v_s (only true for $i = 1$), with coordinate arrays $X_{v_i,j}^{v_s,v_i}$ deleted for $j \in [k']$, so indeed $\mathcal{X}_{v_i}(H') \subset \mathcal{X}_{v_i}(H)$. Now suppose v_{s-1} is deleted from H to give H'. For $1 \leq i \leq s-2$, we have $\mathcal{X}_{v_i}(H') \subset \mathcal{X}_{v_i}(H)$ by the same reasoning as when v_s is deleted. Additionally, we can show $\mathcal{X}_{v_s}(H') = \mathcal{X}_{v_s}(H)$: nodes v_s and v_{s-1} are not adjacent in H (node

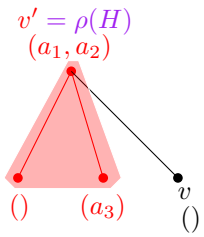

Figure 4-8: Lemma 4.5.4. In the example configuration of size $k = 7$, to delete the root node $v' = \rho(H)$ (purple) without deleting v, one needs to delete all the red vectors and red nodes. This requires 3 node deletions and 3 vectors deletions for a total of $6 = k - 1$ deletions.

deletions can only delete non-root nodes so v_{s-1} is not the root) so all of the coordinate arrays of $\mathcal{X}_{v_s}(H)$ and $\mathcal{X}_{v_s}(H')$ in part 1 of Definition 4.5.5 are the same, and $\mathcal{X}_{v_s}(H)$ and $\mathcal{X}_{v_s}(H')$ have no coordinate arrays in part 2 of Definition 4.5.5 since v_s is the largest node each of $\prec_H$ and $\prec_{H'}$.

Thus, we have that $\mathcal{X}_{v_i}(H') \subseteq \mathcal{X}_{v_i}(H)$ for all nodes v_i in H'. For all nodes v_i in H', we have the stacks $S_{v_i}(H')$ and $S_{v_i}(H)$ are the same, since no vector insertions/deletions were applied. Thus, since stack $S_{v_i}(H)$ satisfies all the coordinate arrays in $\mathcal{X}_{v_i}(H)$, we have $S_{v_i}(H')$ satisfies all the coordinate arrays in $\mathcal{X}_{v_i}(H')$, as desired.

We have shown that if H' is obtained by applying a single vector deletion or node deletion to H, and H is valid, then H' is valid. By the first paragraph of the proof, this completes the proof. □

Lemma for the Yes case. The next lemma is useful for the Yes case.

Lemma 4.5.4. *Suppose H is a size-k configuration containing a non-root leaf node v with $S_v(H) = \emptyset$ and edge (v, v') where $v' = \rho(H)$. Suppose that one applies c full operations among which node v' is deleted but node v is never deleted. Then $c \geq k - 1$.*

Proof. Let $H_0 = H, H_1, \ldots, H_c$ be the sequence of configurations such that H_i is the result of applying a valid full operation to H_{i-1} for $i = 1, \ldots, c$. Let $v'' \notin \{v, v'\}$ be an arbitrary node in H. We claim that node v'' must be deleted before node v'. Let $i \in \{0, \ldots, c\}$ be the largest index such that v'' and v are both in configuration H_i. Node v' is on the path from node v'' to node v in configuration graph H_0. Only leaf nodes can be deleted in a node deletion. Thus, as v and v'' are both in $H_0, \ldots, H_i$, all the nodes on the path from v to v'' are also nodes in $H_0, \ldots, H_i$. In

particular, node v' is in $H_0, \ldots, H_i$, so node v'' must be deleted before v'.

Hence, the only way to delete node v' without deleting node v is to first delete all nodes other than v' (by first deleting the vectors in their stacks and then the node) and then deleting v'. This results in deleting all nodes other than v, which takes at least $\sum_{u \in V(H)}(1 + |S_u(H)|) - (1 + |S_v(H)|) = k - (1 + 0)$ deletions. Since each full operation applies at most one deletion, the number of full operations needed to delete v' without deleting v is at least $k - 1$. □

Permutations commute with valid full operations The next few lemmas justify the informal statement that "permutations commute with valid full operations". This statement is convenient in the Yes case because it allows us to assume that all permutation edges are at the end of a path. Intuitively, we expect this lemma to be true because changing the node labels of a configuration gives essentially the same configuration.

Lemma 4.5.5. *Let $\pi : [2k'] \to [2k']$ be a permutation. Let H be a configuration, and suppose that applying a vector insertion (vector deletion, node insertion, node deletion, flip) on H gives configuration H'. Then there exists a vector insertion (vector deletion, node insertion, node deletion, flip) that, applied to $\pi(H)$, gives $\pi(H')$.*

Proof. A vector $b \in A$ is inserted at node v in H (v is a node label in $[2k']$) to give a configuration H'. Suppose that inserting vector b at node $\pi(v)$ in $\pi(H)$ gives a configuration H''. We claim $H'' = \pi(H')$. By definition of vector insertion, H'' has the same nodes, edges, edge-constraints, root node, and ordering as configuration $\pi(H)$. Furthermore, since H has the same nodes, edges, edge-constraints, root node, and ordering as configuration H', we have $\pi(H)$ and $\pi(H')$, and thus H'' and $\pi(H')$ have the nodes, edges, edge-constraints, root node, and ordering. Furthermore, the stacks $S_{\pi(v)}(H'')$ and $S_{\pi(v)}(\pi(H'))$ are both equal to $S_v(H) + b$, and the stacks $S_{\pi(v')}(H'')$ and $S_{\pi(v')}(\pi(H'))$ are both equal to $S_{v'}(H)$ for nodes $v' \neq v$ in H, so we indeed have $H'' = \pi(H')$.

This proves the lemma for vector insertions. The proofs for vector deletions, node insertions, node deletions, and flips are similar. □

Lemma 4.5.6. *Let $\pi : [2k'] \to [2k']$ be a permutation. If configuration H is valid, then configuration $\pi(H)$ is valid.*

Proof. The root node $\rho(\pi(H))$ of $\pi(H)$ has the same stack as the root node $\rho(H)$ of H, which has at least $(k-2)/2$ vectors. By definition of $\pi(H)$, for each node $v \in V(H)$, the set of coordinate arrays $\mathcal{X}_{\pi(v)}(H)$ is the same as $\mathcal{X}_v(H)$. Since H is valid, $S_v(H)$ satisfies every coordinate array in $\mathcal{X}_v(H)$, so $S_{\pi(v)}(\pi(H)) = S_v(H)$ satisfies every coordinate array in $\mathcal{X}_{\pi(v)}(\pi(H)) = \mathcal{X}_v(H)$. This holds for all v, so $\pi(H)$ is edge-satisfying and thus valid. □

As a corollary of Lemmas 4.5.5 and 4.5.6, we have that permutations commute with valid full operations.

Corollary 4.5.1. *Let $\pi : [2k'] \to [2k']$ be a permutation. Let H be a configuration, and suppose that applying some valid full operation on H gives configuration H'. Then applying some valid full operation on $\pi(H)$ gives $\pi(H')$.*

4.5.4 No case.

We now prove that when Φ has no solution, our Diameter instance has diameter at most k. To do so, we find a length k path between any two configurations H and H'. As sketched in the overview, we apply k full operations to get H' from H, and each operation inserts a vector or node "from H'" and deletes a vector or node "from H". For an example of such a path when $k = 7$, see Figure 4-9.

Let H be an arbitrary size-k configuration with vertices $v_1 \prec_H \cdots \prec_H v_s$ for some $s \geq 1$, where $v_1 = \rho(H)$ is the root, and with edges $v_1 v_i$ with edge-constraint X^{v_1,v_i} for $2 \leq i \leq s$. Let H' be an arbitrary size-k configuration with vertices $v'_1 \prec_{H'} \cdots \prec_{H'} v'_{s'}$ for some $s' \geq 1$, where $v'_1 = \rho(H')$ is the root, and with edges $v'_1 v'_i$ with edge-constraint $Y^{v'_1,v'_i}$ for $2 \leq i \leq s$. By taking a permutation edge (of weight 0) from vertex H' in the Diameter instance G to obtain an equivalent configuration, we may assume without loss of generality that the set of node labels $\{v_1, \ldots, v_s\}$ of H are disjoint from the node labels $\{v'_1, \ldots, v'_{s'}\}$ of H'.

We now define an edge-constraint Z, containing the only "extra" coordinate arrays we need in the path from H to H'. Let Z be a (v_1, v'_1)-edge constraint such that,

- For $i \in [k']$, coordinate array $Z_{v_1,i}$ is satisfied by stack $S_{v_1}(H)$ and, if $i \leq s'$, by stack $S_{v'_i}(H')$,

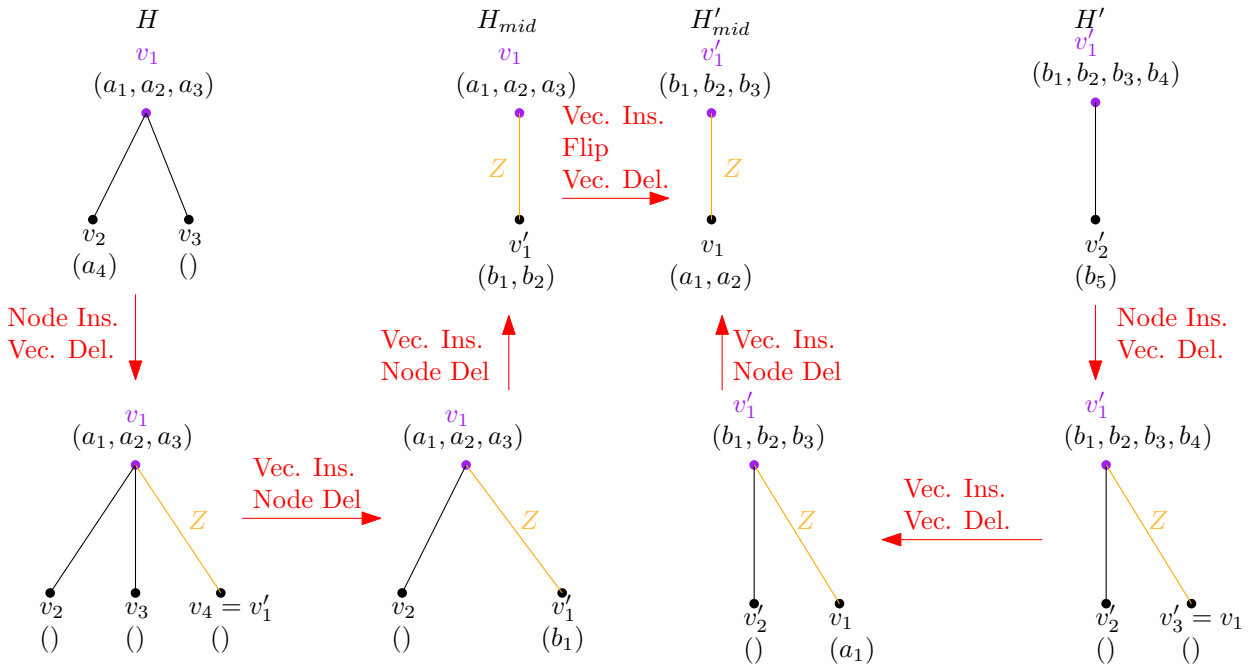

Figure 4-9: The path of length 7 between H and H' for $k = 7$. Full operations are indicated by red arrows and roots are indicated by purple. The "extra" edge-constraint Z that belongs to neither H nor H' is labeled in orange.

- For $i \in [k']$, coordinate array $Z_{v'_1,i}$ is satisfied by stack $S_{v'_1}(H')$ and, if $i \leq s$, by stack $S_{v_i}(H)$, and
- Z_* is satisfied by $S_{v_1}(H)$ and $S_{v'_1}(H')$.

As configurations H and H' are size-k and have at least 1 stack, any stack of H or H' has at most $k-1$ vectors. Hence, the coordinate arrays of Z all exist by Lemma 4.2.2. Note that the definition of Z is symmetric with respect to H and H', in the sense that if we switch H with H' (and s with s' and $(v_1, \ldots, v_s)$ with $(v'_1, \ldots, v'_s)$), the definition of Z stays the same.

We now define two intermediate nodes H_{mid} and H'_{mid}, which are on our desired path from H to H'. Let H_{mid} be the configuration with nodes v_1 and v'_1, with the connecting edge having (v_1, v'_1)-edge constraint Z, where

- $v_1 = \rho(H_{mid})$ is the root,
- $S_{v_1}(H_{mid})$ is the bottom $\lceil (k-2)/2 \rceil$ elements of $S_{v_1}(H)$, and

- $S_{v_1'}(H_{mid})$ is the bottom $\lfloor (k-2)/2 \rfloor$ elements of $S_{v_1'}(H')$.

Let H'_{mid} be the configuration with nodes v_1 and v_1', with the connecting edge having (v_1, v_1')-edge constraint Z, where

- $v_1' = \rho(H'_{mid})$ is the root,
- $S_{v_1'}(H'_{mid})$ is the bottom $\lceil (k-2)/2 \rceil$ elements of $S_{v_1'}(H')$, and
- $S_{v_1}(H'_{mid})$ is the bottom $\lfloor (k-2)/2 \rfloor$ elements of $S_{v_1}(H)$.

We have that H_{mid} and H'_{mid} are valid: by the definition of the edge-constraint Z, we have that $S_{v_1}(H)$ and thus $S_{v_1}(H_{mid})$ satisfies $Z_{v_1,j}$ for all $j \in [k']$, and also satisfies coordinate array $Z_{v_1',1}$ and Z_*. Similarly, $S_{v_1'}(H')$ and thus $S_{v_1'}(H_{mid})$ satisfies $Z_{v_1',j}$ for all $j \in [k']$, and also satisfies coordinate arrays Z_*. Thus, H_{mid} is edge-satisfying and thus valid. By a symmetric argument, H'_{mid} is also valid. Note that H_{mid} and H'_{mid} are symmetric with respect to H and H', in the sense that if we switched H and H', then H_{mid} becomes H'_{mid} and vise-versa.

Claim 2. *One can apply $\lfloor k/2 \rfloor$ valid full operations on H to obtain H_{mid}, and $\lfloor k/2 \rfloor$ valid full operations on H' to obtain H'_{mid}.*

Proof. We prove this for H and H_{mid}, and the result for H' and H'_{mid} follows from a symmetric argument (the symmetry holds because the definition of Z and the definitions of H_{mid} and H'_{mid} are symmetric with respect to H and H'). Let $\tilde{H}$ be the configuration obtained by adding node v_1' to H with stack $S_{v_1'}(H'_{mid})$ (of size $\lfloor (k-2)/2 \rfloor$), with an edge (v_1, v_1') having edge constraint Z, and such that the ordering $\prec_{\tilde{H}}$ agrees with $\prec_H$ on the nodes of H, and v_1' is the largest node of $\prec_{\tilde{H}}$ (see Figure 4-10). Note that $\tilde{H}$ has size larger than k (to be precise, it has size $k + \lfloor k/2 \rfloor$).

We first prove that $\tilde{H}$ is edge-satisfying. First, the set $\mathcal{X}_{v_1'}(\tilde{H})$ has coordinate arrays Z_* and $Z_{v_1',j}$ for $j \in [k']$, by part 1 of Definition 4.5.5, and has no coordinate arrays from part 2 of Definition 4.5.5 as v_1' is the largest node of $\prec_H$. By definition of Z, stack $S_{v_1'}(H')$ satisfies all these coordinate arrays, and thus by Lemma 4.2.2 stack $S_{v_1'}(H_{mid})$ does as well, satisfying the requirement of Definition 4.5.5 for node v_1'. For $i \in [s]$, the set of coordinate arrays in $\mathcal{X}_{v_i}(\tilde{H})$ is the same as the set of coordinate arrays $\mathcal{X}_{v_i}(H)$ plus the coordinate array $Z_{v_1',i}$, and, if $i = 1$, plus

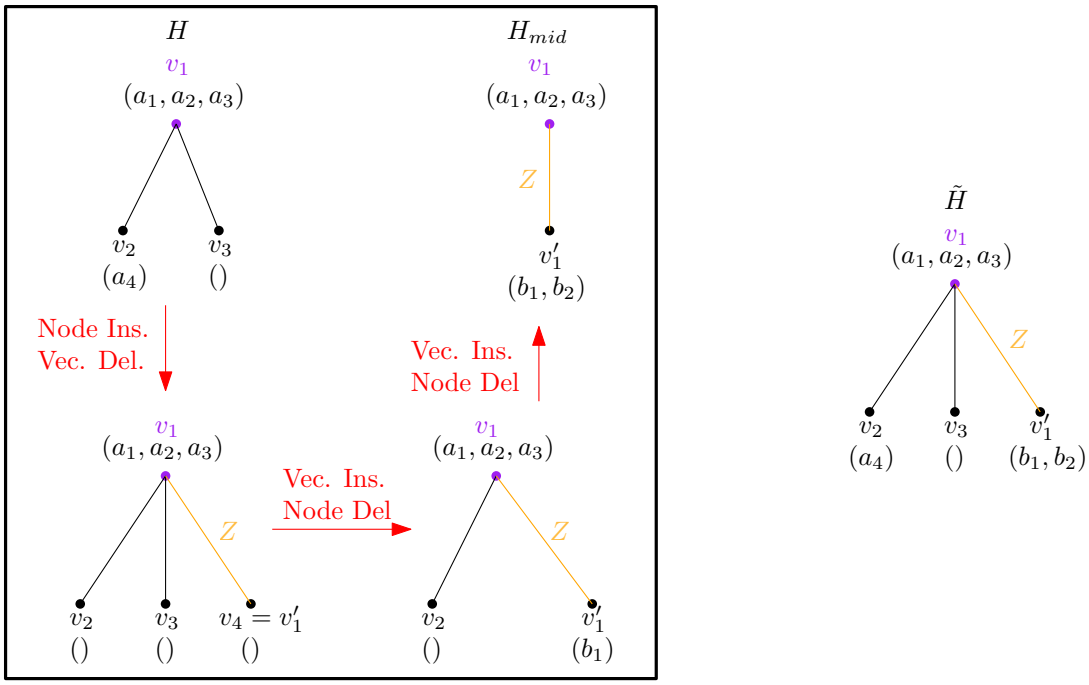

Figure 4-10: Claim 2, the configuration $\tilde{H}$ for Figure 4-9: all configurations on the path from H to H_{mid} are subconfigurations of $\tilde{H}$. By Lemma 4.5.3, showing $\tilde{H}$ is valid implies that the path from H to H_{mid} is valid.

the coordinate arrays Z_* and $Z_{v_1,j}^{v_1',v_1}$ for $j \in [k']$. By definition of Z, we have that $S_{v_i}(\tilde{H}) = S_{v_i}(H)$ satisfies coordinate array $Z_{v_1',i}$. Furthermore, $S_{v_1}(\tilde{H}) = S_{v_1}(H)$ satisfies coordinate arrays Z_* and $Z_{v_1,j}$ for $j \in [k']$. Since configuration H is edge-satisfying and the above coordinate arrays are satisfied, we conclude that configuration $\tilde{H}$ is edge-satisfying.

We now note that H_{mid} can be obtained from H by applying the following half-operations

- Insert node v_1' as the largest node in the ordering
- Insert vectors into v_1' $\lfloor (k-2)/2 \rfloor$ times.
- For each $i = s, s-1, \ldots, 2$, delete vectors from S_{v_i} until the stack is empty, and then delete node v_i.
- Delete vectors from S_{v_1} until the stack has size $\lceil (k-2)/2 \rceil$.

We can check that there are $\lfloor k/2 \rfloor$ insertions and $\sum_{i=1}^{s}(1 + |S_{v_i}|) - (1 + \lceil k-2 \rceil/2) = k - \lceil k/2 \rceil = \lfloor k/2 \rfloor$ deletions. We can obtain H from H_{mid} by alternating applying these insertions

and deletions, giving a configurations $H = H_0, H_{0.5}, H_1, \ldots, H_{\lfloor k/2 \rfloor - 0.5}, H_{\lfloor k/2 \rfloor} = H_{mid}$, so that applying the ith insertion to H_{i-1} gives the size-$k+1$ configuration $H_{i-0.5}$, and applying the ith deletion to $H_{i-0.5}$ gives the size-k configuration H_i. These half-operations indeed satisfy the definition of half-operations: all the vector insertions/deletions are legal, the single node insertion is legal as v'_1 is inserted as the largest node, and all the node deletions are legal as the deleted nodes are always the second-largest node in the ordering. Furthermore, if we perform only the insertions, we obtain configuration $\tilde{H}$. Hence, any $i = 0, 0.5, \ldots, \lfloor k/2 \rfloor$, we can obtain configuration H_i from configuration $\tilde{H}$ by applying vector deletions at node v'_1 until stack $S_{v'_1}$ is the right size, and then applying the first $\lfloor i \rfloor$ node/vector deletions above (at nodes $v_s, v_{s-1}, \ldots$). Thus, for $i = 0, 0.5, \ldots, \lfloor k/2 \rfloor$ configuration H_i is a subconfiguration of configuration $\tilde{H}$. Since configuration $\tilde{H}$ is valid, by Lemma 4.5.3, each H_i and $H_{i+0.5}$ is valid, so we have a sequence of $\lfloor k/2 \rfloor$ valid full operations that gives H_{mid} from H. □

With Claim 2, we have nearly proved the No case. It remains to show that H_{mid} and H'_{mid} are at distance either 0 or 1, depending on the parity of k.

If k is even, then H_{mid} can be obtained by applying a flip to H'_{mid}, and thus the two configurations are at distance 0 in the Diameter graph G. Thus, there is a length $2 \cdot \lfloor k/2 \rfloor = k$ path from H to H' through H_{mid} and H'_{mid} by Claim 2.

If k is odd, then H_{mid} is distance 1 from H'_{mid}: H'_{mid} is obtained from H_{mid} by applying a vector insertion at node v'_1, giving a configuration $H_{mid,+}$, followed by a flip, giving a configuration $H'_{mid,+}$, followed by a vector deletion at node v_1, giving configuration H_{mid}. The flip can be done because $H_{mid,+}$ and $H'_{mid,+}$ both have two nodes, each of which has a stack of size $\lceil (k-2)/2 \rceil$. We now check these half-operations are all valid operations, by checking that configurations $H_{mid,+}$ and $H'_{mid,+}$ are valid configurations. Since no vectors are deleted at node v'_1 from $H_{mid,+}$ to H'_{mid}, we have $S_{v'_1}(H_{mid,+}) = S_{v'_1}(H'_{mid})$ is a substack of $S_{v'_1}(H')$, and similarly $S_{v_1}(H_{mid,+}) = S_{v_1}(H_{mid})$ is a substack of $S_{v_1}(H)$. Hence, by construction of Z and Lemma 4.2.1, stack $S_{v'_1}(H_{mid,+}) = S_{v'_1}(H'_{mid})$ satisfies coordinate array $Z_{v'_1,j}$ for all $j \in [k']$ and also satisfies coordinate array Z_*, and the stack $S_{v_1}(H_{mid,+}) = S_{v_1}(H_{mid})$ at the root node of $H_{mid,+}$ satisfies $Z_{v_1,j}$ for all $j \in [k']$ and also satisfies coordinate arrays $Z_{v'_1,1}$ and Z_*. Hence, configuration $H_{mid,+}$

is edge-satisfying, and thus a valid configuration. By a symmetric argument, configuration $H'_{mid,+}$ is valid. Hence, configurations H_{mid} and H'_{mid} are adjacent in the diameter instance with an edge of weight 1, and we have a path from H to H' through H_{mid} and H'_{mid} of length $1 + 2 \cdot \lfloor k/2 \rfloor = k$ by Claim 2.

In either case, we have shown that, when A has no k orthogonal vectors, then for any two configurations H and H', there is a length k path from H to H'. This completes the proof of the no case.

4.5.5 Yes case.

We now prove that the Diameter of G is at least $2k - 1$ in the Yes case. Suppose A has an orthogonal k-tuple $(a_1, \ldots, a_k)$. Throughout this section fix $v \in [2k']$ to be an arbitrary edge label (say $v = 1$). Let H be the 1-stack configuration with a single node v assigned with a stack $S_v(H) = (a_1, \ldots, a_{k-1})$ (and a trivial ordering). Let H' be the 1-stack configuration with a single node labeled v assigned with a stack $S_v(H') = (a_k, \ldots, a_2)$. We claim configurations H and H' are at distance $2k - 1$ in the Diameter graph G.

Consider a path $H_0 = H, H_1, \ldots, H_{r+1} = H'$ from H to H' using edges of G, and assume for contradiction this path has length $2k - 2$ (if it has length less than $2k - 2$, we may assume without loss of generality that in one of the t-stack vertices for $t \geq 2$, there are trivial valid full operations (e.g., node insertion followed by node deletion), which give self loop edges of weight 1, increasing the path length to $2k - 2$). This path contains some valid full operation edges, possibly some weight-0 flip edges if k is even, and possibly some weight-0 permutation edges between equivalent configurations. By Corollary 4.5.1, we may assume without loss of generality that all weight-0 permutation edges are at the end of the path, and furthermore if there are multiple permutations $\pi_1, \ldots, \pi_\ell : [2k'] \to [2k']$, we may replace them by a single permutation $\pi = \pi_1 \circ \cdots \pi_\ell$ by Lemma 4.5.1. Hence, we may assume that our path has $2k - 2$ valid full operation edges, followed by a single weight-0 edge applying a permutation π.

Thus, we may assume that $r = 2k - 2$, and configuration H' is $\pi(H_{2k-2})$ for some $\pi : [2k'] \to [2k']$, so that configuration H_{2k-2} contains a single stack at node $v' := \pi^{-1}(v)$, and so that for

$i = 1, \ldots, 2k-2$, configuration H_i can be reached from H_{i-1} by an operation edge, and possibly a flip edge. For each $i = 0, \ldots, 2k-3$, the valid full operation on H_i has one valid vector/node insertion, possibly followed by a flip operation, followed by one valid vector/node deletion, possibly followed by a flip, so we can let $H_{i+0.5}$ denote the result of only applying the insertion and possibly a flip to H_i, so that H_{i+1} is the result of applying a deletion, possibly followed by a flip, to $H_{i+0.5}$. By definition of a valid half-operation, configuration $H_{i+0.5}$ is valid (and has size $k+1$).

The following two claims reason about the stacks and the edge-constraints that must be on the path.

Claim 3. *If an edge (w, w') appears in configuration H_i for some integer $i = 1, \ldots, 2k-3$, it also appears (with the corresponding edge-constraint) in configurations $H_{i-0.5}$ and $H_{i+0.5}$.*

Proof. Configuration $H_{i+0.5}$ is obtained by applying a vector or node insertion to H_i, possibly followed by a flip, so no node, and thus no edge is deleted from H_i to $H_{i+0.5}$. Configuration H_i is obtained by applying a vector or node deletion to $H_{i-0.5}$, possibly followed by a flip, so $H_{i-0.5}$ is obtained by possibly applying a flip to H_i, followed by a node or vector insertion, and again no edge is deleted. □

Claim 4. *For $0 \leq s \leq k-1$, we have $S_v(H_s)$ and $S_v(H_{s+0.5})$ both contain $(a_1, \ldots, a_{k-1-s})$ as a substack. For $k-1 \leq s \leq 2k-2$, we have $S_{v'}(H_s)$ and $S_{v'}(H_{s-0.5})$ both contain $(a_k, \ldots, a_{2k-s})$ as a substack.*

Proof. For the first item, we have $S_v(H_0) = (a_1, \ldots, a_{k-1})$, and each of the first s full operations deletes at most one vector from this stack, so stack $S_v(H_s)$ has $(a_1, \ldots, a_{k-1-s})$ as a substack. By Claim 3, $S_v(H_{i+0.5})$ does as well. Similarly, we have stack $S_v(H_{2k-2}) = (a_k, \ldots, a_2)$. Applying $2k-2-s$ full operations from H_{2k-2} gives H_s, but each operation deletes at most one vector from the starting stack $S_{v'}(H_{2k-2}) = (a_k, \ldots, a_2)$. Hence, stack $S_v(H_s)$ has $(a_k, \ldots, a_{2k-s})$ as a substack, and by Claim 3, stack $S_v(H_{s-0.5})$ does as well. □

Let s be the largest index such that node v is in configurations $H_0, \ldots, H_s$ (s exists because H_0 contains node v). Let s' be the smallest index such that node v' is in configurations $H_{s'}, \ldots, H_{2k-2}$ (again s' exists because H_{2k-2} contains node v'). By the maximality of s (and since we assume no

permutation edges are used in $H_0, \ldots, H_{2k-2}$), node v has an empty stack in configuration graph H_s. Node v also has a size $k-1$ stack in H_0. Since each valid full operation can delete at most one vector from some stack, we have that $s \geq k-1$. Similarly, we have that $s' \leq k-1$, so $s' \leq s$. Thus, nodes v and v' both appear in each of the configurations $H_{s'}, \ldots, H_s$. We have three cases, and in each case, we show that our path contradicts Lemma 4.2.3.

Case 1. $v = v'$**.** This implies that $s' = 0$ and $s = r$, and node v appears in every configuration $H_0, \ldots, H_{2k-2}$. We have that the stack $S_v(H_0) = (a_1, \ldots, a_{k-1})$, and $S_v(H_{2k-2}) = (a_k, \ldots, a_2)$. Thus, to obtain $S_v(H_r)$ from $S_v(H_0)$, one needs to apply $k-1$ vector deletions followed by $k-1$ vector insertions. Since each valid full operation applies at most one vector insertion followed by at most one vector deletion, the first $k-1$ full operations of our path must include a vector deletion at node v, and the last $k-1$ edges must include a vector insertion at node v, inserting the vectors $a_k, \ldots, a_2$ in that order. In particular, we have $S_v(H_k) = (a_k)$.

Because valid full operations must have one endpoint with at least two nodes, H_0 to H_1 operation must include a node insertion of some node $w \neq v$ with an edge (v, w). By Claim 3 the edge (v, w) with an edge constraint $X^{v,w}$ appears in configuration $H_{0.5}$, so stack $S_v(H_{0.5}) = (a_1, \ldots, a_{k-1})$ satisfies coordinate array $X^{v,w}_*$. Furthermore, since there are no node-deletions in the first $k-1$ valid full operations (because each full operation deletes either a vector or node, not both), we know the edge (v, w) exists in each of $H_1, \ldots, H_{k-1}$. By Claim 3 the edge (v, w) labeled with the edge constraint $X^{v,w}$ also exist in configuration $H_{k-0.5}$. Additionally, as we reasoned earlier, $S_v(H_{k-0.5}) = (a_k)$, so stack (a_k) satisfies coordinate array $X^{v,w}_*$. However, this means that stacks $(a_1, \ldots, a_{k-1})$ and (a_k) both satisfy $X^{v,w}_*$, which is a contradiction of Lemma 4.2.3.

Case 2. $v \neq v'$ **and nodes** v **and** v' **are adjacent in configuration** $H_{s'}$**.** Clearly we have $s' \geq 1$ and $s \leq 2k-3$ in this case. In configuration $H_{s'}$, node v' is a non-root leaf node with an empty stack $S_{v'}(H_{s'}) = \emptyset$ and incident edge (v, v'). Furthermore, from configuration $H_{s'}$ to H_{s+1}, node v is deleted, but node v' is in configurations $H_{s'}, \ldots, H_{s+1}$. Hence, by Lemma 4.5.4, we have $(s+1) - s' \geq k-1$.

By Claim 3, both configurations $H_{s'-0.5}$ and $H_{s+0.5}$ contain the edge (v, v') with edge-constraint $X^{v,v'}$. By Claim 4, in configuration $H_{s'-0.5}$, node v is labeled with a stack that contains

$(a_1, \ldots, a_{k-s'})$ as a substack, so by Lemma 4.2.1, stack $(a_1, \ldots, a_{k-s'})$ satisfies coordinate array $X_*^{v,v'}$. Similarly, by Claim 4, in configuration $H_{s+0.5}$, node v' is labeled with a stack that contains $(a_k, \ldots, a_{2k-1-s})$ as a substack, so stack $(a_k, \ldots, a_{2k-1-s})$ satisfies coordinate array $X_*^{v,v'}$. Since $k - s' \geq (2k-1-s)-1$, we have that, for $j = k - s'$, both stacks $(a_1, \ldots, a_j)$ and $(a_k, \ldots, a_{j+1})$ satisfies coordinate array $X_*^{v,v'}$, which is a contradiction by Lemma 4.2.3.

Case 3. $v \neq v'$ and nodes v and v' are not adjacent in configuration $H_{s'}$. In any configuration, the root node is adjacent to all other vertices, so v and v' must both be non-root nodes. Suppose that in configuration $H_{s'}$, the root node is $w = \rho(H_{s'})$. Since only leaf nodes in a configuration can be deleted, and since nodes v and v' are not deleted in $H_{s'}, \ldots, H_s$, we have that node w exists and has degree at least two in each of $H_{s'}, \ldots, H_s$, and therefore must be the root node in each of $H_{s'}, \ldots, H_s$. In particular, since the total order $\prec_H$ and root node of a configuration H can only be changed when there are at most two vertices, no full operations from $H_{s'}$ to H_s include flip operations. Consequently, nodes v and v' have the same order with respect to orderings $\prec_{H_{s'}}$ and $\prec_{H_s}$

Assume without loss of generality that $v \prec_{H_{s'}} v'$ and $v \prec_{H_s} v'$ (the reverse direction is symmetric). Let t' be the largest index such that node w is in configuration $H_{t'}$ ($t' \leq 2k-3$ because configuration H_{2k-2} only contains node v'). By maximality of t', from configuration $H_{t'}$ to $H_{t'+1}$, node w is deleted, so by Lemma 4.5.4, $t' - s' \geq k-2$. By Claim 3, both v and v' are in $H_{s'-0.5}$. Let i_v be such that v is the i_vth smallest node in configuration $H_{s'-0.5}$ according to $\prec_{H_{s'-0.5}}$. Because $v \prec_{H_{s'-0.5}} v'$, and since configuration $H_{s'-0.5}$ is valid, Definition 4.5.5 gives that stack $S_v(H_{s'-0.5})$ satisfies coordinate array $X_{v',i_v}^{v',w}$. By Claim 4, $(a_1, \ldots, a_{k-s'})$ is a substack of $S_v(H_{s'-0.5})$, so by Lemma 4.2.1, stack $(a_1, \ldots, a_{k-s'})$ also satisfies coordinate array $X_{v',i_v}^{v',w}$. On the other hand, by Claim 4, $(a_k, \ldots, a_{2k-1-t'})$ is a substack of $S_v(H_{t'+0.5})$. Additionally, by Claim 3, edge (v', w) is also in $H_{t'+0.5}$, so stack $S_v(H_{t'+0.5})$, and thus stack $(a_k, \ldots, a_{2k-1-t'})$, satisfies coordinate array $X_{v',i_v}^{v',w}$. Since $k - s' \geq (2k-1-t)-1$, we have that for $j = k - s'$, stacks $(a_1, \ldots, a_j)$ and $(a_k, \ldots, a_{j+1})$ satisfy the same coordinate array $X_{v',i_v}^{v',w}$, which is a contradiction by Lemma 4.2.3.

In all cases of v and v', we have shown a contradiction. Thus, the path from configuration H to configuration H' in the Diameter instance G cannot have length $2k-2$. Thus, when A has k

orthogonal vectors, the Diameter of G is at least $2k-1$. This completes the proof.

4.6 Main theorem for $k = 5$

In this section, we prove Theorem 4.1.1 (again) for $k = 5$. This proof shows how the $k = 4$ proof in Section 4.3 can be easily modified to give a hardness reduction for $k = 5$. We include this proof because it is simpler than the $k = 5$ instantiation of the general-k proof in Section 4.5, so it may help to reader gain intuition for the general construction. To avoid confusion, we highlight the main differences between the proof in this section and the general proof specialized to $k = 5$.

- In the general proof specialized to $k = 5$, vertices have up to three stacks. In this proof, vertices have up to two stacks. This difference is the main simplification.

- To make this simplification work, we include "coordinate change edges" (as in the $k = 4$ proof). By contrast, the general proof does not have such edges.

- To make this simplification work, we also let coordinate arrays constrain stacks differently. In the general construction, if a coordinate array x constrains two stacks S and S', that means both S and S' satisfy x. Here, we only require $S \circ S'$ or $S' \circ S$ to satisfy x.

Theorem 4.6.1. *Assuming SETH, for all $\varepsilon > 0$ a $(\frac{9}{5}-\varepsilon)$-approximation of Diameter in* unweighted, undirected *graphs on n vertices needs $n^{5/4-o(1)}$ time.*

Proof. Start with a 5-OV instance Φ given by a set $A \subset \{0,1\}^{\mathbb{d}}$ with $|A| = n_{OV}$ and $\mathbb{d} = c \log n_{OV}$. We can check in time n_{OV}^4 where there are 4 orthogonal vectors in A, if so, we know Φ has 5 orthogonal vectors, so assume otherwise. We construct a graph with $\tilde{O}(n_{OV}^4)$ vertices and edges from the 5-OV instance such that (1) if Φ has no solution, any two vertices are at distance 5, and (2) if Φ has a solution, then there exists two vertices at distance 9. Any $(9/5-\varepsilon)$-approximation for Diameter distinguishes between graphs of diameter 5 and 9. Since solving Φ needs $n_{OV}^{5-o(1)}$ time under SETH, a $9/5-\varepsilon$ approximation of diameter needs $n^{5/4-o(1)}$ time under SETH.

Construction of the graph The vertex set $L_1 \cup L_2$ is defined on

$$\begin{aligned} L_1 &= \{(a,b,c,d) \in A^4\}, \\ L_2 &= \big\{(\{S_1,S_2\},x,y) : S_1,S_2 \text{ are stacks with } |S_1|+|S_2| = 3, \\ &\qquad\qquad x,y \in [\mathbb{d}]^3 \text{ are coordinate arrays such that} \\ &\qquad\qquad S_1 \circ S_2 \text{ satisfies } x \text{ and } S_2 \circ S_1 \text{ satisfies } y, \text{ OR} \\ &\qquad\qquad S_1 \circ S_2 \text{ satisfies } y \text{ and } S_2 \circ S_1 \text{ satisfies } x\big\} \end{aligned} \tag{4.6}$$

Throughout, we identify tuples (a,b,c,d) and $(\{S_1,S_2\},x,y)$ with vertices of G, and we denote vertices in L_1 and L_2 by $(a,b,c)_{L_1}$ and $(\{S_1,S_2\},x,y)_{L_2}$ respectively. The (undirected unweighted) edges are the following.

- (L_1 to L_2) Edge between $(a,b,c,d)_{L_1}$ and $(\{(a,b,c),()\},x,y)_{L_2}$ if stack (a,b,c,d) satisfies both x and y.
- (vector change in L_2) For some vector $a \in A$ and stacks S_1, S_2 with $|S_1| \geq 1$, an edge between $(\{S_1,S_2\},x,y)_{L_2}$ and $(\{\text{popped}(S_1), S_2 + a\},x,y)_{L_2}$ if both vertices exist.
- (vector change in L_2, part 2) For some vector $a \in A$ and stacks S_1, S_2 with $|S_1| \geq 1$, an edge between $(\{S_1,S_2\},x,y)_{L_2}$ and $(\{\text{popped}(S_1) + a, S_2\},x,y)_{L_2}$ if both vertices exist.
- (coordinate change in L_2) Edge between $(\{S_1,S_2\},x,y)_{L_2}$ and $(\{S_1,S_2\},x',y')_{L_2}$ if both vertices exist.

There are n_{OV}^4 vertices in L_1 and at most $n_{OV}^3 \mathbb{d}^8$ vertices in L_2. Note that each vertex of L_1 has $O(\mathbb{d}^8)$ neighbors, each vertex of L_2 has $O(n_{OV} + \mathbb{d})$ neighbors. The total number of edges and vertices, and thus the construction time, is $O(n_{OV}^4 \mathbb{d}^8) = \tilde{O}(n_{OV}^4)$. We now show that this construction has diameter 5 when Φ has no solution and diameter at least 9 when Φ has a solution.

5-OV no solution Assume that the 5-OV instance $A \subset \{0,1\}^{\mathbb{d}}$ has no solution, so that no five (or four or three or two) vectors are orthogonal. We begin with the following lemma:

Lemma 4.6.1. *If stacks (a, b) and (a') satisfy x, then (a, b, a') and (a', a, b) satisfy x. If stacks (a, b) and (a', b') satisfy coordinate array x, then the stack (a, b, b') satisfies coordinate array x. If stacks (e', a', b') and (a) satisfy coordinate array x, then stack (a, a', b') satisfies coordinate array x.*

Proof. For the first item, (a, b, a') satisfies x because (a, b) satisfies x and a' is 1 in every coordinate of x. Similarly, (a', a, b) satisfies x because a and a' are 1 in every coordinate of x, and b is 1 in at least 3 coordinates of x.

For the second item, since (a, b) and (a', b') satisfy x, we have $a[x[i]] = 1$ for $i \in [4]$, and there exists $I_2, J_2 \subset [4]$ of size 3 such that $b[x[i]] = 1$ for $i \in I_2$ and $b'[x[i]] = 1$ for $i \in J_2$. We have $|I_2 \cap J_2| = |I_2| + |J_2| - |I_2 \cup J_2| \geq 3 + 3 - 4 = 2$. Thus, $I_1 \supset I_2 \supset (I_2 \cap J_2)$ certifies that (a, b, b') satisfies x.

For the third item, because stack (e', a', b') satisfies x, there exists $[4] = I_1 \supset I_2 \supset I_3$ with $a'[x[i]] = 1$ for $i \in I_2$ and $b'[x[i]] = 1$ for $i \in I_3$. Since $a[x[i]] = 1$ for all $i \in [4]$, we thus have $I_1 \supset I_2 \supset I_3$ certifies that (a, a', b') satisfies x. □

We show that any pair of vertices have distance at most 4, by casework on which of L_1, L_2 the two vertices are in.

- **Both vertices are in L_1:** Let the vertices be $(a, b, c, d)_{L_1}$ and $(a', b', c', d')_{L_1}$. By Lemma 4.2.2 there exists coordinate array x satisfied by both stacks (a, b, c, d) and (a', b', c', d'). Then

$$\begin{aligned}(a, b, c, d)_{L_1} &- (\{(), (a, b, c)\}, x, x)_{L_2} \\ &- (\{(a, b), (a')\}, x, x)_{L_2} \\ &- (\{(a), (a', b')\}, x, x)_{L_2} \\ &- (\{(), (a', b', c')\}, x, x)_{L_2} - (a', b', c', d')_{L_1} \end{aligned} \tag{4.7}$$

  is a valid path. Indeed, the first edge and second vertex are valid because (a, b, c, d) satisfies x (and thus, by Lemma 4.2.1, stack (a, b, c) satisfies x). By the same reasoning the last edge

and fifth vertex are valid. The third vertex is valid because (a) and (a', b') both satisfy x and thus both (a, a', b') and (a', b', a) satisfy x by the first part of Lemma 4.6.1. By the same reasoning, the fourth vertex is valid.

- **One vertex is in L_1 and the other vertex is in L_2 with stacks of size 1 and 2:** Let the vertices be $(a, b, c, d)_{L_1}$ and $(\{(a', b'), (e')\}, x', y')_{L_2}$. By Lemma 4.2.2, there exists a coordinate array x that is satisfied by stacks (a, b, c, d) and (a', b', e'), and there exists a coordinate array y satisfied by both stacks (a, b, c, d) and (e', a', b'). We claim the following is a valid path:

$$\begin{aligned}(a, b, c, d)_{L_1} &- (\{(a, b, c), ()\}, x, y)_{L_2} \\ &- (\{(a'), (a, b)\}, x, y)_{L_2} \\ &- (\{(a', b'), (a)\}, x, y)_{L_2} \\ &- (\{(a', b'), (e')\}, x, y)_{L_2} - (\{(a', b'), (e')\}, x', y')_{L_2}. \end{aligned} \tag{4.8}$$

  The first edge and second vertex are valid because (a, b, c, d) satisfies x.

  For the third vertex, we have (a, b, c, d) and (a', b', e') satisfy coordinate array x, so by Lemma 4.2.1, stacks (a, b) and (a') satisfy coordinate array x. Then by the first part of Lemma 4.6.1, stack (a', a, b) satisfies x. Similarly, (a, b, c, d) and (e', a', b') satisfy coordinate array y, so stacks (a, b) and (e', a') satisfy coordinate array y, so by the second part of Lemma 4.6.1, stack (a, b, a') satisfies y. Thus, the third vertex $(\{(a'), (a, b)\}, x, y)_{L_2}$ is valid.

  For the fourth vertex, we similarly have stacks (a', b') and (a) satisfy x, so stack (a', b', a) satisfy y. Additionally, stacks (e', a', b') and (a) satisfy y so (a, a', b') satisfies y. Thus the fourth vertex $(\{(a', b'), (a)\}, x, y)_{L_2}$ is valid.

  The fifth vertex $(\{(a', b'), (e')\}, x, y)_{L_2}$ is valid because (a', b', e') satisfies x and (e', a', b') satisfy y by construction of x and y.

  Hence, this is a valid path.

- **Both vertices are in L_2 and have two stacks of size 1 and 2:** Let the vertices be $(\{(a, b), (e)\}, x', y')_{L_2}$ and $(\{(a', b'), (e')\}, x'', y'')_{L_2}$. By Lemma 4.2.2, there exists a co-

ordinate array x that is satisfied by (a, b, e) and (e', a', b'), and there exists a coordinate array y satisfied by both stacks (e, a, b) and (a', b', e'). Then the following is a valid path:

$$\begin{aligned}(\{(a,b),(e)\},x',y')_{L_2} &- (\{(a,b),(e)\},x,y)_{L_2} \\ &- (\{(a,b),(a')\},x,y)_{L_2} \\ &- (\{(a),(a',b')\},x,y)_{L_2} \\ &- (\{(e'),(a',b')\},x,y)_{L_2} - (\{(a',b'),(e')\},x'',y'')_{L_2}. \end{aligned} \tag{4.9}$$

By construction of coordinate arrays x and y, vertices $(\{(a,b),(e)\},x,y)_{L_2}$ and $(\{(a',b'),(e')\},x,y)_{L_2}$ are valid. We now show vertex $(\{(a,b),(a')\},x,y)_{L_2}$ is valid, and the fact that vertex $(\{(a),(a',b')\},x,y)_{L_2}$ is valid follows by a symmetric argument. We have stacks (a,b) and (e',a') satisfy x, so (a,b,a') satisfies x by the second part of Lemma 4.6.1. Furthermore (e,a,b) and (a') satisfy y, so stack (a',a,b) satisfies y by the third part of Lemma 4.6.1.

- **One vertex is in L_2 with two stacks of size 3 and 0:** For every vertex $u = (\{(a,b,c),()\},x,y)_{L_2}$ in L_2 with stacks of size 3 and 0, any vertex of the form $v = (a,b,c,d)_{L_1}$ in L_1 has the property that the neighborhood of u is a superset of the neighborhood of v (by consider coordinate change edges from u). Thus, any vertex that v can reach in 5 edges can also be reached by u is 5 edges. In particular, since any two vertices in L_1 are at distance at most 5, any vertex in L_1 is distance at most 5 from any vertex in L_2 with stacks of size 3 and 0. Applying a similar reasoning, any two vertices in L_2 with stacks of size 3 and 0 are at distance at most 5, and any vertex in L_2 with stacks of size 3 and 0 is distance at most 5 from any vertex in L_2 with stacks of size 2 and 1.

We have thus shown that any two vertices are at distance at most 5, proving the diameter is at most 5.

5-OV has solution Now assume that the 5-OV instance has a solution. That is, assume there exists $a_1, a_2, a_3, a_4, a_5 \in A$ such that $a_1[i] \cdot a_2[i] \cdot a_3[i] \cdot a_4[i] \cdot a_5[i] = 0$ for all i. Since we assume there are no 4 orthogonal vectors, we may assume that a_1, a_2, a_3, a_4, a_5 are pairwise distinct.

Suppose for contradiction there exists a path of length at most 8 from $u_0 = (a_1, a_2, a_3, a_4)_{L_1}$ to $u_6 = (a_5, a_4, a_3, a_2)_{L_1}$. Since all vertices in L_2 have self-loops with trivial coordinate-change edges, we may assume the path has length exactly 8. Let the path be $u_0 = (a_1, a_2, a_3, a_4)_{L_1}, u_1, \ldots, u_8 = (a_5, a_4, a_3, a_2)_{L_1}$. We may assume the path never visits L_1 except at the ends: if $u_i = (S)_{L_1} \in L_1$, then $u_{i-1} = (\{\text{popped}(S), ()\}, x, y)_{L_2}$ and $u_{i+1} = (\{\text{popped}(S), ()\}, x', y')_{L_2}$ are in L_2, and in particular u_{i-1} and u_{i+1} are adjacent by a coordinate change edge, so we can replace the path $u_{i-1} - u_i - u_{i+1}$ with $u_{i-1} - u_{i+1} - u_{i+1}$, where the last edge is a self-loop.

For $i = 1, 2, 3, 4$, let p_i denote the largest index such that $u_0, u_1, \ldots, u_{p_i}$ all contain a stack that has stack $(a_1, \ldots, a_i)$ as a substack. In this way, $p_4 = 0$. For $i = 1, \ldots, 4$, let q_i be the smallest index such that vertices $u_{q_i}, \ldots, u_8$ all contain a stack with stack $(a_5, \ldots, a_{6-i})$ as a substack. In this way, $q_4 = 8$. We show that,

Claim 5. *For $i = 1, \ldots, 4$, between vertices u_{p_i} and $u_{q_{5-i}}$, there must be a coordinate change edge.*

Proof. Suppose for contradiction there is no coordinate change edge between u_{p_i} and $u_{q_{5-i}}$.

First, consider $i = 4$. Here, $u_{p_i} = u_0 = (a_1, a_2, a_3, a_4)_{L_1}$. Then, u_{q_1} is a vertex of the form $(\{S_1, S_2\}, x, y)$ where (a_5) is a substack of S_1. Since there is no coordinate change edge, we must have $u_1 = (\{(a_1, a_2, a_3), ()\}, x, y)$ for the same coordinate arrays x and y, so stack (a_1, a_2, a_3, a_4) satisfies both x and y. Then S_1, and thus (a_5), satisfies one of x and y, so there is some coordinate array satisfied by both (a_1, a_2, a_3, a_4) and (a_5), which is a contradiction of Lemma 4.2.3 By a similar argument, we obtain a contradiction with $i = 1$.

Now suppose $i = 3$. Vertex u_{p_3} is of the form $(\{(a_1, a_2, a_3), ()\}, x, y)$. Then stack (a_1, a_2, a_3) satisfies both coordinate arrays x and y. Vertex u_{q_2} is of the form $(\{S_1', S_2'\}, x, y)$ where (a_5, a_4) is a substack of S_1'. Then stack $S_1' \circ S_2'$ satisfies one of x or y, and thus (a_5, a_4), a substack of $S_1' \circ S_2'$, satisfies one of x or y. Thus, there is some coordinate array satisfied by both (a_5, a_4) and (a_1, a_2, a_3), which is a contradiction of Lemma 4.2.3. By a similar argument, we obtain a contradiction with $i = 2$.

Thus, we have shown that for all $i = 1, \ldots, 4$, there must be a coordinate change edge between u_{p_i} and $u_{q_{5-i}}$. $\square$

Since coordinate change edges do not change any vectors, by maximality of p_i, the edge $u_{p_i}u_{p_i+1}$ cannot be a coordinate change edge for all $i = 1, \ldots, 4$. Similarly, by minimality of q_i, the edge $u_{q_i-1}u_{q_i}$ cannot be a coordinate change edge for all $i = 1, \ldots, 4$.

Consider the set of edges

$$u_{p_4}u_{p_4+1}, u_{p_3}u_{p_3+1}, u_{p_2}u_{p_2+1}, u_{p_1}u_{p_1+1}, u_{q_4-1}u_{q_4}, u_{q_3-1}u_{q_3}, u_{q_2-1}u_{q_2}, u_{q_1-1}u_{q_1}. \tag{4.10}$$

By above, none of these edges are coordinate change edges. These edges are among the 8 edges $u_0u_1, \ldots, u_7u_8$. Additionally, the edges $u_{p_i}u_{p_i+1}$ for $i = 1, \ldots, 4$ are pairwise distinct, and the edges $u_{q_i-1}u_{q_i}$ for $i = 1, \ldots, 4$ are pairwise distinct. Edge $u_{p_4}u_{p_4-1}$ cannot be any of $u_{q_i-1}u_{q_i}$ for $i = 1, \ldots, 4$, because we assume our orthogonal vectors a_1, a_2, a_3, a_4, a_5 are pairwise distinct and $u_{p_4-1} = u_1$ does not have any stack containing vector a_5. Similarly, $u_{q_4-1}u_{q_4}$ cannot be any of $u_{p_i}u_{p_i+1}$ for $i = 1, \ldots, 4$. Thus, the edges in (4.10) have at least 5 distinct edges, so our path has at most 3 coordinate change edges. By Claim 5, there must be at least one coordinate change edge. We now casework on the number of coordinate change edges.

Case 1: the path $u_0, \ldots, u_8$ has one coordinate change edge. By Claim 5, since vertex $u_{p_4} = u_0$ is before the coordinate change edge, edge $u_{q_1-1}u_{q_1}$ must be after the coordinate change edge, and similarly edge $u_{p_1}u_{p_1+1}$ must be before the coordinate change edge. Then all of the edges in (4.10) are pairwise distinct, so then the path has 8 edges from (4.10) plus a coordinate change edge, for a total of 9 edges, a contradiction.

Case 2: the path has two coordinate change edges. Again, by Claim 5, for $i = 1, \ldots, 4$, edges $u_{q_i-1}u_{q_i}$ must be after the first coordinate change edge, and edge $u_{p_i}u_{p_i+1}$ must be before the second coordinate change edge. Since we have 8 edges total, we have at most 6 distinct edges from (4.10), so there must be at least two pairs (i, j) such that the edges $u_{p_i}u_{p_i+1}$ and $u_{q_j-1}u_{q_j}$ are equal, and by above this edge must be between the two coordinate change edges. Thus, each of $u_{p_4}u_{p_4+1}, u_{p_3}u_{p_3+1}, u_{p_2}u_{p_2+1}, u_{p_1}u_{p_1+1}$ and $u_{q_4-1}u_{q_4}, u_{q_3-1}u_{q_3}, u_{q_2-1}u_{q_2}, u_{q_1-1}u_{q_1}$ have at least two

edges between the two coordinate change edges. This means that vertices $u_{p_2}, u_{p_1}, u_{q_2}, u_{q_1}$ are all between the two coordinate change edges. By Claim 5, vertices u_{p_3} and u_{q_3} cannot be between the two coordinate change edges. Thus, we must have $u_{p_1}u_{p_1+1} = u_{q_2-1}u_{q_2}$ and $u_{p_2}u_{p_2+1} = u_{q_1-1}u_{q_1}$. Since we use at most 8 edges total and exactly 6 distinct edges from (4.10), we have $q_1 = p_1 = p_2 + 1 = q_2 - 1$. However, this is impossible, because that means node $u_{p_1} = u_{q_1}$ has two stacks, one containing vector a_1 and one containing vector a_5. By maximality of p_1, the stack containing vector a_1 has no other vectors, and by minimality of q_1, the stack containing vector a_5 has no other vectors, so vertex $u_{p_1} = u_{q_1}$ has two stacks with a total of only two vectors, a contradiction of the definition of a vertex in L_2.

Case 3: the path has three coordinate change edges. Since the distinct edges of (4.10) are

$$u_{p_4}u_{p_4+1}, u_{p_3}u_{p_3+1}, u_{p_2}u_{p_2+1}, u_{p_1}u_{p_1+1}, u_{q_4-1}u_{q_4}, \tag{4.11}$$

we must have

$$\begin{aligned} u_{p_3}u_{p_3+1} &= u_{q_1-1}u_{q_1} \\ u_{p_2}u_{p_2+1} &= u_{q_2-1}u_{q_2} \\ u_{p_1}u_{p_1+1} &= u_{q_3-1}u_{q_3} \end{aligned} \tag{4.12}$$

Hence, by Claim 5, there must be a coordinate change edge between any two edges in (4.11), so we must have four coordinate change edges, a contradiction.

This proves that there cannot be a length 8 path from (a_1, a_2, a_3, a_4) to (a_5, a_4, a_3, a_2), showing that the diameter is at least 9, as desired.

□

Chapter 5

Finding Short Cycles: The Girth

This chapter was written with Virginia Vassilevska Williams and focuses on computing the girth of graphs. The girth is one of the most basic graph parameters, and its computation has been studied for many decades. Under widely believed fine-grained assumptions, computing the girth exactly is known to require $mn^{1-o(1)}$ time, both in sparse and dense m-edge, n-node graphs, motivating the search for fast approximations. Fast good quality approximation algorithms for undirected graphs have been known for decades. For the girth in directed graphs, until recently the only constant factor approximation algorithms ran in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent. These algorithms have two drawbacks: (1) they only offer an improvement over the mn running time for dense graphs, and (2) the current fast matrix multiplication methods are impractical. The first constant factor approximation algorithm that runs in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ and all sparsities m was only recently obtained by Chechik et al. [STOC 2020]; it is also combinatorial.

It is known that a better than 2-approximation algorithm for the girth in dense directed unweighted graphs needs $n^{3-o(1)}$ time unless one uses fast matrix multiplication. Meanwhile, the best known approximation factor for a combinatorial algorithm running in $O(mn^{1-\varepsilon})$ time (by Chechik et al.) is 3. Is the true answer 2 or 3?

The main result of this chapter is a (conditionally) tight approximation algorithm for directed graphs. First, we show that under a popular hardness assumption, any algorithm, even one that

exploits fast matrix multiplication, would need to take at least $mn^{1-o(1)}$ time for some sparsity m if it achieves a $(2-\varepsilon)$-approximation for any $\varepsilon > 0$. Second we give a 2-approximation algorithm for the girth of unweighted graphs running in $\tilde{O}(mn^{3/4})$ time, and a $(2+\varepsilon)$-approximation algorithm (for any $\varepsilon > 0$) that works in weighted graphs and runs in $\tilde{O}(m\sqrt{n})$ time. Our algorithms are combinatorial.

We also obtain a $(4+\varepsilon)$-approximation of the girth running in $\tilde{O}(mn^{\sqrt{2}-1})$ time, improving upon the previous best $\tilde{O}(m\sqrt{n})$ running time by Chechik et al. Finally, we consider the computation of roundtrip spanners. We obtain a $(5+\varepsilon)$-approximate roundtrip spanner on $\tilde{O}(n^{1.5}/\varepsilon^2)$ edges in $\tilde{O}(m\sqrt{n}/\varepsilon^2)$ time. This improves upon the previous approximation factor $(8+\varepsilon)$ of Chechik et al. for the same running time.

5.1 Introduction.

One of the most basic and well-studied graph parameters is the *girth*, i.e. the length of the shortest cycle in the graph. Computing the girth in an m-edge, n-node graph can be done by computing all pairwise distances, that is, solving the All-Pairs Shortest Paths (APSP) problem. This gives an $\tilde{O}(mn)$ time algorithm for the general version of the girth problem: directed or undirected integer weighted graphs and no negative weight cycles[1].

The $\tilde{O}(mn)$ running time for the exact computation of the girth is known to be tight, up to $n^{o(1)}$ factors, both for sparse and dense weighted graphs, under popular hardness hypotheses from fine-grained complexity [VW10, LVW18]. In unweighted graphs or graphs with integer weights of magnitude at most M, one can compute the girth in $\tilde{O}(Mn^{\omega})$ time [Sei95, IR78, RV11, CGS15] where $\omega < 2.373$ is the exponent of $n \times n$ matrix multiplication [Vas12, Le 14]. This improves upon mn only for somewhat dense graphs with small weights, and moreover is not considered very practical due to the large overhead of fast matrix multiplication techniques.

Due to the subcubic equivalences of [VW10], however, it is known that even in unweighted

[1]If the weights are nonnegative, running Dijkstra's algorithm suffices. If there are no negative weight cycles, one can use Johnson's trick to make the weights nonnegative at the cost of a single SSSP computation which can be achieved for instance in $\tilde{O}(m\sqrt{n}\log M)$ time if M is the largest edge weight magnitude via Goldberg's algorithm [Gol93], so as long as the weights have at most $\tilde{O}(\sqrt{n})$ bits, the total time is $\tilde{O}(mn)$.

dense graphs, any algorithm that computes the girth in $O(n^{3-\varepsilon})$ time needs to use fast matrix multiplication techniques, unless one can obtain a subcubic time combinatorial Boolean Matrix Multiplication (BMM) algorithm. Thus, under popular fine-grained complexity assumptions, if one wants to have a fast combinatorial algorithm, or an algorithm that is faster than mn for sparser graphs, one needs to resort to *approximation*.

Fast approximation algorithms for the girth in undirected graphs have been known since the 1970s, starting with the work of Itai and Rodeh [IR78]. The current strongest result shows a 2-approximation in $\tilde{O}(n^{5/3})$ time [RV12]; note that if the graph is dense enough this algorithm is sublinear in the input. Such good approximation algorithms are possible for undirected graphs because of known strong structural properties. For instance, as shown by Bondy and Simonovits [BS74], for any integer $k \geq 2$, if a graph has at least $100kn^{1+1/k}$ edges, then it must contain a $2k$ cycle, and this gives an immediate upper bound on the girth. There are no such structural results for directed graphs, making the directed girth approximation problem quite challenging.

Zwick [Zwi02] showed that if the maximum weight of an edge is M, one can obtain in $\tilde{O}(n^{\omega}\log(M/\varepsilon)/\varepsilon)$ time a $(1+\varepsilon)$-approximation for APSP, and this implies the same for the girth of directed graphs. As before, however, this algorithm does not run fast in sparse graphs, and can be considered impractical.

The first nontrivial approximation algorithms (both for sparse graphs and combinatorial) for the girth of directed graphs were achieved by Pachocki et al. [PRS$^+$18]. The current best result by Chechik et al. [CLRS19, CLRS20] achieves for every integer $k \geq 1$, a randomized $O(k\log k)$-approximation algorithm running in time $\tilde{O}(m^{1+1/k})$. The best approximation factor that Chechik et al. obtain in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ is 3, in $\tilde{O}(m\sqrt{n})$ time.

What should be the best approximation factor attainable in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$? It is not hard to show (see e.g. [Vas08a], the construction in Thm 4.1.3) that graph triangle detection can be reduced to triangle detection in a directed graph whose cycle lengths are all divisible by 3. This, coupled with the combinatorial subcubic equivalence between triangle detection and BMM [VW10] implies that any $O(n^{3-\varepsilon})$ time algorithm for $\varepsilon > 0$ that achieves a $(2-\delta)$-approximation for the girth implies an $O(n^{3-\varepsilon/3})$ time algorithm for BMM, and hence fast matrix multiplication

techniques are likely necessary for faster $(2-\varepsilon)$-approximation of the directed girth.

5.1.1 Our results

We first give a simple extension to the above hardness argument for $(2-\varepsilon)$-approximation, giving a conditional lower bound on the running time of $(2-\varepsilon)$-girth approximation algorithms under the so called k-Cycle hardness hypothesis [AHR+19, PVW20, LVW18].

The k-Cycle hypothesis states that for every $\varepsilon > 0$, there is a k such that k-cycle in m-edge directed unweighted graphs cannot be solved in $O(m^{2-\varepsilon})$ time (on a $O(\log n)$ bit word-RAM).

The hypothesis is consistent with all known algorithms for detecting k-cycles in directed graphs, as these run at best in time $m^{2-c/k}$ for various small constants c [YZ04, AYZ97, LVW18, DDV19], even using powerful tools such as matrix multiplication. Moreover, as shown by Lincoln et al. [LVW18] any $O(mn^{1-\varepsilon})$ time algorithm (for $\varepsilon > 0$) that, for odd k, can detect k-cycles in n-node m-edge directed graphs with $m = \Theta(n^{1+2/(k-1)})$, would imply an $O(n^{k-\delta})$ time algorithm for k-clique detection for $\delta > 0$. If the cycle algorithm is "combinatorial", then the clique algorithm would be "combinatorial" as well, and since all known $O(n^{k-\delta})$ time k-clique algorithms use fast matrix multiplication, such a result for k-cycle would be substantial.

In Section 5.5, with a very simple reduction we show:

Theorem 5.1.1. *Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph. Then for every constant k, one can detect whether an m-edge directed graph contains a k-cycle, in $O(m^{2-\varepsilon})$ time, and hence the k-Cycle Hypothesis is false.*

Thus, barring breakthroughs in Cycle and Clique detection algorithms, we know that the best we can hope for using an $O(mn^{1-\varepsilon})$ time algorithm for the girth of directed graphs is a 2-approximation. The proof of Theorem 5.1.1 is presented in section 5.5.

The main result of this chapter is the first ever $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ 2-approximation algorithm for the girth in directed graphs. This result is conditionally tight via the above discussion.

Theorem 5.1.2. *There is an $\tilde{O}(mn^{3/4})$ time randomized algorithm that 2-approximates the girth in directed unweighted graphs whp. For every $\varepsilon > 0$, there is a $(2+\varepsilon)$-approximation algorithm for*

the girth in directed graphs with integer edge weights that runs in $\tilde{O}(m\sqrt{n}/\varepsilon)$ time. The algorithms are randomized and are correct whp.

If one wanted to obtain a $(4+\varepsilon)$-approximation to the girth via Chechik et al.'s $O(k \log k)$ approximation algorithms, the best running time one would be able to achieve is $\tilde{O}(m\sqrt{n})$. Here we show how to get an improved running time for a $(4+\varepsilon)$ approximation.

Theorem 5.1.3. *For every $\varepsilon > 0$, there is a $(4+\varepsilon)$-approximation algorithm for the girth in directed graphs with integer edge weights that runs in $\tilde{O}(mn^{\sqrt{2}-1}/\varepsilon)$ time. The algorithm is randomized and correct whp.*

In fact, we obtain a generalization of the above algorithms that improves upon the algorithms of Chechik et al. for all constants k.

Theorem 5.1.4. *For every $\varepsilon > 0$ and integer $k \geq 1$, there is a $(2k+\varepsilon)$-approximation algorithm for the girth in directed graphs with integer edge weights that runs in $\tilde{O}(mn^{\alpha_k}/\varepsilon)$ time, where $\alpha_k > 0$ is the solution to $\alpha_k(1+\alpha_k)^{k-1} = 1-\alpha_k$. The algorithms are randomized and correct whp.*

For example, let's consider α_1 in the above theorem. It is the solution to $\alpha_1 = 1-\alpha_1$, giving $\alpha_1 = 1/2$ and recovering the result of Theorem 5.1.2 for weighted graphs. On the other hand, α_2 is the solution to $\alpha_2(1+\alpha_2) = 1-\alpha_2$, which gives $\alpha_2 = \sqrt{2}-1$ and recovering Theorem 5.1.3. Finally, say we wanted to get a $6+\varepsilon$ approximation, then we need α_3, which is the solution to $\alpha_3(1+\alpha_3)^2 = 1-\alpha_3$, giving $\alpha_3 \leq 0.354$, and thus there's an $\tilde{O}(mn^{0.354}/\varepsilon)$ time $(6+\varepsilon)$-approximation algorithm. Note that there is only one positive solution to the equation defining α_k in Theorem 5.1.4.

As k grows, α_k grows as $\Theta(\log k/k)$, and so the algorithm from Theorem 5.1.4 has similar asymptotic guarantees as the algorithm of Chechik et al. as it achieves an $O(\ell \log \ell)$ approximation in $\tilde{O}(mn^{1/\ell})$ time. The main improvements lie in the improved running time for small constant approximation factors.

Our approximation algorithms on weighted graphs can be found in section 5.4. If we are aiming for an algorithm running in $T(n,m)$ time, we first suppose that the maximum edge weight of the

graph is M and we obtain an algorithm in $T(n,m)\log M$ time. We then show how to remove the $\log M$ factor at the end of section 5.4.

Roundtrip Spanners. Both papers that achieved nontrivial combinatorial approximation algorithms for the directed girth were also powerful enough to compute sparse approximate roundtrip spanners.

A c-approximate roundtrip spanner of a directed graph $G=(V,E)$ is a subgraph $H=(V,E')$ of G such that for every $u,v\in V$, $d_H(u,v)+d_H(v,u)\leq c\cdot(d_G(u,v)+d_G(v,u))$. Similar to what is known for spanners in undirected graphs, it is known [CD19] that for every integer $k\geq 2$ and every n, every n-node graph contains a $(2k-1+o(1))$-approximate roundtrip spanner on $O(kn^{1+1/k}\log n)$ edges; the $o(1)$ error can be removed if the edge weights are at most polynomial in n and the result then is optimal, up to log factors under the Erdös girth conjecture.

The best algorithms to date for computing sparse roundtrip spanners, similarly to the girth, achieve an $O(k\log k)$ approximation in $\tilde{O}(m^{1+1/k})$ time [CLRS20]. The best constant factor approximation achieved for roundtrip spanners in $O(mn^{1-\varepsilon})$ time for $\varepsilon>0$ is again achieved by Chechik et al.: a $(8+\varepsilon)$ approximate $O(n^{1.5})$-edge (in expectation) roundtrip spanner can be computed in $\tilde{O}(m\sqrt{n})$ expected time. We improve this latter result:

Theorem 5.1.5. *There is an $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ time randomized algorithm that computes a $(5+\varepsilon)$-approximate roundtrip spanner on $\tilde{O}(n^{1.5}\log^2(M)/\varepsilon^2)$ edges whp, for any n-node m-edge directed graph with edge weights in $\{1,\ldots,M\}$.*

5.2 Preliminary Lemmas

We begin with some preliminary lemmas. The first two will allow us to decrease all degrees to roughly m/n, while keeping the number of vertices and edges roughly the same. The last lemma, implicit in [CLRS19], is a crucial ingredient in our algorithms.

The following lemma was proven by Chechik et al. [CLRS19]:

Lemma 5.2.1. *Given a directed graph $G=(V,E)$ with $|V|=n,|E|=m$, we can in $O(m+n)$ time construct a graph $G'=(V',E')$ with $V\subseteq V'$, so that $|V'|\leq O(n)$, $|E'|\leq O(m+n)$, for*

every $v \in V'$, $deg(v) \leq \lceil m/n \rceil$, and so that for every $u, v \in V$, $d_{G'}(u, v) = d_G(u, v)$, and so that any path p between some nodes $u \in V$ and $v \in V$ in G' (possibly $u = v$) is in one-to-one correspondence with a path in G of the same length.

The proof of the above lemma introduces edges of weight 0, even if the graph was originally unweighted. In the lemma below which is proved in Section 5.6, we show how for an unweighted graph we can achieve essentially the same goal, but without adding weighted edges. This turns out to be useful for our unweighted girth approximation.

Lemma 5.2.2. *Given a directed unweighted graph $G = (V, E)$ and $|V| = n, |E| = m$, we can in $\tilde{O}(m + n)$ time construct an unweighted graph $G' = (V', E')$ with $V \subseteq V'$, so that $|V'| \leq O(n \log n)$, $|E'| \leq O(m + n \log n)$, for every $v \in V'$ out-deg$(v) \leq \lceil m/n \rceil$, and so that there is an integer t such that for every $u, v \in V$, $d_{G'}(u, v) = t \cdot d_G(u, v)$, and so that any path p between some nodes $u \in V$ and $v \in V$ in G' (possibly $u = v$) is in one-to-one correspondence with a path in G of length $1/t$ of the length of p.*

In particular, the lemma will imply that the girth of G' is exactly t times the girth of G, and that given a c-roundtrip spanner of G', one can in $\tilde{O}(m + n)$ time obtain from it a c-roundtrip spanner of G. We note that it is easy to obtain the same result but where both the in- and out-degrees are $O(m/n)$ (see the proof in the Section 5.6).

Now we can assume that the degree of each node is no more than $O(m/n)$. This will allow us for instance to run Dijkstra's algorithm or BFS from a vertex within a neighborhood of w nodes in $\tilde{O}(mw/n)$ time.

Another assumption we can make without loss of generality is that our given graph G is strongly connected. In linear time we can compute the strongly connected components and then run any algorithm on each component separately. We know that any two vertices in different components have infinite roundtrip distance.

A final lemma (implicit in [CLRS19]) will be very important for our algorithms:

Lemma 5.2.3. *Let $G = (V, E)$ be a directed graph with $|V| = n$ and integer edge weights in $\{1, \ldots, M\}$. Let $S \subseteq V$ with $|S| > c \log n$ (for $c \geq 100/\log(10/9)$) and let d be a positive integer.*

Let R be a random sample of $c \log n$ nodes of S and define $S' := \{s \in S \mid d(s,r) \leq d,\ \forall r \in R\}$. Suppose that for every $s \in S$ there are at most $0.2|S|$ nodes $v \in V$ so that $d(s,v), d(v,s) \leq d$. Then $|S'| \leq 0.8|S|$.

Proof. The proof will consist of two parts. First we will show that the number of ordered pairs $s, s' \in S$ for which $d(s,s'), d(s',s) \leq d$ is small. Then we will show that if $|S'| > 0.8|S|$, then with high probability, the number of ordered pairs $s, s' \in S$ for which $d(s,s'), d(s',s) \leq d$ is large, thus obtaining a contradiction.

(1) If for every $s \in S$ there are at most $0.2|S|$ nodes $v \in V$ so that $d(s,v), d(v,s) \leq d$, then the number of ordered pairs $s, s' \in S$ for which $d(s,s'), d(s',s) \leq d$ is clearly at most $0.2|S|^2$.

(2) Suppose now that $|S'| > 0.8|S|$. First, consider any $s \in S$ for which there are at least $0.1|S|$ nodes $s' \in S$ such that $d(s,s') > d$. The probability that $d(s,r) \leq d$ for all $r \in R$ is then at most $0.9^{c \log n} \leq 1/n^{100}$. Thus, via a union bound, with high probability at least $1 - 1/n^{99}$, for every $s \in S'$, there are at least $0.9|S|$ nodes $s' \in S$ such that $d(s,s') \leq d$.

Now, if $|S'| > 0.8|S|$, with high probability, there are at least $0.8|S| \times 0.9|S| = 0.72|S|^2$ ordered pairs (s,s') with $s, s' \in S$ and $d(s,s') \leq d$. There are at most $\binom{|S|}{2} \leq |S|^2/2$ ordered pairs (s,s') such that exactly one of $\{d(s,s') \leq d, d(s',s) \leq d\}$ holds. Hence, with high probability there are at least $0.22|S|^2 > 0.2|S|^2$ ordered pairs (s,s') with $s, s' \in S$ and both $d(s,s') \leq d$ and $d(s',s) \leq d$. Contradiction. □

5.3 2-Approximation for the Girth in Unweighted Graphs

Here we show how to obtain a genuine 2-approximation for the girth in unweighted graphs.

Theorem 5.3.1. *Given a directed unweighted graph G on m edges and n nodes, one can in $\tilde{O}(mn^{3/4})$ time compute a 2-approximation to the girth.*

Note that this is the first part of Theorem 5.1.2. The pseudocode for the algorithm of Theorem 5.3.1 can be found in Algorithm 1, and we will refer to it at each stage of the proof.

We will consider two cases for the girth: when it is $\geq n^\delta$ and when it is $< n^\delta$, for some $\delta > 0$ we will eventually set to $1/4$. We will assume that all out-degrees in the graph are $O(m/n)$.

5.3.1 Large girth.

Pick a random sample R of $100n^{1-\delta}\log n$ nodes, run BFS to and from each $s \in R$. Return

$$\min_{s\in R}\min_{v\neq s} d(s,v)+d(v,s).$$

If the girth is $\geq n^{\delta}$, with high probability, R will contain a node s on the shortest cycle C. Since any cycle must contain two distinct nodes, $\min_{s\in R}\min_{v\neq s} d(s,v)+d(v,s)$ is the weight of a shortest cycle that contains some node of R, and with high probability it must be the girth. Thus in $\tilde{O}(mn^{1-\delta})$ time we have computed the girth exactly. See Procedure HIGHGIRTH in Algorithm 1.

5.3.2 Small girth.

Now let us assume that the girth is at most n^{δ}. For a vertex u and integer $j \in \{0,\ldots,n^{\delta}\}$, define

$$B^j(u) := \{x \in V \mid d(u,x) = j\} \text{ and } \bar{B}^j(u) := \{x \in V \mid d(u,x) \leq j\}.$$

We will try all choices of integers i from 3 to n^{δ} to estimate the girth when it is $\leq i$.

Our algorithm first computes a random sample Q of size $O(n^{1-t}\log n)$ for a parameter t, does BFS from and to all nodes in Q, and computes for each $i \in \{1,\ldots,n^{\delta}\}$, $V_i' = \{v \in V \mid \exists q \in Q:\ d(v,q) \leq i \text{ and } d(q,v) \leq i\}$. The running time needed to do this for all $i \leq n^{\delta}$ is $\tilde{O}(mn^{1-t+\delta})$ [2].

If $V_i' \neq \emptyset$, the girth of G must be $\leq 2i$.

Now, pick the smallest i for which $V_{i+1}' \neq \emptyset$. Then $V_k' = \emptyset$ for all $k \leq i$, and we have certified that the girth is $\leq 2i+2$. If the girth is $\geq i+1$, we already have a 2-approximation. Otherwise, the girth must be $\leq i$.

Consider any $u \in V$, and $j \leq i$. Suppose that for all $j \leq i$, $|B^j(u)| \leq 100n^t$. Then, for u and for all $v \in B^j(u)$ for $j \leq i$, we could compute the distances from u to v in G efficiently: We do this by running BFS from u but stopping when a vertex outside of $\cup_{j=0}^{i} B^j(u)$ is found. Note that

[2]The running time is actually less, $\tilde{O}(n^{2-t+\delta} + mn^{1-t})$ but this won't matter for our algorithm.

the number of vertices in $\cup_{j=0}^{i} B^j(u)$ is $O(n^t \cdot i)$, and since we assumed that the degree of every vertex is $O(m/n)$, we get a total running time of $O(mn^{t-1} \cdot i)$. If this works for all vertices u, then we would be able to compute all distances up to i exactly in total time $O(m \cdot in^t) \leq O(mn^{t+\delta})$.

Unfortunately, however, some $B^j(u)$ balls can be larger than $100n^t$. In this case, for every $j \leq i$, we will compute a small set of nodes $B'^j(u)$ that will be just as good as $B^j(u)$ for computing short cycles.

Claim 6. *Fix i: $1 \leq i \leq n^\delta$. Suppose that for every $j \leq i$ we are given black box access to sets $B'^j(u) \subseteq \bar{B}_j(u)$ of nodes such that (1) In $t(n)$ time we can check whether a node is in $B'^j(u)$, (2) $|B'^j(u)| \leq 100n^t$ whp, and (3) for any cycle C of length $\leq i$ containing u, and every $j \leq i$, any node of C that is in $B^j(u)$ is also in $B'^j(u)$.*

Then there is an $O(mn^{t-1+\delta}t(n))$ time algorithm that can find a shortest cycle through u, provided that cycle has length $\leq i$.

Proof. Let us assume that there is some cycle C of length $\leq i$ containing u. Also, assume that we are given the sets $B'^j(u)$ for all $j \leq i$ as in the statement of the lemma.

Then we can compute a modified BFS out of u. We will show by induction that when considering distance $j \leq i$, our modified BFS will have found a set $N_j(u)$ of nodes such that for every $x \in N_j(u)$, $d(u,x) \leq j$, and so that for any cycle C of length $\leq i$ containing u, any node of C that is in $B^j(u)$ is also in $N_j(u)$.

Initially, $N_0(u) = \{u\}$, so the base case is fine. Let's make the induction hypothesis for j that for every $x \in N_j(u)$, $d(u,x) \leq j$, and for a shortest cycle C of length $\leq i$ containing u, any node of C that is in $B^j(u)$ is also in $N_j(u)$.

Our modified BFS proceeds as follows: Given $N_j(u)$, we go through each $z \in N_j(u)$, and if $z \in B'^j(u)$, we go through all out-neighbors y of z, and if y has not been visited until now, we place y into $N_{j+1}(u)$. See Procedure MODBFS in Algorithm 1 (parameter t is set to $1/2$ here).

Clearly, since $d(u,z) \leq j$ (by the induction hypothesis), we have that $d(u,y) \leq j+1$ for each out-neighbor y of z. Now consider a shortest cycle C containing u of length $\leq i$. To complete the induction we only care about $j < |C|$.

Assume that the induction hypothesis for j holds. Let x be the node on C at distance $j+1$

from u along C, and let x' be its predecessor on C, i.e. the node on C at distance j from u along C. Since C is a shortest cycle containing u and since $x' \neq x$, we must have that $d(u, x') = j$ so that $x' \in B^j(u)$. Also, either $u = x$, or $d(u, x) = j + 1$ and so $x \in B^{j+1}(u)$.

We know by the induction hypothesis that $x' \in N_j(u)$ and also that $x' \in B'^j(u)$ by the definition of $B'^j(u)$. Thus, we would have gone through the edges out of x', and x would have been discovered. If $u = x$, then the cycle C will be found. Otherwise, $d(u, x) = j + 1$, and x cannot have been visited until now, so our modified BFS will insert x into $N_{j+1}(u)$ thus completing the induction.

The running time of the modified BFS is determined by the fact that there are $i \leq n^\delta$ levels, each of $N_j(u) \cap B'^j(u)$ contains $\leq O(n^t)$ nodes, and we traverse the $O(m/n)$ edges out of every $x \in N_j(u) \cap B'^j(u)$. The running time is thus asymptotically $t(n) \times n^\delta \times n^t \times m/n$ which is $O(mn^{t+\delta-1}t(n))$. □

Now we want to explain how to compute the sets $B'^j(u)$. We use Lemma 5.2.3 from the preliminaries. Suppose that the girth is at most i and for every $k \leq i$, $V'_k = \emptyset$.

Let u be a node on a cycle C of length at most i. Let x be any node on C so that $x \in B^j(u)$ for some integer $j \leq i$. Then we must have that for every $y \in \bar{B}^j(u)$:

$$d(x, y) \leq d(x, u) + d(u, y) \leq |C| - d(u, x) + d(u, y) \leq i - j + j = i.$$

This inequality is crucial for our algorithm. See Figure 5-1 for a depiction of it.

In other words, we obtain that x is in $\{w \in B^j(u) \mid d(w, y) \leq i,\ \forall y \in \bar{B}^j(u)\}$.

Suppose that we are able to pick a random sample $R^j(u)$ of $c \log n$ vertices from $\bar{B}^j(u)$ (we will show how later). Then we can define

$$\bar{B}'^j(u) := \{z \in \bar{B}^j(u) \mid d(z, y) \leq i,\ \forall y \in R^j(u)\}.$$

Using Lemma 5.2.3 we will show that if $|\bar{B}^j(u)| \geq 10n^t$, then $|\bar{B}'^j(u)| \leq 0.8\bar{B}^j(u)$ and if x is in $\{w \in B^j(u) \mid d(w, y) \leq i,\ \forall y \in \bar{B}^j(u)\}$, then whp $x \in \bar{B}'^j(u)$. We will then repeat the argument to obtain $B'^j(u)$ of size $O(n^t)$.

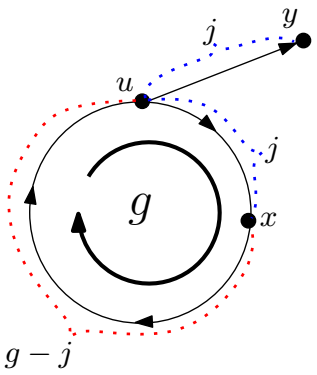

Figure 5-1: Here there is a cycle of length g containing u. A node x on the cycle is at distance j from u along the cycle and another node y is at distance $\leq j$ from u. Then the distance from x to y is at most g since one way to go from x to y is to go from x to u along the cycle at a cost of $g - j$, and then from u to y at a cost of $\leq j$. If the cycle is a shortest cycle containing u and if $x \neq u$, then the distance in the graph from u to x is j, as the path along the cycle needs to be a shortest path.

Consider any $s \in V$ with at least $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s,v), d(v,s) \leq i$. As $|\bar{B}^j(u)| \geq 10n^t$ (as otherwise we would be done), $0.2|\bar{B}^j(u)| \geq 2n^t$, and so with high probability, for s with the property above, our earlier random sample Q contains some q with $d(s,q), d(q,s) \leq i$, and so $V_i' \neq \emptyset$ which we assumed didn't happen. Thus with high probability, for every $s \in V$, there are at most $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s,v), d(v,s) \leq i$. Hence we also have that every $z \in \bar{B}^j(u)$ has at most $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s,v), d(v,s) \leq i$.

Thus we can apply Lemma 5.2.3 to $\bar{B}^j(u)$ and conclude that $|\bar{B}'^j(u)| \leq 0.8|\bar{B}^j(u)|$, while also any node $x \in B^j(u)$ on the cycle C (containing u) is also in $\bar{B}'^j(u)$.

We will iterate this process until we arrive at a subset of $\bar{B}^j(u)$ that is smaller than $10n^t$ and still contains all $x \in B^j(u)$ on an $\leq i$-length cycle C.

We do this as follows. Let $B_0^j(u) = \bar{B}^j(u)$. For each $k = 0, \ldots, 2\log n$, let $R_k^j(u)$ be a random sample of $O(\log n)$ vertices of $B_k^j(u)$. Define $B_{k+1}^j(u) = \{z \in B_k^j(u) \mid d(z,y) \leq i,\ \forall y \in \cup_{\ell=0}^k R_\ell^j(u)\}$. We get that for each k, $|B_k^j(u)| \leq 0.8^k|\bar{B}^j(u)|$ so that at the end of the last iteration, $|B_{2\log n}^j(u)| \leq 10n^t$ and we can set $B'^j(u)$ to $B_{2\log n}^j(u)$.

It is not immediately clear how to obtain the random sample $R_k^j(u)$ from $B_k^j(u)$ as $B_k^j(u)$ is unknown. We do it in the following way, adapting an argument from Chechik et al. [CLRS20]. For each $j \leq i$ and $k \leq 2\log n$ we independently obtain a random sample $S_{j,k}$ of V by sampling

each vertex independently with probability $p = 100 \log n/n^t$. For each of the (in expectation) $O(n^{1-t+\delta} \log^2(n))$ vertices in the sets $S_{j,k}$ we run BFS to and from them, to obtain all their distances.

Now, for $j \leq i$ and k, to obtain the random sample $R_k^j(u)$ of the unknown $B_k^j(u)$, we assume that we already have $R_\ell^j(u)$ for $\ell < k$, and define

$$T_k^j(u) = \{s \in S_{j,k} \mid s \in \bar{B}^j(u) \text{ and } d(s,y) \leq i,\ \forall y \in \cup_{\ell<k} R_\ell^j(u)\}.$$

Forming the set $T_k^j(u)$ is easy since we have the distances $d(s,v)$ for all $s \in S_{j,k}$ and $v \in V$, so we can check whether $s \in \bar{B}^j(u)$ and $d(s,y) \leq i,\ \forall y \in \cup_{\ell<k} R_\ell^j(u)$ in polylogarithmic time for each $s \in S_{j,k}$. See Procedure RANDOMSAMPLES in Algorithm 1.

Now since $S_{j,k}$ is independent from all our other random choices, $T_k^j(u)$ is a random sample of $B_k^j(u)$ essentially created by selecting each vertex with probability p. If $B_k^j(u) \geq 100n^t$, with high probability, $T_k^j(u)$ has at least $10 \log n$ vertices so we can pick $R_k^j(u)$ to be a random sample of $10 \log n$ vertices of $T_k^j(u)$, and they will also be a random sample of $10 \log n$ vertices of $B_k^j(u)$.

Once we have the sets $R_k^j(u)$ for each u and $j \leq i$, $k \leq 2 \log n$, we run our modified BFS from each u from Claim 6 where when we are going through the vertices $x \in N_j(u)$ we check whether $x \in B'^j(u)$ by checking whether $d(x,r) \leq i$ for every $r \in \cup_k R_k^j(u)$. This only gives a polylogarithmic overhead so we can run the modified BFS in time $\tilde{O}(mn^{t-1+\delta})$ time. We can run it through all $u \in V$ in total time $\tilde{O}(mn^{t+\delta})$ time, and in this time we will be able to compute the length of the shortest cycle if that cycle is of length $\leq i$.

Putting it all together. In $\tilde{O}(mn^{1-\delta})$ time we compute the girth exactly if it is $\geq n^\delta$. In $\tilde{O}(mn^{1-t+\delta})$ time, we obtain i so that we have a 2-approximation of the girth if the girth is $> i$. In additional $\tilde{O}(mn^{1-t+\delta} + mn^{t+\delta})$ time we compute the girth exactly if it is $\leq i$.

To optimize the running time we set $t = 1/2$, $1 - \delta = 0.5 + \delta$, obtaining $\delta = 1/4$, and a running time of $\tilde{O}(mn^{3/4})$. The final algorithm is in Algorithm 1.

Algorithm 1: 2-Approximation algorithm for the girth in unweighted graphs.

Procedure HIGHGIRTH($G = (V, E)$)
  Let $R \subseteq V$ be a uniform random sample of $100n^{3/4} \log n$ nodes.
  foreach $s \in R$ **do**
    Do BFS from s in G
  Let g be the length of the shortest cycle found by the BFS searches.
  Return g.

Procedure RANDOMSAMPLES($G = (v, E), i$)
  foreach $j \in \{1, \ldots, i\}$ **do**
    foreach $k \in \{1, \ldots, 2 \log n\}$ **do**
      Let $S_{j,k} \subseteq V$ be a uniform random sample of $100\sqrt{n} \log n$ vertices.
      foreach $s \in S_{j,k}$ **do**
        Do BFS to and from s to compute for all v, $d(s, v)$ and $d(v, s)$.
  foreach $u \in V$ **do**
    foreach $j \in \{1, \ldots, i\}$ **do**
      $R^j(u) \leftarrow \emptyset$.
      foreach $k \in \{1, \ldots, 2 \log n\}$ **do**
        $T_k^j(u) \leftarrow \{s \in S_{j,k} \mid d(u, s) \leq j$ and for all $y \in R^j(u) : \ d(s, y) \leq i\}$.
        if $|T_k^j(u)| < 10 \log n$ **then**
          $R^j(u) \leftarrow R^j(u) \cup T_k^j(u)$
          Exit this loop (over k).
        else
          Let $R_k^j(u)$ be a uniform random sample of $10 \log n$ nodes from $T_k^j(u)$.
          $R^j(u) \leftarrow R^j(u) \cup R_k^j(u)$.
  Return the sets $R^j(u)$ for all $j \leq i$, $u \in V$, and $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$.

Procedure MODBFS($G = (v, E), u, i, R^1(u), \ldots, R^i(u)), d(\cdot)$
  // $d(\cdot)$ contains $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$.
  $Visited \leftarrow$ empty hash table
  $N_0 \leftarrow \{u\}$
  $Visited.insert(u)$
  foreach j *from* 0 *to* $i - 1$ **do**
    $N_{j+1} \leftarrow$ empty linked list
    foreach $x \in N_j$ **do**
      if *for every* $s \in R^j(u)$, $d(x, s) \leq i$ **then**
        foreach y *s.t.* $(x, y) \in E$ *and* $y \notin Visited$ **do**
          if $y = u$ **then**
            Stop and return $j + 1$
          $N_{j+1}.insert(y)$
          $Visited.insert(y)$
  Return ∞ // No $\leq i$ length cycle found through u

Procedure GIRTHAPPROX($G = (V, E)$)
  $g_{high} \leftarrow$ HIGHGIRTH(G)
  Let $Q \subseteq V$ be a uniform random sample of $100n^{1/2} \log n$ nodes.
  foreach $s \in Q$ **do**
    Do BFS from and to s in G
  Let i be the minimum integer s.t. $\exists s \in Q$ and $\exists v \in V$ with $d(s, v) \leq i + 1$ and $d(v, s) \leq i + 1$.
  $g_{med} \leftarrow 2(i + 1)$
  Let i be the min of i and $n^{1/4}$
  Run RANDOMSAMPLES(G, i) to obtain sets $R^j(u)$ for all $j \leq i$, $u \in V$, and $d(\cdot)$ containing $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$
  foreach $u \in V$ **do**
    $g_u \leftarrow$ MODBFS($G, u, i, R^1(u), \ldots, R^i(u), d(\cdot)$)
  $g \leftarrow \min\{g_{high}, g_{med}, \min_{u \in V} g_u\}$
  Return g

5.4 Weighted Graphs: Girth and Roundtrip Spanner.

One of the main differences between our weighted and unweighted algorithms is that for weighted graphs we do not go through each distance value i up to n^δ, but we instead process intervals of possible distance values $[(1 + \varepsilon)^i, (1 + \varepsilon)^{i+1})$ for small $\varepsilon > 0$. This will affect the approximation, so that we will get a $(2+O(\varepsilon))$-approximation. However, it will also enable us to have a smaller running time of $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$, and to be able to output an $\tilde{O}(n^{1.5}\log(M)/\varepsilon)$-edge $(5+O(\varepsilon))$-approximate roundtrip spanner in $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon^2)$ time, where M is the maximum edge weight.

Fix $\varepsilon > 0$. For a vertex u and integer j, define (differently from the previous section)

$$B^j(u) := \{x \in V \mid (1+\varepsilon)^j \leq d(u, x) < (1+\varepsilon)^{j+1}\} \text{ and } \bar{B}^j(u) := \{x \in V \mid d(u, x) < (1+\varepsilon)^{j+1}\}.$$

We include a boundary case $B^\emptyset(u) := \{x \in V \mid d(u, x) = 0\}$. Recall that we originally started with a graph with positive integer weights, but our transformation to vertices of degree $O(m/n)$ created some 0 weight edges. We note that any distance of 0 involves at least one of the auxiliary vertices and no roundtrip distance can be 0.

In our algorithms including our $(2 + \epsilon)$-approximation algorithm, we do a restricted version of

Dijkstra from every vertex where before running these Dijkstras, we need to efficiently sample a set of vertices $R^j(u)$ of size $O(\log n)$ from a subset of $B^j(u)$, without computing the set $B^j(u)$. The following lemma is given as input the target approximation factor 2β, a parameter i as an estimated size of cycles the algorithm is handling at a given stage and a parameter α as the target running time $\tilde{O}(mn^\alpha)$ of our algorithms. It outputs the sample sets in this running time. The proof of the lemma is similar to the sampling method of the previous section and is included in section 5.6.

Lemma 5.4.1. *Let M be the maximum edge weight of the graph and suppose that $i \in \{1, \ldots, \log_{1+\epsilon} Mn\}$, $\beta > 0$ and $0 < \alpha < 1$ are given. Suppose that Q is a given sampled set of size $\tilde{O}(n^\alpha)$ vertices. Let $d = \beta(1+\epsilon)^{i+1}$. Let $V_i' = \{v \in V \mid \exists q \in Q : \ d(v,q) \leq d$ and $d(q,v) \leq d\}$. In $\tilde{O}(mn^\alpha)$ time, for every $u \in V$ and every $j = \{1, \ldots, \log_{(1+\epsilon)}(Mn)\}$, one can output a sample set $R_i^j(u)$ of size $O(\log^2 n)$ from $\bar{Z}_i^j(u) = \bar{B}^j(u) \setminus V_i'$, where the number of vertices in $Z_i^j(u) = B^j(u) \setminus V_i'$ of distance at most d from all vertices in $R_i^j(u)$ is at most $O(n^{1-\alpha})$ whp.*

Now we focus on our $(2 + O(\epsilon))$-approximation algorithm for the girth and $(5 + O(\epsilon))$-approximate roundtrip spanner. We are going to prove the following Theorem, which consists of Theorem 5.1.5 and the second part of Theorem 5.1.2 with a $\log M$ factor added to their running times.

Theorem 5.4.1. *Let G be an n-node, m-edge directed graph with edge weights in $\{1, \ldots, M\}$. Let $\varepsilon > 0$. One can compute a $(5+\varepsilon)$-roundtrip spanner on $\tilde{O}(n^{1.5} \log^2 M/\varepsilon^2)$ edges in $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ time, whp. In $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$ time, whp, one can compute a $(2+\varepsilon)$-approximation to the girth.*

We will start with a sampling approach, similar to that in the unweighted girth approximation. The pseudocode of the girth algorithm can be found in Algorithm 2, and we will refer to it at each stage of the proof.

Lemma 5.4.2. *Let $G = (V, E)$ be a directed graph with $|V| = n$ and integer edge weights in $\{1, \ldots, M\}$. Let d be a positive integer, $\varepsilon \geq 0$, and let $Q \subseteq V$ be a random sample of $100\sqrt{n}\log n$ vertices. In $\tilde{O}(m\sqrt{n})$ time we can compute shortest paths trees $T^{in}(q), T^{out}(q)$ into and out of*

each $q \in Q$. Let H be the subgraph of G consisting of the edges of these trees $T^{in}(q), T^{out}(q)$. Let $V' = \{v \in V \mid \exists q \in Q,\ d(v,q) \leq d$ and $d(q,v) \leq d\}$. Then:

- **Girth approximation:** *If $V' \neq \emptyset$, then the girth of G is at most $2d$.*
- **Additive distance approximation:** *For any $u, v \in V$, if the shortest u to v path contains a node of V', then $d_H(u,v) \leq d(u,v) + 2d$.*
- **Sparsity:** *The number of edges in H is $\tilde{O}(n^{1.5})$.*

Proof. Given a directed $G = (V, E)$ with $|V| = n$, $|E| = m$ and edge weights in $\{1, \ldots, M\}$, let us first take a random sample $Q \subseteq V$ of $100\sqrt{n}\log n$ vertices. Run Dijkstra's algorithm from and to every $q \in Q$. Determine $V' \subseteq V$ defined as those $v \in V$ for which there is some $q \in Q$ with $d(v,q), d(q,v) \leq d$. If $V' \neq \emptyset$, we get that the girth of G is at most $2d$. Suppose that we insert all edges of the in- and out- shortest paths trees rooted at all $q \in Q$ into a subgraph H. Then we have only inserted $\tilde{O}(n^{1.5})$ edges as each tree has $\leq n - 1$ edges.

Consider some $u, v \in V$ such that there is some node $x \in V'$ on the shortest $u - v$ path. Let $q \in Q$ be such that $d(x,q), d(q,x) \leq d$. Then

$$d_H(u,v) \leq d(u,q) + d(q,v) \leq d(u,x) + d(x,q) + d(q,x) + d(x,v) \leq d(u,v) + 2d.$$

□

Our approach below will handle the roundtrip spanner and the girth approximation at the same time.

We will try all choices of integers i from 0 to $\log_{1+\varepsilon}(Mn)$ to estimate roundtrip distances in the interval $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$, and to estimate the girth if it is $< (1+\varepsilon)^{i+1}$.

Fix a choice for i for now.

Our algorithm first applies the approach of Lemma 5.4.2 by setting $d = (1+\varepsilon)^{i+2}$ (we will see later why). We compute a random sample Q of size $O(\sqrt{n}\log n)$, do Dijkstra's from and to all nodes in Q, and add the edges of the computed shortest paths trees to our roundtrip spanner H.

We also compute

$$V_i' = \{v \in V \mid \exists q \in Q : \ d(v,q) \leq (1+\varepsilon)^{i+2} \text{ and } d(q,v) \leq (1+\varepsilon)^{i+2}\}.$$

By Lemma 5.4.2, if $V_i' \neq \emptyset$, the girth of G must be $\leq 2(1+\varepsilon)^{i+2}$. For the choice of i where $(1+\varepsilon)^i \leq g \leq (1+\varepsilon)^{i+1}$, we will get an approximation factor of $2(1+\varepsilon)^2 \leq 2(1+3\varepsilon)$. Just as with the algorithm for unweighted graphs, we can pick the minimum i so that $V_i' \neq \emptyset$, use $2(1+\varepsilon)^{i+2}$ as one of our girth estimates and then proceed from now on with a single value $i-1$ considering only the interval $[(1+\varepsilon)^{i-1}, (1+\varepsilon)^i)$.

By Lemma 5.4.2, we also get that for any $u, v \in V$ for which the u-v shortest path contains a node of V_i', H gives a good additive estimate of $d(u,v)$, i.e. $d(u,v) \leq d_H(u,v) \leq d(u,v) + 2(1+\varepsilon)^{i+2}$.

Suppose that also $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$, and that we somehow also get a good estimate for $d(v,u)$ (either because the v-u shortest path contains a node of V', or by adding more edges to H), so that also $d(v,u) \leq d_H(v,u) \leq d(v,u) + 2(1+\varepsilon)^{i+2}$. Then,

$$d(u \leftrightarrows v) \leq d_H(u \leftrightarrows v) \leq d(u \leftrightarrows v) + 4(1+\varepsilon)^{i+2} \leq d(u \leftrightarrows v)(1+4(1+3\varepsilon)) \leq d(u \leftrightarrows v)(5+12\varepsilon).$$

In other words, we would get a $5+O(\varepsilon)$-roundtrip spanner, as long as by adding $\tilde{O}(n^{1.5})$ edges to H, we can get a good additive approximation to the weights of the u-v shortest paths that do not contain nodes of V_i', for all u, v with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$. We will in fact compute these shortest paths exactly. For the girth g itself, we will show how to compute it exactly, if no node of V_i' hit the shortest cycle, where i is such that $(1+\varepsilon)^{i-1} \leq g \leq (1+\varepsilon)^i$.

Fix i. Let $Z_i = V \setminus V_i'$ and $d = (1+\varepsilon)^{i+1}$. We can focus on the subgraph induced by Z_i.

Consider any $u \in Z_i$, and $j \leq i$. Define $Z_i^j(u) = Z_i \cap B^j(u)$ and $\bar{Z}_i^j(u) = Z_i \cap \bar{B}^j(u)$. We also add the boundary case $Z_i^\emptyset = Z_i \cap B^\emptyset(u) = \{x \in Z_i \mid d(u,x) = 0\}$.

If for all $j \in \{\emptyset\} \cup \{1, \ldots, i\}$, $|Z_i^j(u)| \leq 100\sqrt{n}$, running Dijkstra's algorithm from u in the graph induced by Z_i, up to distance $(1+\varepsilon)^{i+1}$ would be cheap. Unfortunately, however, some $Z_i^j(u)$ balls can be larger than $100\sqrt{n}$. In this case, similarly to our approach for the unweighted

case, we will replace $Z_i^j(u)$ with a set $Z_i'^j(u) \subseteq \bar{Z}_i^j(u)$ of size $O(\sqrt{n})$ with the guarantee that for any $v \in V$ with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$ for which the shortest u-v path does not contain a node of V_i', every node of this u-v shortest path that is in $Z_i^j(u)$ is also in $Z_i'^j(u)$.

The following lemma shows how to use such replacement sets.

Lemma 5.4.3. *Let u and i be fixed. Suppose that for every $j \in \{\emptyset\} \cup \{1, \ldots, i\}$ we are given black box access to sets $Z_i'^j(u) \subseteq \bar{Z}_i^j(u)$ of nodes such that (1) Checking whether a node z is in $Z_i'^j(u)$ takes $t(n)$ time, (2) $|Z_i'^j(u)| \leq 100\sqrt{n}$ whp, and (3) for any v such that $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$, and every $j \leq i$, every node on the shortest path P from u to v that is in $Z_i^j(u)$ is also in $Z_i'^j(u)$.*

Then there is an $\tilde{O}(m \log(M) t(n)/(\varepsilon\sqrt{n}))$ time algorithm that finds a shortest path from u to any v with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$ and s.t. the shortest u-v path does not contain a node of V_i'. The algorithm returns $\tilde{O}(n^{0.5} \log(M)/\varepsilon)$ edges whose union contains all these shortest paths.

Proof. Assume we have the sets $Z_i'^j(u)$ for $j \in \{\emptyset\} \cup \{1, \ldots, i\}$ as in the statement of the lemma.

Then we will define a modified Dijkstra's algorithm out of u. The algorithm begins by placing u in the Fibonacci heap with $d[u] = 0$ and all other vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether x is in $Z_i'^j$. If it is not, we ignore it and extract a new vertex from the heap. Otherwise if $x \in Z_i'^j$, we go through all its out-edges (x, y), and if $d[y] > d[x] + w(x, y)$, we update $d[y] = d[x] + w(x, y)$. For any new cycle to u found, we update the best weight found, and in the end we return it. See Procedure MODDIJKSTRA in Algorithm 2.

Since we only go through the edges of at most $O(\sqrt{n} \log(Mn)/\varepsilon)$ vertices and the degrees are all $O(m/n)$, the runtime is $O(m \log(Mn)/(\varepsilon\sqrt{n}))$. For the same reason, the modified shortest paths tree whose edges we add to our roundtrip spanner has at most $O(\sqrt{n} \log(Mn)/\varepsilon)$ edges.

Let v be such that $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$ and for which the shortest u-v path does not contain a node of V_i'. We will show by induction that our modified Dijkstra's algorithm will compute the shortest path from u to v exactly.

The induction will be on the distance from u. Let's call the nodes on the shortest u to v path, $u = u_0, u_1, \ldots, u_t = v$. The induction hypothesis for u_k is that u_k is extracted from the heap with $d[u_k] = d(u, u_k)$. Let us show that u_{k+1} will also be extracted from the heap with $d[u_{k+1}] = d(u, u_{k+1})$. The base case is clear since u is extracted first.

When u_k is extracted from the heap, by the induction hypothesis, $d[u_k] = d(u, u_k)$. Let j be such that $(1+\varepsilon)^j \leq d[u_k] < (1+\varepsilon)^{j+1}$. As no node on the u-v shortest path is in V_i', we get that $u_k \in Z_i^j$. By the assumptions in the lemma, we also have that $u_k \in Z_i'^j$. Thus, when u_k is extracted, we will go over its edges. In particular, (u_k, u_{k+1}) will be scanned, and $d[u_{k+1}]$ will be set to (or it already was) $d[u_k] + w(u_k, u_{k+1}) = d(u, u_{k+1})$. This completes the induction.

It is also not hard to see that the girth will be computed exactly if u is on a shortest cycle, the girth is in $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ and V_i' is empty. □

Now we compute the sets $Z_i'^j(u)$. First consider $u, v \in V$ with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$. Let x be any node on the u to v roundtrip path (cycle) so that $x \in Z_i^j(u)$ for some integer $j \leq i$. Recall that this means $(1+\varepsilon)^j \leq d(u,x) < (1+\varepsilon)^{j+1}$. Then for every y with $d(u,y) < (1+\varepsilon)^{j+1}$ and so for each $y \in \bar{Z}_i^j(u)$ we must have (see Figure 5-2) that

$$d(x,y) \leq d(x,u) + d(u,y) \leq d(u \leftrightarrows v) - d(u,x) + d(u,y) \leq d(u \leftrightarrows v) - (1+\varepsilon)^j + (1+\varepsilon)^{j+1}$$

$$= d(u \leftrightarrows v) + \varepsilon(1+\varepsilon)^j \leq d(u \leftrightarrows v) + \varepsilon(1+\varepsilon)^i \leq d(u \leftrightarrows v)(1+\varepsilon) \leq (1+\varepsilon)^{i+2}.$$

In other words, x must be in $\{w \in \bar{Z}_i^j(u) \mid d(w,y) \leq (1+\varepsilon)^{i+2},\ \forall y \in \bar{Z}_i^j(u)\}$.

We apply Lemma 5.4.1 for $\beta = (1+\epsilon)$ and $\alpha = 1/2$. It outputs sets $R_i^j(u)$ of size $O(\log^2 n)$ vertices, where the number of vertices in $\bar{Z}_i^j(u)$ that are at distance $(1+\epsilon)^{i+2}$ from all vertices in $R_i^j(u)$ is $O(\sqrt{n})$ (See Procedure RANDOMSAMPLESWT in Algorithm 2). So all vertices $x \in Z_i^j(u)$ that are in a roundtrip path $u - v$ with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$ are in this set, so we let $Z_i'^j(u) = \{w \in \bar{Z}_i^j(u) | d(w,y) \leq (1+\epsilon)^{i+2},\ \forall y \in R_i^j(u)\}$.

Now that we have the random samples, we implement the modified Dijkstra's algorithm from Lemma 5.4.3 with only a polylogarithmic overhead as follows:

Fix some j. Let's look at the vertices x with $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$ that the modified

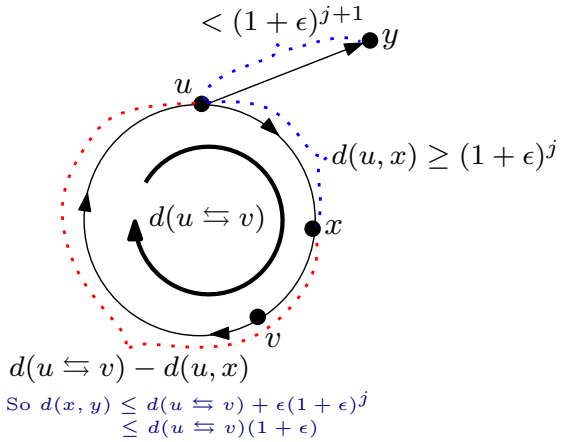

Figure 5-2: Here u and v have roundtrip distance more than $(1+\varepsilon)^j$. A node x on the shortest u-v path is at distance at least $(1+\varepsilon)^j$ from u, and another node y is at distance at most $(1+\varepsilon)^{j+1}$ from u. Then the distance from x to y is at most $d(u \leftrightarrows v)(1+\varepsilon)$ since one way to go from x to y is to go from x to u along the u-v roundtrip cycle at a cost of at most $d(u \leftrightarrows v) - (1+\varepsilon)^j$, and then from u to y at a cost of at most $(1+\varepsilon)^{j+1}$.

Dijkstra's algorithm extracts from the heap. Since $d[x]$ is always an overestimate, $d(u,x) \leq d[x] < (1+\varepsilon)^{j+1}$, and so $x \in \bar{B}^j(u)$. Now, since x is already in $\bar{B}^j(u)$, to check whether $x \in Z_i'^j(u)$, we only need to check whether $x \in Z_i$ (easy) and whether $d(x,y) \leq (1+\varepsilon)^{i+2}$ for all $y \in R_i^j(u)$ (this takes $O(\log^2 n)$ time since we have all the distances to the nodes in the random samples).

The final running time is $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ since we need to run the above procedure $O(\log(Mn)/\varepsilon)$ times, once for each i, and each procedure costs $\tilde{O}(m\log(M)\sqrt{n}/\varepsilon)$ time. As we mentioned before, to estimate the girth to within a $(2+\varepsilon)$-factor, we do not need to run the procedure for all i but (as with the algorithm for unweighted graphs), only for the minimum i for which $V'_{i+1} \neq \emptyset$. Thus the running time for the girth becomes $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$. See Procedure GIRTHAPPROXWT in Algorithm 2.

5.4.1 $(4+\epsilon)$-Approximation Algorithm for the Girth in $\tilde{O}(mn^{\sqrt{2}-1})$ Time

In this section we are going to prove the modified version of Theorem 5.1.3, where a $\log M$ factor is added to the running time with M being the maximum edge weight.

Theorem 5.4.2. *For every $\varepsilon > 0$, there is a $(4+\varepsilon)$-approximation algorithm for the girth in directed graphs with edge weights in $\{1,\ldots,M\}$ that runs in $\tilde{O}(mn^{\sqrt{2}-1}\log(M)/\varepsilon)$ time.*

Algorithm 2: $2+\varepsilon$-Approximation algorithm for the girth in weighted graphs.

Procedure RANDOMSAMPLESWT($G=(v,E), i, \varepsilon$)
  foreach $j \in \{1,\ldots,i\}$ **do**
    foreach $k \in \{1,\ldots,2\log n\}$ **do**
      Let $S_{j,k} \subseteq V$ be a uniform random sample of $100\sqrt{n}\log n$ vertices.
      foreach $s \in S_{j,k}$ **do**
        Run Dijkstra's to and from s to compute for all v, $d(s,v)$ and $d(v,s)$.
  foreach $u \in V$ **do**
    foreach $j \in \{0,\ldots,i\}$ **do**
      $R^j(u) \leftarrow \emptyset$.
      foreach $k \in \{1,\ldots,2\log n\}$ **do**
        $T_k^j(u) \leftarrow \{s \in S_{j,k} \mid d(u,s) < (1+\varepsilon)^{j+1}$ and for all $y \in R^j(u):\ d(s,y) \le (1+\varepsilon)^{i+2}\}$.
        if $|T_k^j(u)| < 10\log n$ **then**
          $R^j(u) \leftarrow R^j(u) \cup T_k^j(u)$
          Exit this loop (over k).
        else
          Let $R_k^j(u)$ be a uniform random sample of $10\log n$ nodes from $T_k^j(u)$.
          $R^j(u) \leftarrow R^j(u) \cup R_k^j(u)$.
  Return the sets $R^j(u)$ for all $j \le i$, $u \in V$, and $d(s,v), d(v,s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$.

Procedure MODDIJKSTRA($G=(v,E), u, i, \varepsilon, R^1(u),\ldots,R^i(u)), d(\cdot)$
  // $d(\cdot)$ contains $d(s,v), d(v,s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$.
  $F \leftarrow$ empty Fibonacci heap
  $Extracted \leftarrow$ empty hash table
  $F.insert(u,0)$
  $g_u \leftarrow \infty$
  while *F is nonempty* **do**
    $(x, d[x]) \leftarrow F.extractmin$
    $Extracted.insert(x)$
    if *for every* $s \in R^j(u)$, $d(x,s) \le (1+\varepsilon)^{i+2}$ **then**
      foreach *y s.t.* $(x,y) \in E$ **do**
        if $y \notin Extracted$ **then**
          if *y is in F* **then**
            $F.DecreaseKey(y, d[x]+w(x,y))$
          else
            $F.insert(y, d[x]+w(x,y))$
        if $y = u$ **then**
          $g_u \leftarrow \min\{g_u, d[x]+w(x,y)\}$
  Return g_u

Procedure GIRTHAPPROXWT($G = (V, E), \varepsilon$)
  Let $Q \subseteq V$ be a uniform random sample of $100n^{1/2} \log n$ nodes.
  foreach $s \in Q$ **do**
    Do Dijkstra's from and to s in G
  Let i be the minimum integer s.t. $\exists s \in Q$ and $\exists v \in V$ with $d(s, v) < (1 + \varepsilon)^{i+2}$ and $d(v, s) < (1 + \varepsilon)^{i+2}$.
  $g_{med} \leftarrow \min_{s \in Q, v \in V} d(s, v) + d(v, s)$ // $g_{med} < 2(1 + \varepsilon)^{i+2}$
  if $i < 0$ **then**
    // Here $i = -1$ and $d(s, v) = d(v, s) = 1$
    Return g_{med}
  Run RANDOMSAMPLESWT(G, i, ε) to obtain sets $R^j(u)$ for all $j \leq i$, $u \in V$, and $d(\cdot)$ containing $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$
  foreach $u \in V$ **do**
    $g_u \leftarrow$ MODDIJKSTRA($G, u, i, \varepsilon, R^1(u), \ldots, R^i(u), d(\cdot)$)
  $g \leftarrow \min\{g_{med}, \min_{u \in V} g_u\}$
  Return g

Proof. Suppose that we want an $\tilde{O}(mn^\alpha)$ time girth approximation algorithm. Let $\beta = 2(1 + \epsilon)$. As a first step, we sample a set Q of $\tilde{O}(n^\alpha)$ vertices and do in and out Dijkstra from them.

We let $V'_i = \{v \in V \mid \exists q \in Q : \ d(v, q) \leq \beta(1 + \varepsilon)^{i+1}$ and $d(q, v) \leq \beta(1 + \varepsilon)^{i+1}\}$. If $V'_i \neq \emptyset$ for some i, then we have that the girth g is at most $2\beta(1 + \epsilon)^{i+1}$. If $(1 + \epsilon)^i \leq g \leq (1 + \epsilon)^{i+1}$, this is a $2\beta(1 + \epsilon) \leq 4(1 + 3\epsilon) = 4 + O(\epsilon)$ approximation.

So take the minimum i where $V'_{i+1} \neq \emptyset$. Let $g' = (1 + \epsilon)^{i+1}$ be our current upper bound for the girth g. We initially mark all vertices "on", meaning that they are not processed yet. For each on vertex u, we either find the smallest cycle of length at most g' passing through u where all vertices of the cycle are on, or conclude that there is no cycle of length at most g' passing through u. When a vertex u is processed, we mark it as "off". We proceed until all vertices are off.

We apply Lemma 5.4.1 for $\beta = 2(1 + \epsilon)$. Note that since $V'_i = \emptyset$, $Z^j_i(u) = B^j(u)$ is all the vertices at distance $[(1 + \epsilon)^j, (1 + \epsilon)^{j+1})$ from u. The lemma outputs sets $R^j_i(u) \subseteq Z^j_i(u)$, where $|R^j_i(u)| = O(\log^2 n)$ and the number of vertices in $B^j(u)$ at distance $\beta g'$ from $R^j_i(u)$ is at most $O(n^{1-\alpha})$ whp. Fix some on vertex u. We do modified Dijkstra from u up to vertices with distance at most $g'/2$ from u as follows:

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether $d(x,r) \leq g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$ for all $r \in R_i^{j'}(u)$ for all j'. If x does not satisfy this condition, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x,y), and if $d[y] > d[x] + w(x,y)$, we update $d[y] = d[x] + w(x,y)$. We stop when the vertex u extracted from the heap has $d[u] > g'/2$.

Let $S_i(u)$ be the set of all the vertices visited in the modified out-Dijkstra. Simillarly, let $T_i(u)$ be all the vertices visited in the analogous modified in-Dijkstra (using an analogous version of Lemma 5.4.1).

Suppose that there is a vertex v with $d(u \leftrightarrows v) \leq g'$, where all vertices in the uv cycle C are on. Without loss of generality, suppose that $d_C(u,v) \leq g'/2$. So $d(u,v) \leq g'/2$. Moreover, suppose that $v \in Z_i^j(u)$, i.e. $(1+\epsilon)^j \leq d(u,v) \leq (1+\epsilon)^{j+1}$. So for any vertex $w \in Z_i^{j'}(u)$ for some j' we have that $d(v,w) \leq d(v,u) + d(u,w) \leq g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$. Since all vertices on the uv path that is part of the cycle are on and the length of this path is at most $g'/2$, we visit v in the out-Dijkstra, i.e. $v \in S_i(u)$. Similarly, if $d(v,u) \leq g'/2$, we visit v in the in-Dijkstra and so $v \in T_i(u)$.

If both $S_i(u)$ and $T_i(u)$ have size at most n^α, we do Dijkstra from u in the induced subgraph on $S_i(u) \cup T_i(u)$, and see if there is a cycle of length at most g' passing through u (and find the smallest such cycle), which takes $O(\frac{m}{n} n^\alpha)$ time. We take the length of this cycle as one of our estimates. The modified in and out Dijkstras take $O(\log^2 n . \frac{\log nM}{\varepsilon} . n^\alpha . \frac{m}{n})$, as checking the conditions for each x extracted from the heap takes $O(\log^2 n . \frac{\log nM}{\varepsilon})$ time. So in $\tilde{O}(\frac{\log M}{\varepsilon} n^\alpha . \frac{m}{n})$ time we process u and mark it as "off and proceed to another vertex.

Suppose $S_i(u)$ has size bigger than n^α (the case where $T_i(u)$ has size bigger than n^α is similar). Note that by Lemma 5.4.1 we have $|S_i(u)| \leq O(n^{1-\alpha})$ because for each $r \in R_i^j(u)$, we have that $d(x,r) \leq g' - (1+\varepsilon)^j + (1+\varepsilon)^{j+1} \leq g' + \varepsilon(1+\varepsilon)^j \leq g' + \varepsilon g'/2 \leq \beta g'$. So it is a subset of vertices that are at distance at most $\beta g'$ from all samples in R_i^j for all j. Our new goal is the following: We want to either find the smallest cycle of length at most g' passing through $S_i(u)$ that contains no

off vertices, or say that there is no cycle of length $\leq g'$ passing through any of the vertices in $S_i(u)$ whp.

For this, we do another Modified Dijkstra from u as follows:

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether $d(x, r) \leq \beta g' = 2(1+\varepsilon)g'$ for all $r \in R_i^j(u)$. If it is not, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x, y), and if $d[y] > d[x] + w(x, y)$, we update $d[y] = d[x] + w(x, y)$. We stop when the vertex u extracted from the heap has $d[u] > 3g'/2$.

We show that if there is a cycle of length at most g' going through $v \in S_i(u)$ containing to off vertex, all vertices of the cycle are among the vertices we visit in the modified Dijkstra: Suppose that $d(w \leftrightarrows v) \leq g'$, and suppose that $v \in Z_i^j(u)$ and $w \in Z_i^{j'}(u)$. Then for every $r \in R_i^{j'}(u)$, we have that $d(w, r) \leq d(w, v) + d(v, r) \leq g' - d(v, w) + g' - (1+\varepsilon)^j + (1+\varepsilon)^{j'+1}$. Since $d(v, w) \geq (1+\varepsilon)^{j'} - (1+\varepsilon)^{j+1}$, we have $d(w, r) \leq 2g' + \varepsilon(1+\varepsilon)^{j'} + \varepsilon(1+\varepsilon)^j \leq 2g' + 3\varepsilon g'/2 + \varepsilon g'/2 = \beta g'$. Since the uw path that goes through v is a path of length at most $\beta g'$ that has no off vertices, we visit w in the modified Dijkstra.

By Lemma 5.4.1 the total number of vertices visited in the modified Dijkstra is at most $O(n^{1-\alpha})$. Let the subgraph on these vertices be G'. We recurse on G', and find a $4 + O(\varepsilon)$ approximation of the girth in G'. The girth in G' is a lower bound on the minimum cycle of length $\leq g'$ passing through any vertex in $S_i(u)$ that has no off vertex. We take this value as one of our estimates. So we have processed all vertices in $S_i(u)$ and we mark them off. This takes $O(\frac{m}{n}.\frac{\log M}{\varepsilon}.((n^{1-\alpha})^{1+\alpha}))$, and we have marked off at least n^α vertices. So we spend $O(\frac{m}{n}.\frac{\log M}{\varepsilon}.n^{1-\alpha^2-\alpha})$ for processing each vertex. Letting $1-\alpha^2-\alpha = \alpha$, we have that $\alpha = \sqrt{2}-1$. So the total running time is $\tilde{O}(mn^{\sqrt{2}-1}\log(M)/\varepsilon)$. Our final estimate of the girth is the minimum of all the estimates we get through processing vertices. $\square$

5.4.2 $(2k+\epsilon)$-Approximation Algorithm For the Girth

In this section we are going to prove a modified version of Theorem 5.1.4, where a $\log M$ factor is added to the running time with M being the maximum edge weight. The proof is a generalization of the proof of Theorem 5.4.2.

Theorem 5.4.3. *For every $\varepsilon > 0$ and integer $k \geq 1$, there is a $(2k+\varepsilon)$-approximation algorithm for the girth in directed graphs with edge weights in $\{1, \ldots, M\}$ that runs in $\tilde{O}(mn^{\alpha_k} \log(M)/\varepsilon)$ time, where $\alpha_k > 0$ is the solution to $\alpha_k(1+\alpha_k)^{k-1} = 1 - \alpha_k$.*

Suppose that we are aiming for a $2k(1+O(\epsilon))$ approximation algorithm for the girth, in $\tilde{O}(mn^{\alpha} \log M/\varepsilon)$ time, where we set α later. So basically we want to spend $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}n^{\alpha})$ per vertex. Let $\beta = k + k^2\epsilon + k\epsilon = k + O(\varepsilon)$. As before, first we sample a set Q of $\tilde{O}(n^{\alpha})$ and do in and out Dijkstra from each vertex $q \in Q$. Let i_{min} be the minimum number i such that the set $V_i' = \{v \in V \mid \exists q \in Q : \ d(v,q) \leq \beta(1+\varepsilon)^{i+1}$ and $d(q,v) \leq \beta(1+\varepsilon)^{i+}\}$ is non-empty. So our initial estimate of the girth is $2\beta(1+\epsilon)^{i_{min}+1}$.

Let $i = i_{min} - 1$ and let $g' = (1+\varepsilon)^{i+1}$ be our estimate of the girth. Initially we mark all vertices as "on", and as we process each vertex, we either find a smallest cycle of length at most g' with no "off" vertex, or we say that there is no cycle of length at most g' passing through it whp, and we mark the vertex as off.

We apply Lemma 5.4.1 for $\beta = k+k^2\epsilon+k\epsilon$ and the set Q as input. It gives us the sets $R_i^j(u)$ of size $O(\log^2 n)$ for all j, such that the number of vertices in $\bar{B}^j(u) = \{w \in V | d(u,w) \leq (1+\epsilon)^{j+1}\}$ that are at distance at most $\beta g'$ from all $r \in R_i^j(u)$ is at most $O(n^{1-\alpha})$ whp.

We take an on vertex u and do "modified" Dijkstra from (to) u, stopping at distance $g'/2$, such that the set of vertices we visit contains any cycle of length g' that passes through u that has no off vertex. We explain this modified Dijkstra later.

We call the set of vertices that we visit in the modified out-Dijkstra $S_i^1(u)$. If $S_i^1(u) \leq n^{\alpha}$, we do an analogous modified in-Dijkstra from u, and let $T_i^1(u)$ be the set of vertices visited in this in-Dijkstra. If $T_i^1(u) \leq n^{\alpha}$, then we do Dijkstra from u in the subgraph induced by $S_i^1(u) \cup T_i^1(u)$, and hence find a smallest cycle of length $\leq g'$ that passes through u with no off vertex. We take

the length of this cycle as one of our estimates for the girth. If there is no such cycle, we don't have any estimate from u. Now we mark u as off and proceed the algorithm by taking another on vertex. Our modified Dijkstras takes $O(\log^2 n.\frac{\log Mn}{\varepsilon}.\frac{m}{n}|S|)$ time if S is the set of vertices visited by the Dijkstra. Hence for processing u we spend $O(\log^2 n.\frac{\log Mn}{\varepsilon}.\frac{m}{n}n^\alpha)$ time.

So suppose that either $S_i^1(u)$ or $T_i^1(u)$ have size bigger than n^α. Without loss of generality assume that $|S_i^1(u)| \geq n^\alpha$ (the other case is analogous). For $1 \leq l \leq k$, define sets $S_i^l(u)$ as the set of on vertices $w \in V$ such that there is a path of length at most $(2l-1)g'/2$ from u to w that contains no off vertex, and if $w \in B^j(u)$, then for all $r \in R_i^{j'}(u)$ for all j', we have $d(w,r) \leq (l+l^2\varepsilon)g' + (1+\epsilon)^{j'+1} - (1+\epsilon)^j$. Once we explain our modified Dijkstras, it will be clear that S_i^1 defined here is indeed the set of vertices visited in the first modified out-Dijkstra.

We set $S_i^0(u) = \{u\}$. We prove the following useful lemma in the Section 5.6.

Lemma 5.4.4. *For all $l \in \{1,\ldots,k\}$, we have that $S_i^{l-1}(u) \subseteq S_i^l(u)$. Moreover, if $w \in V$ is in a cycle of length at most g' with some vertex in $S_i^{l-1}(u)$ such that the cycle contains no off vertex, then we have $w \in S_i^l(u)$.*

Our algorithm will do at most k modified Dijkstras from u, where we prove that the set of vertices visited in the lth Dijkstra is $S_i^l(u)$. After performing each Dijkstra we decide if we continue to the next modified Dijkstra from u or proceed to another on vertex.

Suppose that at some point we know that the set $S_i^{l-1}(u)$ is the set of vertices visited in the $(l-1)$th modified Dijkstra, and we want to proceed to the lth Dijkstra. Our new goal is the following: We want to catch a minimum cycle of length $\leq g'$ passing through S_i^l with no off vertex. For this, we do the lth modified Dijkstra form u as follows.

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether $d(x,r) \leq (l+l^2\varepsilon)g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$ for all $r \in R_i^{j'}(u)$ for all j'. If x does not satisfy this condition, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x,y), and if $d[y] > d[x] + w(x,y)$, we update $d[y] = d[x] + w(x,y)$. We stop when the vertex u extracted from the heap has $d[u] > (2l-1)g'/2$.

It is clear by definition that the set of vertices that this modified Dijkstra visits is $S_i^l(u)$. Now if $|S_i^l(u)| \leq c(|S_i^{l-1}(u)|.n^\alpha)^{\frac{1}{1+\alpha}}$ for some constant c, we recurse on the subgraph induced by $S_i^l(u)$, i.e. $G[S_i^l(u)]$, to get an $2k + O(\varepsilon)$ approximation of the girth on this subgraph. The girth in $G[S_i^l(u)]$ is a lower bound on the minimum cycle of length $\leq g'$ passing through $S_i^{l-1}(u)$ with no off vertex. So we take this value as one of our estimates and we mark all vertices of $S_i^{l-1}(u)$ as off. The running time of this recursion is $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}|S_i^l(u)|^{1+\alpha})$ as the average degree is $O(\frac{m}{n})$. Since we process $S_i^{l-1}(u)$ vertices in this running time, we spend $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}.|S_i^l(u)|/|S_i^{l-1}(u)|) \leq \tilde{O}(\frac{m}{n}.\frac{\log M}{\varepsilon}.n^\alpha)$ for each vertex.

Note that $|S_i^k(u)| \leq O(n^{1-\alpha})$. This is because for all $x \in S_i^k \cap B^j(u)$ and for all $r \in R_i^j(u)$, we have that $d(x,r) \leq (k + k^2\varepsilon)g' + (1+\epsilon)^{j+1} - (1+\epsilon)^j \leq (k + k^2\varepsilon)g' + \epsilon(1+\epsilon)^j \leq (k+k^2\varepsilon)g' + \epsilon(2k-1)g'/2 \leq (k+k^2\epsilon+k\varepsilon)g' = \beta g'$. So $S_i^k(u)$ is a subset of all vertices in $B^j(u)$ with distance at most $\beta g'$ from all $r \in R_i^j(u)$, and so by Lemma 5.4.1 it has size at most $O(n^{1-\alpha})$.

When all vertices are marked off, we take the minimum value of all the estimates as our estimate for g.

Since we have that $S_i^l(u) \leq O(n^{1-\alpha})$, if we set α appropriately, for some $l < k$ we have that $|S_i^{l+1}(u)| \leq (|S_i^l(u)|.n^\alpha)^{\frac{1}{1+\alpha}}$. For $k = 1$, setting $\alpha = 1/2$ gives us the algorithm of Theorem 5.4.1. For $k > 1$, the following lemma determines α. The proof of the lemma can be found in Section 5.6.

Lemma 5.4.5. *For $k > 1$, let the sets S_i^l for $l = 1, \ldots, k$ be such that $S_i^l \subseteq S_i^{l+1}$ for all $l < k$, $S_i^1 \geq n^\alpha$ and $S_i^k \leq O(n^{1-\alpha})$. Let $0 < \alpha < 1$ satisfy $\alpha(1+\alpha)^{k-1} = 1 - \alpha$. Then there is $l < k$ and a constant c such that $|S_i^{l+1}| \leq c(|S_i^l|.n^\alpha)^{\frac{1}{1+\alpha}}$.*

Note that for $k = 2$, Lemma 5.4.5 sets $\alpha = \sqrt{2} - 1$ and thus gives us the algorithm of Theorem 5.4.2.

5.4.3 Removing the $\log M$ factor

In this subsection we show how to remove the $\log M$ factor in the running times of our algorithms where M is the maximum edge weight, resulting in strongly polynomial algorithms.

Assume that we have a $(2k+\epsilon)$-approximation algorithm A for the girth in $\tilde{O}(mn^{\alpha_k}\log M/\epsilon)$

running time for some $0 \leq \alpha_k \leq 1$. We want to obtain an algorithm that gives us a $(2k + O(\epsilon))$-approximation of the girth in $\tilde{O}(mn^{\alpha_k}/\epsilon)$ time.

First, suppose that we know the smallest number W such that there is a cycle with all edge weights at most W. Then by the definition of W we have that $W \leq g$ and $g \leq nW$. Moreover, note that the edges of any cycle with total weight at most $(2k + O(\epsilon))g$ cannot have weights more than $3knW$, so we can remove any edge with weight more than $3knW$. Let $R = W\epsilon/n$. Let H be a copy of G, with the weight $w_G(e)$ of the edge e replaced by $w_H(e) = \lfloor w_G(e)/R \rfloor$. Note that the weights of H are bounded by $O(n^2/\epsilon')$.

Now consider a cycle C in G. Suppose that C has n_C edges. Let $w_G(C)$ and $w_H(C)$ be the sum of the edge-weights of C in G and H respectively. For any edge e, we have that $w_G(e) - R \leq R \cdot w_H(e) \leq w_G(e)$. This gives us

$$w_G(C) - Rn_C \leq R \cdot w_H(C) \leq w_G(C). \tag{5.1}$$

Note that if C is the cycle with minimum length in G, then we have that $Rg' \leq Rw_H(C) \leq g$, where g' is the girth of H.

Now we apply algorithm A on H, which takes $\tilde{O}(mn^{\alpha_k}/\epsilon)$ time. Suppose that it outputs a cycle C such that $g' \leq w_H(C) \leq (2k + \varepsilon)g'$. Since $g \geq Rg'$ and by equation 5.1 we have $w_G(C) \leq Rw_H(C) + Rn_c \leq (2k + \varepsilon)Rg' + Rn \leq (2k + 2\varepsilon)g$. The last inequality uses the fact that $Rn = W\varepsilon \leq g\varepsilon$.

It suffices to show how we obtain W. We sort the edges of G in $\tilde{O}(m)$ time, so that the edge weight are $w_1 \leq \ldots \leq w_m$. We find W using binary search and DFS as follows: Suppose that we are searching for W in the interval $w_i \leq \ldots \leq w_j$ for $1 \leq i \leq j \leq m$. Let $r = (i + j)/2$, we remove all the edges with weight more than w_r and then do DFS in the remaining graph to see if it has a cycle. If it does, we update $j = r$, otherwise we update $i = r$. Note that this process takes $\tilde{O}(m)$ time.

5.5 Hardness

Our hardness result is based on the following k-Cycle hypothesis (see [LVW18, AHR$^+$19, PVW19]).

Hypothesis 5 (k-Cycle Hypothesis)**.** *In the word-RAM model with $O(\log m)$ bit words, for any constant $\varepsilon > 0$, there exists a constant integer k, so that there is no $O(m^{2-\varepsilon})$ time algorithm that can detect a k-cycle in an m-edge graph.*

All known algorithms for detecting k-cycles in directed graphs with m edges run at best in time $m^{2-c/k}$ for various small constants c [YZ04, AYZ97, LVW18, DDV19], even using powerful tools such as fast matrix multiplication. Refuting the k-Cycle Hypothesis above would resolve a big open problem in graph algorithms. Moreover, as shown by Lincoln et al. [LVW18] any algorithm for directed k-cycle detection, for k-odd, with running time $O(mn^{1-\varepsilon})$ for $\varepsilon > 0$ whenever $m = \Theta(n^{1+2/(k-1)})$ would imply an $O(n^{k-\delta})$ time algorithm for k-clique detection for $\delta > 0$. If the cycle algorithm is "combinatorial", then the clique algorithm would be "combinatorial" as well, and since all known $O(n^{k-\delta})$ time k-clique algorithms use fast matrix multiplication, such a result for k-cycle would be substantial.

We will show that under Hypothesis 5, approximating the girth to a factor better than 2 would require $mn^{1-o(1)}$ time, and so up to this hypothesis, our approximation algorithm is optimal for the girth in unweighted graphs.

Theorem 5.5.1. *Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph. Then for every constant k, one can detect whether an m-edge directed graph contains a k-cycle, in $O(m^{2-\varepsilon})$ time, and hence the k-Cycle Hypothesis is false.*

Proof. The proof is relatively simple. Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph.

Now let $k \geq 3$ be any constant integer and let G be an n-node, m-edge graph. First randomly color each vertex of G with one of k colors. Let C be any k-cycle in G. With probability $1/k^k$, for each $i = 0, \ldots, k-1$, the ith vertex of C is colored i.

Now, for each $0 \leq i \leq k-1$, let V_i be the vertices colored i. For each vertex $u \in V_i$, and each directed edge (u, v) out of u, keep (u, v) if and only if $v \in V_{i+1}$ where the indices are taken mod k. This builds a graph G' which is a subgraph of G and contains a k-cycle if G does with probability $\geq 1/k^k$.

G' has two useful properties. (1) Any cycle of G' has length divisible by k, and (2) (which follows from (1)) the girth of G' is k if G' contains a k-cycle and it is $\geq 2k$ otherwise.

As G' has at most m edges (it is a subgraph of G), we can use our supposedly fast $2-\delta$ approximation algorithm to determine whether the girth is k or larger in $O(m^{2-\varepsilon})$ time. By iterating the construction $O(k^k \log n)$ times, we get that the k-cycle problem in G can be solved in $\tilde{O}(k^k m^{2-\varepsilon})$ time, and as k is a constant, we are done. The approach can be derandomized with standard techniques (e.g. [AYZ95]). □

5.6 Omitted proofs

Proof of Lemma 5.2.2. We start with a simple claim which is proved at the end:

Claim 7. *Let $q \geq 2$ be an integer. Let $L \geq 1$ be an integer. There is a directed rooted tree with $\leq 3L$ nodes, L leaves, with every node of outdegree $\leq q$ and such that every root to leaf path has the same length $\lceil \log_q L \rceil$.*

The idea of the proof is to represent every edge (u, v) of G by a t-length path from u to v via some auxiliary nodes, so that the total number of auxiliary nodes is small, and the degree of every node is small as well.

Let $q = \max\{2, \lceil m/n \rceil\}$. Consider some node u and its out-neighbors $v_1, \ldots, v_{deg(u)}$. Remove the edge from u to v_j for each j. Let d be the smallest power of q that is larger than $deg(u)$, i.e. $q^{d-1} < deg(u) \leq q^d$ and $d = \lceil \log_q deg(u) \rceil$.

Using the construction of Claim 7, create a partial q-ary tree T_u of at most $3\lceil deg(u)/q \rceil$ aux-

iliary nodes, with $\lceil deg(u)/q \rceil$ leaves, and so that the leaves are all at depth $\lceil \log_q(\lceil deg(u)/q \rceil) \rceil = d - 1$. Then, make the original out-neighbors $v_1, \ldots, v_{deg(u)}$ of u children of the leaves of T_u so that every leaf of T_u has at most q children.

Let $t = \lceil \log_q n \rceil$. Notice that since $deg(u) < n$, we have that $d \leq t$. If $d = t$, set $r_u = u$. If $d < t$, add another $t - d$ new auxiliary nodes $u_1, \ldots, u_{t-d}$, connect them into a directed path $u_1 \to \ldots \to u_{t-d}$ and then add the edge (u_{t-d}, r_u). Let $u_1 = u$. This completes a directed tree $T(u)$ rooted at u such that the number of edges on any root-to-leaf path is t. See Figure 5-3 for example trees.

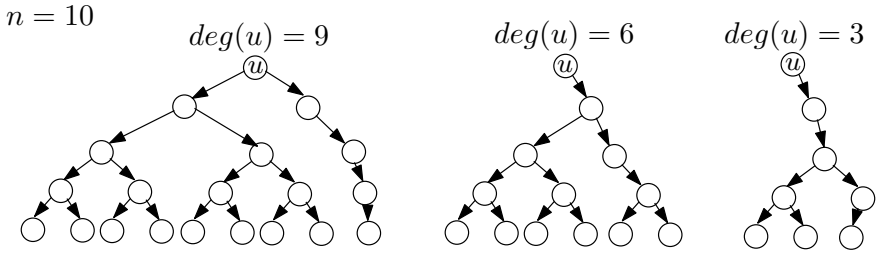

Figure 5-3: Here we give examples of the construction of $T(u)$ when $t = 4$ (e.g. when $n = 10$), and when the out-degree of u is $9, 6$ and 3.

The obtained graph is unweighted. Notice that for each j, the original edge (u, v_j) is replaced by a path in $T(u)$ of length exactly t, and hence for every $u, v \in V$, $d_{G'}(u, v) = t d_G(u, v)$.

Since the auxiliary nodes do not create new cycles, any cycle C in G' must correspond to a cycle in G that can be obtained from C by replacing each subpath between nodes of V with the edge corresponding to it, and the girth of G' is exactly t times the girth of G. Similarly, if we had a c-roundtrip spanner over the new graph G', we can obtain a c-spanner of G by replacing each auxiliary path between vertices of V with the corresponding edge of G. The number of edges does not increase.

Every vertex in the new graph has out-degree at most q. If we would like the in-degrees to be bounded by q as well, we can perform the same procedure (with edge directions reversed) on the in-neighborhoods.

The total number of auxiliary vertices added to $T(u)$ is

$$t + 3\lceil deg(u)/q \rceil \leq \log n + 3 + 3n \cdot deg(u)/m.$$

Over all vertices the total number of auxiliary vertices is at most

$$\sum_{u \in V} \left[\log n + 3 + 3\frac{n \cdot deg(u)}{m} \right] = O(n \log n).$$

□

Proof of Claim 7. It is easy to see that if $q^{d-1} < L \leq q^d$, we can always take a complete q-ary tree on q^d leaves and remove enough leaves until we only have L. This would definitely achieve the depth requirement. However, if we are not careful, we might have more than $3L$ nodes in the tree. Here we do a more fine-tuned analysis to have both the size and the depth of the tree under control.

Let us consider the q-ary representation of L: $L = a_{d-1}q^{d-1} + a_{d-2}q^{d-2} + \ldots + a_0$. Here each $a_j \in \{0, \ldots, q-1\}$.

We will show inductively how to build a rooted tree with out-degree $\leq q$ so that every leaf is at depth d. The base case is when $d = 1$, so that $L = a_0$. Then we simply have a root with a_0 children.

Suppose that $d > 1$. Let us assume that for every integer $\ell < q^{d-1}$ we can create a rooted tree with outdegree at most q, ℓ children all of depth $d-1$. Consider now $L = a_{d-1}q^{d-1} + a_{d-2}q^{d-2} + \ldots + a_0$. Create a root r with $a_{d-1} + 1$ children. The first a_{d-1} children are roots of complete q-ary trees with q^{d-1} leaves. These have depth $d-1$, and together with the edge from r to their roots, they have depth d. The last child of r is a root of a directed tree formed inductively to have $a_{d-2}q^{d-2} + \ldots + a_0$ leaves (all of depth $d-1$) and out-degree $\leq q$. As $d = \lceil \log_q L \rceil$, we are done with the depth argument.

As for the number of nodes in the tree, we prove it by induction. The base case is when $d = 1$, so $L = a_0$ and the number of nodes in the tree is $L + 1 \leq 3L$ (as $L \geq 1$). Suppose the number of nodes in the tree is $\leq 3L$ for all $L < q^{d-1}$. Consider $L = a_{d-1}q^{d-1} + a_{d-2}q^{d-2} + \ldots + a_0$. The number of nodes in the tree is then at most

$$1 + a_{d-1}\frac{q^d}{q-1} + 3 \cdot (L - a_{d-1}q^{d-1}),$$

where 1 is for the root, $\frac{q^d}{q-1}$ is the number of nodes of a complete q-ary tree with q^{d-1} leaves and $(L - a_{d-1}q^{d-1})$ is the number of leaves left after the first $a_{d-1}q^{d-1}$ are covered by the complete q-ary trees. The expression above is

$$\leq 3L + 1 + \frac{a_{d-1}}{q-1}(q^d - 1 - 3q^d + 3q^{d-1}) \leq 3L + \frac{a_{d-1}}{q-1}(-2q^d + 3q^{d-1} + q - 2).$$

Now, since $d \geq 2$ ($d = 1$ was the base case), and $q \geq 2$, we have that

$$-2q^d + 3q^{d-1} + q - 2 = q^{d-1}(3 - 2q) + q - 2 \leq -q^{d-1} + q - 2 \leq -2 < 0,$$

and hence the number of nodes is $\leq 3L$.

□

Proof of Lemma 5.4.1. First suppose that we are able to pick a random sample $R_i^j(u)$ of $c \log n$ vertices from $\bar{Z}_i^j(u)$. Then we can define $B_i^j(u) = \{z \in \bar{Z}_i^j(u) \mid d(z, y) \leq d,\ \forall y \in R_i^j(u)\}$.

Consider any $s \in V$ with at least $0.2|\bar{Z}_i^j(u)|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq d$. As $\bar{Z}_i^j(u) \geq 10n^{1-\alpha}$ (as otherwise we would be done and the sampled vertices would work), $0.2|\bar{Z}_i^j(u)| \geq 2n^{1-\alpha}$, and so with high probability, for s with the property above, Q contains some q with $d(s, q), d(q, s) \leq d$, and so $s \in V_i'$. Thus with high probability, for every $s \in Z_i$, there are at most $0.2|\bar{Z}_i^j(u)|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq d$.

We will iterate this sampling process until we arrive at a subset of $\bar{Z}_i^j(u)$ that is smaller than $10n^{1-\alpha}$ that contains all the vertices in $Z_i^j(u)$ with distance at most d to all the sampled vertices, as follows:

Let $Z_{i,0}^j(u) = \bar{Z}_i^j(u)$. For each $k = 0, \ldots, 2 \log n$, let $R_{i,k}^j(u)$ be a random sample of $O(\log n)$ vertices of $Z_{i,k}^j(u)$. Define $Z_{i,k+1}^j(u) = \{z \in \bar{Z}_i^j(u) \mid d(z, y) \leq d \forall y \in \cup_{\ell=0}^k R_{i,\ell}^j(u)\}$. We get that for each k, $|Z_{i,k}^j(u)| \leq 0.8^k|\bar{Z}_i^j(u)|$ so that at the end of the last iteration, $|Z_{i,2\log n}^j| \leq 10n^{1-\alpha}$. Hence we get the set $Z_i'^j(u)$ that we are after as $Z_{i,2\log n}^j$.

It is not immediately clear how to obtain the random sample $R_{i,k}^j(u)$ from $Z_{i,k}^j(u)$ as $Z_{i,k}^j(u)$ is unknown. We do it in the following way. For each i, j, k we independently obtain a random sample $S_{i,j,k}$ of Z_i by sampling each vertex independently with probability $p = 100 \log n/n^\alpha$. For

each of the (in expectation) $O(n^\alpha \log^4(n))$ vertices in the sets $S_{i,j,k}$ we run Dijkstra's to and from them, to obtain all their distances.

Now, for a fixed i, $j \leq i$, k, to obtain the random sample $R^j_{i,k}(u)$ of the unknown $Z^j_{i,k}(u)$, we assume that we already have $R^j_{i,\ell}(u)$ for $\ell < k$, and define

$$T^j_{i,k}(u) = \{s \in S_{i,j,k} \mid s \in \bar{Z}^j_i(u) \text{ and } d(s,y) \leq d \forall y \in \cup_{\ell<k} R^j_{i,\ell}(u).\}$$

Forming the set $T^j_{i,k}(u)$ is easy since we have the distances $d(s,v)$ for all $s \in S_{i,j,k}$ and $v \in V$, so we can check whether $s \in \bar{B}^j(u)$ and $s \in Z_i$ (thus checking that $s \in \bar{Z}^j_i(u)$) and $d(s,y) \leq d \forall y \in \cup_{\ell<k} R^j_{d,\ell}(u)$ in polylogarithmic time for each $s \in S_{i,j,k}$.

Now since $S_{i,j,k}$ is independent from all our other random choices, $T^j_{i,k}(u)$ is a random sample of $Z^j_{i,k}(u)$ essentially created by selecting each vertex with probability p. If $Z^j_{i,k}(u) \geq 100n^{1-\alpha}$, with high probability, $T^j_{i,k}(u)$ has at least $10 \log n$ vertices so we can pick $R^j_{i,k}(u)$ to be a random sample of $10 \log n$ vertices of $T^j_{i,k}(u)$, and they will also be a random sample of $10 \log n$ vertices of $Z^j_{i,k}(u)$. So we let $R^j_i(u) = \cup_k R^j_{i,k}(u)$, which has size $O(\log^2 n)$. The running time of this sampling procedure comes from the Dijkstras we perform from $S_{i,j,k}$s and hence it is $\tilde{O}(mn^\alpha)$. $\square$

Proof of Lemma 5.4.4. First it is clear that for all $v \in S^{l-1}_i(u)$ and each $r \in R^{j'}_i(u)$, we have $d(v,r) \leq ((l-1) + \varepsilon(l-1)^2)g' + (1+\epsilon)^{j'+1} - (1+\epsilon)^j \leq (l + \varepsilon l^2)g' + (1+\epsilon)^{j'+1} - (1+\epsilon)^j$, and so $v \in S^l_i(u)$.

Now suppose that for $v \in S^{l-1}_i(u)$ and $w \in V$, we have $d(v \leftrightarrows w) \leq g'$. Suppose that $v \in B^{j_1}(u)$, $w \in B^{j_2}(u)$. For $r \in R^{j_3}_i$, we have

$$d(w,r) \leq d(w,v) + d(v,r) \leq d(w,v) + ((l-1) + (l-1)^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_1}.$$

If $j_1 \geq j_2$, then since $d(w,v) \leq g'$, we have

$$d(w,r) \leq (l + (l-1)^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_1} \leq (l + l^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_2}.$$

If $j_1 < j_2$, then we have $d(w,v) \leq g' - d(v,w) \leq g' - [(1+\epsilon)^{j_2} - (1+\epsilon)^{j_1+1}]$. Using the fact

that $(1+\varepsilon)^{j1} \leq (l-1)g'$ we have that

$$\begin{aligned} d(w,r) &\leq (l+(l-1)^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_2} + \varepsilon(1+\varepsilon)^{j_1} \\ &\leq (l+(l-1)^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_2} + \varepsilon(l-1)g' \\ &\leq (l+l^2\varepsilon)g' + (1+\epsilon)^{j_3+1} - (1+\epsilon)^{j_2}. \end{aligned}$$

We have that $d(u,w) \leq d(u,v) + d(v,w) \leq (2l-3)g'/2 + g' \leq (2l-1)g'/2$. If the path vw contains no off vertex, then there uw path passing through v contains no off vertex and so there is a path of length at most $(2l-1)g'/2$ with all vertices. So $w \in S_i^l(u)$. □

Proof of Lemma 5.4.5. Assume that $|S_i^k| \leq Cn^{1-\alpha}$ for some constant $C > 1$. Suppose that for all $l < k$, we have that $|S_i^{l+1}| > C(|S_i^l|n^\alpha)^{\frac{1}{1+\alpha}}$. Using $|S_i^1| > n^\alpha$, we have that $|S_i^k| > Cn^{\frac{\alpha}{(1+\alpha)^{1-k}}+\alpha\sum_{j=1}^{k-1}\frac{1}{(1+\alpha)^j}}$. Since $1-\frac{1}{(1+\alpha)^{k-1}} = \alpha\sum_{j=1}^{k-1}\frac{1}{(1+\alpha)^j}$ and we have that $\alpha(1+\alpha)^{k-1} = 1-\alpha$ iff $\alpha + \frac{\alpha-1}{(1+\alpha)^{k-1}} = 0$, we obtain that $|S_i^k| > Cn^{1-\alpha}$, which is a contradiction. □

5.7 Most recent results on this problem

Following our results in this chapter, Chechik and Lifshitz [CL21] made the following improvements. They provide a $\tilde{O}(n^2)$ time algorithm that computes a 2-multiplicative approximation of the girth of an n-node m-edge directed graph with non-negative edge weights. They also provide an additional algorithm that computes a 2-multiplicative approximation of the girth in $\tilde{O}(m\sqrt{n})$ time (recall that we give a $2+\epsilon$ approximation in this running time). Their results naturally provide algorithms for improved constructions of 4-roundtrip spanners, the analog of spanners in directed graphs.

Chapter 6

Finding Long Shortest Paths and Short Cycles: Variants

In this chapter we study variants of distance problems. First, we study a variant that imposes constraints on which pairs of points we are interested in as opposed to considering *all* pairs of points. Second, we study a symmetric distance measure for directed graphs. For both of these variants we provide approximation algorithms and conditional lower bounds. We provide sufficient background in each subsection and hence they can be read independently.

6.1 Bichromatic and ST distance problems

This section was written with authors Virginia Vassilevska Williams, Nikhil Vyas and Nicole Wein and focuses on the first type of variants of distance problems.

Some of the most fundamental and well-studied graph parameters are the Diameter (the largest shortest paths distance) and Radius (the smallest distance for which a "center" node can reach all other nodes). The natural and important ST-variant considers two subsets S and T of the vertex set and lets the ST-diameter be the maximum distance between a node in S and a node in T, and the ST-radius be the minimum distance for a node of S to reach all nodes of T. The *bichromatic* variant is the special case in which S and T partition the vertex set.

In this section of this chapter we present a comprehensive study of the *approximability* of ST and Bichromatic Diameter, Radius, and Eccentricities, and variants, in graphs with and without directions and weights. We give the first nontrivial approximation algorithms for most of these problems, including time/accuracy trade-off upper and lower bounds. We show that nearly *all* of our obtained bounds are tight under the Strong Exponential Time Hypothesis (SETH), or the related Hitting Set Hypothesis.

For instance, for Bichromatic Diameter in undirected weighted graphs with m edges, we present an $\tilde{O}(m^{3/2})$ time [1] $5/3$-approximation algorithm, and show that under SETH, neither the running time, nor the approximation factor can be significantly improved while keeping the other unchanged.

6.1.1 Introduction

A fundamental and very well studied problem in algorithms is the Diameter of a graph, where the output is the largest (shortest path) distance over all pairs of vertices. Over the years many different algorithms have been developed for the problem, both in theory (e.g. [ACIM99, PRT12, RV13, CLR+14, BRS+18]) and in practice (e.g. [CGLM12, TK11, MLH09]).

A very natural variant is the so called ST-Diameter problem [BRS+18]: given a graph and two subsets S and T of its vertex set, determine the largest distance between a vertex of S and a vertex of T. In the Subset version of ST-Diameter, we have $S = T$. Bichromatic Diameter is the version of ST-Diameter for which S and T partition the vertex set. Besides Diameter, the Radius (the smallest distance for which a "center" node can reach all other nodes) and Eccentricities (the largest distance out of every vertex) problems are also very well studied, and analogous ST, Subset, and Bichromatic versions are easy to define.

All of these parameters are simple to compute by computing all pairwise distances in the graph, i.e. by solving All-Pairs Shortest Paths (APSP). In sparse n-node graphs, where the number of edges m is $\tilde{O}(n)$, APSP still needs $\Omega(n^2)$ time, as this is the size of the output, whereas it is apriori unclear whether this much time is needed for computing the Diameter, Radius and Eccentricities

[1] $\tilde{O}$ notation hides polylogarithmic factors.

or their ST and bichromatic variants, as the output is small.

A related extremely well-studied problem in computational geometry is Bichromatic Diameter on point sets (commonly known as Bichromatic Farthest Pair), where one seeks to determine the farthest pair of points in a given set of points in space (see e.g. [Yao82, DG04, Wil18, AESW91, KI92]). Another related problem is the Subset version of spanners (e.g. [Kle06, CGK13]), as well as the ST version of spanners (e.g. [CE06, Kav17]). Furthermore, the ST, Subset, and Bichromatic versions of many problems have been of great interest; for instance Steiner Tree, Subset TSP, and a number of problems in computational geometry such as Bichromatic Matching (e.g. [Ind07]) and Bichromatic Line Segment Intersection (e.g. [CEGS94]).

There are several known approximation algorithms for the standard version of Diameter, most of which have been developed in the last 6 years. Trivially, running Dijkstra's algorithm from an arbitrary vertex gives a simple $\tilde{O}(m)$ time 2-approximation algorithm for directed and weighted graphs. Non-trivial algorithms achieve an improved approximation factor with an increased runtime: Building on Aingworth et al. [ACIM99], Roditty and Vassilevska W. [RV13] showed for instance that an "almost" 1.5 approximation for Diameter can be computed in $\tilde{O}(m\sqrt{n})$ time in m-edge n-vertex directed weighted graphs—the approximation factor is 1.5 if the Diameter is divisible by 3, and there is a slight additive error otherwise. Chechik et al. [CLR$^+$14] gave a true 1.5 approximation at the expense of increasing the runtime to $\tilde{O}(mn^{2/3})$, and Cairo, Grossi and Rizzi [CGR16] generalized the approach giving an $\tilde{O}(mn^{1/(k+1)})$ time, "almost" $2-1/2^k$ approximation algorithm for all $k \geq 1$ which works only in undirected graphs.

In STOC'18, Backurs et al. [BRS$^+$18] gave the first non-trivial approximation algorithms for ST-Diameter: an $\tilde{O}(m^{3/2})$ time 2-approximation and an $\tilde{O}(m)$ time 3-approximation. They also showed that these algorithms cannot be improved significantly, unless the Strong Exponential Time Hypothesis (SETH) fails. Backurs et al. did not provide algorithms for ST-Eccentricities or ST-Radius, and they did not study the natural Subset and Bichromatic versions. They also only focused on undirected graphs.

We study the following natural and fundamental questions:

How well can ST-Eccentricities and ST-Radius be approximated? Are any interesting

approximation algorithms possible for directed graphs for any of the ST-variants? Does the approximability of the problems change when one turns to the Subset versions in which $S = T$, or the Bichromatic versions in which S and T are required to partition the vertex set?

Our Results

We present a comprehensive study of the approximability of the ST, Subset and Bichromatic variants of the Diameter, Radius and Eccentricities problems in graphs, both with and without directions and weights. We obtain the first non-trivial approximation algorithms for most of these problems, including time/accuracy trade-off upper and lower bounds. We show that nearly *all* of our approximation algorithms are tight under SETH (or under the related Hitting Set Hypothesis for Radius). Additionally, we study a parameterized version of these problems.

Our results are summarized in Tables 6.1-6.4.

All our algorithms in m-edge, n-node graphs, run in $\tilde{O}(m^{3/2})$ time or in $\tilde{O}(m\sqrt{n})$ time when a small additive error is allowed. For sparse graphs the $m^{3/2}$ runtime beats the fastest APSP algorithms [Cha07, PR05, Pet04] as they run in $\tilde{O}(mn)$ time. The $m\sqrt{n}$ time of the algorithms that allow small additive error beat the APSP algorithms for every graph sparsity.

Bichromatic Diameter and Radius. Our first contribution is an algorithm with the same running time as the 2-approximation ST-Diameter algorithm of [BRS$^+$18], achieving a better, $5/3$ approximation for Bichromatic Diameter. In other words, when S and T partition the vertex set of the graph, ST-Diameter can be approximated much better! Moreover, we show that under SETH, neither the runtime nor the approximation factor of our algorithm can be improved. The result is summarized in Theorem 6.1.1 below, and proven in Theorems 6.1.7 and 6.1.25.

Theorem 6.1.1. *There is a randomized $\tilde{O}(m^{3/2})$ 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 $3D_{ST}/5 \leq D' \leq D_{ST}$ with high probability, where D_{ST} is the ST-Diameter of G.*

Moreover, if there is an $O(m^{3/2-\varepsilon})$ time $5/3$-approximation algorithm for some $\varepsilon > 0$, or if there is an $O(m^{2-\varepsilon})$ time $(5/3-\varepsilon)$-approximation algorithm for the problem, then SETH is false.

[2] *with high probability* means with probability at least $1 - 1/n^c$ for all constants c.

	Upper Bounds			Lower Bounds	
Problem	Runtime	Approx.	Comments	Runtime	Approx.
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***	$m^{2-o(1)}$	$3/2 - \delta$
Radius	$O(m + n \log n)$	almost 2	unweighted		
	$\tilde{O}(m\sqrt{n})$	almost $5/3$	unweighted, nearly **tight***	$m^{2-o(1)}$	$5/3 - \delta$
	$\tilde{O}(m^{3/2})$	$5/3$	weighted, **tight***	"	"
	$O(m\|B\|)$	almost $3/2$	unweighted, **tight***	$m^{2-o(1)}$	$3/2 - \delta$
Eccentricities	$O(m + n \log n)$	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**	"	"
	$O(m\|B\|)$	almost $5/3$	unweighted, **tight***	$m^{2-o(1)}$	$5/3 - \delta$

Table 6.1: Bichromatic undirected results. All of our parameterized algorithms and near-linear time algorithms, except for directed Subset Radius and Eccentricities, are deterministic. The rest are randomized and work with high probability[2]. Our lower bounds for Diameter and Eccentricities are under SETH and our lower bounds for Radius are under the Hitting Set (HS) Hypothesis, defined later. All of our lower bounds hold even for unweighted graphs. The trade-off lower bounds in terms of k hold for any integer $k \geq 2$. δ is any constant > 0. B and B' are parameters defined in our parameterized algorithms. The lower bound constructions for the parameterized algorithms have $|B| = \tilde{O}(1)$
* Multiplicative approximation factor is tight, but not runtime.

We also obtain an $\tilde{O}(m\sqrt{n})$ time algorithm that achieves an "almost" $5/3$-approximation: the guarantee for unweighted graphs is $3D_{ST}/5 - 6/5 \leq D' \leq D_{ST}$. We also obtain a near-linear time algorithm for weighted graphs that returns an estimate D' with $D_{ST}/2 - W/2 \leq D' \leq D_{ST}$ where W is the minimum weight of a $S \times T$ edge. Using our general theorem 6.1.25, we get that this result is also essentially tight, as a $(2-\varepsilon)$-approximation for $\varepsilon > 0$ running in near-linear time would refute SETH.

To obtain our improvements for Bichromatic Diameter over the known ST-Diameter algo-

	Upper Bounds			Lower Bounds	
Problem	Runtime	Approx.	Comments	Runtime	Approx.
Diameter	$\tilde{O}(m^{3/2})$	2	weighted, **tight***	$m^{2-o(1)}$	$2-\delta$
	$O(m\|B'\|)$	almost $3/2$	unweighted, **tight***	$m^{2-o(1)}$	$3/2-\delta$
Radius	N/A	N/A	weighted, **tight**	$m^{2-o(1)}$	any finite
Eccentricities	N/A	N/A	weighted, **tight**	$m^{2-o(1)}$	any finite

Table 6.2: Bichromatic directed results. See caption of Table 6.1.

	Upper Bounds			Lower Bounds	
Problem	Runtime	Approx.	Comments	Runtime	Approx.
Diameter [BRS$^+$18]	$O(m+n\log n)$	3	weighted, **tight**	$m^{1+o(1)}$	$3-\delta$
	$\tilde{O}(m\sqrt{n})$	almost 2	unw, nearly **tight**	$m^{\frac{k}{k-1}-o(1)}$	$3-2/k-\delta$
	$\tilde{O}(m^{3/2})$	2	weighted, **tight**	"	"
Radius	$O(m+n\log n)$	3	weighted		
	$\tilde{O}(m\sqrt{n})$	almost 2	unw, nearly **tight***	$m^{2-o(1)}$	$2-\delta$
	$\tilde{O}(m^{3/2})$	2	weighted, **tight***	"	"
Eccentricities	$O(m+n\log n)$	3	weighted, **tight**	$m^{1+o(1)}$	$3-\delta$ [BRS$^+$18]
	$\tilde{O}(m\sqrt{n})$	almost 2	unw, nearly **tight**	$m^{\frac{k}{k-1}-o(1)}$	$3-2/k-\epsilon$ [BRS$^+$18]
	$\tilde{O}(m^{3/2})$	2	weighted, **tight**	"	"

Table 6.3: ST undirected results. See caption of Table 6.1. unw means unweighted.

rithms, 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.

We explain our technique in more detail for the bichromatic diameter problem, and similar ideas are used for our algorithms for the other problems. Let $s^* \in S$ and $t^* \in T$ be end-points of an ST-Diameter path. Similarly to prior Diameter algorithms, our goal is to run Dijkstra's algorithm from some $s \in S$ which is close to s^*, and hence far from t^*, or from some $t \in T$

	Upper Bounds			Lower Bounds	
Problem	Runtime	Approx.	Comments	Runtime	Approx.
Diameter	$\tilde{O}(m)$	2	weighted, directed, **tight**	$m^{2-o(1)}$	$2-\delta$
Radius	$\tilde{O}(m)$	2	weighted, undirected, **tight**	$m^{2-o(1)}$	$2-\delta$
	$\tilde{O}(m/\delta)$	$2+\delta$	weighted, directed, **tight** up to an additive δ	"	"
Eccentricities	$\tilde{O}(m/\delta)$	$2+\delta$	weighted, directed, **tight** up to an additive δ	$m^{2-o(1)}$	$2-\delta$

Table 6.4: Subset results. See caption of Table 6.1.

which is close to t^* and hence far from s^* (by the triangle inequality). Our $5/3$-approximation algorithms are a delicate 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^*. If (1) is not good enough, in theme (2) 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. 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. In order to better separate the ideas in our algorithms, we explain them in several steps, where Theme (1) can be seen in the first steps and Theme (2) appears towards the last steps.

Following a similar approach to our Bichromatic Diameter algorithms, we develop similar algorithms for Bichromatic Radius. First, we give a simple near-linear time almost 2-approximation algorithm, and then we adapt the $5/3$-approximation for Bichromatic Diameter to also give a $5/3$-approximation for Bichromatic Radius. Moreover, we show that any better approximation factor requires essentially quadratic time, under the Hitting Set (HS) Hypothesis of [AVW16] (see also [GIKW17]).

Theorem 6.1.2. *There is a randomized $\tilde{O}(m^{3/2})$ 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 R' such that $R_{ST} \leq R' \leq 5R_{ST}/3$ with high probability, where R_{ST} is the ST-Radius of G. Moreover, if there is a $5/3-\varepsilon$ approximation algorithm running in $O(m^{2-\delta})$ time for any $\varepsilon, \delta > 0$,*

then the HS Hypothesis is false.

Similarly to the Bichromatic Diameter algorithm, if one is satisfied with a slight additive error, one can improve the runtime to $\tilde{O}(m\sqrt{n})$.

ST-Eccentricities and ST-Radius. Prior work only considered ST-Diameter but did not consider the more general ST-Eccentricities problem in which one wants to approximate for every $s \in S, \varepsilon_{ST}(s) := \max_{t \in T} d(s,t)$.

Here we show that one can achieve exactly the same approximation factors for ST-Eccentricities as for ST-Diameter. Since any conditional lower bound for ST-Diameter also applies for the ST-Eccentricities problem, the algorithms we obtain are conditionally optimal, similarly to the ST-Diameter algorithms in [BRS$^+$18]. Interestingly, we show that the same conditional lower bounds apply for Bichromatic Eccentricities (Proposition 6), and therefore our ST-Eccentricities algorithms are optimal even for the Bichromatic case.

Theorem 6.1.3. *There is a randomized $\tilde{O}(m^{3/2})$ time algorithm, that given an undirected graph $G = (V, E)$ with nonnegative integer edge weights and $S, T \subseteq V$, can output for every $s \in S$, an estimate $\varepsilon'(s)$ such that $\varepsilon_{ST}(s)/2 \leq \varepsilon'(s) \leq \varepsilon_{ST}(s)$ with high probability. Moreover, if there is a $2-\varepsilon$ approximation algorithm running in $O(m^{2-\delta})$ time for any $\varepsilon, \delta > 0$ or a 2-approximation algorithm running in $O(m^{3/2-\varepsilon})$ time for $\varepsilon > 0$, even for the Bichromatic case when $T = V \setminus S$, then SETH is false.*

Again, as before, one can improve the runtime to $\tilde{O}(m\sqrt{n})$ with a slight additive error, and there is a simple near-linear time 3-approximation algorithm which is tight under SETH, similar to the one in [BRS$^+$18] for ST-Diameter. A simple argument shows that these algorithms imply algorithms with the same running time and approximation factor for ST-Radius.

Bichromatic and ST Problems in Directed Graphs. Using simple reductions we first show that there can be no $O(m^{2-\varepsilon})$ time (for $\varepsilon > 0$) algorithms that achieve any finite approximation for ST-Diameter or ST-Eccentricities (under SETH), or ST-Radius (under HS). Interestingly, the same holds for Bichromatic Eccentricities (under SETH, Proposition 7) and Bichromatic Radius (under HS, Proposition 8), but not Bichromatic Diameter! Surprisingly, unlike those two prob-

lems, Bichromatic Diameter does admit a finite, in fact 2-approximation algorithm running in subquadratic time, and this algorithm is conditionally optimal:

Theorem 6.1.4. *There is a randomized $\tilde{O}(m^{3/2})$ time algorithm, that given a directed graph $G = (V, E)$ with nonnegative integer edge weights and $S \subseteq V, T = V \setminus S$, can output an estimate D' such that $D_{ST}/2 \leq D' \leq D_{ST}$ with high probability, where D_{ST} is the ST-Diameter of G.*

Moreover, if there is an $O(m^{2-\varepsilon})$ time $2-\delta$-approximation algorithm for the problem for some $\varepsilon, \delta > 0$, then SETH is false.

The previously known techniques for approximating Diameter in directed graphs fail here. The main issue is that the prior techniques were general enough that they also gave algorithms for Eccentricities and Radius as a byproduct. In the Bichromatic case, however, there is a genuine difference between Diameter and Radius, as we noted above, and new techniques are needed. Here again it turns out that combining theme (2) with a delicate argument is sufficient to get conditionally tight algorithms under SETH.

Subset Versions. Recall that Subset Diameter, Radius, and Eccentricities are the versions of the corresponding ST problems with the constraint that $S = T$. Interestingly, Subset Diameter, Radius, and Eccentricities all exhibit the same sharp threshold behavior. For all three problems, there are near-linear time algorithms that achieve a 2 (or almost 2) approximation, as well as conditional lower bounds that show that there is no $2 - \delta$ approximation in $m^{2-o(1)}$ time.

Parameterized Algorithms. We consider the Bichromatic Diameter, Radius, and Eccentricities problems parameterized by the size of the *boundary* between the S and T sets. If S' is the set of vertices in S that have a neighbor in T, and T' is the set of vertices in T that have a neighbor in S, then the boundary B is whichever of S' or T' is smaller in size. Our lower bound constructions already have small boundary so they rule out algorithms even for graphs with small boundary. However, interestingly we obtain near-linear time algorithms for graphs with small boundary that achieve *better* multiplicative approximation factors than the optimal non-parameterized algorithms. This is not a contradiction because our parameterized algorithms have a constant additive error, while the apparently contradictory lower bounds do not tolerate additive error.

6.1.2 Preliminaries

Given a graph $G = (V, E)$ (directed or undirected, weighted or unweighted), let $d(u, v)$ denote the distance from $u \in V$ to $v \in V$. For a subset $X \subseteq V$ and $v \in V$, define $d(v, X) := \min_{x \in X} d(v, x)$. Similarly $d(X, v) := \min_{x \in X} d(x, v)$.

Unless otherwise stated, m denotes the number of edges and n the number of vertices of the underlying graph. Without loss of generality, we can assume that all undirected graphs are connected, and all directed graphs are weakly connected, so that $m \geq n - 1$.

The *Eccentricity* $\varepsilon(v)$ of a vertex $v \in V$ is $\max_{u \in V} d(v, u)$. The *Diameter* $D(G)$ of G is $\max_{v \in V} \varepsilon(v)$, and the *Radius* $R(G)$ of G is $\min_{v \in V} \varepsilon(v)$.

Given $S, T \subseteq V$, we define analogous parameters as follows. The ST-Eccentricity $\varepsilon_{ST}(v)$ of $v \in S$ is $\max_{u \in T} d(v, u)$. The ST-Diameter $D_{ST}(G)$ is $\max_{v \in S} \varepsilon_{ST}(v)$, and the ST-Radius $R_{ST}(G)$ is $\min_{v \in S} \varepsilon_{ST}(v)$.

The above parameters are called *Bichromatic* Eccentricities, Diameter, and Radius if S and T form a partition of V, i.e. $T = V \setminus S$.

The above parameters are called *Subset* Eccentricities, Diameter, and Radius if $S = T$ and are notated with subscript S instead of ST.

Preliminaries for algorithms

Lemma 6.1.1. *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 6.1.2. *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 6.1.1, 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 6.1.1 and 6.1.2 hold when the sample is from a set of edges. Here is the analogue of Lemma 6.1.2. The other lemma is similar.

Lemma 6.1.3. *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.

Preliminaries for lower bounds

The Strong Exponential Time Hypothesis (SETH) asserts that on a Word-RAM with $O(\log n)$ bit words, there is no $(2-\varepsilon)^n$ time (possibly randomized) algorithm for some constant $\varepsilon > 0$ that can determine whether a given CNF-Formula with n variables and $O(n)$ clauses is satisfiable. (This version of SETH is equivalent to the original formulation by Impagliazzo, Paturi and Zane [IP01a].) By a result of Williams [Wil05], the following Orthogonal Vectors (OV) Problem requires $n^{2-o(1)}\text{poly}\,(d)$ time (on a word-RAM with $O(\log n)$ bit words), unless SETH fails: given

two sets $U, V \subseteq \{0,1\}^d$ with $|U| = |V| = n$ and $d = \omega(\log n)$, determine whether there are $u \in U, v \in V$ with $u \cdot v = 0$.

Given an arbitrary instance of OV with $d = \tilde{O}(1)$ (while respecting $d = \omega(\log n)$, e.g. $d = \Theta(\log^2 n)$), consider the following graph representation, which we call the ***OV-graph***: the vertex set consists of a node for every $u \in U$, for every $v \in V$ and for every coordinate $c \in [d] = C$, and there is an edge $(x \in U \cup V, c \in C)$ if and only if $x[c] = 1$. OV is then equivalent to the question of whether there exist $u \in U, v \in V$ such that $d(u, v) > 2$. In fact, it is equivalent to distinguishing whether for every $u \in U, v \in V$, $d(u, v) = 2$ (no OV-solution), or there is some $u \in U, v \in V$ such that $d(u, v) \geq 4$ (OV-solution). In other words, if we set $S = U, T = V$, the ST-Diameter of the OV-graph is 2 if and only if there is no OV-solution and at least 4 otherwise. Because the OV graph has $m = \tilde{O}(n)$, under SETH, any $(2 - \delta)$-approximation algorithm for ST-Diameter requires $m^{2-o(1)}$.

A related problem to OV is the Hitting Set (HS) problem [AVW16, GIKW17, Vas15]: given two sets $U, V \subseteq \{0,1\}^d$ with $|U| = |V| = n$ and $d = \omega(\log n)$, determine whether there is $u \in U$ such that for all $v \in V$, $u \cdot v \neq 0$. A common hypothesis is that (on the word-RAM) HS requires $n^{2-o(1)}$ time.

If we form the OV-graph on the HS instance input, then the HS problem becomes equivalent to determining whether there is some $u \in U$ such that for all $v \in V$, $d(u, v) \leq 2$. In other words, if we set $S = U, T = V$, the ST-Radius of the OV-graph is 2 if and only if there is a HS-solution and at least 4 otherwise. Thus, under the HS hypothesis, any $(2 - \delta)$-approximation algorithm for ST-Radius requires $m^{2-o(1)}$.

Additionally for our constructions we assume that if there is a HS solution u' then for all $c \in C$, $d(u', c) \leq 3$. This is because for every coordinate index i there must be $v \in V$ with $v[i] = 1$ as otherwise we can just delete the i^{th} bit from all vectors.

Let $k \geq 2$ be an integer. Then, a generalization of the OV problem is k-OV: given k sets $U_1, \ldots, U_k \subseteq \{0,1\}^d$, are there $u_1 \in U_1, \ldots, u_k \in U_k$ so that $\sum_{c=1}^{d} \prod_{i=1}^{k} u_i[c] = 0$? It is known that, under SETH, when $d = \omega(\log n)$, there is no $n^{k-o(1)}$ time algorithm for k-OV (in the word RAM model) [Wil05].

Similar to the OV-graph, Backurs et al. [BRS$^+$18] define a graph for k-OV which we will refer to as the k-OV-graph. We do not explicitly define the k-OV-graph here; instead we list its properties in the following theorem.

Theorem 6.1.5 ([BRS$^+$18])**.** *Let $k \geq 2$. 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.*

Organization

In Section 6.1.3 we present our algorithms for Bichromatic Diameter, Eccentricities, and Radius. In Section 6.1.4 we present our algorithms for ST-Eccentricities and Radius. In Section 6.1.5 we present our algorithms for Subset Diameter, Eccentricities, and Radius. In Section 6.1.6 we present our parameterized algorithms for Bichromatic Diameter, Radius, and Eccentricities. In Section 6.1.7 we present all of our conditional lower bounds.

6.1.3 Algorithms for Undirected Bichromatic Diameter, Eccentricities and Radius

Undirected Bichromatic Diameter

We begin with a simple near-linear time algorithm.

Proposition 1. *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 6.1.6. *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. However, we include such analyses in our proofs for Bichromatic Radius (Theorem 6.1.9) and ST-Eccentricities (Theorem 6.1.14) and the analysis for Diameter is analogous.

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 6.1.2 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,

$$D_4 \geq d(t(s'), s^*) \geq d(s^*, t^*) - d(t(s'), t^*) \geq D - d(t(s'), s') - d(s', t^*) \geq D - 1 - 2D/5 + 1 = 3D/5.$$

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

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 6.1.7. *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 6.1.3 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 end points. 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 $3D/5 \leq D'' \leq D$. $\square$

Undirected Bichromatic Radius

We begin with a simple near-linear time algorithm that achieves almost a 2-approximation.

Theorem 6.1.8. *Let $G = (V,E)$ be an undirected graph with nonnegative edge weights w. Let*

$S \subseteq V, T = V \setminus S$. *There is an* $O(m + n \log n)$ *time algorithm that outputs an estimate* R' *such that* $R_{ST} \leq R' \leq 2R_{ST} + \min_{s \in S, t \in T, (s,t) \in E} w(s,t)$. *If* G *is unweighted, the algorithm runs in* $O(m+n)$ *time and* $R_{ST} \leq R' \leq 2R_{ST} + 1$.

Proof. The algorithm is as follows. Let $(s,t) \in E$ be the smallest weight edge among those with $s \in S, t \in T$. Run Dijkstra's algorithm from s and output $R' = \max_{t' \in T} d(s,t')$.

Clearly $R_{ST} \leq R'$. Let $s^* \in S$ be the true ST-center. Then for all $t' \in T$, $d(s,t') \leq d(s,s^*) + R_{ST}$. On the other hand, $d(s,s^*) \leq w(s,t) + d(t,s^*) \leq w(s,t) + R_{ST}$, and hence $R' \leq w(s,t) + 2R_{ST}$.

For unweighted graphs, $w(s,t) = 1$ and we can run BFS instead of Dijkstra's algorithm. □

We now present a $\tilde{O}(m\sqrt{n})$ algorithm for Bichromatic Radius, similar in spirit to our Bichromatic Diameter algorithm.

Theorem 6.1.9. *There is an* $\tilde{O}(m\sqrt{n})$ *time algorithm, that given an undirected unweighted graph* $G = (V,E)$ *and* $S \subseteq V, T = V \setminus S$, *can output estimates* R'_{ST} *such that* $R_{ST} \leq R'_{ST} \leq 5R_{ST}/3 + 5/3$. *If* R_{ST} *is divisible by* 3, $R_{ST} \leq R'_{ST} \leq 5R_{ST}/3 + 1$.

Proof. Let $s^* \in S$ be the ST-center of G and let $R = R_{ST}$ be the ST-Radius.

(Algorithm Step 1): The algorithm samples $S_1 \subseteq S$ of size $c\sqrt{n} \ln n$ uniformly at random. For every $s \in S_1$, run BFS and find $t(s) \in T$ which is closest to s. Let $T_2 = \{t(s) \mid s \in S_1\}$.

Then sample $T_1 \subseteq T$ of size $c\sqrt{n} \ln n$ uniformly at random. For every $t \in T_1$, run BFS and find $s(t) \in S$ which is closest to t. Let $S_2 = \{s(t) \mid t \in T_1\}$.

Let $s_0 \in S$ be the node minimizing $\max_{t \in T_1 \cup T_2} d(s_0,t)$. Let $R_1 = \max_{t \in T} d(s_0,t)$. Let $w \in T$ be the node maximizing $d(w, T_1 \cup T_2)$.

(Analysis Step 1): We know that $\max_{t \in T_1 \cup T_2} d(s^*,t) \leq R$, and hence $\max_{t \in T_1 \cup T_2} d(s_0,t) \leq R$.

Suppose that for every $t \in T$, $d(t, T_1 \cup T_2) \leq 2R/3$. Then, $d(s_0,t) \leq R + 2R/3 = 5R/3$ and hence $R_1 \leq 5R/3$ and s_0 would be a good approximate center. Thus, we can assume that there exists some t with $d(t, T_1 \cup T_2) > 2R/3$, and in particular, $d(w, T_1 \cup T_2) > 2R/3$.

Moreover, suppose that there is some $s \in S_1$ such that $d(w,s) \leq R/3$. Then, $d(w,t(s)) \leq d(w,s) + d(s,t(s)) \leq 2d(w,s) \leq 2R/3$, contradicting the fact that $d(w, T_1 \cup T_2) > 2R/3$. Thus, we must have that $d(w,S_1) > R/3$.

Now, since T_1 is random of size $c\sqrt{n}\ln n$, by Lemma 6.1.2, the number of nodes of T at distance $\leq 2R/3$ from w is at most $\sqrt{n}$, whp. Similarly, since S_1 is random of size $c\sqrt{n}\ln n$, by Lemma 6.1.2, the number of nodes of S at distance $\leq R/3$ from w is at most $\sqrt{n}$, whp.

(Algorithm Step 2): Run BFS from w. Take the closest $\sqrt{n}$ nodes T_w of T at distance from w. Run BFS from all $t \in T_w$, and find $s(t) \in S$ closest to t. Run BFS from each $s(t)$.

Let $R_2 := \min_{t' \in T_w} \max_{t \in T} d(s(t'), t)$.

(Analysis Step 2): Since $|T_w| \leq \sqrt{n}$, the runtime of this step is $O(m\sqrt{n})$.

Since $w \in T$, we know that $d(w, s^*) \leq R$. Now consider the node b on the shortest w to s^* path P_{ws^*} for which $d(w, b) \leq 2R/3$, but such that the node b' after it on P_{wt^*} has $d(w, b') > 2R/3$. Since the graph is unweighted, we get that $d(w, b) = \lfloor 2R/3 \rfloor \geq 2R/3 - 2/3$.

Let us consider $d(b, s^*) = d(w, s^*) - d(w, b)$. Since $d(w, s^*) \leq R$ and $d(w, b) \geq 2R/3 - 2/3$, $d(b, s^*) \leq R/3 + 2/3$.

Suppose that $b \in T$. By our previous argument, as $d(w, b) \leq 2R/3$, b must be in T_w. Then we ran BFS from $s(b)$ and $d(s(b), s^*) \leq d(s(b), b) + d(b, s^*) \leq 2d(b, s^*) \leq 2R/3 + 4/3$, and hence $R_2 \leq 2R/3 + R + 4/3 = 5R/3 + 4/3$. Thus if R_2 is not a good approximation, then $b \in S$.

(Algorithm Step 3): Take the $\sqrt{n}$ closest nodes of S to w. Call these S_w. Run BFS from every $s \in S_w$. Set $R_3 := \min_{s \in S_w} \max_{t \in T} d(s, t)$.

(Analysis Step 3): Since $|S_w| \leq \sqrt{n}$, the runtime of this step is $O(m\sqrt{n})$.

Let a be the node on P_{ws^*} with $d(w, a) \leq R/3$ but so that the node a' after a on P_{ws^*} has $d(w, a') > R/3$. We have that $d(w, a) = \lfloor R/3 \rfloor \geq R/3 - 2/3$.

Suppose that $a \in S$. As $d(w, a) \leq R/3$ and a is among the closest $\sqrt{n}$ nodes to w by our previous argument, we ran BFS from a.

However, also $d(a, s^*) = d(w, s^*) - d(w, a) \leq R - R/3 + 2/3 = 2R/3 + 2/3$, and hence $R_3 \leq 2R/3 + R + 2/3 = 5R/3 + 2/3$. If R_3 is not a good approximation, it must be that $a \in T$.

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

Hence for every $t \in T$, $d(s(t'), t) \leq 5R/3 + 5/3$. If R is divisible by 3, the only source of additive error is the $+1$ from using the edge $(t', s(t'))$ instead of (t', s').

Hence if we set $R' = \min\{R_1, R_2, R_3\}$, we have $R \leq R' \leq 5R/3 + 5/3$. If R is divisible by 3, $R \leq R' \leq 5R/3 + 1$. □

We now use edge sampling to remove the additive error and make the algorithm work for weighted graphs as well, at the expense of increasing the runtime to $\tilde{O}(m^{3/2})$.

Theorem 6.1.10. *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 estimates R'_{ST} such that $R_{ST} \leq R'_{ST} \leq 5R_{ST}/3$.*

Proof. Let $s^* \in S$ be the ST-center of G and let $R = R_{ST}$ be the ST-Radius.

(Algorithm Step 1): We sample $c\sqrt{m} \ln n$ edges $E' \subseteq E$ uniformly at random. Let S_1 be the endpoints that are in S and let T_1 be the endpoints in T. For every $s \in S_1$, run Dijkstra and find $t(s) \in T$ which is closest to s. Let $T_2 = \{t(s) \mid s \in S_1\}$.

For every $t \in T_1$, run Dijkstra and find $s(t) \in S$ which is closest to t. Let $S_2 = \{s(t) \mid t \in T_1\}$.

Let $s_0 \in S$ be the node minimizing $\max_{t \in T_1 \cup T_2} d(s_0, t)$. Run Dijkstra from s_0. Let $R_1 = \max_{t \in T} d(s_0, t)$. Let $w \in T$ be the node maximizing $d(w, T_1 \cup T_2)$.

(Analysis Step 1): The algorithm runs in $\tilde{O}(m^{3/2})$ time.

We know that $\max_{t \in T_1 \cup T_2} d(s^*, t) \leq R$, and hence $\max_{t \in T_1 \cup T_2} d(s_0, t) \leq R$.

Suppose that for every $t \in T$, $d(t, T_1 \cup T_2) \leq 2R/3$. Then, $d(s_0, t) \leq R + 2R/3 = 5R/3$ and hence $R_1 \leq 5R/3$ and s_0 would be a good approximate center. Thus, we can assume that there exists some t with $d(t, T_1 \cup T_2) > 2R/3$, and in particular, $d(w, T_1 \cup T_2) > 2R/3$.

Moreover, suppose that there is some $s \in S_1$ such that $d(w, s) \leq R/3$. Then, $d(w, t(s)) \leq d(w, s) + d(s, t(s)) \leq 2d(w, s) \leq 2R/3$, contradicting the fact that $d(w, T_1 \cup T_2) > 2R/3$. Thus, we must have that $d(w, S_1) > R/3$.

Now, since E' is random of size $c\sqrt{m} \ln n$, by Lemma 6.1.3, the number of edges (t, g) where $t \in T, g \in V$ and $d(w, t) \leq 2R/3$ is at most $\sqrt{m}$, whp. Similarly, the number of edges (s, g) where $s \in S, g \in V$ and $d(s, w) \leq R/3$ is at most $\sqrt{m}$, whp.

(Algorithm Step 2): Run Dijkstra from w. Consider the edges (t, b) with $t \in T$ sorted in nondecreasing order according to $d(w, t)$. Let E^T be the first $\sqrt{m}$ edges in this sorted order. Run Dijkstra from both endpoints of each edge in E^T. Call T_w those endpoints that are in T and S_w^1 those in S. Let $R_2 := \min_{s \in S_w^1} \max_{t \in T} d(s, t)$.

For every $t \in T_w$, determine a closest node $s(t) \in T$ to t, and run Dijkstra's algorithm from $s(t)$ as well. Let $R_3 := \min_{t \in T_w} \max_{t' \in T} d(s(t), t')$.

(Analysis Step 2): Since $|E^T| \leq \sqrt{m}$, the runtime of this step is $\tilde{O}(m^{3/2})$.

If $R_2 \leq 5R/3$ or $R_3 \leq 5R/3$, we are done. So let us assume that $R_2, R_3 > 5R/3$. Also, since $w \in T$, we know that $d(w, s^*) \leq R$.

Now consider the node b on the shortest w to s^* path P_{ws^*} for which $d(w, b) \leq 2R/3$, but such that the node b' after it on P_{ws^*} has $d(w, b') > 2R/3$.

Suppose that $b \in T$. Then since $d(w, b) \leq 2R/3$ and since by the previous argument the edges from T nodes at distance $2R/3$ from w is at most $\sqrt{m}$, (b, b') must be among the edges in E^T. We thus run Dijkstra's from both b and b'.

Let us consider $d(b', s^*) = d(w, s^*) - d(w, b')$. Since $d(w, s^*) \leq R$ and $d(w, b') > 2R/3$, $d(b', s^*) < R/3$. If $b' \in S$, then since we ran Dijkstra's algorithm from b', we got $R_2 \leq 4R/3$. If $b' \in T$, then we ran Dijkstra's algorithm from $s(b')$ and $d(s(b'), s^*) \leq d(s(b'), b') + d(b', s^*) \leq 2d(b', s^*) < 2R/3$, and hence $R_3 \leq 2R/3 + R = 5R/3$. Thus if neither R_2, nor R_3 are good approximations, then $b \in S$.

(Algorithm Step 3): Consider the edges (s, b) with $s \in S$ sorted in nondecreasing order according to $d(w, s)$. Let E^S be the first $\sqrt{m}$ edges in this sorted order. Run Dijkstra from both endpoints of each edge in E^S. Call S_w^2 those endpoints that are in S. Let $R_4 := \min_{s \in S_w^2} \max_{t \in T} d(s, t)$.

(Analysis Step 3): As $|E^S| = \sqrt{m}$, $|S_w| \leq 2\sqrt{m}$, so this step runs in $\tilde{O}(m^{3/2})$ time.

If $R_4 \leq 5R/3$, we would be done, so assume that $R_4 > 5R/3$. Let a be the node on P_{ws^*} with $d(w, a) \leq R/3$ but so that the node a' after a on P_{ws^*} has $d(w, a') > R/3$. Suppose that $a' \in S$. Then since $d(w, a) \leq R/3$, $(a, a') \in E^S$, $a' \in S_w^2$ and we ran Dijkstra's algorithm from a'. However, also $d(a', s^*) = d(w, s^*) - d(w, a') < R - R/3 = 2R/3$, and hence $R_4 \leq 2R/3 + R =$

$5R/3$. Since $R_4 > 5R/3$, it must be that $a' \in T$.

Now, since $a' \in T$ and $b \in S$, somewhere on the a' to b shortest path P_{ab}, there must be an edge (t', s') with $s' \in S, t' \in T$. However, since t' is before b, we have that $d(w, t') \leq d(w, b) \leq 2R/3$. Thus, $(t', s') \in E^T$ and we ran Dijkstra's algorithm from s'. However, $d(s', s^*) = d(w, s^*) - d(w, s') \leq d(w, s^*) - d(w, a') < R - R/3 = 2R/3$, and hence $R_2 \leq R + 2R/3 = 5R/3$.

Hence if we set $R' = \min\{R_1, R_2, R_3, R_4\}$, we have $R \leq R' \leq 5R/3$ □

Undirected Bichromatic Eccentricities.

In the next section we will give approximation algorithms for ST-Eccentricities in undirected graphs which imply algorithms for bichromatic Eccentricities in undirected graphs with same guarantees. We reproduce them here for convenience.

Proposition 2. *There is an $O(m + n \log n)$ 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 $\varepsilon'_{ST}(v)$ for each node $v \in S$ such that $\varepsilon_{ST}(v)/3 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Theorem 6.1.11. *There is an $\tilde{O}(m\sqrt{n})$ time algorithm, that given an unweighted graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, can output an estimate $\varepsilon'_{ST}(v)$ for each $v \in S$ such that $\varepsilon_{ST}(v)/2 - 5/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$. If $\varepsilon_{ST}(v)$ is divisible by 2, $\varepsilon_{ST}(v)/2 - 2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Theorem 6.1.12. *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 estimates $\varepsilon'_{ST}(v)$ for each $v \in S$, such that $\varepsilon_{ST}(v)/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Directed Bichromatic Diameter

Theorem 6.1.13. *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 6.1.3, this edge is among the $\sqrt{m}$ closest edges to w. Hence $D_2 \geq d(s^*,t) \geq d(s^*,t^*) - d(t,t^*) \geq D - d(t,t^*) \geq D - d(w,t^*) \geq D/2$ □

6.1.4 Algorithms for ST-Eccentricities and Radius

All of the algorithms in this section are for undirected graphs; we later prove that the directed versions of these problems do not admit truly subquadratic algorithms with any finite approximation factor.

We do not give algorithms for ST-Diameter, as tight algorithms were already given in [BRS+18].

ST-Eccentricities

We begin with a near-linear time 3-approximation algorithm.

Proposition 3. *There is an $O(m + n\log n)$ time algorithm, that given an undirected graph $G = (V,E)$ with nonnegative integer edge weights and $S, T \subseteq V$, can output an estimate $\varepsilon'_{ST}(v)$ for each node $v \in S$ such that $\varepsilon_{ST}(v)/3 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Proof. The algorithm is as follows. Pick an arbitrary node $t \in T$ and run Dijkstra's algorithm from it. Let t' be a node in T maximizing $d(t',t)$, and run Dijkstra's algorithm from t'. For each $v \in S$,

output $\varepsilon'_{ST}(v) = \max\{d(v,t), d(v,t')\}$.

Clearly $\varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$. Now suppose that $v' \in T$ is the farthest node from v in T. So we have $\varepsilon_{ST}(v) = d(v,v') \leq d(v,t) + d(t,v') \leq d(v,t) + d(t,t') \leq d(v,t) + d(t,v) + d(v,t') \leq 3\varepsilon'_{ST}(v)$, where the first and third inequalities are from triangle inequality and the second inequality is from the definition of t'. □

Now we turn to our 2-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 6.1.14. *There is an $\tilde{O}(m\sqrt{n})$ time algorithm, that given an undirected unweighted graph $G = (V,E)$ and $S, T \subseteq V$, can output an estimate $\varepsilon'_{ST}(v)$ for each $v \in S$ such that $\varepsilon_{ST}(v)/2 - 5/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$. If $\varepsilon_{ST}(v)$ is divisible by 2, $\varepsilon_{ST}(v)/2 - 2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Proof. For each $v \in S$, let v' be the farthest node from v, i.e. $d(v,v') = \varepsilon_{ST}(v)$.

(Algorithm Step 1): The algorithm samples $X \subset V$ of size $c\sqrt{n}\ln n$ uniformly at random. For every $x \in X$, run BFS and find $t(x) \in T$ which is closest to x (if $x \in T$, $t(x) = x$). Let $T_X = \{t(x) | x \in X\}$.

Run BFS from each node $t \in T_X$. For each $v \in S$ let $\varepsilon^{(1)}_{ST}(v) = \max_{t \in T_X} d(v,t)$.

Let $w \in T$ be the node maximizing $d(w, T_X)$.

(Analysis Step 1): This step of the algorithm runs in $\tilde{O}(m\sqrt{n})$.

Suppose there is some node $t \in T_X$ such that $d(v',t) \leq \varepsilon_{ST}(v)/2$. Then $d(v,t) \geq d(v,v') - d(v',t) \geq \varepsilon_{ST}(v)/2$, and so $\varepsilon^{(1)}_{ST}(v)$ is a good approximation for v. Thus we can assume that $d(v', T_X) > \varepsilon_{ST}(v)/2$, and so $d(w, T_X) > \varepsilon_{ST}(v)/2$. Now since X is random of size $c\sqrt{n}\ln n$, by Lemma 6.1.2, the number of nodes of T at distance $\leq \varepsilon_{ST}(v)/2$ from w is at most $\sqrt{n}$ whp.

Moreover, suppose that there is some node $x \in X$ such that $d(w,x) \leq \varepsilon_{ST}(v)/4$. Then $d(w,t(x)) \leq d(w,x) + d(x,t(x)) \leq 2d(w,x) \leq \varepsilon_{ST}(v)/2$, contradicting the fact that $d(w,T_X) > \varepsilon_{ST}(v)/2$. Thus, we must have that $d(w,X) > \varepsilon_{ST}(v)/4$.

Now, since X is random of size $c\sqrt{n}\ln n$, by Lemma 6.1.2, the number of nodes at distance $\leq \varepsilon_{ST}(v)/4$ from w is at most $\sqrt{n}$ whp.

(Algorithm Step 2): Run BFS from w. For each $v \in S$, let $\varepsilon^{(2)}_{ST}(v) = d(v, w)$.

Take the closest $\sqrt{n}$ nodes of $V \setminus T$ to w. Call these Y. Run BFS from all $y \in Y$, and let $e(y) = \max_{t \in T} d(y, t)$. Let $\varepsilon^{(3)}_{ST}(v) = \max_{y \in Y} e(y) - d(v, y)$.

(Analysis Step 2): If $d(v, w) \geq \varepsilon_{ST}(v)/2$, then $\varepsilon^{(2)}_{ST}(v)$ is a good estimate. So assume that $d(v, w) \leq \lceil \varepsilon_{ST}/2 \rceil - 1 \leq \varepsilon_{ST}/2 - 1/2$.

Now consider the node a on the shortest w to v path P_{wv} for which $d(w, a) \leq \varepsilon_{ST}(v)/4$, but such that the node a' after it on P_{wv} has $d(w, a') > \varepsilon_{ST}(v)/4$. Since the graph is unweighted, we get that $d(w, a) = \lfloor \varepsilon_{ST}(v)/4 \rfloor \geq \varepsilon_{ST}(v)/4 - 3/4$.

If $a \in V \setminus T$, then by the previous argument since $d(a, w) \leq \varepsilon_{ST}(v)/4$, $a \in Y$ and we run BFS from a. Since $e(a) \geq d(a, v') \geq d(v, v') - d(a, v)$ and $d(a, v) = d(w, v) - d(a, w)$, we have $e(a) \geq 3\varepsilon_{ST}(v)/4 - 1/4$. So $e(a) - d(v, a) \geq \varepsilon_{ST}(v)/2 - 1/2$. Moreover, if a' is the farthest node from a in T, then $\varepsilon_{ST}(v) \geq d(v, a') \geq d(a, a') - d(v, a) = e(a) - d(v, a)$, and hence $\varepsilon^{(3)}_{ST}(v)$ is a good estimate.

So assume that $a \in T$.

(Algorithm Step 3): Take the closest $\sqrt{n}$ nodes of T to w. Call these T_w. Run BFS from all $t \in T_w$ and find $y(t) \in V \setminus T$. Run BFS from each $y(t)$, and let $e(y(t)) = \max_{t' \in T} d(y(t), t')$. Let $\varepsilon^{(4)}_{ST}(v) = \max_{t \in T_w} e(y(t)) - d(v, y(t))$.

(Analysis Step 3): Consider the node b on P_{wv} for which $d(w, b) \leq 3\varepsilon_{ST}(v)/8$, but such that the node b' after it on P_{wv} has $d(w, b') > 3\varepsilon_{ST}(v)/8$. Since the graph is unweighted, we get that $d(w, b) = \lfloor 3\varepsilon_{ST}(v)/8 \rfloor \geq 3\varepsilon_{ST}(v)/8 - 7/8$.

If $b \in T$, then since $d(w, b) \leq \varepsilon_{ST}(v)/2$, by previous argument $b \in T_w$ and we run BFS from b. Since $d(v, b) = d(w, v) - d(w, b) \leq \varepsilon_{ST}(v)/8 + 3/8$, we have that $d(v, y(b)) \leq d(v, b) + d(b, y(b)) \leq 2d(v, b) \leq \varepsilon_{ST}(v)/4 + 3/4$. Similar to the previous step, we get that $e(y(b)) - d(v, y(b)) \geq d(y(b), v') - d(v, y(b)) \geq d(v, v') - 2d(v, y(b)) \geq \varepsilon_{ST}(v)/2 - 3/2$. By considering the farthest node from $y(b)$ in T, we can show that $e(y(b)) - d(v, y(b)) \leq \varepsilon_{ST}(v)$ and hence $\varepsilon^{(4)}_{ST}(v)$ is a good approximate. So if $\varepsilon^{(4)}_{ST}(v)$ is not a good approximate, it must be that $b \in V \setminus T$.

Now, since $a \in T$ and $b \in V \setminus T$, somewhere on the a to b shortest path P_{ab}, there must be an edge (t', y') with $t' \in T$ and $y' \in V \setminus T$. However, since t' is on P_{wv}, we have $d(w, t') \leq d(v, w) <$

$\varepsilon_{ST}(v)/2$. Thus, $t' \in T_w$ and we run BFS from $y(t')$. However, $d(t', y(t')) \leq d(t', y') = 1$, and hence $d(y(t'), v) \leq d(y(t'), t') + d(t', v) \leq 1 + d(w, v) - d(w, t') \leq 1 + \varepsilon_{ST}(v)/2 - 1/2 - d(w, a) \leq \varepsilon_{ST}(v)/4 + 5/4$. So we get that

$$e(y(t')) - d(y(t'), v) \geq d(y(t'), v') - d(y(t'), v) \geq d(v, v') - 2d(y(t'), v) \geq \varepsilon_{ST}(v)/2 - 5/2$$

Moreover, if y' is the farthest node $y(t')$ in T, then $\varepsilon_{ST}(v) \geq d(v, y') \geq d(y', y(t')) - d(v, y(t')) = e(y(t')) - d(v, y(t'))$. Hence if for each $v \in S$ we set $\varepsilon'_{ST}(v) = \max\{\varepsilon^{(1)}_{ST}(v), \varepsilon^{(2)}_{ST}(v), \varepsilon^{(3)}_{ST}(v), \varepsilon^{(4)}_{ST}(v)\}$, we have $\varepsilon_{ST}(v)/2 - 5/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$. □

We now use edge sampling to remove the additive error from the above algorithm and make the algorithm work for weighted graphs as well, at the expense of increasing the runtime to $\tilde{O}(m^{3/2})$.

Theorem 6.1.15. *There is an $\tilde{O}(m^{3/2})$ time algorithm, that given an undirected graph $G = (V, E)$ with nonnegative integer edge weights and $S, T \subseteq V$, can output estimates $\varepsilon'_{ST}(v)$ for each $v \in S$, such that $\varepsilon_{ST}(v)/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$.*

Proof. For each $v \in S$, let v' be the farthest node from v, i.e. $d(v, v') = \varepsilon_{ST}(v)$.

(Algorithm Step 1): We sample $c\sqrt{m} \ln n$ edges $E' \subseteq E$ uniformly at random. Run Dijkstra from both endpoints of edges in E' (we call these vertices $V(E')$), and for each endpoint x, find $t(x) \in T$ which is closest to x. Let $T_{E'} = \{t(x) | x \in V(E')\}$.

Run Dijkstra from each node in $T_{E'}$, and for each $v \in S$, let $\varepsilon^{(1)}_{ST}(v) = d(v, T_{E'})$.

Let $w \in T$ be the node maximizing $d(w, T_{E'})$.

(Analysis Step 1): Since $V(E') = \tilde{O}(\sqrt{m}) = |T_{E'}|$, this step takes $\tilde{O}(m^{3/2})$ time.

Suppose there is some node $t \in T_{E'}$ such that $d(v', t) \leq \varepsilon_{ST}(v)/2$. Then $d(v, t) \geq d(v, v') - d(v', t) \geq \varepsilon_{ST}(v)/2$, and so $\varepsilon^{(1)}_{ST}(v)$ is a good approximation for $\varepsilon_{ST}(v)$. Thus we can assume that $d(v', T_{E'}) > \varepsilon_{ST}(v)/2$, and so $d(w, T_{E'}) > \varepsilon_{ST}(v)/2$. Now since E' is random of size $c\sqrt{m} \ln n$, by Lemma 6.1.3, the number of edges (t, g) where $t \in T, g \in V$ and $d(w, t) \leq \varepsilon_{ST}(v)/2$ is at most $\sqrt{m}$, whp.

Moreover, suppose that there is some edge $(x, b) \in E'$ such that $d(w, x) \leq \varepsilon_{ST}(v)/4$. Then $d(w, t(x)) \leq d(w, x) + d(x, t(x)) \leq 2d(w, x) \leq \varepsilon_{ST}(v)/2$, contradicting the fact that $d(w, T_{E'}) >$

$\varepsilon_{ST}(v)/2$. Thus, we must have that $d(w, V(E')) > \varepsilon_{ST}(v)/4$.

Now, since E' is random of size $c\sqrt{n}\ln n$, by Lemma 6.1.3, the number of edges $(x, g) \in E'$ where $g \in V$ such that $d(w, x) \leq \varepsilon_{ST}(v)/4$ is at most $\sqrt{m}$, whp.

(Algorithm Step 2): Run Dijkstra from w. For each $v \in S$, let $\varepsilon^{(2)}_{ST}(v) = d(v, w)$.

Consider the edges (y, b) sorted in nondecreasing order according to $d(w, y)$. Let E'' be the first $\sqrt{m}$ edges in this sorted order. Let Y be the endpoints of edges in E'' that are in $V \setminus T$. Run Dijkstra from each node in Y and let $e(y) = \max_{t \in T} d(y, t)$. Let $\varepsilon^{(3)}_{ST}(v) = \max_{y \in Y} e(y) - d(v, y)$.

(Analysis Step 2): Since $|Y| = \tilde{O}(\sqrt{m})$, this step takes $\tilde{O}(m^{3/2})$ time.

If $d(v, w) \geq \varepsilon_{ST}(v)/2$, then $\varepsilon^{(2)}_{ST}(v)$ is a good approximation. So assume that $d(v, w) < \varepsilon_{ST}(v)/2$.

Now consider the node a on the shortest w to v path P_{wv} for which $d(w, a) \leq \varepsilon_{ST}(v)/4$, but such that the node a' after it on P_{wv} has $d(w, a') > \varepsilon_{ST}(v)/4$.

Since $d(w, a) \leq \varepsilon_{ST}(v)/4$, by the previous argument the number of edges from the nodes at distance $\varepsilon_{ST}(v)/4$ from w is at most $\sqrt{m}$, and so (a, a') must be among the edges in E''. Suppose that $a' \in V \setminus T$. We thus run Dijkstra from a'.

Let us consider $d(a', v) = d(w, v) - d(w, a')$. Since $d(w, a') > \varepsilon_{ST}(v)/4$ and $d(w, v) < \varepsilon_{ST}(v)/2$, $d(a', v) < \varepsilon_{ST}(v)/4$. Thus $e(a') \geq d(a', v') \geq d(v, v') - d(a', v) > 3\varepsilon_{ST}(v)/4$. So $e(a') - d(a', v) > \varepsilon_{ST}(v)/2$. Now if a'' is the farthest node from a' in T, then $\varepsilon_{ST}(v) \geq d(v, a'') \geq d(a', a'') - d(v, a') = e(a') - d(v, a')$, and hence $\varepsilon^{(3)}_{ST}(v)$ is a good approximation.

So we assume that $a' \in T$.

(Algorithm Step 3): Consider the edges (t, b) with $t \in T$ sorted in nondecreasing order according to $d(w, t)$. Let E^T be the first $\sqrt{m}$ edges in this sorted order. Run Dijkstra from both endpoints of each edge in E^T (call these nodes $V(E^T)$), and find $y(x) \in V \setminus T$ closest to x, for each $x \in V(E^T)$. Run Dijkstra from each $y(x)$, and let $e(y(x)) = \max_{t \in T} d(y(x), t)$. Let $\varepsilon^{(4)}_{ST}(v) = \max_{x \in V(E^T)} e(y(x)) - d(v, y(x))$.

(Analysis Step 3):

Consider the node b on P_{wv} for which $d(w, b) \leq 3\varepsilon_{ST}(v)/8$, but such that the node b' after it on P_{wv} has $d(w, b') > 3\varepsilon_{ST}(v)/8$.

Suppose that $b' \in T$, then since $d(w,b) \leq \varepsilon_{ST}(v)/2$, by the previous argument $(b,b') \in E^T$ and we run Dijkstra from $y(b')$. Let us consider $d(y(b'),v) \leq d(v,b') + d(b',y(b')) \leq 2d(v,b')$. Since $d(v,b') = d(v,w) - d(w,b') < \varepsilon_{ST}(v)/8$, $d(y(b'),v) < \varepsilon_{ST}(v)/4$. Similar as in the previous step, we get that $\varepsilon_{ST}(v) \geq e(y(b')) - d(y(b'),v)$ and also $e(y(b')) - d(y(b'),v) \geq d(y(b'),v') - d(y(b'),v) \geq d(v,v') - 2d(y(b'),v) > \varepsilon_{ST}(v)/2$, thus $\varepsilon^{(4)}_{ST}(v)$ is a good approximation. So if $\varepsilon^{(4)}_{ST}(v)$ is not a good approximation, it must be that $b' \in V \setminus T$.

Now, since $a' \in T$ and $b' \in V \setminus T$, somewhere on the a' to b' shortest path $P_{a'b'}$, there must be an edge (t,x) with $t \in T$ and $x \in V \setminus T$. However, since t is on P_{wv}, we have $d(w,t) \leq d(v,w) < \varepsilon_{ST}(v)/2$. Thus, $(t,x) \in E^T$ and we run Dijkstra from x.

Let us consider $y(x)$. Since $x \in V \setminus T$, $y(x) = x$. Moreover since x is after a' on $P_{a'b'}$, $d(w,x) \geq d(w,a') > \varepsilon_{ST}(v)/4$, and thus $d(x,v) = d(v,w) - d(w,x) < \varepsilon_{ST}(v)/4$. So $e(y(x)) - d(y(x),v) = e(x) - d(x,v) \geq d(x,v') - d(x,v) \geq d(v,v') - 2d(x,v) > \varepsilon_{ST}(v)/2$.

Hence if for each $v \in S$ we set $\varepsilon'_{ST}(v) = \max\{\varepsilon^{(1)}_{ST}(v), \varepsilon^{(2)}_{ST}(v), \varepsilon^{(3)}_{ST}(v), \varepsilon^{(4)}_{ST}(v)\}$, we have $\varepsilon_{ST}(v)/2 \leq \varepsilon'_{ST}(v) \leq \varepsilon_{ST}(v)$. □

ST-Radius

A simple argument shows that given any approximation algorithm for ST-Eccentricities, one obtains an approximation algorithm for ST-Radius with the same approximation factor. First, run the ST-Eccentricities algorithm and let v be the vertex with the smallest estimated Eccentricity $\epsilon'(v)$. Then run Dijkstra's algorithm from v and report $\epsilon_{ST}(v)$ as the ST-Radius estimate R'. Let R be the true ST-Radius of the graph and let c be the true ST-center. If α is the approximation ratio for the ST-Eccentricities algorithm then $\epsilon_{ST}(v) \leq \alpha\epsilon'(v) \leq \alpha\epsilon_{ST}(v)$ and $\epsilon_{ST}(c) \leq \alpha\epsilon'(c) \leq \alpha\epsilon_{ST}(c)$. By choice of v, $\epsilon'(v) \leq \epsilon'(c)$. Thus, $\alpha R = \alpha\epsilon_{ST}(c) \geq \alpha\epsilon'(c) \geq \alpha\epsilon'(v) \geq \epsilon_{ST}(v) = R'$. Clearly $R' \geq R$, so $R \leq R' \leq \alpha R$.

Thus, we get the following theorems from our algorithms for ST-Eccentricities.

Theorem 6.1.16. *There is an $O(m + n\log n)$ time algorithm, that given an undirected graph $G = (V,E)$ with nonnegative integer edge weights and $S, T \subseteq V$, can output an estimate R' such that $R_{ST}/3 \leq R' \leq R_{ST}$.*

Theorem 6.1.17. *There is an $\tilde{O}(m\sqrt{n})$ time algorithm, that given an undirected unweighted graph $G = (V, E)$ and $S, T \subseteq V$, can output an estimate R' such that $R_{ST}/2 - 5/2 \leq R' \leq R_{ST}$.*

Theorem 6.1.18. *There is an $\tilde{O}(m^{3/2})$ time algorithm, that given an undirected graph $G = (V, E)$ with nonnegative integer edge weights and $S, T \subseteq V$, can output estimates R' such that $R_{ST}/2 \leq R' \leq R_{ST}$.*

6.1.5 Algorithms for Subset Diameter, Eccentricities, and Radius

We obtain 2-approximations for Subset Diameter in directed graphs and Subset Radius in undirected graphs simply by running Dijkstra's algorithm from an arbitrary vertex $s \in S$. We obtain an almost 2-approximation in almost linear time for directed Subset Eccentricities (and thus directed Subset Radius) by a slight modification of an algorithm for (non-Subset) Eccentricities in directed graphs from [BRS$^+$18].

Proposition 4 (Directed Subset Diameter)**.** *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, $D_S/2 \leq \max\{D_1, D_2\} \leq D_S$. □

Proposition 5 (Undirected Subset Radius)**.** *There is an $\tilde{O}(m)$ time algorithm, that given an undirected graph $G = (V, E)$ with nonnegative integer weights and $S \subseteq V$, outputs an estimate R' such that $R_S/2 \leq D' \leq R_S$.*

Proof. Run Dijkstra's algorithm from s and return $R' = \max_{s' \in S} d(s, s')$.

Let $c^* \in S$ be the true center. Then since $d(c^*, s') \leq R_S$ for all $s' \in S$, the triangle inequality implies that for all s', $d(s, s') \leq 2R_S$. Thus, $R_S \leq R' \leq 2R_S$. □

Theorem 6.1.19 (Directed Subset Eccentricities)**.** *Suppose that we are given a directed graph $G = (V, E)$ with nonnegative integer weights. For any $1 > \tau > 0$ we can in $\tilde{O}(m/\tau)$ time output for all $v \in S$ an estimate $\varepsilon'(v)$ such that $\frac{1-\tau}{2}\varepsilon_S(v) \leq \varepsilon'(v) \leq \varepsilon_S(v)$.*

Proof. The algorithm proceeds in iterations and maintains a set U of nodes for which we still do not have a good Eccentricity estimate. In each iteration either we get a good estimate for many new vertices and hence remove them from U, or we remove all vertices from U that have large Eccentricities, and for the remaining nodes in U we have a better upper bound on their Eccentricities. After a small number of iterations we have a good estimate for all vertices of the graph. Initially $U = S$ and we will end with $|U| \leq O(1)$. When $|U| \leq O(1)$ we can evaluate $\varepsilon_S(v)$ for all $v \in U$ in the total time of $O(m)$.

Also we maintain a value D that upper bounds the largest Eccentricity of a vertex in U. That is, $\varepsilon_S(v) \leq D$ for all $v \in U$. Initially we set $D = n^C$ for some large enough constant $C > 0$ (we assume that the set S is strongly connected). The algorithm proceeds in phases. Each phase takes $\tilde{O}(m)$ time and either $|U|$ decreases by a factor of at least 2 or D decreases by a factor of at least $1/(1-\tau)$. After $O(\log(n)/\tau)$ phases either $|U| \leq O(1)$ or $D < 1$.

For a subset $U \subseteq V$ of vertices and a vertex $x \in V$ we define a set $U_x \subseteq S$ to contain those $|U_x| = |U|/2$ vertices from U that are closest to x (according to distance $d(\cdot, x)$). The ties are broken by taking the vertex with the smaller id. Given a subset $U \subseteq V$ of vertices and a threshold D, a phase proceeds as follows.

- We sample a set $A \subseteq U$ of $O(\log n)$ random vertices from the set U. By Lemma 6.1.1, with high probability for all $x \in V$ we have $A \cap U_x \neq \emptyset$.
- Let w be the vertex in S that maximizes $d(A, w)$. We can find it by constructing a vertex y adjacent to every vertex in A and running Dijkstra's algorithm from y.
- We consider two cases.

  Case $d(U \setminus U_w, w) \geq \frac{1-\tau}{2}D$**.** For all $x \in U \setminus U_w$ we have $\frac{1-\tau}{2}D \leq \varepsilon_S(x) \leq D$ and we assign the estimate $\varepsilon'(x) = \frac{1-\tau}{2}D$. This gives us that $\frac{1-\tau}{2}\varepsilon_S(x) \leq \frac{1-\tau}{2}D = \varepsilon'(x) \leq \varepsilon_S(x)$ for all $x \in U \setminus U_w$. We update U to be U_w. This decreases the size of U by a factor of 2 as

required.

Case $d(U \setminus U_w, w) < \frac{1-\tau}{2}D$. Set $U' = U$. For every vertex $v \in U$ evaluate $r_v := \max_{x \in A} d(v, x)$. We can evaluate these quantities by running Dijkstra's algorithm from every vertex in A and following the incoming edges. If $r_v \geq \frac{1-\tau}{2}D$, then assign the estimate $\varepsilon'(v) = \frac{1-\tau}{2}D$ and remove v from U'. Similarly as in the previous case we have $\frac{1-\tau}{2}\varepsilon_S(v) \leq \varepsilon'(v) \leq \varepsilon_S(v)$ for all $v \in U \setminus U'$. Below we will show that for every $v \in U'$ we have $\varepsilon_S(v) \leq (1-\tau)D$. Thus we can update $U = U'$ and decrease the threshold D to $(1-\tau)D$ as required.

Correctness We have to show that, if there exists $v \in U'$ such that $\varepsilon_S(v) > (1-\tau)D$, then we will end up in the first case (this is the contrapositive of the claim in the second case). Since $v \in U'$ we must have that $d(v, x) \leq \frac{1-\tau}{2}D$ for all $x \in A$. Since $\varepsilon_S(v) > (1-\tau)D$, we must have that there exists $v' \in S$ such that $d(v, v') > (1-\tau)D$. By the triangle inequality we get that $d(x, v') > \frac{1-\tau}{2}D$ for every $x \in A$. By choice of w, we have $d(A, w) > \frac{1-\tau}{2}D$. Since $A \cap U_w \neq \emptyset$, we have $d(U \setminus U_w, w) \geq \frac{1-\tau}{2}D$ and we will end up in the first case.

The guarantee on the approximation factor follows from the description. □

Directed Subset Radius Using the argument from Section 6.1.4, we obtain an algorithm for Directed Subset Radius from our algorithm for Directed Subset Eccentricities.

Theorem 6.1.20 (Directed Subset Radius)**.** *Suppose that we are given a directed graph $G = (V, E)$ with nonnegative integer weights. For any $1 > \tau > 0$ we can in $\tilde{O}(m/\tau)$ time output an estimate R' such that $R_S \leq R' \leq \frac{2}{1-\tau}R_S$.*

6.1.6 Parameterized Algorithms for Bichromatic Diameter, Radius, and Eccentricities

In this section we give algorithms for Bichromatic Diameter, Radius, and Eccentricities with runtimes parameterized by the size of the *boundary* B. Let S' be the set of vertices in S that have a neighbor in T and let T' be the set of vertices in T that have a neighbor in S. Let B be whichever of S' or T' is smaller in size.

Undirected Parameterized Bichromatic Diameter

Theorem 6.1.21. *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 $2D_{ST}(G)/3 - 1 \leq D' \leq D_{ST}(G)$.*

Proof. **(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. □

Undirected Parameterized Bichromatic Radius

Theorem 6.1.22. *There is an $O(m|B|)$ time algorithm that, given an unweighted undirected graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, returns an estimate R' such that $R_{ST}G \leq R' \leq 3R_{ST}(G)/2 + 3$.*

Proof. **(Algorithm):** If $B \subseteq S$, we run BFS from all $v \in B$ and let R_1 be the minimum Eccentricity found; that is, $R_1 = \min_{v \in B} \varepsilon_{ST}(v)$. If $B \subseteq T$, for every $v \in B$, we let v_S be an arbitrary

neighbor of v such that $v_S \in S$, and run BFS from v_S. In this case we let $R_1 = \min_{v \in B} \varepsilon_{ST}(v_S)$. Let U be the set of vertices that we have run BFS from so far.

Then, let $s \in S$ be the vertex that is closest to *all* vertices in U; that is, let s be the vertex that minimizes $\max_{v \in U} d(s, v)$. Run BFS from s and let $R_2 = \varepsilon_{ST}(s)$. Return $\min\{R_1, R_2\}$.

(Analysis): Let $c^* \in S$ be the true center and let R denote $R_{ST}(G)$; that is, $\varepsilon_{ST}(c^*) = R$. If there exists a vertex $v \in U$ such that $d(c^*, v) \leq R/2$, then since $U \subseteq S$ and by the triangle inequality, $\varepsilon_{ST}(v) \leq 3R/2$ and we are done.

If we are not done by the previous step, c^* must be of distance at least $R/2$ from *every* vertex in U, and thus of distance at least $R/2 - 1$ from every vertex in B. We observe that the shortest path between s and any vertex in T must contain a vertex in B. Thus, every vertex in T must be of distance at most $R/2 + 1$ from *some* vertex in B, and thus of distance at most $R/2 + 2$ from some vertex in U.

Since for all $v \in T$, $d(c^*, v) \leq R$, the triangle inequality implies that for all $v \in U$, $d(c^*, v) \leq R + 1$. Therefore, by choice of s, for all $v \in U$, $d(s, v) \leq R + 1$. We claim that $\varepsilon_{ST}(s) \leq 3R/2$. Consider an arbitrary vertex $t \in T$. Let u be a vertex in U such that $d(u, t) \leq R/2 + 2$; such a u exists by the previous paragraph. Then, $d(s, u) + d(u, t) \leq (R + 1) + (R/2 + 2) = 3R/2 + 3$. Thus, $\varepsilon_{ST}(s) \leq 3R/2 + 3$. □

Undirected Parameterized Bichromatic Eccentricities

Theorem 6.1.23. *There is an $O(m|B|)$ time algorithm that, given an unweighted undirected graph $G = (V, E)$ and $S \subseteq V, T = V \setminus S$, returns for every $v \in S$ an estimate $\varepsilon'(v)$ such that $3\varepsilon_{ST}(v)/5 - 1 \leq \varepsilon'(v) \leq \varepsilon_{ST}(v)$.*

Proof. **(Algorithm):** Suppose $B \subseteq S$. For every vertex $u \in B$, we run BFS from u, let u' be the vertex in T that maximizes $d(u, u')$, and run BFS from u'. Then for every vertex $u \in B$ we let u_T be an arbitrary neighbor of u such that $u_T \in T$ and run BFS from u_T. Then, let $t \in T$ be the farthest vertex from B; that is, t is the vertex in T that maximizes $d(B, t)$. Let T'' be the set of vertices in T that we have run BFS from. For every vertex $v \in S$, we return the estimate $\varepsilon'(v) = \max_{t'' \in T''} d(v, t'')$.

We use a similar algorithm for when $B \subseteq T$: For every vertex $u \in B$, we run BFS from u,

let u' be the vertex in T that maximizes $d(u, u')$, and run BFS from u'. Then, let $t \in T$ be the farthest vertex from B; that is, t is the vertex in T that maximizes $\min_{u \in B} d(u, t)$. Let T'' be the set of vertices in T that we we have run BFS from. For every vertex $v \in S$, we return the estimate $\varepsilon'(v) = \max_{t'' \in T''} d(v, t'')$.

(Analysis): Suppose $B \subseteq S$. If there exists a vertex in $u \in B$ such that $d(v, u) \geq 3\varepsilon_{ST}(v)/5$, then $d(v, u_T) \geq 3\varepsilon_{ST}(v)/5 - 1$ so we are done. On the other hand, suppose $B \subseteq T$. If there exists a vertex in $u \in B$ such that $d(v, u_T) \geq 3\varepsilon_{ST}(v)/5 - 1$, then we are done. Otherwise, v is of distance at most $3\varepsilon_{ST}(v)/5$ from *every* vertex in B. Thus, regardless of whether $B \subseteq S$ or T, if we are not already done, v is of distance at most $3\varepsilon_{ST}(v)/5$ from *every* vertex in B.

Then, since every path from v to any vertex in T must contain a vertex in B, there must exist a vertex in T that is of distance at least $2\varepsilon_{ST}(v)/5$ from *every* vertex in B. In particular, t must be of distance at least $2\varepsilon_{ST}(v)/5$ from every vertex in B.

Let v' be the true farthest vertex from v; that is, $d(v, v') = \varepsilon_{ST}(v)$. If there exists a vertex in $u \in B$ such that $d(v, u) \leq \varepsilon_{ST}(v)/5$, then by the triangle inequality $d(u, v') \geq 4\varepsilon_{ST}(v)/5$, so $d(u, u') \geq 4\varepsilon_{ST}(v)/5$. Applying the triangle inequality again, $d(v, u') \geq 3\varepsilon_{ST}(v)/5$, so we are done. Otherwise, every vertex $u \in B$ is of distance at least $\varepsilon_{ST}(v)/5$ from v.

We claim that if we are not already done, $d(v, t) \geq 3\varepsilon_{ST}(v)/5$. We observe that every path from v to t must contain a vertex in B. Let $u \in B$ be a vertex on the shortest path from v to t. Then, $d(v, t) = d(v, u) + d(u, t) \geq \varepsilon_{ST}(v)/5 + 2\varepsilon_{ST}(v)/5 = 3\varepsilon_{ST}(v)/5$. □

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 6.1.24. *There is an $O(m|B'|)$ time algorithm that, given an unweighted directed graph $G = (V, E)$ and $S \subseteq V, T = V \backslash S$, 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 $D_2 \geq 2D/3 + 1$. □

6.1.7 Conditional Lower Bounds

Bichromatic Diameter, Eccentricities, and Radius

Undirected Bichromatic Diameter The following theorem implies that our algorithms for undirected Bichromatic Diameter from Theorem 6.1.7 and Proposition 1 are tight under SETH.

Theorem 6.1.25. *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 6.1.25 implies that our $m^{3/2}$ time $5/3$-approximation algorithm from Theorem 6.1.7 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 1 is tight under SETH.

Theorem 6.1.25 follows from the following lemma.

Lemma 6.1.4. *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.

Construction of the graph. We begin with the k-OV-graph from Theorem 6.1.5. 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 of the construction.

Case 1: The k-OV instance has no solution. By property 3 of Theorem 6.1.5 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 6.1.5 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 6.1.5, $d((a_0, \ldots, a_{k-2}) \in S, (a_1, \ldots, a_{k-1}) \in L_k) \geq 3k - 2$. Thus, $d((a_0, \ldots, a_{k-2}) \in S, (a_1, \ldots, a_{k-1}) \in L_{2k-1})) \geq 4k - 3$. □

Undirected Bichromatic Eccentricities The following proposition implies that our algorithms for undirected Bichromatic Eccentricities from Theorem 6.1.12 and Proposition 2 are tight under SETH.

Proposition 6. *Under SETH, for every $k \geq 2$, every algorithm that can distinguish between Bichromatic Eccentricities k and $3k-2$ in undirected unweighted graphs requires $m^{1+1/(k-1)-o(1)}$ time.*

In particular setting $k = 2$ and 3 in Theorem 6 implies that our $m^{3/2}$ time 2-approximation algorithm from Theorem 6.1.12 is tight under SETH in approximation factor and runtime, respectively. Furthermore, setting k to be arbitrarily large implies that our $\tilde{O}(m)$ time almost 3-approximation algorithm from Proposition 2 is tight under SETH.

Proposition 6 follows from the following lemma.

Lemma 6.1.5. *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. Let S_0 be a particular subset of S.*

1. *If the k-OV instance has no solution, then for* all *vertices $v \in S_0$ we have $\varepsilon_{ST}(v) \leq k$.*
2. *If the k-OV instance has a solution, then there exists a vertex $v \in S_0$ such that $\varepsilon_{ST}(v) \geq 3k-2$.*

Proof. We begin with the k-OV-graph from Theorem 6.1.5. Let $T = L_k$ and let S contain the rest of the vertices in the graph. Let $S_0 = L_0$.

If the k-OV instance has no solution then by property 3 of Theorem 6.1.5 for all $u \in L_0$ and $v \in T$, $d(u, v) = k$. Thus, for all $u \in L_0$, $\varepsilon_{ST}(u) = k$.

Suppose the k-OV instance has a solution $(a_0, \ldots, a_{k-1})$. Then by property 4 of Theorem 6.1.5, $d((a_0, \ldots, a_{k-2}) \in L_0, (a_1, \ldots, a_{k-1}) \in T) \geq 3k-2$, so $\varepsilon_{ST}(a_0, \ldots, a_{k-2}) \geq 3k-2$. □

Undirected Bichromatic Radius The following theorem implies that our $\tilde{O}(m^{3/2})$ time $5/3$-approximation algorithm for undirected Bichromatic Radius from Theorem 6.1.10 is tight in approximation factor under the HS hypothesis.

Theorem 6.1.26. *Under the HS hypothesis, any algorithm for Bichromatic Radius that achieves a $(5/3-\delta)$-approximation factor for $\delta > 0$ in m-edge undirected unweighted 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. Create G' which has the same vertex set as $G(U,V)$ except instead of having a vertex for every $v \in V$ it has two copies $v_1 \in V_1$ and $v_2 \in V_2$.

The edges for G' are: for $u \in U, c \in C$, we add (u,c) as an edge iff $u[c] = 1$. For $v \in V, c \in C$, we add (c, v_1) as an edge iff $v[c] = 1$. For each $v \in V$ we add an edge (v_1, v_2). Set $S = U$ and $T = V_1 \cup V_2 \cup C$. The number of edges in the graph is $O(nd)$.

Suppose that there is no HS solution, then for all $u \in U$ there is some $v \in V$ so that $u \cdot v = 0$ and hence $d(u, v_2) \geq 5$. If there is an HS solution $u \in U$, then for all $t \in T$, $d(u,t) \leq 3$. □

Directed Bichromatic Diameter The following theorem implies that our $m^{3/2}$ 2-approximation algorithm for directed Bichromatic Diameter from Theorem 6.1.13 has a tight approximation factor under SETH.

Theorem 6.1.27. *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 Bichromatic Eccentricities

Proposition 7. *Under SETH, any algorithm for Bichromatic Eccentricities 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 $S = U \cup C$, $T = V$. Notice this is an instance of Bichromatic Eccentricities.

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-Eccentricity for every $u \in U \subseteq S$ is ∞, and otherwise it is 2. Any finite approximation to the ST-Eccentricities can distinguish between ∞ and 2, and thus can solve OV. (Notice, we do not even need the Eccentricities of nodes in C.) Thus, there can be no $m^{2-\varepsilon}$ time algorithm for $\varepsilon > 0$ that achieves a finite approximation factor if SETH holds. □

Directed Bichromatic Radius

Proposition 8. *Under the HS hypothesis, any algorithm for Bichromatic Radius that achieves a finite approximation factor in m-edge directed graphs requires $m^{2-o(1)}$ time.*

Proof. The proof is similar to that for Bichromatic Eccentricities. Given an instance $U, V \subseteq \{0,1\}^d$ of HS, let $G(U,V)$ be its OV-graph. Now, direct the edges from U to C and from C to V, and add an extra node z so that for every $u \in U$ there is a directed edge (u,z). Set $S = U \cup C$, $T = V \cup \{z\}$.

First, if the ST-Radius is finite, the ST-center (the node achieving the Radius) must be in U, since no node in C can reach z, by construction. The distance $d(u,z)$ is 1 for all $u \in U$. 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 a HS solution, then the ST-Radius is 2, and otherwise it is ∞. Any finite approximation to the ST-Radius can distinguish between ∞ and 2, and thus can solve HS. Thus, there can be no $m^{2-\varepsilon}$ time algorithm for $\varepsilon > 0$ that achieves a finite approximation factor if the HS hypothesis holds. □

ST-Diameter, Eccentricities, and Radius

Undirected ST-Diameter and Eccentricities For undirected graphs, Backurs et al. [BRS$^+$18] give a time-accuracy trade-off lower bound for ST-Diameter that immediately extends to ST-Eccentricities (since any Eccentricities algorithm gives a Diameter algorithm with the same running time and accuracy by taking the maximum of Eccentricities).

The following theorem shows that our algorithms for ST-Eccentricities from Theorem 6.1.15 and Proposition 3 are tight under SETH.

Theorem 6.1.28 ([BRS$^+$18])**.** *Under SETH, for every $k \geq 2$, every algorithm for ST-Diameter (and thus ST-Eccentricities) that achieves a $((4k-3)/(2k-1)-\delta)$-approximation for $\delta > 0$ in undirected unweighted graphs requires $m^{1+1/(k-1)-o(1)}$ time.*

In particular, setting $k = 2$ and 3 in Theorem 6.1.28 shows that our $m^{3/2}$ time 2-approximation algorithm for ST-Eccentricities from Theorem 6.1.15 is tight under SETH, in terms of both approximation factor and runtime. Furthermore, setting k to be arbitrarily large implies that our $\tilde{O}(m)$ time 3-approximation algorithm for ST-Eccentricities from Proposition 3 is tight under SETH.

Undirected ST-Radius The following proposition shows that our $\tilde{O}(m^{3/2})$ time 2-approximation algorithm for undirected ST-Radius from Theorem 6.1.18 has a tight approximation factor under the HS hypothesis.

Proposition 9. *Under the HS hypothesis, any algorithm for ST-Radius that achieves a $(2-\delta)$-approximation for $\delta > 0$ in m-edge undirected graphs requires $m^{2-o(1)}$ time.*

Proof. Given an instance $U, V \in \{0,1\}^d$ of HS, let G be the OV-graph defined on this instance. Let $S = U$ and $T = V$. Suppose that there is a node $u \in U$ which is not orthogonal to any node in V. Then for each $v \in V$, $d(u,v) = 2$ by using the coordinate node on which both u and v are 1. So in this case the ST-Radius is 2. Suppose on the other hand that no such node in U exists, so that for each node $u \in U$, there is a node $v \in V$ such that $u \cdot v = 0$. Then $d(u,v) \geq 4$. Since $S = U$, the ST-Radius is at least 4 in this case.

So any $(2-\delta)$-approximation algorithm can distinguish between ST-Radius 2 and 4, and thus

solve HS. Therefore, there can be no $m^{2-\epsilon}$ time algorithm for $\epsilon > 0$ that achieves a $(2-\delta)$-approximation factor if HS hypothesis holds. □

Directed ST-Diameter

Proposition 10. *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 to the 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. □

Directed ST-Eccentricities and Radius Propositions 7 and 8 immediately carry over to Directed ST-Eccentricities and ST-Radius since the Bichromatic version is a special case of the ST version. We state the results here for convenience.

Proposition 11. *Under SETH, any algorithm for ST-Eccentricities that achieves a finite approximation factor in m-edge directed graphs requires $m^{2-o(1)}$ time.*

Proposition 12. *Under the HS hypothesis, any algorithm for ST-Radius that achieves a finite approximation factor in m-edge directed graphs requires $m^{2-o(1)}$ time.*

Subset Diameter, Eccentricities, and Radius

Subset Diameter and Eccentricities The following proposition implies that our $\tilde{O}(m)$ time 2-approximation algorithm for Subset Diameter from Proposition 4 is tight under SETH, and that our near-linear time almost 2-approximation algorithm for Subset Eccentricities from Theorem 6.1.19 is essentially tight under SETH.

Proposition 13. *Under SETH, any algorithm for Subset Diameter (and thus Subset Eccentricities) that achieves a $(2-\delta)$-approximation factor 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. □

Subset Radius The following proposition implies that our $\tilde{O}(m)$ time 2-approximation algorithm for Subset Radius from Proposition 5 is tight under the HS hypothesis.

Proposition 14. *Under the HS hypothesis, any algorithm for Subset Radius that achieves a $(2-\delta)$-approximation factor for $\delta > 0$ in m-edge undirected graphs requires $m^{2-o(1)}$ time.*

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

If there is no HS solution, then in the original OV-graph, for all $s \in U$, there exists some $s' \in V$ such that $d(s, s') \geq 4$. We note that the addition of the vertices u and v does not change this fact. Furthermore, for all vertices $s \in V$, $d(v, s) = 4$. Thus, the Subset Radius is at least 4.

On the other hand, if there is a HS solution, then there exists a vertex $s \in U$ such that for all vertices $s' \in V$, $d(s, s') = 2$. Also, $d(s, v) = 2$. Thus, the Subset Radius is 2. □

Parameterized Bichromatic Diameter, Eccentricities, and Radius

In this section we show that modifications of our lower bound constructions show that our algorithms parameterized by the boundary size $|B|$ for Bichromatic Diameter, Eccentricities, and Radius are conditionally tight. Recall that for undirected graphs, S' is the set of vertices in S that have a neighbor in T, T' is the set of vertices in T that have a neighbor in S, and B is whichever of S' or T' is smaller in size. Since these our parameterized algorithms for undirected graphs have additive error, instead of showing that e.g. distinguishing between values 2 and 3 is hard, we will

give results of the form "for all ℓ, distinguishing between e.g. 2ℓ and 3ℓ is hard". This proves that even algorithms with constant additive error cannot achieve a better multiplicative approximation factor than e.g. $3/2$.

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 6.1.21 is tight under SETH for $|B| = \omega(\log n)$.

Theorem 6.1.29. *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.

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, $d(u', v') = d(u', u) + d(u, v) + d(v, v') \geq 6\ell$. □

Undirected Parameterized Bichromatic Eccentricities The following proposition implies that the multiplicative factor in our $\tilde{O}(m|B|)$ time almost $5/3$-approximation algorithm for undirected Bichromatic Eccentricities from Theorem 6.1.23 is tight under SETH for $|B| = \omega(\log n)$.

Proposition 15. *For any integer $\ell > 0$, under SETH any algorithm for Bichromatic Eccentricities in undirected unweighted graphs that distinguishes for all vertices v between $\varepsilon_{ST}(v) = 3\ell$ and $\varepsilon_{ST}(v) = 5\ell$ requires $m^{2-o(1)}$ time, even for graphs with $|B| = d = \tilde{O}(1)$.*

Proof.

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 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 S contain U, C, and the vertices that subdivide the edges between U and C. Let T contain the remaining vertices.

Analysis We note that $S' = C$ and $|C| = d$ so $|B| = d = \tilde{O}(1)$.

If there is no OV solution, then for all pairs of vertices $u \in U$, $v \in V$, $d(u,v) = 2\ell$. Every vertex in T is of distance at most ℓ from some vertex in T so for all vertices $u \in U$, $\varepsilon_{ST}(u) \leq 3\ell$.

If there is an OV solution $u \in U$, $v \in V$, $d(u,v) \geq 4\ell$. Let $v' \in V'$ be the vertex matching to v. Then, $d(u,v') \geq 5\ell$ so $\varepsilon_{ST}(v) \geq 5\ell$. □

Undirected Parameterized Bichromatic Radius The following theorem implies that the multiplicative factor in our $\tilde{O}(m|B|)$ time almost $3/2$-approximation algorithm for undirected Bichromatic Radius from Theorem 6.1.22 is tight under the HS hypothesis for $|B| = \omega(\log n)$.

Theorem 6.1.30. *For any integer $\ell > 0$, under the HS hypothesis any algorithm for Bichromatic Radius in undirected unweighted graphs that distinguishes between Bichromatic Radius 4ℓ and 6ℓ requires $m^{2-o(1)}$ time, even for graphs with $|B| = d = \tilde{O}(1)$.*

Proof.

Construction Given an instance $U, V \in \{0,1\}^d$ of HS, we begin with two copies of the construction from Theorem 6.1.29, U_1', U_1, C_1, V_1, V_1', and U_2', U_2, C_2, V_2, V_2'. We then merge each vertex in U_1' with its corresponding vertex in U_2'.

Analysis We note that $T' = C_1 \cup C_2$ and $|C_1| = |C_2| = d$ so $|B| = 2d = \tilde{O}(1)$.

It will be convenient to imagine that the graph is layered from left to right as V_2', V_2, C_2, U_2, U_1', U_1, C_1, V_1, V_1'.

If there is no HS solution, then for all $u_1 \in U_1$, there exists some $v_1 \in V_1$ such that $d(u_1, v_1) \geq 4\ell$ and for all $u_2 \in U_2$, there exists some $v_2 \in V_2$ such that $d(u_2, v_2) \geq 4\ell$. Let u be any vertex in S that lies in U_1' or to the right of U_1'. Since any path u to a vertex in V_2' contains a vertex in U_2, there exists $v \in V_2'$ such that $d(u, v) \geq 6\ell$. Symmetrically, if u is a vertex in S that lies to the left of U_1', there exists $v \in V_1'$ such that $d(u, v) \geq 6\ell$. Thus, the Bichromatic Radius is at least 6ℓ.

On the other hand, if there is a HS solution, then there exists a vertex $u \in U_1$ such that for all vertices $v \in V_1$, $d(u, v) = 2\ell$. Let u' be the vertex in U_1' matched to u and let u'' be the vertex in U_2 matched to u'. Then, for all vertices $v \in V_2$, $d(u'', v) = 2\ell$. Thus, for all vertices $v \in V_1 \cup V_2$, $d(u', v) = 3\ell$, so for all vertices $v \in T$, $d(u', v) \leq 4\ell$. Thus, the Bichromatic Radius is at most 4ℓ. □

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 6.1.29 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 6.1.24 is tight under SETH for $|B'| = \omega(\log n)$.

Proposition 16. *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.

Construction We begin with the construction from Theorem 6.1.29. 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 6.1.29. 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. □

6.2 Min-distance problems in general graphs

This section was written with authors Virginia Vassilevska Williams, Nikhil Vyas, Nicole Wein, Yinzhan Xu and Yuancheng Yu, and focuses on the second type of variants of distance problems in general graphs. We study fundamental graph parameters such as the Diameter and Radius in directed graphs, when distances are measured using a somewhat unorthodox but natural measure: the distance between u and v is the *minimum* of the shortest path distances from u to v and from v to u. The center node in a graph under this measure can for instance represent the optimal location for a hospital to ensure the fastest medical care for everyone, as one can either go to the hospital, or a doctor can be sent to help.

By computing All-Pairs Shortest Paths, all pairwise distances and thus the parameters we study can be computed exactly in $\tilde{O}(mn)$ time for directed graphs on n vertices, m edges and nonnegative edge weights. Furthermore, this time bound is tight under the Strong Exponential Time Hypothesis [Roditty-Vassilevska W. STOC 2013] so it is natural to study how well these parameters can be *approximated* in $O(mn^{1-\varepsilon})$ time for constant $\varepsilon > 0$. Abboud, Vassilevska Williams, and Wang [SODA 2016] gave a polynomial factor approximation for Diameter and Radius, as well as a constant factor approximation for both problems in the special case where the graph is a DAG. We greatly improve upon these bounds by providing the first constant factor approximations for Diameter, Radius and the related Eccentricities problem in general graphs. Additionally, we provide a hierarchy of algorithms for Diameter that gives a time/accuracy trade-off.

6.2.1 Introduction

The diameter, radius and eccentricities of a graph are fundamental parameters that have been extensively studied [Chu87, Hak64, CD94, Epp99, ACIM99, CDHP01, CDV02, DH04, BMBST07, BK07, Wul08, Yus10, Cha12, FHW12, WY13, RV13, CLR$^+$14, AGV15, BCH$^+$15] (and many others). The eccentricity of a vertex v is the largest distance between v and any other vertex. The diameter is the maximum eccentricity of a vertex in the graph, thus measuring how far apart two nodes can be, and the radius is the minimum eccentricity, measuring the maximum distance to the most central node.

The distance between two vertices in an undirected graph is just the shortest path distance $d(\cdot, \cdot)$ between them. For directed graphs, however, this notion of distance d is no longer necessarily symmetric, and rather than being a distance *between* two nodes, it measures the distance in a given direction. Several related notions of pairwise distance that are symmetric have been studied. These include the roundtrip distance [CW99] which for two vertices u and v is just $d(u, v) + d(v, u)$, the max-distance [AVW16] which is $\max\{d(u, v), d(v, u)\}$, and the min-distance [AVW16] which is $\min\{d(u, v), d(v, u)\}$.

Each of these notions of distance has a particular application. For instance, one would have to pay the roundtrip distance when going to the store and back. On the other hand, if one needs medical assistance, one could either go to the hospital, or have a physician come to the home — the time to receive care is then measured by the min-distance. Another example of min-distance is in symmetric-key encryption: any pair of parties can create a shared private key by using only one-way communication.

For each notion of distance, the diameter, radius and eccentricity parameters are well-defined. Given the shortest path distances $d(\cdot, \cdot)$ for all vertices, the parameters for each distance measure can be computed in $O(n^2)$ time in n vertex graphs. The fastest known algorithms for All-Pairs Shortest Paths (APSP) [Wil14, Pet02, PR05] give the fastest known algorithms to compute these parameters exactly, running in $n^3/\exp(\sqrt{\log n})$ time and $O(mn + n^2 \log\log n)$, respectively on m-edge, n-vertex graphs. Furthermore, under the Strong Exponential Hypothesis, there is no $O(m^{2-\varepsilon})$ time algorithm for Diameter in unweighted graphs (and thus also for any of these notions

of Diameter and Eccentricities in directed graphs) [RV13]. For Radius, the same lower bound holds but under the “Hitting Set" conjecture [AVW16].

As exact computation is expensive, it makes sense to resort to approximation algorithms. For the shortest path distance versions of Diameter, Eccentricities and Radius, there are several fast algorithms that achieve various small constant approximation ratios [RV13, CLR$^+$14, CGR16, BRS$^+$18]. For instance, for Diameter, a folklore linear time algorithm can achieve a 2-approximation, and an $\tilde{O}(m^{3/2})$ time[3] algorithm can achieve a $3/2$-approximation [RV13, CLR$^+$14].

Many of these algorithms [RV13, CLR$^+$14, BRS$^+$18] work for any distance measure that satisfies the triangle inequality. Thus they work for the shortest paths distance, max-distance and roundtrip distance. The min-distance however does not satisfy the triangle inequality: e.g. you might have edges (x, y) and (z, y), and thus the min-distance between x and y and between y and z are both 1, yet there may be no directed path between x and z in any direction, so that the min-distance between them may be ∞.

This issue makes it much more difficult to design fast approximation algorithms for Min-Diameter, Min-Radius and Min-Eccentricities (the parameters of interest under the min-distance). The only known nontrivial algorithms are by Abboud et al. [AVW16]. For Min-Diameter [AVW16] gives a near-linear time 2-approximation algorithm if the input is a directed acyclic graph. For general graphs, the only nontrivial fast approximation algorithm is an $\tilde{O}(mn^{1-\varepsilon})$ time n^{ε}-approximation algorithm for any constant $\varepsilon > 0$. (No constant factor approximation algorithm is known that runs significantly faster than just computing APSP.) For Min-Radius, [AVW16] gives an $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for directed acyclic graphs. For general graphs, they only achieve a very weak n-approximation in near-linear time that checks if the Min-Radius is finite. There are no known approximation algorithms for Min-Eccentricities faster than just computing APSP.

[3]We use $\tilde{O}$ notation to hide polylogarithmic factors

Our Results

The main goal of this section is to obtain new fast, $O(mn^{1-\varepsilon})$ time for some constant $\varepsilon > 0$, algorithms for Min-Diameter, Min-Radius and Min-Eccentricities (thus beating the $\tilde{O}(mn)$ time of exact computation). We achieve this by developing powerful new techniques that can handle the complications that arise due to the fact that the min-distance does not satisfy the triangle inequality.

Our results are as follows. For Min-Diameter we achieve a hierarchy of algorithms trading off running time with approximation accuracy.

Theorem 6.2.1. *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 tradeoff 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, whereas for general graphs the only known efficient algorithm could achieve an n^{ε}-approximation.

For Min-Radius, we also achieve the first constant factor approximation algorithm for general graphs running in $O(mn^{1-\varepsilon})$ time for some constant $\varepsilon > 0$. Such a result was only known for directed acyclic graphs, whereas for general graphs the only known efficient algorithm could only check if the Min-Radius is finite.

Theorem 6.2.2. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph G with edge weights positive and polynomial in n, can output an estimate R' such that $R \leq R' \leq (3 + \delta)R$ with high probability, where R is the min-radius of G.*

Finally, we obtain the first $O(mn^{1-\varepsilon})$ time (for constant $\varepsilon > 0$) constant factor approximation algorithms for the Min-Eccentricities of all vertices in a graph. For unweighted graphs we are able

to obtain a close to 3 approximation in $\tilde{O}(m\sqrt{n})$ time. For weighted graphs, our approximation factor grows to 5, while the running time is the same. Previously, the only algorithm to approximate the Min-Eccentricities computed them exactly via an APSP computation.

Theorem 6.2.3. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (5+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of vertex s in G.*

Theorem 6.2.4. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta^2)$ time randomized algorithm, that given a directed unweighted graph $G = (V, E)$, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (3+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

Our Techniques

To obtain our results, we develop powerful new techniques which we outline below.

Partial search graphs. The idea of partial search graphs is used in the algorithms of [AVW16] for Min-Radius and Min-Diameter on DAGs. These algorithms use 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 desired quantity (either min-diameter or min-radius), then the partial search from some vertex v returns a good estimate of the min-eccentricity of v, which in turn is a good estimate for the desired quantity. 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 some interval containing v (though defining the exact intervals requires some work). 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 carefully define a DAG-like structure in general graphs. Such a structure may be of independent interest.

Defining a DAG-like structure in general graphs. It would be ideal to directly reduce the problem on general graphs to the problem on DAGs, however it is very unclear how to do this. Instead, we recognize that it suffices to define 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 $T_v = \{u : d(u,v) > d(v,u)\}$[4]. 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 runtime analysis. The obtained partial ordering is the starting point for all of our algorithms.

Min-Diameter: 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 distances in the original graph.

[4] 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

Min-Radius: refined DAG-like structure Our Min-Radius algorithm is much more delicate than our Min-Diameter algorithm due to the fact that for Min-Radius we care about small distances instead of large distances. In particular, the graph augmentation idea from our Min-Diameter algorithm does not help for Min-Radius because although the augmentations do not distort large distances much, they heavily distort small distances. Furthermore, the previously mentioned DAG-like structure for general graphs does not suffice for Min-Radius. However we use it as a starting point to define a more refined DAG-like partial ordering. Most of our algorithm is concerned with precisely arranging vertices in this partial ordering. Specifically, we structure the partial ordering to satisfy *roughly* the following property: for every pair of vertices u, v such that u appears before v in the partial ordering, $d(v, u)$ is large while $d(u, v)$ is small.

Notation

Given a graph $G = (V, E)$, $n = |V|$ and $m = |E|$. Graphs are directed and have non-negative weights polynomial in n unless otherwise specified. For any pair of vertices u and v, the *distance from u to v* $d(u, v)$ is the length of the shortest directed path from u to v. When the context is not clear, we write $d_G(u, v)$ to specify the graph G. The *min-distance* between a pair of vertices u and v is $d_{min}(u, v) = \min\{d(u, v), d(v, u)\}$. The *min-diameter* of a graph is $\max_{u,v \in V} d_{min}(u, v)$. The *min-radius* of a graph is $\min_{v \in V} \max_{u \in V} d_{min}(u, v)$. For any vertex v, the *min-eccentricity* of v is $\varepsilon(v) = \max_{u \in V} d_{min}(u, v)$. When the context is not clear, we say $\varepsilon_G(v)$ to specify the graph G. Note that we do not use the *min* subscript to denote the min-eccentricity of a vertex. 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.

Organization

In Section 6.2.2 we give an overview of all of our algorithms, in Section 6.2.3 we describe a graph partitioning procedure that begins all of our algorithms, in Section 6.2.4 we describe our Min-Diameter algorithms, in Section 6.2.5 we describe our Min-Radius algorithm, and in Section 6.2.6 we describe our Min-Eccentricities algorithm.

6.2.2 Overview of Algorithms

We use the algorithms from [AVW16] for Min-Diameter and Min-Radius on DAGs as inspiration. For each problem, we first outline the DAG algorithm and then provide intuition for how to apply these ideas to general graphs.

Min-Diameter

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.

Min-Radius

Algorithm for DAGs

We begin by outlining the $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for Min-Radius on DAGs from [AVW16], which is very different from and more involved than the Min-Diameter algorithm

on DAGs. We begin by considering a topological ordering of the vertices and performing Dijkstra's algorithm from a set W of $\tilde{O}(\sqrt{n})$ evenly spaced vertices including the first and last vertex. If a vertex $v \in W$ has min-eccentricity at most twice the true min-radius R then we have obtained a 2-approximation. (We do not know R in advance but we repeatedly run the algorithm with different values of R to perform a binary search on R.)

Otherwise, we will define intervals in the ordering such that the min-center c cannot be contained in any of these intervals. A key observation is that if there is a pair of vertices (u, v) such that u appears before v in the topological ordering and $d(u, v) > 2R$, then the min-center c cannot fall between u and v in the topological ordering. This is because if it did, then $d(u, c) \leq R$ and $d(c, v) \leq R$, so $d(u, v) \leq 2R$, a contradiction. We define the intervals that cannot contain c as follows: for all $v \in W$ we let a_v be the first vertex in the ordering such that $d(a_v, v) > 2R$ (if it exists, otherwise $a_v = v$) and define b_v to be the last vertex in the ordering such that $d(v, b_v) > 2R$ (if it exists, otherwise $b_v = v$). Then, the key observation implies that c cannot fall in the interval $[a_v, b_v]$ in the ordering. Now, we have a set of possibly overlapping intervals that cannot contain c. We take the union of these intervals to get a set of disjoint intervals that cannot contain c.

Every vertex u that does not appear in such an interval, falls between two consecutive intervals I_u and I'_u. We define the partial search graph of u to be the graph induced by the set of vertices in I_u or I'_u or between I_u and I'_u. After performing the partial searches, the algorithm returns 3 times the minimum min-radius of all partial search graph. Next we give the idea of the analysis, which demystifies the factor of 3 in the returned value.

We claim that if the min-eccentricity of a vertex with respect to its partial search graph is at most R, then its min-eccentricity with respect to the full graph is at most $3R$, and the min-eccentricity of the true min-center with respect to its partial search graph is at most R (because for any path in a DAG whose starting and ending points are in a certain interval, every vertex in the path is in that interval). Thus, assuming the claim, $3R$ is a 3-approximation for the min-radius. We now outline the proof of the claim. Let u be the min-center with the minimum min-radius R of all partial search graphs. Let $v \in W$ such that a_v is the first vertex (in the topological order) of I_u, then $v \in I_u$ and $d(v, u) \leq R$. Furthermore, by the definition of a_v, all vertices that appear before the beginning

of the interval I_u have distance at most $2R$ to v, and thus distance at most $3R$ to u. A symmetric argument holds for vertices that appear after the end of the interval I'_u. Hence the min-eccentricity of u with respect to the full graph is at most $3R$.

This algorithm runs in time $O(m\sqrt{n})$ because the vertices of W are evenly spaced so there are no more than $\sqrt{n}$ vertices between each pair of consecutive intervals. This implies that in the partial searches, each edge is only scanned $O(\sqrt{n})$ times. (Furthermore, repeatedly running the algorithm to binary search for R adds a logarithmic factor to the runtime.)

Algorithm for general graphs

We now give a high-level outline of our $\tilde{O}(m\sqrt{n})$ time 3-approximation algorithm for Min-Radius. This algorithm is much more delicate than our Min-Diameter algorithm, hence more of the details are deferred to the full description. We begin by running Dijkstra's algorithm from a set W of $\tilde{O}(\sqrt{n})$ randomly sampled vertices to recursively partition the vertices into S_v and T_v as outlined in Section 6.2.1. This defines an initial DAG-like structure, however our analysis requires constructing a much more refined DAG-like structure.

Perhaps counter-intuitively, it makes sense to place vertices that are *far* from each other in the graph *close* to each other in the DAG-like structure. The reason for this is illuminated by the Min-Radius algorithm on DAGs, in which we find pairs of vertices u, v that are far from each other and apply the key observation that the min-center cannot be between u and v in the topological ordering. Intuitively, it is as if we collapse the interval between u and v in the DAG since we do not have to search within this interval for the min-center. An analogous key observation is true for general graphs: if there is a pair of vertices (u, v) with $d_{min}(u, v) > 2R$, then either $c \in S_u \cap S_v$ or $c \in T_u \cap T_v$. This is because if $c \in T_u \cap S_v$, then $d(u, c) \leq R$ and $d(c, v) \leq R$ so $d(u, v) \leq 2R$, a contradiction; the last case $c \in S_u \cap T_v$ is symmetric. In our algorithm for general graphs, we ensure that far vertices are near each other in the DAG-like structure by doing the following: we let the *far graph* G_{far} be an undirected graph on V with an edge between $u \in W$ and $v \in V$ if $d_{min}(u, v) > 2R$. All vertices in W that are in the same connected component in G_{far} will be grouped in the DAG-like structure. We let F_i be the set of vertices in W that are in the i^{th} connected component of G_{far}.

To construct the DAG-like structure, we show that precisely chosen groups of F_is can be merged to create *supercomponents*, which constitute a DAG-like structure in the following sense: there is an ordering of supercomponents such that for every pair of vertices $u, v \in W$ where the supercomponent containing u appears before that containing v, $d(u, v)$ is small and $d(v, u)$ is large. Specifically, we define the *close graph* H whose vertex set is the set of F_is. We add a directed edge between a pair of vertices in H if there exists a short path (length $\leq 5R$) between the corresponding F_is. Then we merge all F_is that appear in the same strongly connected component of H into a supercomponent. This contraction of strongly connected components of H results in a DAG, which defines the ordering of the supercomponents.

Now that we have arranged the vertices in W into a DAG-like structure, we would like to fit every vertex in the graph into this structure. Based on the precise way that we have defined the supercomponents, we can use an intricate argument to show *roughly* the following property: for every vertex v there exists an i such that for every vertex $u \in W$ in the first i supercomponents, $d(u, v)$ is small and for every vertex $u \in W$ in the remaining supercomponents, $d(v, u)$ is small.

After fitting every vertex into the refined DAG-like ordering, we can define each partial search graph to be an interval in the ordering that is large enough to contain several supercomponents. In the algorithm for DAGs, there were two important properties of the partial search graphs: (1) the min-eccentricity of the true min-center with respect to its partial search graph is at most R, and (2) if the min-eccentricity of a vertex with respect to its partial search graph is at most R then its min-eccentricity with respect to the full graph is at most $3R$. We show that due to the precise structure of the supercomponents, refinements of properties (1) and (2) are also true for general graphs.

Intuitively, property (1) is roughly true because for every pair of vertices $u, v \in W$ such that u's supercomponent appears before v's in the ordering, $d(v, u) > 5R$, since otherwise this pair of supercomponents would be in the same strongly connected component of H and would have been merged into a single supercomponent. This implies that paths of length at most R to or from the min-center cannot stray beyond its partial search graph. Intuitively, property (2) is roughly true because for every pair of vertices $u, v \in W$ such that u's supercomponents appears before v's in

the ordering, $d(u, v) \leq 2R$ because otherwise, u and v would be in the same component of G_{far} and thus be in the same supercomponent. Thus, like the argument for DAGs, for all u, all vertices that appear before u's partial search graph G_u have distance at most $2R$ to each supercomponent in G_u, and thus distance at most $3R$ to u. A symmetric argument holds for vertices after u in the ordering.

Min-Eccentricities

Our Min-Eccentricities algorithm is a modification of our Min-Radius algorithm. In our Min-Radius algorithm, we identify a vertex whose min-eccentricity is at most about $3R$, where R is the true min-radius. In our Min-Eccentricities algorithm, we show that with some extra bookkeeping, the algorithm can identify *all* vertices with min-eccentricity at most about 5ρ for any ρ. We run the algorithm repeatedly, increasing ρ by a factor of $(1 + \delta)$ at each execution until we have estimated the min-eccentricity of every vertex.

The major modification of the Min-Radius algorithm here is that if one of the vertices that we run Dijkstra from has min-eccentricity at most 3ρ, we cannot stop running the algorithm, as we can in the Min-Radius algorithm. Instead, we use this vertex as a tool to find vertices with min-eccentricity at most 5ρ.

6.2.3 Preliminary Graph Partitioning

In this section we describe a graph partitioning procedure we use as a first step in our Min-Diameter, Min-Radius, and Min-Eccentricities algorithms. The goal of this partitioning is to define a DAG-like structure in general directed graphs.

Definition 6.2.1. *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 runtime 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 6.2.1. *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 6.2.1 to recursively construct a balanced partition of the vertices into a given number of of sets.

Lemma 6.2.2. *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 $\mathcal{V}_q = \{V_1 \dots V_{q+1}\}$.

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 6.2.1, 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, 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 6.2.1 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 6.2.1. 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 runtime is $\tilde{O}(mn^{1-c})$.

□

6.2.4 Min-Diameter Algorithm

Throughout this section, let D be the min-diameter, and let s^*, t^* the endpoints of the min-diameter. In this section we prove the time/accuracy trade-off theorem for Min-Diameter.

Theorem 6.2.5. *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.*

We first prove a special case of Theorem 6.2.5 where $\ell = 1$.

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

Theorem 6.2.6. (Theorem 6.2.5 *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 6.2.2 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 define $D' = \max_{w \in W} \varepsilon(w)$. 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.

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 6.2.3. *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 6.2.4. *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 6.2.2, property 2, there exists $w \in W$ such

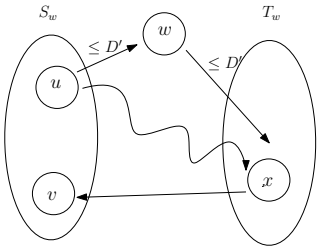

Figure 6-1: The case where $u, v \in S_w$ and the shortest path from u to v contains a node $x \in T_w \cup \{w\}$.

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.

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 6.2.5. *For every i, and every pair of vertices $u, v \in V_i$, $d_i'(u, v) \geq d(u, v) - 2D'$; that is, $d_{G_i^S}(u, v) \geq d(u, v) - 2D'$ 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,

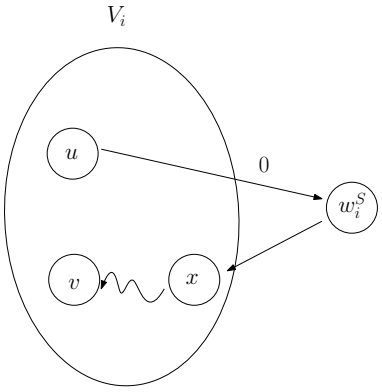

Figure 6-2: A shortest u, v path in G_i^S that contains w_i^S. The path goes from u, directly to w_i^S using a weight 0 edge, then directly to a vertex x, and finally reaches v.

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 6.2.4 $\max_{i,u\in V_i,v\in V_i} \min\{d'_i(u,v), d'_i(v,u)\} \leq \max_{i,u\in V_i,v\in V_i} d_{min}(u,v) \leq D$. 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 6.2.2, 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 6.2.3, $\varepsilon(w) \geq D/2$. 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 6.2.5, $\min\left(d'_i(s^*,t^*), d'_i(t^*,s^*)\right) \geq d_{min}(s^*,t^*) - 2D' = D - 2D'$. Since $\max\{D-2D', D'\} \geq D/3$, we get a 3-approximation.

Runtime analysis It takes $\tilde{O}(m\sqrt{n})$ time to perform the partitioning from Lemma 6.2.2 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 6.2.2 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.

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 6.2.2 to sample $q = \sqrt{n}$ vertices, we will apply Lemma 6.2.2 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 6.2.2 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 the 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 6.2.4 and Lemma 6.2.5.

Lemma 6.2.6. *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 6.2.4, 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 6.2.7. *For every i, and every pair of vertices $u, v \in V_i$, $d_{G_i}(u, v) \geq d(u, v) - 4D'$.*

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 6.2.5 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 6.2.5 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 guarantee: $D/(4\ell - 1) \leq \tilde{D} \leq D$. We prove the result inductively. When $\ell = 1$, it is exactly Theorem 6.2.6. 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 6.2.6, 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 6.2.2, 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 6.2.3, $\varepsilon(w) \geq D/2$. 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 6.2.7, $\min\{d_{G_i}(s^*, t^*), d_{G_i}(t^*, s^*)\} \geq d_{min}(s^*, t^*) - 4D' = D - 4D'$. Since we calculate a $(4\ell - 5)$-approximation of G_i's min diameter,

our estimate is at least

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

Runtime analysis It takes $\tilde{O}(mn^{1/(\ell+1)})$ time to perform the partitioning from Lemma 6.2.2 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 6.2.2, 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 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 6.2.2 at most $poly(n)$ times in the recursion and this the only randomization so the whole algorithm works with high probability.

6.2.5 Min-Radius Algorithm

Theorem 6.2.7. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm that, given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate R' such that $R \leq R' \leq (3 + \delta)R$ with high probability, where R is the min-radius of the G.*

Proof. We fix a value r and our algorithm either certifies that $R > r$ or $R \leq 3r$. Then by a binary search argument we get a $(3 + \delta)$-approximation as follows. Let $\delta' = \delta/3$. Starting from $r = 1$, we run the algorithm and increase r for each run. If the output of the algorithm is that $R \leq 3r$, then we stop. Otherwise (if $R > r$), we run the algorithm with the new value $r_{new} = (1 + \delta')r$. This contributes a multiplicative factor of $\log_{1+\delta'} R = \tilde{O}(1/\delta)$ to the total runtime. Suppose that for some value of r we have $R \leq 3r$. So from the previous run of the algorithm, we know that $R > r/(1 + \delta')$. Letting $R' = 3r$, we have $R \leq 3r = R' < 3(1 + \delta')R = (3 + \delta)R$, which means that R' is a $(3 + \delta)$-approximation. Now we present the algorithm.

Algorithm Step 1: Preliminaries

Let c be the min-center (which is unknown). First we remove all the edges with weight more than r, because if $R \leq r$, this removal does not change the min-radius. Then we sample a set W

of $\sqrt{n}$ vertices according to Lemma 6.2.2. For every vertex $v \in W$, we run Dijkstra's algorithm from and to v to obtain the min-distance between v and all other vertices. If there exists a vertex $v \in W$ with $\varepsilon(v) \leq 3r$, we have certified that $R \leq 3r$ so we are done.

Algorithm Step 2: Constructing the "far graph"

Now we can assume that for each $v \in W$, $\varepsilon(v) > 3r$. We say that a pair of vertices is *far* if their min-distance is more than $2r$, and let the *far graph* G_{far} be an undirected unweighted graph on V defined as follows: for each $u \in W$ and $v \in V$, (u, v) is an undirected edge if u and v are far. We partition W based on the connected components of G_{far}. Specifically, for all i define Z_i to be the i^{th} connected component of G_{far} which contains at least one vertex in W. Let $F_i = W \cap Z_i$, note that F_i is non-empty.

Analysis Step 2

Remember that we defined $S_U = \bigcap_{v \in U} S_v$ and $T_U = \bigcap_{v \in U} T_v$.

By constructing G_{far}, we prune the set of candidate min-centers, as specified in the following lemma.

Lemma 6.2.8. *If $R \leq r$, then for any F_i either $c \in S_{F_i}$ or $c \in T_{F_i}$.*

Proof. First note that we have $S_{F_i} \cup T_{F_i} \neq V \setminus F_i$. We know that $c \notin F_i$ as $F_i \subseteq W$. By way of contradiction, assume that there are two vertices $u, v \in F_i$ such that $c \in S_u \cap T_v$. Consider a path in G_{far} from u to v. There must be a pair of adjacent vertices (u', v') on the path such that $c \in S_{u'} \cap T_{v'}$. Then, by definition, u' and v' are far (with respect to the original graph G). Since $c \in S_{u'} \cap T_{v'}$, we have $d(v', c) \leq r$ and $d(c, u') \leq r$, so by the triangle inequality $d(v', u') \leq 2r$. Thus u' and v' are not far, a contradiction. □

Algorithm Step 3: Defining a DAG-like structure

a) Constructing the "close graph" The purpose of constructing the close graph is that it allows us to either perform Dijkstra's algorithm from some additional vertices and obtain a good estimate (see step b), or "merge" some connected components of the far graph to further prune the set of vertices that could be the min-center (see step c). The *close graph* G_{close} is an unweighted directed

graph with one vertex f_i for each F_i. For all i and j, let (f_i, f_j) be an edge in G_{close} if for some $u \in F_i$ and some $v \in F_j$, $d(u, v) \leq 5r$.

b) Additional Dijkstra We now perform Dijkstra's algorithm from some additional vertices, which are carefully chosen so that either we find a vertex with small min-eccentricity and are done in this step, or we can define a DAG-like structure in the graph (step c). We compute the strongly connected components (SCCs) of G_{close}. For each SCC $Q = (V_Q, E_Q)$, find $E'_Q \subseteq E_Q$ with $|E'_Q| \leq 2|V_Q|$ such that $Q' = (V_Q, E'_Q)$ is strongly connected; it is simple to show that such an E'_Q exists and we include the proof in subsection 6.2.7 for completeness (Lemma 6.2.19). Let $E' = \cup_Q E'_Q$. Note that every edge $e \in E'$ corresponds to a path P_e of length at most $5r$ in the original graph G. For each $e \in E'$, find an ordered set V_e of at most 9 vertices on P_e that divide P_e into subpaths of length at most r; it is simple to show that such a V_e exists and we include the proof in subsection 6.2.7 for completeness (Lemma 6.2.18). We run Dijkstra's algorithm from every vertex in V_e and if we find a vertex v with $\varepsilon(v) \leq 3r$ then we are done.

c) Constructing the DAG of "supercomponents" Let H be the DAG created by contracting every strongly connected component of G_{close} into a single vertex. That is, there is an edge from u to v in H if the strongly connected component v is reachable from the strongly connected component u. Let k be the number of vertices in H; we number the vertices in H from 1 to k according to a topological ordering. For each $j \in [k]$, we merge the set of F_i's represented by vertex j in H into a *supercomponent* W_j. Formally, if we define F_u to be the connected component of G_{far} that contains u, a vertex $u \in W$ is in supercomponent W_j if f_u is in the strongly connected component of H represented by vertex j.

d) Fitting the remaining vertices into the DAG structure In the previous step, we defined a DAG-like structure on the vertices in W. Now we place the rest of the vertices into this structure. We partition the rest of the vertices based on whether they could potentially be the min-center. We define the vertex sets C and B next and in the analysis we prove that $c \in C$ (among other properties of C and B). We will use the following notation: for any distance $d > 0$, let $S_v^d = \{u \in S_v : d(u, v) \leq d\}$ and let $T_v^d = \{u \in T_v : d(v, u) \leq d\}$. Remember that for any set U of vertices, we defined $S_U^d = \bigcap_{v \in U} S_v^d$, and $T_U^d = \bigcap_{v \in U} T_v^d$.

- For $i = 1, \ldots, k+1$, let $v \in C_i$ if for all $j < i$, $v \in T^{2r}_{W_j}$ and for all $j \geq i$, $v \in S^{2r}_{W_j}$. Let $C = \cup_{i=1}^{k+1} C_i$.

- For $i = 2, \ldots, k+1$, let $v \in B_i$ if $v \notin C$ and i is the largest integer for which $v \in T^{2r}_{W_{i-1}}$. Let $v \in B_1$ if there is no such i and $v \notin C$. Let $B = \cup_{i=1}^{k+1} B_i$.

Analysis Step 3

Figure 6-3 shows a summary of the structure of the graph which we will describe in the following observations and lemmas.

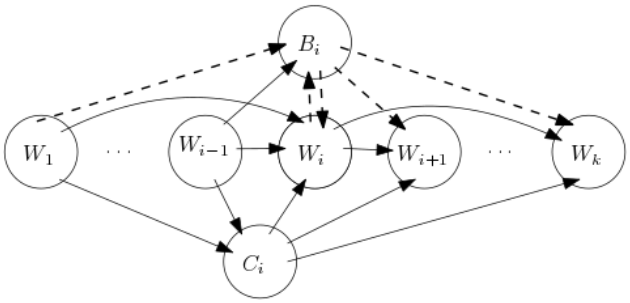

Figure 6-3: The graph structure for the sets W_i, B_i and C_i. Solid lines are paths of length at most $2r$ between any member of the outgoing set to any member of the incoming set. Dashed lines are paths of length at most $2r$ which might not exist between all pairs, which is expressed more accurately in Lemma 6.2.11.

We first observe two important properties of supercomponents:

Observation 6.2.1. *For every pair of vertices $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $d(v_j, v_i) > 5r$.*

This is true because if $d(v_j, v_i) \leq 5r$, then there is an edge from f_j to f_i in G_{close}, so there is an edge from j to i in H. Since $i < j$, this contradicts the topological ordering of H.

Observation 6.2.2. *For every pair of vertices $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $v_i \in S^{2r}_{W_j}$ and $v_j \in T^{2r}_{W_i}$.*

This is true because v_i and v_j are in different F_k's since W_i and W_j are collections of disjoint sets of F_i's. So v_i and v_j are not far i.e. $d_{min}(v_i, v_j) \leq 2r$ and by Observation 6.2.1 we know that

$d(v_j, v_i) > 5r > 2r$, so it must be that $d(v_i, v_j) \leq 2r$. Since this is true for all vertices $v_j \in W_j$, we have $v_i \in S^{2r}_{W_j}$. Similarly, $v_j \in T^{2r}_{W_i}$.

We now prove a refinement of Lemma 6.2.8 where we consider supercomponents instead of far graph components. This further prunes the vertices that could potentially be the min-center.

Lemma 6.2.9. *If $R \leq r$, then for each $i = 1, \ldots, k$, either $c \in S_{W_i}$ or $c \in T_{W_i}$.*

Proof. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $c \in S_u \cap T_v$. By Lemma 6.2.8, u and v must be in different F_i's say F_u and F_v.

Recall that by the definition of a supercomponent, f_u and f_v are in the same strongly connected component of G_{close}. So there is a path P from f_u to f_v in G_{close} such that all of its edges are in E'. By Lemma 6.2.8 Since $c \in S_u \cap T_v$, we have that $c \in S_{F_u} \cap T_{F_v}$. So there are two consecutive nodes f_j and $f_{j'}$ on P (in that order) such that $c \in S_{F_j} \cap T_{F_{j'}}$.

Recall that each edge $e \in E'$ corresponds to a path P_e of length at most $5r$ in the original graph. Let e be the edge $(f_j, f_{j'})$ and consider P_e and V_e, where V_e is the set of vertices that divides P_e into subpaths of length at most r. Since the endpoints of P_e are in F_j and $F_{j'}$ respectively, there exists a pair of vertices u', v' consecutive in V_e (in that order) such that $c \in S_{u'} \cap T_{v'}$. We note that $d(u', v') \leq r$.

Now we claim that $\varepsilon(v') \leq 3r$. This is because $d(v', c) \leq r$ and $d(c, v') \leq d(c, u') + d(u', v') \leq 2r$. Consider an arbitrary vertex $w \in V$. Either $d(c, w) \leq R$ or $d(w, c) \leq R$. If $d(c, w) \leq R$ then $d(v', w) \leq d(v', c) + d(c, w) \leq 2r$. If $d(w, c) \leq R$, then $d(w, v') \leq d(w, c) + d(c, v') \leq 3r$. In this case, the algorithm would have stopped after step 3b.

□

We now prove that $c \in C$, which further prunes the vertices that could potentially be the min-center.

Lemma 6.2.10. *If $R \leq r$, then $c \in C$.*

Proof. By Lemma 6.2.9, either $c \in S_{W_i}$ or $c \in T_{W_i}$. Since c is the min-center and $R \leq r$, if $c \in S_{W_i}$ then $c \in S^{2r}_{W_i}$, and similarly if $c \in T_{W_i}$ then $c \in T^{2r}_{W_i}$. We claim that for each $i < j$, $S^{2r}_{W_i} \cap T^{2r}_{W_j} = \emptyset$, which completes the proof. Suppose otherwise and let i and j be such that $i < j$

and there is a vertex $v \in S^{2r}_{W_i} \cap T^{2r}_{W_j}$. Then for every vertex $v_i \in W_i$ and $v_j \in W_j$, $d(v_j, v) \leq 2r$ and $d(v, v_i) \leq 2r$, so $d(v_j, v_i) \leq 4r$. This contradicts Observation 6.2.1. □

Now we prove that the vertices in B fit into the DAG structure in a similar but weaker sense than the vertices in C:

Lemma 6.2.11. *Consider a node $v \in B_i$. Then for all $z \geq i$ except for at most two values, we have $v \in S^{2r}_{W_z}$. And for all $z \leq i$ except for at most two values, we have $v \in T^{2r}_{W_z}$.*

Proof. We first observe that there is at most one j such that v is far from some vertex in W_j. This is because if v were far from two vertices u, w in different supercomponents, then G_{far} would contain the edges (u, v) and (w, v) making u and w in the same connected component of G_{far}, and thus in the same supercomponent. We fix j and consider two cases:

Case 1: Suppose by way of contradiction that for some node $w \in W_z$ for some $z < i$, $z \neq j$, we have $v \in S^{2r}_w$. We know that $z < i - 1$, since by definition of B_i, we have $v \in T^{2r}_{W_{i-1}}$. Let $w' \in W_{i-1}$ be an arbitrary node, then $d(w', w) \leq d(w', v) + d(v, w) \leq 2r + 2r < 5r$, a contradiction to Observation 6.2.1.

Case 2: Now suppose that for some node $w \in W_z$ for some $z > i$, $z \neq j$, we have $v \in T^{2r}_w$. We will show that $j = i$ and $z = i + 1$; that is, for all $z' \geq i + 2$, we have that $v \in S^{2r}_{W_{z'}}$. If there is some node $w' \in W_i$ such that $v \in S^{2r}_{w'}$, then $d(w, w') \leq d(w, v) + d(v, w') \leq 2r + 2r < 5r$, a contradiction to Observation 6.2.1. Assume that there is no such w' i.e. $d(v, w') > 2r$ for all $w' \in W_i$. Then for every node $w' \in W_i$, either v and w' are far or $d(w', v) \leq 2r$. If for all $w' \in W_i$, $d(w', v) \leq 2r$, then $v \in T^{2r}_{W_i}$, which cannot happen since by the definition of B_i, i is the biggest integer that $v \in T^{2r}_{W_{i-1}}$. Thus, v is far from some vertex in W_i so we have that $j = i$. If $z > i + 1$, then by definition of B_i there is some vertex $u \in W_{i+1}$ such that $v \in S^{2r}_u$. So $d(w, u) \leq d(w, v) + d(v, u) \leq 2r + 2r < 5r$, a contradiction to Observation 6.2.1. So it must be that $z = i + 1$. So for all $z' \geq i + 2$, we have that $v \in S^{2r}_{W_{z'}}$. □

We have observed stronger properties than Lemma 6.2.11 for vertices $v \in W_i$ (Observation 6.2.2) and $v \in C_i$ (by definition), so we have the following corollary.

Corollary 6.2.1. *Lemma 6.2.11 is true for all $v \in B_i \cup C_i \cup W_i$. Moreover, for such v's, we have $v \in T^{2r}_{W_{i-1}}$.*

Algorithm Step 4: Partial search

From each of the potential min-centers, we will run Dijkstra's algorithm on a small subgraph of G. For each $i = 1, \ldots, k+1$, let G_i be the subgraph of G induced by $W_{i-6} \cup \ldots \cup W_{i+3} \cup B_{i-6} \cup \ldots \cup B_{i+3} \cup C_{i-5} \cup \ldots \cup C_{i+3}$. Define $\bar{C}_i$ to be the set of nodes $v \in C_i$ such that v is within min-distance r from all vertices in W (we know this set of nodes because we have already run Dijkstra's algorithm from and to every vertex in W). For every vertex v in $\bar{C}_i$, run Dijkstra's algorithm from v with respect to the graph G_i. If v is within min-distance r from all nodes in $U_i = C_i \cup B_{i-2} \cup B_{i-1} \cup B_i$, we will show that $R \leq 3r$. If there is no such v, we will show that $r < R$.

Analysis Step 4

The following two claims prove that our algorithm either certifies that $R > r$ or $R \leq 3r$.

Claim 8. *For some i, if $c \in \bar{C}_i$ and $R \leq r$, then for all $u \in U_i$, the min-distance between c and u with respect to G_i is at most r.*

Claim 9. *If a vertex $v \in \bar{C}_i$ is within min-distance r from all vertices in U_i with respect to the graph G_i, then $R \leq \varepsilon(v) \leq 3r$.*

Proof of Claim 8. We will prove something slightly stronger: for all i any shortest path in G between two nodes $u, u' \in U_i$ that has length at most r is completely contained in G_i.

By way of contradiction, suppose that the shortest path P from u to u' is not completely in G_i. Define V_r and V_l to be sets of nodes on the right and left of G_i respectively, i.e. $V_r = W_{i+4} \cup \ldots \cup W_k \cup B_{i+4} \cup \ldots \cup B_{k+1} \cup C_{i+4} \cup \ldots \cup C_{k+1}$ and $V_l = W_1 \cup \ldots \cup W_{i-7} \cup B_1 \cup \ldots \cup B_{i-7} \cup C_1 \cup \ldots \cup C_{i-6}$.

First suppose that P contains some node $v_r \in V_r$. There is some $j > i+3$ such that $v_r \in B_j \cup C_j \cup W_j$. So by Corollary 6.2.1, $v_r \in T^{2r}_{W_{j-1}}$. Furthermore, Corollary 6.2.1 implies that there is some $j' \in \{i, i+1, i+2\}$ such that $u' \in S^{2r}_{W_{j'}}$. Pick $w_{j'} \in W_{j'}$ and $w_{j-1} \in W_{j-1}$. We have that $d(w_{j-1}, w_{j'}) \leq d(w_{j-1}, v_r) + d(v_r, u') + d(u', w_{j'}) \leq 2r + r + 2r = 5r$. Since $j - 1 > j'$, this contradicts Observation 6.2.1. This case is shown in Figure 6-4.

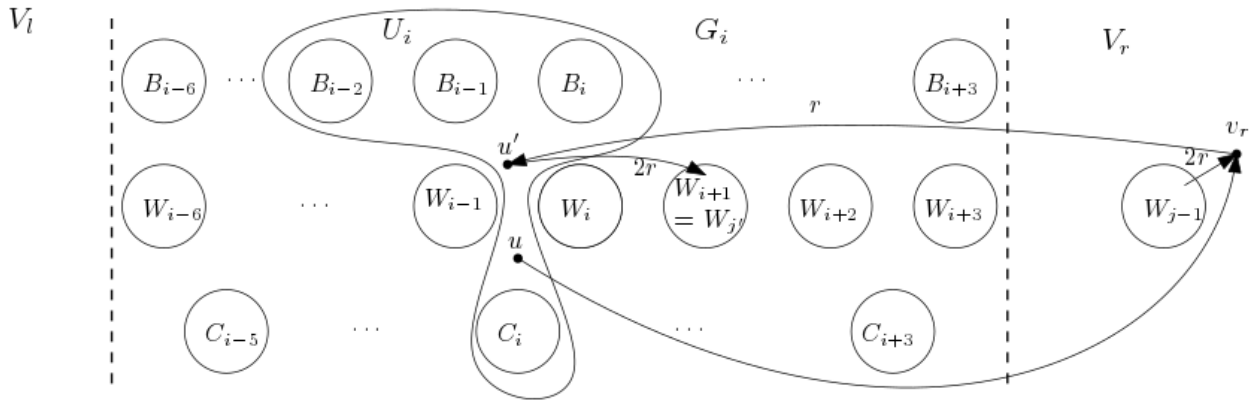

Figure 6-4: First case in Claim 8 where the path P from u to u' passes through some vertex $v_r \in V_r$. In this figure $j' = i + 1$. The upper bound on the weight of each part of the path from W_{j-1} to $W_{j'}$ is written on the edges.

Now suppose that P contains some node v_l in V_l. The argument in this case is symmetric to the previous case. Since $u \in U_i$, there is some $j \in \{i-2, i-1, i\}$ such that $u \in B_j \cup C_j$, and hence by Corollary 6.2.1, $u \in T^{2r}_{W_{j-1}}$. Furthermore, Corollary 6.2.1 implies that there is at least one value $j' \in \{i-5, i-4, i-3\}$ such that $v_l \in S^{2r}_{W_{j'}}$. Pick $w_{j'} \in W_{j'}$ and $w_{j-1} \in W_{j-1}$. We have that $d(w_{j-1}, w_{j'}) \leq d(w_{j-1}, u) + d(u, v_l) + d(v_l, w_{j'}) \leq 2r + r + 2r = 5r$. Since $j' < j - 1$, this contradicts Observation 6.2.1. □

Proof of Claim 9. We show that for any node $u \in V$ we have $d_{min}(u, v) \leq 3r$. We have 3 cases:

Case 1: $u \in W$: From the definition of $\bar{C}_i$, we know that v has min-distance at most r to all vertices in W.

Case 2: $u \in C_j$ for some $j = 1, \ldots, k+1$. If $j = i$, then $u \in U_i$ so we know that $d_{min}(u, v) \leq r$. If $j > i$, then pick some vertex $w_i \in W_i$. By the definition of C_i and C_j we know that $d(v, u) \leq d(v, w_i) + d(w_i, u) \leq r + 2r = 3r$. Similarly, if $j \leq i - 1$, pick some vertex $w_{i-1} \in W_{i-1}$. Then $d(u, v) \leq d(u, w_{i-1}) + d(w_{i-1}, v) \leq 2r + r = 3r$.

Case 3: $u \in B_j$ for some $j = 1, \ldots, k+1$. If $j \in \{i-2, i-1, i\}$, then since $v \in \bar{C}_i$ we know that $d_{min}(u, v) \leq r$. So first suppose that $j \leq i - 3$. Then by Lemma 6.2.11, there is at least one $j' \in \{j+1, \ldots, i-1\}$, such that $u \in S^{2r}_{W_{j'}}$. Pick some node $w_{j'} \in W_{j'}$. So by the definition of C_i we have that $d(u, v) \leq d(u, w_{j'}) + d(w_{j'}, v) \leq 2r + r = 3r$. Now suppose that $j \geq i + 1$. Then by definition of B_j we know that $u \in T^{2r}_{W_{j-1}}$. Pick some vertex $w_{j-1} \in W_{j-1}$. Since $j - 1 \geq i$ and by

the definition of C_i, we have that $d(v,u) \leq d(v,w_{j-1}) + d(w_{j-1},u) \leq r + 2r = 3r$.

□

Runtime analysis

We analyze the running time of each step.

Step 1, preliminaries: $\tilde{O}(m\sqrt{n})$. This is because each Dijkstra in Step 1 takes $\tilde{O}(m)$ time and $|W| = \sqrt{n}$.

Step 2, constructing the "far graph": $\tilde{O}(n\sqrt{n})$. Each edge in the far graph has at least one endpoint in W, and so the construction of the far graph takes $O(n\sqrt{n})$ time. Note that the existence of each edge in the far graph was determined in Step 1.

Step 3, defining a DAG-like structure:

a, constructing the "close graph": $\tilde{O}(n\sqrt{n})$. This is because the connected components of the far graph can be determined in $O(n\sqrt{n})$ time since it has that many edges. The number of components containing a node in W are not more than $|W|$, and so the close graph which is on at most $|W|$ nodes can be constructed in time $O(|W|^2) = O(n)$.

b, additional Dijkstra: $\tilde{O}(m\sqrt{n})$. This is because by Lemma 6.2.19, the number of vertices we run Dijkstra from in SCC Q of the close graph is at most $9|E_Q| \leq 18|V_Q|$. Since the number of vertices in close graph is at most $|W|$, we run Dijkstra from at most $18|W| = \tilde{O}(\sqrt{n})$ vertices. Also running the algorithm of Lemma 6.2.19 takes $O(|E_Q|) = O(|V_Q|^2)$ for each SCC Q, which takes $O(|W|^2) = \tilde{O}(n)$ time in total.

c, constructing the DAG of "supercomponents": $\tilde{O}(n)$. This is because H has at most $|W|$ vertices, so obtaining the DAG ordering of H takes at most $|W|^2 = \tilde{O}(n)$ time.

d, fitting the remaining vertices into the DAG structure: $\tilde{O}(n\sqrt{n})$. For each vertex in V, it takes $O(|W|)$ time to see which set it belongs to, since it only depends on its distances to and from the vertices in W.

Step 4, partial search: $\tilde{O}(m\sqrt{n})$. The Dijkstras ran in G_i take $\tilde{O}(m_i|C_i|)$ time, where m_i is the number of edges with at least one endpoint in G_i. By Lemma 6.2.2, property 3, with high probability $|C_i| = O(\sqrt{n})$. We know that $\bar{C}_i \subseteq C_i$ and so the running time of this step is $O(\sqrt{n}\sum_{i=1}^{k+1} m_i)$. Now since each node is in at most 10 G_is, we have that each edge is also in at most 20 G_is, and

hence $\sum_{i=1}^{k+1} m_i \leq 20m$.

So overall the algorithm runs in $\tilde{O}(m\sqrt{n})$ time.

□

6.2.6 Min-Eccentricities Algorithm

The min-eccentricities algorithm is similar to the min-radius algorithm. Below we will describe the modifications.

Theorem 6.2.8. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta)$ time randomized algorithm, that given a directed weighted graph $G = (V, E)$ with weights positive and polynomial in n, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (5+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

Proof. We fix a value ρ and our algorithm certifies for each $s \in V$ that either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 5\rho$ with high probability. Starting from $\rho = 1$, we will run the algorithm and increase ρ for each run. We will call the vertices for which we have certified $\varepsilon(s) \leq 5\rho$ for earlier values of ρ as *marked*. Let $\delta' = \delta/5$. Starting from $\rho = 1$, we run the algorithm. If the output of the algorithm is that $\varepsilon(s) \leq 5\rho$ and s was unmarked, then we will mark s and set $\varepsilon'(s) = 5\rho$. Then, we run the algorithm with the new value $\rho_{new} = (1+\delta')\rho$. Since $\varepsilon(s) \leq poly(n)$ for all $s \in V$, this contributes a multiplicative factor of $\log_{1+\delta'} n = \tilde{O}(1/\delta)$ to the total runtime. Suppose that for some value of ρ and for some vertex s we have $\varepsilon(s) \leq 5\rho$ and s was unmarked. From the previous run of the algorithm, we know that $\varepsilon(s) > \rho/(1+\delta')$. Then for $\varepsilon'(s) = 5\rho$, we have $\varepsilon'(s) \geq \varepsilon(s)$ and $\varepsilon'(s) \leq 5(1+\delta')\varepsilon(s) = (5+\delta)\varepsilon(s)$, which means that $\varepsilon'(s)$ is a $(5+\delta)$-approximation of $\varepsilon(s)$. After running the whole algorithm for this value of ρ we will also mark all such vertices s. Now we present the algorithm.

Throughout the algorithm ρ behaves analogously to r in the min-radius algorithm. Whenever we say that a certain part of the algorithm is the same we mean that it is same after replacing r by ρ. Note that any vertex s with $\varepsilon(s) = \rho$ satisfies the property that its min-distance to any vertex is at most ρ. This is analogous to the center vertex c in the Min-Radius algorithm using $r = \rho$.

Algorithm Step 1: Preliminaries

First we remove all the edges with weight more than ρ, because if for a vertex s with $\varepsilon(s) \leq \rho$, this removal does not change the min-eccentricity of s. Then we sample a set W of $\sqrt{n}$ vertices according to Lemma 6.2.2. For every vertex $v \in W$, we run Dijkstra's algorithm from and to v to obtain the min-distance between v and all other vertices. This means we know $\varepsilon(v)$ for all $v \in W$ and in particular we know if $\varepsilon(v) > \rho$ or $\varepsilon(v) \leq 3\rho$. We use the vertices in W with min-eccentricity less than 3ρ to detect vertices with min-eccentricity less than 5ρ in the graph.

Algorithm Step 2: Constructing the "far graph"

The far graph and the F_i's are defined the same way as in the min-radius algorithm.

Analysis Step 2

The purpose of constructing G_{far} is to prune the set of vertices that could potentially have low min-eccentricity. Next we state a modified Lemma 6.2.8.

Lemma 6.2.12 (Modification of Lemma 6.2.8)**.** *If for a vertex $s \in V \setminus W$, $\varepsilon(s) \leq \rho$, then for any F_i, either $s \in S_{F_i}$ or $s \in T_{F_i}$.*

Proof. For $s \in F_i$ note that $F_i \subseteq W$ and hence we know $\varepsilon(s)$ and have certified either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 3\rho$. For the other vertices the proof is analogous to that of Lemma 6.2.8 □

Algorithm Step 3: Defining a DAG-like structure

a) Constructing the "close graph" The purpose of constructing the close graph is that it allows us to perform Dijkstra's algorithm from some additional vertices and certify some vertices as having min-eccentricities $\leq 5\rho$. Then we "merge" some connected components of the far graph to further prune the set of vertices that could be having small min-eccentricities (see step c). G_{close} is defined as in the min-radius algorithm.

b) Additional Dijkstra This step of the algorithm diverges from the min-radius algorithm at the end, and hence we state it in full detail. Similar to the min-radius algorithm, we perform Dijkstra's algorithm from some additional vertices, which are chosen so that we detect more vertices with low min-eccentricity and at the end define a DAG-like structure (step c). Recall that we compute the strongly connected components (SCCs) of G_{close}. For each SCC $Q = (V_Q, E_Q)$, find $E'_Q \subseteq E_Q$

with $|E'_Q| \leq 2|V_Q|$ such that $Q' = (V_Q, E'_Q)$ is strongly connected (where the existence of E'_Q is shown in Lemma 6.2.19). Let $E' = \cup_Q E'_Q$. Recall that every edge $e \in E'$ corresponds to a path P_e of length at most 5ρ in the original graph G. For each $e \in E'$, find an ordered set V_e of at most 9 vertices on P_e that divide P_e into sections of length at most ρ (see Lemma 6.2.18). For each $e \in E'$, we run Dijkstra's algorithm from and to every vertex in V_e. This means we know $\varepsilon(v)$ for all $v \in V_e$; and in particular we know whether $\varepsilon(v) > \rho$ or $\varepsilon(v) \leq 3\rho$. Now here is the new part of the algorithm in this step: For every consecutive pair of vertices (a, b) in V_e over all e with $\varepsilon(a), \varepsilon(b) \leq 3\rho$ we certify for all $s \in S_b^\rho \cap T_a^\rho$ that $\varepsilon(s) \leq 5\rho$.

c) Constructing the DAG of "supercomponents" The graphs H, W_i's and the "supercomponents" are defined as in the min-radius algorithm.

d) Fitting the remaining vertices into the DAG structure In the previous step, we defined a DAG-like structure on the vertices of W. Now we place the rest of the vertices into this structure. We partition the rest of the vertices based on whether they haven't been certified to have eccentricity $\leq 5\rho$ *and* could potentially have small eccentricity. Vertex sets C and B are defined as in the min-radius algorithm. In the analysis we prove that all vertices which haven't been certified to have eccentricity $\leq 5\rho$ *and* could potentially have small eccentricity must be in C, among other properties of C and B.

Analysis Step 3

First note that one major difference of this algorithm and the min-radius algorithm is in part b; in the min-radius algorithm we stop whenever we find a good approximate center among the vertices in V_es, but here we can only upper bound the eccentricity of some vertices by 5ρ if we find vertices with eccentricity $\leq 3\rho$ among V_es.

We first show that if for some vertex s and for some consecutive pair of vertices (a, b) in V_e such that $\varepsilon(a), \varepsilon(b) \leq 3\rho$ and $s \in S_b^\rho \cap T_a^\rho$, then $\varepsilon(s) \leq 5\rho$. This is derived by Lemma 6.2.13 which we state bellow, by the following substitution: let $c = s$, $\gamma_1 = \rho, \gamma_2 = 2\rho$ and $\gamma_3 = 3\rho$.

Lemma 6.2.13. *Consider vertices b, c such that $d(b, c) \leq \gamma_1$, $d(c, b) \leq \gamma_2$ and $\varepsilon(b) \leq \gamma_3$ then $\varepsilon(c) \leq \gamma_3 + \max(\gamma_1, \gamma_2)$.*

Proof. Consider a vertex v, as $\varepsilon(b) \leq \gamma_3$ either $d(v,b) \leq \gamma_3$ or $d(b,v) \leq \gamma_3$. If $d(v,b) \leq \gamma_3$ then $d(v,c) \leq d(v,b) + d(b,c) \leq \gamma_3 + \gamma_1$. Otherwise $d(b,v) \leq \gamma_3$ then $d(c,v) \leq d(c,b) + d(b,v) \leq \gamma_3 + \gamma_2$. In both cases $\varepsilon(c) \leq \gamma_3 + \max(\gamma_1, \gamma_2)$. □

Now we observe an important property of supercomponents with an analogous proof to that of Observation 6.2.1.

Observation 6.2.3 (Modification of Observation 6.2.1)**.** *For every pair of vertices in $v_i \in W_i$ and $v_j \in W_j$ with $i < j$, $d(v_j, v_i) > 5\rho$.*

We now prove a modification of Lemma 6.2.9. This further prunes the vertices that could potentially have small eccentricity.

Lemma 6.2.14 (Modification of Lemma 6.2.9)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq 5\rho$ then for each $i = 1, \ldots, k$, either $s \in S_{W_i}$ or $s \in T_{W_i}$.*

Proof. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $s \in S_u \cap T_v$ and $\varepsilon(s) \leq \rho$. By Lemma 6.2.12, u and v must be in different F_i's say F_u and F_v.

Recall that by the definition of a supercomponent, f_u and f_v are in the same strongly connected component of G_{close}. So there is a path P from f_u to f_v in G_{close} such that all of its edges are in E'. By Lemma 6.2.12 since $s \in S_u \cap T_v$, we have that $s \in S_{F_u} \cap T_{F_v}$. So there are two consecutive nodes f_j and $f_{j'}$ on P (in that order) such that $s \in S_{F_j} \cap T_{F_{j'}}$.

Recall that an edge $e \in E'$ corresponds to a path P_e of length at most 5ρ in the original graph. Let e be the edge $(f_j, f_{j'})$ and consider P_e and V_e. Since the endpoints of P_e are in F_j and $F_{j'}$ respectively, there exists a pair of vertices u', v' consecutive in V_e (in that order) such that $s \in S_{u'} \cap T_{v'}$. We note that $d(u', v') \leq \rho$.

Recall that we assumed that $\varepsilon(s) \leq \rho$. Note as well that $d(v', s) \leq \rho$ and $d(s, v') \leq d(s, u') + d(u', v') \leq 2\rho$. Then, using Lemma 6.2.13 with $b = s, \gamma_3 = \rho, c = v', \gamma_1 = 2\rho, \gamma_2 = \rho$, we get that $\varepsilon(v') \leq \rho + \max\{2\rho, \rho\} = 3\rho$. A symmetric argument holds for u', giving $\varepsilon(u'), \varepsilon(v') \leq 3\rho$. In this case, the algorithm would have already marked s in step 3b as it is in the intersection of $S^{\rho}_{u'} \cup T^{\rho}_{v'}$. □

We now prove that for vertices s which have small min-eccentricity and have not been certified as such, $s \in C$. The proof is analogous to that of Lemma 6.2.10.

Lemma 6.2.15 (Modification of Lemma 6.2.10)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq 5\rho$ then $s \in C$.*

Now we prove that the vertices in B fit into the DAG structure in a similar but weaker sense than the vertices in C. The proofs are analogous to those of Lemma 6.2.11 and Corollary 6.2.1.

Lemma 6.2.16 (Modification of Lemma 6.2.11)**.** *Consider a node $v \in B_i$. Then for all $z \leq i$ except for at most two values, we have $v \in T^{2\rho}_{W_z}$. And for all $z \geq i$ except for at most two values, we have $v \in S^{2\rho}_{W_z}$.*

Corollary 6.2.2 (Modification of Corollary 6.2.1)**.** *Lemma 6.2.16 is true for all $v \in B_i \cup C_i \cup W_i$. Moreover for all $v \in B_i \cup C_i \cup W_i$, we have $v \in T^{2\rho}_{W_{i-1}}$.*

Algorithm Step 4: Partial search

From each of the potential vertices with small min-eccentricity in C, we will run Dijkstra's algorithm on a small subgraph of G. G_i and U_i are defined as in the min-radius algorithm. Define $\bar{C}_i$ to be the set of nodes $v \in C_i$ such that v is within min-distance ρ from all vertices in W (we know this set of nodes because we have already run Dijkstra's algorithm from and to every vertex in W). From each node $v \in \bar{C}_i$ run Dijkstra's algorithm from and to v with respect to the graph G_i. If v is within min-distance ρ from all nodes in U_i, we will show that this certifies that $\varepsilon(s) \leq 3\rho$ and otherwise $\varepsilon(s) > \rho$.

Analysis Step 4

The following two claims prove that our algorithm for vertices $s \in C$ either certifies that $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq 3\rho$. The proofs are analogous to those of Claim 8 and Claim 9.

Claim 10 (Modification of Claim 8)**.** *If $s \in C_i$ and $\varepsilon(s) \leq \rho$, then for all $u \in U_i$, the min-distance between c and u with respect to G_i is at most ρ.*

Claim 11 (Modification of Claim 9)**.** *If a vertex s is within min-distance ρ from all vertices in U_i in G_i, then $\varepsilon(s) \leq 3\rho$.*

For all the vertices for which we haven't certified either $\varepsilon(s) \leq 3\rho$ or $\varepsilon(s) \leq 5\rho$ we know that $\varepsilon(s) > \rho$ and can certify that.

□

The runtime is $\tilde{O}(m\sqrt{n})$ with analogous runtime analysis to that of the min-radius algorithm.

$(3+\delta)$-approximation for unweighted graphs

In this part we show that given an unweighted graph, by a slight modification of the min-eccentricity algorithm in Theorem 6.2.8, we are able to improve the approximation factor of the min-eccentricity problem to match that of the min-radius problem, namely we present a $(3+\delta)$-approximation algorithm for every $\delta > 0$.

Theorem 6.2.9. *For any constant δ with $1 > \delta > 0$, there is an $\tilde{O}(m\sqrt{n}/\delta^2)$ time randomized algorithm, that given a directed unweighted graph $G = (V, E)$, can output an estimate $\varepsilon'(s)$ for every vertex $s \in V$ such that $\varepsilon(s) \leq \varepsilon'(s) \leq (3+\delta)\varepsilon(s)$ with high probability, where $\varepsilon(s)$ is the min-eccentricity of the vertex s in G.*

Proof. There are only two parts of the algorithm in Theorem 6.2.8 that change:

(1) Letting $\delta' = \delta/5$, in each run of the algorithm, for each vertex s, we certify that either $\varepsilon(s) > \rho$ or $\varepsilon(s) \leq (3+\delta')\rho$ (instead of $\varepsilon(s) \leq 5\rho$). The subsequent changes follow naturally: We start from $\rho = 1$ and we run the algorithm and increase ρ by a factor of $(1+\delta')$. We call the vertices for which we have certified $\varepsilon(s) \leq (3+\delta')\rho$ for earlier values of ρ as *marked*, and if for an unmarked vertex s the output of the algorithm is $\varepsilon(s) \leq (3+\delta')\rho$, then we let $\varepsilon'(s) = (3+\delta')\rho$. If for some value of ρ and for some vertex s we have $\varepsilon(s) \leq (3+\delta')\rho$ and s was unmarked, then from the previous run of the algorithm, we know that $\varepsilon(s) > \rho/(1+\delta')$. So for $\varepsilon'(s) = (3+\delta')\rho$, we have $\varepsilon'(s) \geq \varepsilon(s)$ and $\varepsilon'(s) \leq (3+\delta')(1+\delta')\varepsilon(s) = (3+\delta)\varepsilon(s)$.

(2) In step 3, part b of the algorithm (Additional Dijkstra), recall that each edge $e \in E'$ is a path of length at most 5ρ in G. Now instead of dividing each e into at most 9 subpaths of length at most ρ, we divide it into subpaths of length at most $\delta'\rho/2 \geq 1$ using at most $20/\delta' - 1 = O(1/\delta')$ vertices which we call V_e. The rest of this step follows naturally: We run Dijkstra from and to each $v \in V_e$, so we know that whether $\varepsilon(v) > \rho$ or $\varepsilon(v) < (2+\delta'/2)\rho$. For every consecutive pair of

vertices (a, b) in V_e over all e with $\varepsilon(a), \varepsilon(b) \leq (2 + \delta'/2)\rho$ we certify for all $s \in S_b^\rho \cap T_a^\rho$ that $\varepsilon(s) \leq (3 + \delta')\rho$. This is indeed true by Lemma 6.2.13 (in the statement of the lemma, let $c = s$, $\gamma_1 = \delta\rho/2, \gamma_2 = (1 + \delta/2)\rho$ and $\gamma_3 = (2 + \delta'/2)\rho$).

First note that by this change the number of vertices that we do Dijkstra from/to in step 3(b) of the algorithm is now $O(|W|/\delta') = \tilde{O}(\sqrt{n}/\delta') = \tilde{O}(\sqrt{n}/\delta)$ (see runtime analysis of step 3(b) in Theorem 6.2.7). The runtime of the other steps are not changed, so the overall runtime of the algorithm is $\tilde{O}(m\sqrt{n}/\delta^2)$.

The main issue in the min-eccentricity algorithm that didn't allow us to get a $(3 + \delta')$ approximation is that we could have potentially big weighted edges, and that didn't let us divide 5ρ-length paths into smaller parts. The analysis of this part is due to Lemma 6.2.14, which is modified as in Lemma 6.2.17.

□

Lemma 6.2.17 (Modification of Lemma 6.2.14)**.** *If for a vertex $s \in V$, $\varepsilon(s) \leq \rho$ and we haven't yet certified $\varepsilon(s) \leq (3 + \delta')\rho$ then for each $i = 1, \ldots, k$, either $s \in S_{W_i}$ or $s \in T_{W_i}$.*

Proof. The proof is similar to that of Lemma 6.2.14, with a change at the end of the argument because of our finer division of paths. Fix i and suppose by way of contradiction that there are nodes $u, v \in W_i$ such that $s \in S_u \cap T_v$ and $\varepsilon(s) \leq \rho$. Similar to Lemma 6.2.14, we can assume that there are two vertices u', v' that we have done Dijkstra from such that $s \in S_{u'} \cap T_{v'}$ and $d(u', v') \leq \rho\delta'/2$.

Now we claim that $\varepsilon(v') \leq (2 + \delta'/2)\rho$. Note that $d(v', s) \leq \rho$ and $d(s, v') \leq d(s, u') + d(u', v') \leq (1 + \delta'/2)\rho$. Consider an arbitrary vertex $w \in V$. Either $d(s, w) \leq \rho$ or $d(w, s) \leq \rho$. If $d(s, w) \leq \rho$ then $d(v', w) \leq d(v', s) + d(s, w) \leq 2\rho$. If $d(w, s) \leq \rho$, then $d(w, v') \leq d(w, s) + d(s, v') \leq (2 + \delta'/2)\rho$. A symmetric argument holds for u'. In this case, the algorithm would have already marked s in step 3b as it is in the intersection of $S_{u'}^\rho \cap T_{v'}^\rho$. □

6.2.7 Omitted Proofs

Lemma 6.2.18. *Given a weighted graph G and a path P in G from v to u of length at most zr for some integers z and r, one can find in $O(|P|)$ time vertices $v_1, \ldots, v_{z'}$ such that $z' \leq 2z - 1$ and*

they divide P into subpaths of length at most r if there are no edges of weight more than r on the path. Equivalently, $|P_{v_iv_{i+1}}| \leq r$, for $i = 0, \ldots, z$, where $v_0 = v, v_{z'+1} = u$ and $P_{v_iv_{i+1}}$ is the part of the path P between v_i and v_{i+1}.

Proof. Start from $v_0 = v$ and go through the path until the last vertex w such that $d(v, w) \leq r$ but $d(v, w') > r$ where w' is the node right after w on the path. Note that since there are no edges of weight more than r, such w exists. Let $v_1 = w$. Starting from v_1, we can do the same and find all vertices $v_2, \ldots, v_{z'}$. It is remained to prove that $z' < 2z$. By the definition of v_1, we know that $d(v_0, v_2) > r$. Similarly, we can argue that $d(v_i, v_{i+2}) > r$ for all $i = 0, \ldots, z' - 1$. So $d(v_0, v_{2i}) > ir$. Since $|P| \leq zr$, we have $z' \leq 2z - 1$. We went through the vertices of P once, so the running time is linear in terms of the length of the path. □

Lemma 6.2.19. *There is an algorithm that given a strongly connected graph $H = (V, E)$, outputs in $O(|E|)$ time a subset $E' \subseteq E$ of size at most $2(|V| - 1)$ such that $H' = (V, E')$ is strongly connected.*

Proof. For any vertex v do a BFS to and from v and denote by E' the union of edges in the two computed BFS trees. $H' = (V, E')$ is strongly connected as for every ordered pair of vertices (a, b) we can go from a to b by following the path $a \to v \to b$. It is clear that since E' is the union of two trees, $|E'| \leq 2(|V| - 1)$.

□

6.3 Min-distance problems in directed acyclic graphs

This section was written with Jenny Kaufmann, and we focus on min-distance problems in Directed Acyclic Graphs (DAGs).

Graph parameters such as the diameter, radius, and vertex eccentricities are not defined in a useful way in DAGs using the standard measure of distance, since for any two nodes, there is no path between them in one of the two directions. So it is natural to consider the distance between two nodes as the length of the shortest path in the direction in which this path exists, motivating

the definition of the min-distance in DAGs. Recall that The min-distance between two nodes u and v is the minimum of the shortest path distances from u to v and from v to u.

As with the standard distance problems, the Strong Exponential Time Hypothesis [Impagliazzo-Paturi-Zane 2001, Calabro-Impagliazzo-Paturi 2009] leaves little hope for computing min-distance problems faster than computing All Pairs Shortest Paths, which can be solved in $\tilde{O}(mn)$ time. So it is natural to resort to approximation algorithms in $\tilde{O}(mn^{1-\epsilon})$ time for some positive ϵ. Abboud, Vassilevska W., and Wang [SODA 2016] first studied min-distance problems achieving constant factor approximation algorithms on DAGs, and Dalirrooyfard *et al* [ICALP 2019] gave the first constant factor approximation algorithms on general graphs for min-diameter, min-radius and min-eccentricities (see section 6.2). Abboud *et al* obtained a 3-approximation algorithm for min-radius on DAGs which works in $\tilde{O}(m\sqrt{n})$ time, and showed that any $(2-\delta)$-approximation requires $n^{2-o(1)}$ time for any $\delta > 0$, under the Hitting Set Conjecture. We close the gap, obtaining a 2-approximation algorithm which runs in $\tilde{O}(m\sqrt{n})$ time. As the lower bound of Abboud *et al* only works for sparse DAGs, we further show that our algorithm is conditionally tight for dense DAGs using a reduction from Boolean matrix multiplication. Moreover, Abboud *et al* obtained a linear time 2-approximation algorithm for min-diameter along with a lower bound stating that any $(3/2-\delta)$-approximation algorithm for sparse DAGs requires $n^{2-o(1)}$ time under SETH. We close this gap for dense DAGs by obtaining a $3/2$-approximation algorithm which works in $O(n^{2.350})$ time and showing that the approximation factor is unlikely to be improved within $O(n^{\omega-o(1)})$ time under the high dimensional Orthogonal Vectors Conjecture, where ω is the matrix multiplication exponent.

6.3.1 Introduction

Min-distance is a particularly natural notion of distance in directed acyclic graphs (DAGs), where the standard notion of distance is infinite in at least one direction for any given pair of vertices in a DAG. For example, in a topologically ordered DAG where the edges are directed from left to right, the min-diameter is simply the largest distance $d(u,v)$ where u is to the left of v.

More formally, for a vertex $v \in V$, the *min-eccentricity* $\epsilon(v)$ is $\max_{w\in V} d_{\min}(v,w)$, or in other

words, the largest min-distance between v and any other vertex. The *min-diameter* of a graph is $\max_{v \in V} \epsilon(v)$. Note that the min-diameter is the only meaningful notion of diameter for DAGs: all other notions are infinite. The *min-radius* of a graph is $\min_{v \in V} \epsilon(v)$. A *center* is a vertex whose min-eccentricity is equal to the min-radius of the graph.

All-Pairs Shortest Paths (APSP) is the problem of computing the distance between u and v for every pair of vertices $u, v \in V$. In a graph G with m edges, n vertices, and nonnegative edge weights polynomial in n, APSP can easily be computed in $\tilde{O}(mn)$ time[5], by running Dijkstra's algorithm from every vertex[6]. Computing eccentricities, diameter, or radius with any of the notions of distance is no harder than computing APSP.

For the standard notion of distance, under the Strong Exponential Time Hypothesis (SETH) [IP01a, CIP09], there is no *truly subquadratic* time algorithm for diameter (and thus nor for eccentricities) in unweighted graphs: that is, no such algorithm runs in time $O(m^{2-\epsilon})$ for $\epsilon > 0$ [RV13]. This lower bound also holds for the other notions of diameter (and eccentricities) [DWV$^+$19]. For radius, the same lower bound holds but under the Hitting Set Conjecture [AVW16].

Since quadratic time is expensive on large graphs, we resort to approximation algorithms. Many constant factor approximation algorithms were known for all notions of diameter, eccentricities and radius, except for the min-distance notion until recently. For example, for the standard diameter and roundtrip diameter there is a folklore linear time 2-approximation algorithm, and for max-diameter and standard diameter, a conditionally tight $3/2$-approximation algorithm is known in $\tilde{O}(m\sqrt{n})$ time [RV13].

Only recently Dalirrooyfard *et al* [DWV$^+$19] showed constant factor approximation algorithms for min-distance problems in general graphs that run in $O(mn^{1-\epsilon})$ time for some fixed $\epsilon > 0$. More specifically, they obtained a 3-approximation algorithm for min-diameter in $\tilde{O}(m\sqrt{n})$ time, a $(3+\delta)$-approximation algorithm for min-radius in $\tilde{O}(m\sqrt{n}/\delta)$ time, and a $(3+\delta)$-approximation algorithm for min-eccentricities in $\tilde{O}(m\sqrt{n}/\delta^2)$ time, for any $\delta > 0$.

The reason it is hard to obtain approximation algorithms for min-diameter, min-radius, and min-eccentricities is that min-distance does not obey the triangle inequality. Hence the typical

[5]The tilde hides polylogarithmic factors.

[6]Faster algorithms are known by Pettie [Pet02] and Pettie and Ramachandran [PR02] for sparse graphs.

Problem	Upper bound	Lower bound	Reference
min-diameter	2 in $O(m)$	$(\frac{3}{2}-\delta)$ needs $m^{2-o(1)}$	[AVW16]
	$\frac{3}{2}$ in $O(n^{2.350})$ (dense, unweighted)	$(\frac{3}{2}-\delta)$ needs $n^{\omega-o(1)}$ $(*)$	this work
min-radius	3 in $\tilde{O}(m\sqrt{n})$	$(\mathbf{2}-\delta)$ needs $m^{2-o(1)}$	[AVW16]
	$\mathbf{2}$ in $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$	$(\mathbf{2}-\delta)$ needs $n^{\omega-o(1)}$ $(*)$	this work
	k in $\tilde{O}(\min(mn^{1/k}, m^{\frac{2^{k-1}}{2^k-1}}n))$		this work
min-eccentri.	$3+\delta$ in $\tilde{O}(m\sqrt{n}/\delta^2)$		[DWV+19]
	$k+\delta$ in $\tilde{O}(\min(mn^{1/k}/\delta, m^{\frac{2^{k-1}}{2^k-1}}n/\delta))$		this work

Table 6.5: Results on min-distance problems on DAGs. The $(*)$ marks lower bounds that are for dense DAGs. Our $(2-\delta)$ lower bound for min-radius is based on Triangle Detection and our $(\frac{3}{2}-\delta)$ lower bound for min-diameter is based on high dimensional OV. Our k and $(k+\delta)$-approximation algorithms are for any integer $k \geq 2$. Conditionally tight bounds are in bold.

approaches to find algorithms that work for other notions of distance do not work for min-distance, as they crucially rely on the triangle inequality.

On the bright side, since DAGs have more structure, it is easier to find algorithms for them. The best known subquadratic time algorithm for min-diameter in DAGs is a linear time 2-approximation algorithm, and the best subquadratic time algorithm for min-radius is a 3-approximation algorithm in $\tilde{O}(m\sqrt{n})$ time [AVW16]. However, neither of these algorithms were proven to be conditionally tight.

Previously, the only known conditional lower bounds for these problems were due to Abboud, Vassilevska W., and Wang [AVW16]. They showed that under the Orthogonal Vectors Conjecture from fine-grained complexity (and consequently under SETH [Wil05]), there is no $(3/2-\delta)$-approximation algorithm for any $\delta > 0$ for min-diameter which runs in truly subquadratic time on sparse DAGs. Moreover, under the Hitting Set Conjecture, there is no $(2-\delta)$-approximation algorithm for any $\delta > 0$ for min-radius which runs in truly subquadratic time on sparse DAGs.

Our results

We obtain fast algorithms for min-diameter, min-eccentricities and min-radius with improved approximation factors. Our results can be seen in Table 6.5.

Min-Eccentricities and Min-Radius

We obtain the first known subquadratic time $(2+\delta)$-approximation algorithm for min-eccentricities in DAGs for any $\delta > 0$, and the first known subquadratic time 2-approximation algorithm for min-radius in DAGs. These algorithms run in time $\tilde{O}(\min(m\sqrt{n}/\delta, m^{2/3}n)/\delta)$ and $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$ respectively. Note that our algorithms in this section are combinatorial: they do not exploit fast matrix multiplication and are potentially practical. Our results are *conditionally optimal* in both sparse and dense graphs: For sparse graphs, if the Hitting Set Conjecture is true, then our min-radius result is tight and our min-eccentricity result is essentially tight, in the sense that no approximation factor smaller than 2 can be achieved in subquadratic time for either of these problems [AVW16]. For dense graphs, our 2-approximation algorithm works in $\tilde{O}(n^{7/3})$ time, and we show that there is no $(2-\delta)$-approximation algorithm for min-radius (and hence min-eccentricities) in $O(n^{\omega-\epsilon})$ for $\epsilon > 0$, if the best algorithm for Triangle Detection runs in time $\Omega(n^{\omega-o(1)})$. Here $\omega < 2.37286$ [AV21] is the exponent of matrix multiplication.

More generally, we obtain a series of algorithms trading off runtime and accuracy.

Theorem 6.3.1. *For integer $k \geq 2$ and every $\delta > 0$, there is a $(k+\delta)$-approximation algorithm for min-eccentricities in DAGs which runs in $\tilde{O}(\min(mn^{1/k}/\delta, m^{2^{k-1}/(2^k-1)}n/\delta))$ time.*

For every integer $k \geq 2$, there is a k-approximation algorithm for min-radius in DAGs which runs in $\tilde{O}(\min(mn^{1/k}, m^{2^{k-1}/(2^k-1)}n))$ time.

As mentioned earlier, the case $k = 2$ gives a 2-approximation algorithm for min-radius running in time $\tilde{O}(\min(m\sqrt{n}, m^{2/3}n))$. For $m = \tilde{O}(n^{1.5})$, this matches the runtime and improves the approximation factor of the previous best known algorithm for this problem (from [AVW16]). For $m = \omega(n^{1.5+o(1)})$, it improves both the approximation factor and the runtime.

Our min-eccentricity $(2+\delta)$-approximation algorithm borrows a key idea from the 3-approximation algorithm of [AVW16] and combines it with a new binary search technique. The idea is to partition the DAG into intervals and do local APSP searches to find local paths, then combine these local paths with "outer" paths to guarantee a low enough min-distance to any vertex in the graph. In [AVW16], these outer paths were found by using a clever choice of intervals; our algorithm instead applies binary search to find sets which can be used as jumping-off points for the

outer paths, allowing us to shorten the lengths of these paths and also allowing us to approximate all min-eccentricities, not only min-radius. Our $(k+\delta)$-approximation algorithm is achieved by recursively running our approximation algorithm on the intervals instead of running local APSP, which allows us to improve the runtime.

For sparse graphs, Abboud, Vassilevska W., and Wang [AVW16] already showed that a $(2-\delta)$-approximation for min-radius needs $\Omega(m^{2-o(1)})$ time under the Hitting Set Conjecture, so our 2-approximation algorithm is conditionally tight for sparse graphs. We show that the approximation factor of our algorithm is conditionally tight for the dense case as well by reducing Triangle Detection to $(2-\delta)$-approximation of min-radius for any $\delta > 0$. The best running time for Triangle Detection in n-node graphs is conjectured to be $\Omega(n^{\omega-o(1)})$ by many papers (see for example [ABW15, BW17]), where $\omega < 2.37286$ [AV21] is the exponent of fast matrix multiplication. Note that, since $m = O(n^2)$, our algorithm runs in $\tilde{O}(n^{7/3})$ time, which is faster than $O(n^\omega)$ for the current best bound on ω. Since the algorithm of Theorem 6.3.1 is combinatorial, if we restrict to combinatorial algorithms then there is no *truly subcubic* (meaning $O(n^{3-\epsilon})$ for $\epsilon > 0$) time $(2-\delta)$-approximation algorithm for min-radius provided that there is no truly subcubic time combinatorial algorithm for Boolean matrix multiplication (BMM). This is because BMM and Triangle Detection are subcubic equivalent [VW10]. Note that our reduction graph in Theorem 6.3.2 is an unweighted DAG.

Theorem 6.3.2. *If there is a $T(n,m)$-time algorithm for $(2-\delta)$-approximation of min-radius in $O(n)$-node $\tilde{O}(m)$-edge DAGs for some $\delta > 0$, then there is an $\tilde{O}(T(n,m)+m)$-time algorithm for Triangle Detection on graphs with n nodes and m edges.*

Corollary 6.3.1. *Assuming the best algorithm for Triangle Detection runs in time $\Omega(n^{\omega-o(1)})$, there is no algorithm for $(2-\delta)$-approximation of min-radius in n-node dense DAGs that runs in time $O(n^{\omega-\epsilon})$ for any $\delta, \epsilon > 0$.*

Moreover, there is no $O(n^{3-\epsilon})$-time combinatorial algorithm for $(2-\delta)$-approximation of min-radius in n-node dense DAGs with $\epsilon, \delta > 0$ if there is no $O(n^{3-\epsilon'})$-time combinatorial algorithm for BMM with $\epsilon' > 0$.

Improving the running time using Fast Matrix Multiplication In DAGs with small integer edge weights, we further improve the running times for all k in Theorem 6.3.1 by applying a result of Zwick in [Zwi02] on the runtime of APSP in such graphs. We describe our result in more detail in Section 2. In particular, in DAGs with constant integer edge weights, including unweighted DAGs, our result in the case $k = 2$ is as follows:

Theorem 6.3.3. *For every $\delta > 0$, there is an $\tilde{O}(\min(m\sqrt{n}/\delta, m^{0.605}n/\delta))$-time $(2+\delta)$-approximation algorithm for min-eccentricities in DAGs with constant integer edge weights.*

There is an $\tilde{O}(\min(m\sqrt{n}, m^{0.605}n))$-time 2-approximation algorithm for min-radius in DAGs with constant integer edge weights.

Min-Diameter

We obtain a $3/2$-approximation algorithm for min-diameter in unweighted DAGs, where the approximation factor is conditionally optimal in dense graphs. Specifically, our algorithm improves on the standard APSP runtime for any graph with $m = \omega(n^{1+o(1)})$ edges. This is the first known $3/2$-approximation algorithm for min-diameter in dense DAGs that runs faster than the best constant factor approximation algorithm for APSP, which runs in $\tilde{O}(n^\omega)$ time in unweighted directed graphs [Zwi02].

Theorem 6.3.4. *There is an $O(m^{0.414}n^{1.522} + n^{2+o(1)})$-time $3/2$-approximation algorithm for min-diameter in unweighted DAGs.*

This algorithm relies on the sparse matrix multiplication algorithm of Yuster and Zwick [YZ05]. In dense graphs with $m = O(n^2)$, its runtime is $O(n^{2.350})$. In relatively sparse graphs, with $m = O(n^{1.154+o(1)})$, the second term dominates, so the runtime is $O(n^{2+o(1)})$.

Our techniques, which mix known diameter techniques with sparse matrix multiplication, are informally as follows: We first construct a covering set, which will intersect any sufficiently large set. We run BFS from all vertices in the covering set, and check whether any min-distances found were large. If not, then for each vertex u, we will define a set of vertices that are relatively "close" to u on its right; if this set is large it will intersect the covering set, allowing us to find paths from u to some vertices to its right, using a "close" vertex in the covering set as a jumping-off point.

The remaining vertices w, for which this method did not construct a $u \to w$ path, must have the property that any $u \to w$ path must intersect a relatively small subset of the set of vertices "close" to u (note that this set may have been small to begin with, in which case we can skip the previous step). Symmetrically, for each vertex w we can construct the corresponding relatively small subset of vertices "close" to w on its left, and then to bound the min-distance between u and w we check whether these two small subsets share a vertex in common. We use sparse matrix multiplication to detect this set intersection.

The conditional lower bound of [AVW16] says that if the Orthogonal Vectors Conjecture is true then min-diameter cannot be $(3/2 - \delta)$-approximated in truly subquadratic time in sparse graphs. There is no known $3/2$-approximation algorithm for min-diameter on DAGs that works faster than APSP, neither for dense graphs nor for sparse graphs. So the question is: Is $3/2$ the right bound for inapproximability of min-diameter in DAGs? We answer this question in the affirmative for dense DAGs. Theorem 6.3.4 gives the first $3/2$-approximation algorithm that works faster than APSP, and it is optimal conditioned on *high dimensional OV* using the same reduction as [AVW16]. High dimensional OV can be used for obtaining lower bounds for dense graphs. In high dimensional OV, the dimension of the vectors can be as big as $O(n)$, and using a simple reduction to Boolean matrix multiplication, the best known algorithm for it is in time $O(n^\omega)$.

High dimensional OV gives a conditional lower bound of $\Omega(n^{\omega - o(1)})$ time for $(3/2 - \delta)$-approximation of min-diameter for any $\delta > 0$. Our algorithm gives an upper bound of $O(n^{2.350})$ for $m = \Theta(n^2)$, which is faster than $O(n^\omega)$ for the current best bound on ω. We note while we provide tight results for dense DAGs, the gap between the lower bound and upper bound for computing min-diameter on sparse DAGs is still open.

Preliminaries

All graphs in this section are directed graphs. Given a graph G, n denotes the number of vertices and m denotes the number of edges. We will assume $m \geq n - 1$ since otherwise all min-eccentricities are infinite, a case that is easily checked. All edge weights are assumed to be nonnegative and polynomial in n; if w_{max} is the maximum edge weight and w_{min} is the minimum edge weight, we let $M = \max\{w_{max}, 1/w_{min}\}$. We write $G[S]$ to denote the subgraph of G

induced by vertex set S. For a vertex v, we write $N_D^{\text{in}}(v)$ (respectively, $N_D^{\text{out}}(v)$) to denote the set of vertices u such that $d(u, v) \leq D$ (respectively, $d(v, u) \leq D$).

For $v \in V$ and $W \subseteq V$, we define $d_{\text{min}}(W, v) = d_{\text{min}}(v, W)$ as $\min_{w \in W} d_{\text{min}}(v, w)$, and we define the min-eccentricity of W as $\epsilon(W) = \max_{v \in V} d_{\text{min}}(W, v)$.

Given two sets $U, W \subseteq V$, if every $u \in U$ appears prior to (respectively, after) every $w \in W$ in a topological ordering of the vertices of G, we say that U is the left (respectively, right) of W with respect to the topological ordering. When U or W consists of a single vertex $\{x\}$, we omit the brackets. If $W \subseteq U \subseteq V$, we denote the subset of vertices in U that lie to the left (right) of W by $L_U(W)$ (respectively, $R_U(W)$). If $U = V$, we omit the subscript. A vertex set W is called *topologically consecutive* with respect to a topological ordering if its vertices are consecutive; i.e., if $W = V \setminus (L(W) \cup R(W))$. In general, the relevant topological ordering will be clear, and we will omit reference to it.

Let $\omega(1, r, 1)$ be the exponent of the runtime of multiplying $n \times n^r$ by $n^r \times n$ matrices. Let $\omega = \omega(1, 1, 1)$ be the square matrix multiplication exponent. [AV21] showed that $\omega < 2.37286$.

For specifying lower bounds, we use the following problems with their corresponding running time conjectures.

Orthogonal Vectors (OV) Given two lists A, B of n d-dimensional Boolean vectors, determine whether there are vectors $a \in A$ and $b \in B$ such that a and b are *orthogonal*; i.e. there is no $i \in [d]$ such that the ith bits of both a and b are 1. When $d = \Omega(\log n)$, the *OV Conjecture* [Wil05] says that there is no algorithm that can solve the OV problem in time $O(n^{2-\epsilon})$ for any fixed $\epsilon > 0$. The OV Conjecture is implied by the Strong Exponential Time Hypothesis (SETH) [Wil05].

High Dimensional Orthogonal Vectors In high dimensional OV, the dimension d can be as high as $O(n)$. There is a simple reduction from high dimensional OV to matrix multiplication: Given two lists $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}$ of d-dimensional Boolean vectors, let M and N be two $n \times d$ and $d \times n$ Boolean matrices, where $M[i, j] = 1$ if a_i is 1 in bit j, and $N[j, k] = 1$ if b_k is 1 in bit j, for $j = 1, \ldots, d$ and $i, k = 1, \ldots, n$. If MN has a zero entry, the vector pair corresponding to that entry are orthogonal. This gives a $O(n^\omega)$ algorithm for high dimensional OV, and there are no faster algorithms known for it up to polylogarithmic factors. Moreover, OV is equivalent to the

problem of distinguishing diameter 2 vs 3 [RV13], and so high dimensional OV is equivalent to distinguishing diameter 2 vs 3 in dense graphs. A well-known open problem is whether diameter 2 vs 3 can be solved faster than matrix multiplication (see for example [ACIM99]). Hence, it is conjectured that high dimensional OV cannot be solved in $O(n^{\omega-\epsilon})$ time for any $\epsilon > 0$.

Hitting Set (HS) Given two lists $A, B \in \{0,1\}^d$, determine whether there is a vector $a \in A$ that is not orthogonal to any vector $b \in B$. When $d = \Omega(\log n)$, the *Hitting Set Conjecture* [AVW16] says that there is no algorithm that can solve the Hitting Set problem in time $O(n^{2-\delta})$ for any fixed $\delta > 0$.

Boolean Matrix Multiplication (BMM) We abbreviate multiplying two Boolean $n \times n$ matrices over the (AND, OR)-semiring by BMM. It is conjectured that there is no combinatorial algorithm solving BMM in $O(n^{3-\epsilon})$ time for any fixed $\epsilon > 0$, and the best algebraic algorithm for it is in $O(n^{\omega+o(1)})$ time for $\omega < 2.37286$ [AV21].

Triangle Detection [VW10] Given a tripartite graph $G(A, B, C, E)$ where A, B and C are the three parts of the vertex set and E is the edge set, determine if there are $a \in A$, $b \in B$, and $c \in C$ such that abc is a triangle. Vassilevska W. and Williams [VW10] showed that considering only combinatorial algorithms, Triangle Detection and BMM are subcubic equivalent, meaning that a truly subcubic combinatorial algorithm in one results in a truly subcubic combinatorial algorithm in the other. Moreover, the best (algebraic) algorithm for Triangle Detection is through BMM. Thus the best running time for Triangle Detection is $O(n^{\omega})$, and it is conjectured (see for example [ABW15, BW17]) that there is no algorithm faster than $O(n^{\omega})$ for detecting a triangle.

6.3.2 Min-Eccentricities and Min-Radius

We present two different versions of our min-eccentricity and min-radius approximation algorithms, one which works in general weighted DAGs and is combinatorial and one with a lower runtime upper bound which only works in DAGs with small integer edge weights. The algorithms are identical except in how they compute APSP; the former computes APSP in the standard combinatorial way, while the latter uses Zwick's fast APSP algorithm for graphs with small integer edge weights. Here, $\mu(t)$ is the value satisfying $\omega(1, \mu(t), 1) = 1 + 2\mu(t) - t$.

Theorem 6.3.5 ([Zwi02])**.** *APSP can be computed in $O(n^{2+\mu(t)})$ time in directed graphs with integer edge weights bounded by n^t, where $t < 3 - \omega$.*

Both versions of our algorithms use a common technique to compute min-distances to and from a vertex set. Given a graph G and a vertex set $W \subseteq V$, we construct a graph G' by adding a vertex y and adding weight-0 edges (w, y) for all $w \in W$. We then run Dijkstra into y in G'. We refer to this procedure as *running Dijkstra into W*. The symmetric procedure, in which the weight-0 edges point out of an added vertex y' and we run Dijkstra out of y', will be referred to as *running Dijkstra out of W*. Then for $x \in V$, $d_{\min}(x, W) = \min(d(x, y), d(y', x))$, a value which we can now compute. We added $|W|$ edges and ran Dijkstra in G', so in total the procedure takes time $O(|W| + m \log n) = O(m \log n)$.

Our min-eccentricity and min-radius approximation algorithms will be based on the following proposition. Let $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$.

Proposition 17. *For any $k \geq 2$, there is an $O(\min(mn^{1/k} \log^2 n, m^{2^{k-1}/(2^k-1)} n \log^2 n))$-time algorithm which takes as input a DAG G and a parameter r, and certifies for each vertex v that $\epsilon(v) > r$ or that $\epsilon(v) \leq kr$.*

In DAGs with integer edge weights bounded by n^t, where $t < 3 - \omega$, there is a version of this algorithm which runs in $O(\min(mn^{1/k} \log^2 n, m^{c_k(\mu(t))} n \log^2 n))$-time.

In [GU17], Le Gall and Urrutia showed that $\mu = \mu(0) < 0.529$. Thus in DAGs with constant integer edge weights (so that $t = 0$), the runtime of the algorithm of Proposition 17 is $\tilde{O}(\min(mn^{1/k}, m^{c_k(0.529)} n))$ time. When $k = 2$, $c_k(0.529) < 0.605$, leading to the special case stated in Theorem 6.3.3.

The algorithms of Proposition 17 will be described and proven correct in subsection 2.1, and their runtimes will be analyzed in Lemma 6.3.1 in subsection 6.3.2. Then by binary searching over $r \in [0, Mn]$, these algorithms can be used to obtain the min-eccentricity approximation algorithms of Theorems 6.3.6 and 6.3.7 and the min-radius approximation algorithms of Theorems 6.3.8 and 6.3.9.

Theorem 6.3.6. *Let $k \geq 2$ be an integer. For any $\delta > 0$, there is an $\tilde{O}(\min(mn^{1/k}/\delta,$*

$m^{2^{k-1}/(2^k-1)}n/\delta))$*-time algorithm which, given a DAG* G*, outputs for every vertex* $v \in V$ *an estimate* $\epsilon'(v)$ *such that* $\epsilon(v) \leq \epsilon'(v) < (k+\delta)\epsilon(v)$.

Theorem 6.3.7. *Let* $k \geq 2$ *be an integer. For any* $\delta > 0$*, there is an* $\tilde{O}(\min(mn^{1/k}/\delta, m^{c_k(\mu(t))}n/\delta))$ *time algorithm which, given a DAG* G *with integer edge weights bounded by* n^t *for* $t < 3-\omega$*, outputs for every vertex* $v \in V$ *an estimate* $\epsilon'(v)$ *such that* $\epsilon(v) \leq \epsilon'(v) < (k+\delta)\epsilon(v)$.

Proof. First we have all the vertices as "unmarked." We do binary search in $[0, Mn]$ by starting with $r = 1$ in Proposition 17 and incrementing $r' = (1+\delta/k)r$ at each step. At each step, we run the algorithm given in Proposition 17, and for each unmarked v that is reported as having $\epsilon(v) \leq kr$, we set $\epsilon'(v) = kr$ and mark v. At the end we set $\epsilon'(v) = \infty$ for any remaining unmarked vertices.

Suppose a vertex v was marked at the step corresponding to r. Then $r/(1+\delta/k) < \epsilon(v) \leq kr$, so $\epsilon(v) \leq \epsilon'(v) = kr < (k+\delta)\epsilon(v)$. The binary search adds an $O(\log_{1+\delta/k} Mn) = O((\log Mn)/\delta)$ factor to the runtime. Since $\log Mn$ is polylogarithmic in n, this gives the time bounds stated. □

Theorem 6.3.8. *Let* $k \geq 2$ *be an integer. There is an* $\tilde{O}(\min(mn^{1/k}, m^{2^{k-1}/(2^k-1)}n))$*-time algorithm which, given a DAG* G*, outputs an approximation* R' *such that if* R *is the min-radius of* G*,* $R \leq R' < kR$.

Theorem 6.3.9. *Let* $k \geq 2$ *be an integer. There is an* $\tilde{O}(\min(mn^{1/k}, m^{c_k(\mu(t))}n))$*-time algorithm which, given a DAG* G *with integer edge weights bounded by* n^t *for* $t < 3-\omega$*, outputs an approximation* R' *such that if* R *is the min-radius of* G*,* $R \leq R' < kR$.

Proof. We do binary search in $[0, Mn]$, running the algorithm given by Proposition 17 at each step as follows: We keep two numbers A_i and B_i at step i which are the lower bound and upper bound to the min-radius R. At step 1 we have $A_1 = 0$ and $B_1 = Mn$. At step i, we have A_i, B_i such that $A_i < R \leq B_i$. Let $C_i = B_i - kA_i$. If C_i is smaller than the minimum positive edge weight, then any path of length at most B_i must have length at most kA_i, so in this case we terminate the binary search and let $R' = kA_i$. We now have $R \leq R' < kR$ as desired.

If C_i is not smaller than the minimum positive edge weight, let $r = A_i + \frac{C_i}{k+1}$, and run the algorithm given by Proposition 17. If the algorithm reports that there is a vertex v with $\epsilon(v) < kr$,

then let $A_{i+1} = A_i$ and $B_{i+1} = kr = kA_i + \frac{k}{k+1}C_i$, as we have the min-radius is between A_{i+1} and B_{i+1}. Note that in this case $C_{i+1} = B_{i+1} - kA_{i+1} = \frac{k}{k+1}C_i$. Otherwise, if the algorithm reports that every vertex has $\epsilon(v) \geq r$, then the min-radius is at least $A_{i+1} := r = A_i + \frac{C_i}{k+1}$ and is less than $B_{i+1} := B_i$. In this case $C_{i+1} = B_i - k(A_i + \frac{C_i}{k+1}) = \frac{k}{k+1}C_i$. Thus, at each step, the size of C_i shrinks by a factor of $\frac{k}{k+1}$. Hence, for constant k, the algorithm will in $O(\log Mn)$ steps find bounds A_i, B_i such that C_i is smaller than the minimum positive edge weight.

□

Algorithm Description and Correctness

We now describe and prove the correctness of the algorithm of Proposition 17 by induction on k. For convenience, we use $k = 1$ as a base case; in this case we simply run an APSP computation. Our algorithm for $k > 1$ is as follows.

First, topologically sort the vertices and partition them into p consecutive sets $W_1, \ldots W_p$ of size $|W_i| = n/p$. The runtime-minimizing value of p will be chosen later.

For each i, run Dijkstra to and from W_i. If $\epsilon(W_i) > r$, then we can report $\epsilon(w) > r$ for all $w \in W_i$. Otherwise, $\epsilon(W_i) \leq r$. In this case, we will apply Claim 12, below, twice. Recall that for $S \subseteq W \subseteq V$, $L_W(S)$ is the set of vertices in W that are to the left of all vertices in S in the topological ordering.

Claim 12. *Let $W \subseteq V$ be a topologically consecutive subset of a topologically ordered DAG G, and let r be a parameter such that $\epsilon(W) \leq r$. In $O(m \log^2 n)$ time, one can find a nonempty topologically consecutive subset $S \subseteq W$ such that:*

(a) $\epsilon(S) \leq r$.

(b) If $w \in L_W(S)$, $\epsilon(w) > r$.

(c) If $|S| > 1$, *all vertices* $s \in S$ *satisfy* $\epsilon(s) > r$.

Proof. We will use a binary search argument to find S. We will induct on an index j. Let $S^0 = W$. Assume that $S^j \subseteq W$ is topologically consecutive, that $\epsilon(S^j) \leq r$, and that for every $w \in L_W(S^j)$,

$\epsilon(w) > r$. These all hold for $j = 0$. If $S^j = \{s\}$ consists of a single vertex, let $S = S^j$; then we are done.

Otherwise, let S^j_L be the subset of S^j containing its first $|S^j|/2$ vertices in the topological ordering and let $S^j_R = S^j \setminus S^j_L$. So S^j_L and S^j_R are the left and right halves of S^j, respectively; hence both S^j_L and S^j_R are topologically consecutive. See Figure 6-5.

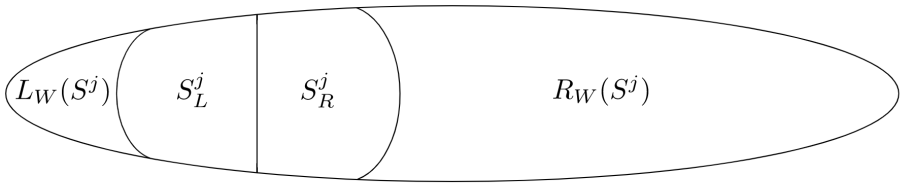

Figure 6-5: S^j is partitioned into two halves, S^j_L and S^j_R.

Run Dijkstra from S^j_L and from S^j_R. If either of these sets has min-eccentricity at most r, we will continue the induction: If $\epsilon(S^j_L) \leq r$, we let $S^{j+1} = S^j_L$. Then $L_W(S^{j+1}) = L_W(S^j)$, so for every $w \in L_W(S^{j+1})$, $\epsilon(w) > r$. Alternatively, if $\epsilon(S^j_L) > r$ but $\epsilon(S^j_R) \leq r$, we let $S^{j+1} = S^j_R$. Then $L_W(S^{j+1}) = L_W(S^j) \cup S^j_L$, so for every $w \in L_W(S^{j+1})$, $\epsilon(w) > r$.

Otherwise, $\epsilon(S^j_L) > r$ and $\epsilon(S^j_R) > r$. In this case we halt the induction and let $S = S^j$. Every $w \in L_W(S^j) \cup S^j$ satisfies $\epsilon(w) > r$, so S has the properties desired.

At each step, the size of the set S^j halves, so there are at most $\log |W| \leq \log n$ iterations. In each iteration, we perform a constant number of Dijkstras, so the runtime is $O(m \log^2 n)$. □

For each i such that $\epsilon(W_i) \leq r$, let S_i be the subset constructed by applying Claim 12 to the set $W = W_i$. For each $w \in L_{W_i}(S_i)$, we report that $\epsilon(w) > r$; this holds by Claim 12b. If S_i consists of a single vertex $\{s\}$, we can determine that for any $v \in L(W_i)$, $d_{\min}(v, s) \leq \epsilon(s) \leq r \leq kr$, by Claim 12a. Otherwise, $|S_i| > 1$, so we report that $\epsilon(s) > r$ for all $s \in S_i$; this holds by Claim 12c.

Using a recursive application of our algorithm to the graph $G_i = G[W_i]$, we can certify, for every vertex $w \in W_i$, that $\epsilon_{G_i}(w) > r$ or that $\epsilon_{G_i}(w) \leq (k-1)r$. Consider any $w \in R_{W_i}(S_i)$. If we determined that $\epsilon_{G_i}(w) > r$, we report that $\epsilon(w) > r$; this holds since $\epsilon(w) \geq \epsilon_{G_i}(w)$. Otherwise, consider any $v \in L(W_i)$. Since $\epsilon(S_i) \leq r$, there is some $s \in S_i$ such that $d_{\min}(v, s) = d(v, s) \leq r$. Then since $\epsilon_{G_i}(w) \leq (k-1)r$ and since w is to the right of s in the topological ordering, we have

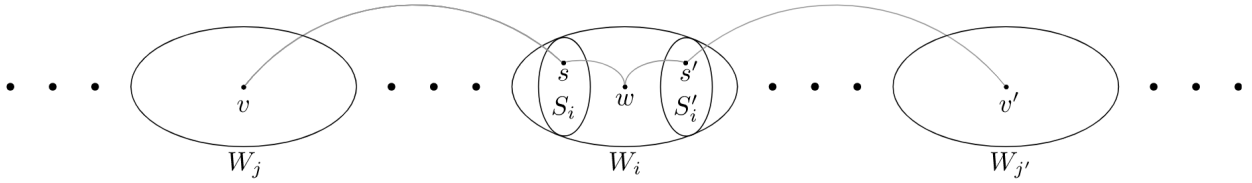

Figure 6-6: A representation of the $v \to w$ and $w \to v'$ paths, via the sets S_i and S'_i constructed with Claim 12. The outer subpaths are of length $\leq r$, and the inner subpaths are of length $\leq (k-1)r$.

$d_{\min}(v, w) \leq d(v, s) + d(s, w) \leq r + (k-1)r = kr$. See Figure 6-6.

Thus, our algorithm has certified for each $w \in W_i$ that $\epsilon(w) > r$ or that $d_{\min}(v, w) \leq kr$ for all $v \in L(W_i)$. By a symmetric argument, we can construct the set S'_i obtained by applying Claim 12 to the graph G with the edges reversed; see Figure 6-6. Then as above we can determine for each $w \in W_i$ that $\epsilon(w) > r$ or that $d_{\min}(w, v') \leq kr$ for all $v' \in R(W_i)$. Since W_i is a topologically consecutive set, $V \setminus W_i = L(W_i) \cup R(W_i)$. So for any $w \in W_i$, if we determine that $d_{\min}(w, v) \leq kr$ for all $v \in L(W_i)$ and for all $v \in R(W_i)$ we report that $\epsilon(w) \leq kr$; otherwise we report $\epsilon(w) > r$.

Runtime Analysis

In this section we analyze the runtime of the algorithm of Proposition 17, and we give full descriptions of how to prove Theorems 6.3.6-6.3.9 from Proposition 17 using binary search.

Recall that $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$.

Lemma 6.3.1. *The algorithm of Proposition 17 runs in time*

$$O(\min(mn^{1/k} \log^2 n, m^{2^{k-1}/(2^k-1)} n \log^2 n))$$

assuming APSP computations are done in $\tilde{O}(mn)$ time.

On graphs with integer edge weights bounded by n^t for $t < 3 - \omega$, the algorithm runs in time

$$O(\min(mn^{1/k} \log^2 n, m^{mc_k(\mu(t))} n \log^2 n)),$$

assuming APSP computations are done in $O(n^{2+\mu(t)})$ time using Zwick's fast APSP algorithm

[Zwi02].

Proof. To simultaneously analyze both versions of the algorithm, our algorithm's runtime will be described in terms of a placeholder τ, such that APSP computations within the algorithm are done in $O(n^{2+\tau}\log n)$ time. To obtain the runtime bound for general weighted DAGs, we will let $\tau = 1$, and note $c_k(1) = \frac{2^{k-1}}{2^k-1}$. To obtain the runtime bound for DAGs with integer edge weights bounded by n^t for $t < 3-\omega$, we will let $\tau = \mu(t)$.

Topologically sorting the graph takes $O(m\log n)$ time which is absorbed into the final runtime.

In order to use $k = 1$ as a base case, our inductive hypothesis will assume a slightly weaker claim about the runtime: in the inductive step for k, we will assume there is an $O(\min(mn^{1/(k-1)}\log^2 n, n^{2c_{k-1}(\tau)+1}\log^2 n))$-time algorithm which certifies for each $v \in V$ that $\epsilon(v) > r$ or that $\epsilon(v) \leq (k-1)r$. Note that $n^{2c_{k-1}} \geq m^{c_{k-1}}$. Then in the base case where $k = 1$, APSP takes time $O(\min(mn\log n, n^{2+\tau}\log n))$, satisfying the inductive hypothesis.

Consider $k > 1$. Running Dijkstra to and from W_i for each i takes $O(mp\log n)$. It takes time $O(mp\log^2 n)$ to apply Claim 12 twice for each i, to construct sets S_i and symmetric sets S_i' (constructed in the same way as the sets S_i but with left and right swapped, pictured in Figure 6-6).

We also do recursive calls of our algorithm on at most p subgraphs, induced by sets W_i. Below, we analyze the runtime of the recursive calls in two different ways, giving us two upper bounds on the algorithm's runtime.

Analysis 1 Let $m_i = |E(G[W_i])|$; then note $\sum_i m_i \leq m$. For each i, the recursive call on W_i takes time $O(m_i(n/p)^{1/(k-1)}\log^2 n)$, so in total the recursive calls take time $O(m(n/p)^{1/(k-1)}\log^2 n)$. Let $p = n^{1/k}$, so that $mp = m(n/p)^{1/(k-1)}$. Then the runtime is $O(mn^{1/k}\log^2 n)$.

Analysis 2 Since $|W_i| = n/p$, a recursive call on $G[W_i]$ takes time $O((n/p)^{2c_{k-1}(\tau)+1}\log^2 n)$. We do at most p such calls, so the total runtime of the recursive calls is $O((n/p)^{2c_{k-1}(\tau)}n\log^2 n)$. Now, we choose p so that $mp = (n/p)^{2c_{k-1}(\tau)}n$. Then $m = (n/p)^{2c_{k-1}(\tau)+1}$. Recall that $c_k(\tau) = \frac{2^{k-2}(1+\tau)}{2^{k-1}(1+\tau)-\tau}$ and note that $2c_{k-1}(\tau)+1 = \frac{2^{k-1}(1+\tau)-\tau}{2^{k-2}(1+\tau)-\tau} = \frac{2c_{k-1}(\tau)}{c_k(\tau)}$. Thus, $m^{c_k(\tau)} = (n/p)^{2c_{k-1}(\tau)}$. So the runtime of the algorithm is $O((n/p)^{2c_{k-1}(\tau)}\cdot n\log^2 n) = O(m^{c_k(\tau)}n\log^2 n)$. Since $m = O(n^2)$, this satisfies the inductive hypothesis. □

Lower Bounds

In this section, using an essentially linear time reduction, we reduce Triangle Detection to $(2-\delta)$-approximation of min-radius.

Reminder of Theorem 6.3.2 *If there is a $T(n,m)$-time algorithm for $(2-\delta)$-approximation of min-radius in $O(n)$-node $\tilde{O}(m)$-edge DAGs for some $\delta > 0$, then there is an $\tilde{O}(T(n,m)+m)$-time algorithm for Triangle Detection on graphs with n nodes and m edges.*

Proof. We are going to use two gadgets from previous works:

- DAG gadget [AVW16]: Given a set X of n nodes $v_1, \ldots, v_n$ and a constant integer parameter $t \geq 2$, the gadget creates a DAG $DG_t(X)$ with at most $O(n)$ nodes and $O(n \log n)$ edges such that in the topological order of $DG_t(X)$, $v_i < v_{i+1}$, and for any two nodes of $DG_t(X)$ x, y where $x < y$ in the topological order, $d(x,y) \leq t+1$.

- Connectivity gadget [AR18]: Let $X = \{v_1, \ldots, v_n\}$, and let $X' = \{v'_1, \ldots, v'_n\}$ be a copy of X, where both X and X' are independent sets. Then we can add a connectivity gadget $U(X)$ along with edges from X to $U(X)$ and from $U(X)$ to X', such that $|U(X)| = O(\log n)$, for all $i \neq j$ we have $d(v_i, v'_j) = 2$, and there is no path from v_i to v'_i.

Now let $G = (A, B, C, E_G)$ be an instance of Triangle Detection, with n nodes and m edges. We create a DAG G^* such that if G has a triangle (YES case), the min-radius of G^* is $t+1$, and if G doesn't have a triangle (NO case), the min-radius of G^* is $2t$. We let t be an integer such that $2 - \delta/2 < \frac{2t}{t+1}$, so that a fast $(2-\delta)$-approximation algorithm is also a fast $(\frac{2t}{t+1} - \delta/2)$-approximation algorithm, and hence it can distinguish min-diameter $t+1$ vs $2t$.

We define G^* as follows: G^* has A, B, and C as part of its vertex set. Let $A'_1, A'_2, \ldots, A'_{t+1}$ be copies of A. Add $E_G(A,B)$ to G^* with edges directed from A to B, and add $E_G(B,C)$ with edges directed from B to C. For any $c \in C$ and $a \in A$, add an edge from c to $a' \in A'_2$ if a and c are attached in G, where a' is the copy of a in A'_2. For each $i = 1, \ldots, t$, connect the copy of a in A'_i to the copy of a in A'_{i+1} for all $a \in A$.

Now we add the two gadgets. Add the connectivity gadget $U(A)$ between A and A'_1. Add two copies of $DG_t(A)$ sharing A, and denote the union of these copies by $DAG(A)$. Also add a

node y, and add edges from all nodes in A to y; this guarantees that the center of G^* must be in $DAG(A)$.

To make all nodes in A at distance $t+1$ to A'_1, make $t-1$ copies of $U(A)$, $U_1, \ldots, U_{t-1}$. For each $i = 1, \ldots, t-1$, connect the copy of u in U_i to the copy of u in U_{i+1}, for any $u \in U(A)$, where $U_t = U(A)$. Add edges from all nodes in $A \cup B \cup C$ to all nodes in U_1.

To make all nodes in A at distance $t+1$ to B and C, let $x_1, \ldots, x_t$ be a path of length $t-1$. Connect all nodes of A to x_1, and connect x_t to all nodes of $B \cup C$. See Figure 6-7 for the construction. Note that G^* is a DAG, with the order of sets of vertices being $DAG(A), y, x_1, \ldots, x_t, B, C, U_1, \ldots, U_{t-1}, U(A), A'_1, \ldots, A'_{t+1}$. Moreover, $G^*[A \cup B \cup C \cup A_2]$ has m edges corresponding to the original edges of G^*, and besides those we only added $O(n \log n)$ edges to G^*. So G^* has $O(n)$ nodes and $O(m + n \log n)$ edges.

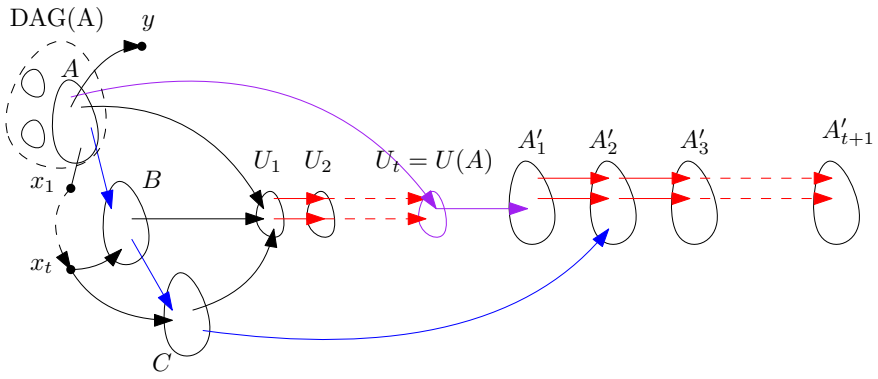

Figure 6-7: Graph G^* created from the Triangle Detection instance G. Blue edges are edges in G, red edges are between two nodes that are copies of the same vertex. Purple edges are part of the connectivity gadget. Dashed lines are subpaths.

We will show that if the Triangle Detection instance is a YES instance, then there is a node $a \in A$ such that $\epsilon(a) = t+1$. If the Triangle Detection instance is a NO instance, then we show that for all nodes in G^*, their min-eccentricity is at least $2t$.

YES case. Let abc be a triangle in G. We show that $\epsilon(a) = t+1$. Note that $d_{\min}(a, \bar{a}) \leq t+1$ for all $\bar{a} \in DAG(A)$. We already know that $d(a, s) \leq t+1$ for any $s \in B \cup C \cup \{x_1, \ldots, x_t, y\}$. For any $u \in U_i$ for $i \leq t+1$, $d(a, u) \leq t+1$ using the path going through $U_1, \ldots U_{i-1}$. Since for any $z' \in A'_1$, there is a $u \in U(A)$ that has an edge to z', we have $d(a, z') \leq t+1$. Now for all

$z' \in A'_2$ where z' is a copy of $z \in A$ and $z \neq a$, we have $d(a, z') = 3$ through $U(A)$ and A'_1 (using the edges of the connectivity gadget). For $z = a$, using the triangle edges going from A to B to C, we have that $d(a, z') = 3$. So for all $z' \in A'_2 \cup \ldots \cup A'_{t+1}$, we have $d(a, z') \leq t + 1$.

NO case. Suppose that there is no triangle in G. First, note that the min-eccentricities of the vertices outside $DAG(A)$ are infinite, because there is no path between them and y. Moreover, if $z \in DAG(A) \setminus A$, it has a copy $z' \in DAG(A) \setminus A$ (in the other copy of $DG_t(A)$), and there is no path between z and z'. This is because this path must go through A, and since $DAG(A)$ consists of two copies of $DG_t(A)$ sharing A, the set of nodes in A that z has a path to (from) is exactly the same as the set of nodes in A that z' has a path to (from). So there is no $a \in A$ such that that z has a path to a and z' has a path from a.

Now it remains to compute the min-eccentricities of the vertices in A. Let $a \in A$, and let $a'_{t+1} \in A'_{t+1}$ be the copy of a. We show that $d(a, a'_{t+1}) = 2t$. Let P be a shortest path from a to a'_{t+1}. First note that any path from a to a'_{t+1} must go through $a'_2 \in A'_2$, where a'_2 is a copy of a, and we have $d(a'_2, a'_{t+1}) = t - 1$. We also know that there is no path from a to a'_2 using the edges from A to $U(A)$, because this path would need to contain a path between a and $a'_1 \in A_1$ in $G^*[A \cup U(A) \cup A'_1]$, and from the construction of the connectivity gadget there is no such path. If P does not use any $C \times A'_2$ edge, then the path must go through U_i for all i, and hence it is of length $2t$. So if the min-eccentricity of a is smaller than $2t$, the path P uses a $C \times A'_2$ edge ca'_2 for some $c \in C$. If x_1 is on the ac path, then the path goes through x_i for all i, and hence it is of length $2t$. Then x_1 is not on the path, so the ac path must go through B. In particular, there is a $b \in B$ such that $ab, bc \in E(G^*)$. Since $ca'_2 \in E(G^*)$, this implies that abc is a triangle in G, which is a contradiction. So $\epsilon(a) \geq 2t$.

□

6.3.3 Min-diameter

Our min-diameter approximation algorithm relies on Yuster and Zwick's fast sparse matrix multiplication algorithm. Here, we define $\alpha = \max\{0 \leq r \leq 1 \mid \omega(1, r, 1) = 2\}$ and $\beta = \frac{\omega-2}{1-\alpha}$.

Theorem 6.3.10 ([YZ05]). *If A and B are n by n matrices with at most l nonzero entries each,*

then A and B can be multiplied in $O(l^{\frac{2\beta}{\beta+1}}n^{\frac{2-\alpha\beta}{\beta+1}} + n^{2+o(1)})$ time.[7]

This sparse matrix multiplication algorithm will be used to prove the following proposition.

Proposition 18. *There is an $O(m^{\frac{2\beta}{3\beta+1}}n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$-time algorithm which, given an unweighted DAG G and a parameter D', reports that the min-diameter D of G satisfies $D \leq \frac{3D'}{2}$ or that it satisfies $D > D'$.*

The algorithm of Proposition 18 will be described and proven to work in subsection 3.1, and its runtime will be analyzed in Lemma 6.3.3 in subsection 6.3.3. Then Proposition 18 allows us to obtain the min-diameter approximation algorithm given in Theorem 6.3.11 below.

Theorem 6.3.11. *There is an $O(m^{\frac{2\beta}{3\beta+1}}n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$-time algorithm which, given an unweighted DAG G, outputs an estimate D_0 for its min-diameter D such that $D \leq D_0 < \frac{3D}{2}$.*

Proof. To obtain our approximation D_0, we binary search over D' in $[0, n]$ by applying the algorithm of Proposition 18 logarithmically many times; note that polylogarithmic factors are $n^{o(1)}$ so they do not affect the runtime bound. Let C be the smallest value found in the binary search such that the algorithm reports that $D \leq \frac{3C}{2}$; then $D > C - 1$. Let $D_0 = \frac{3C}{2}$. Then $D \leq D_0 < \frac{3D}{2}$, as desired. □

Note that since $\alpha > 0.31389$ [GU17] and $\omega < 2.37286$ [AV21], we can use $\beta \simeq 0.5435$. This gives the runtime of $O(m^{0.414}n^{1.522} + n^{2+o(1)})$ stated in Theorem 6.3.4.

Algorithm Description and Correctness

Our algorithm takes as input an unweighted DAG G, an integer D', and a parameter $\epsilon \in [0, 1]$, and reports that $D > D'$ or that $D \leq \frac{3D'}{2}$. (The runtime-minimizing value of ϵ will be determined later.)

If at any point, a BFS finds a pair of vertices at min-distance more than D', the algorithm reports that $D > D'$; hence in what follows we will assume that this does not occur. We initially have all pairs of vertices "unmarked," and mark the pairs for which we know that there is a path from one to the other of length at most $\frac{3D'}{2}$.

[7] To be precise, given known bounds $\alpha \geq a, \omega \leq c$, one can define $b = \frac{c-2}{1-a}$, and then equivalents of Theorem 6.3.10 hold for any such pair of values a, b, not just for the "true" values α, β. This is implicit in [YZ05].

The algorithm first takes two preliminary steps: it topologically sorts the graph, and it constructs for each vertex two topologically sorted lists, one of its in-neighbors and one of its out-neighbors.

Our algorithm will then use the greedy set cover algorithm, described in the following lemma. This lemma, and a related randomized version, are standard techniques used in graph distance algorithms (see for example [ACIM99, RV13, CLR$^+$14, AWW16]). A proof may be found in [VWSK16].

Lemma 6.3.2. *Let $|V| = n$, let $p = O(n)$, and let $X_1, \ldots X_p \subseteq V$ be sets of size $|X_i| \geq n^\epsilon$ for $\epsilon \in [0, 1]$. Then there is an $O(n^{1+\epsilon})$-time algorithm which constructs a set $S \subseteq V$ of size $\tilde{O}(n^{1-\epsilon})$ such that $S \cap X_i \neq \varnothing$ for all i.*

For any $u \in V$, if $|N^{\text{out}}_{D'/2}(u)| < n^\epsilon$ let $X_u = N^{\text{out}}_{D'/2}(u)$ and otherwise let X_u be the left-most n^ϵ vertices in $N^{\text{out}}_{D'/2}(u)$. So in particular, $|X_u| \leq n^\epsilon$. We can compute X_u as follows: we maintain a list of the $\leq n^\epsilon$ left-most vertices we have found so far that are at distance $< D'/2$ from u. At each step, for each vertex in the list, we consider its left-most out-neighbor that is not yet in our set; we add the left-most such out-neighbor to the set. We halt when there are no more such out-neighbors not in our set, or after adding n^ϵ vertices to our set. Likewise, for any $w \in V$, let $Y_w = N^{\text{in}}_{D'/2}(w)$ if $|N^{\text{in}}_{D'/2}(w)| < n^\epsilon$, and otherwise let Y_w consist of the right-most n^ϵ vertices in $N^{\text{in}}_{D'/2}(w)$. We can compute the sets Y_w in a manner symmetric to how we computed the sets X_u. Then we can use Lemma 6.3.2 to construct a set S of size $\tilde{O}(n^{1-\epsilon})$ such that for all u having $|N^{\text{out}}_{D'/2}(u)| \geq n^\epsilon$, $S \cap X_u$ is nonempty, and for all w having $|N^{\text{in}}_{D'/2}(w)| \geq n^\epsilon$, $S \cap Y_w$ is nonempty.

Run BFS into and out of every $s \in S$. We may assume that $d_{\text{min}}(s, x) \leq D'$ for all $s \in S, x \in V$.

We will construct matrices A and B with rows and columns indexed by vertices in V, as follows: For each vertex $t \in X_u$, let $A[u, t] = 1$. For each vertex $t \in Y_w$, let $B[t, w] = 1$. Multiply A and B using the sparse matrix multiplication algorithm of Theorem 6.3.10.

Now, we will consider any pair of vertices (u, w) where u is to the left of w, $u \in R(N^{\text{in}}_{D'/2}(w) \cap S)$, and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$. We have that $(A \cdot B)[u, w] > 0$ if and only if $d(u, w) \leq D'$. Indeed, if $d(u, w) \leq D'$, then there is some intermediate vertex t such that $d(u, x) \leq D'/2$ and

$d(x, w) \leq D'/2$. Suppose that $t \notin X_u$. Then since X_u is defined as the left-most n^ϵ vertices in $N^{\text{out}}_{D'/2}(u)$, this implies that $|N^{\text{out}}_{D'/2}(u)| > n^\epsilon$ and hence that $|X_u| = n^\epsilon$. Then there is some $s \in S \cap X_u$. Since $t \notin X_u$, t is to the right of all vertices in X_u, and in particular t is to the right of s. This implies $t \notin L(N^{\text{out}}_{D'/2}(u) \cap S)$. But since $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ and t lies between u and w, this is a contradiction. Thus, t must be in X_u, and by symmetry, t is in Y_w. So $A[u, t] = 1$ and $B[t, w] = 1$, meaning $(A \cdot B)[u, w] > 0$. Likewise, if $(A \cdot B)[u, w] > 0$, then there exists $t \in X_u \cap Y_w$ such that $d(u, t) \leq D'/2$ and $d(t, w) \leq D'/2$, so $d(u, w) \leq D'$. Therefore, we will mark all pairs (u, w) such that $(A \cdot B)[u, w] > 0$.

Now, consider any $u \in V$ and any $w \notin L(N^{\text{out}}_{D'/2}(u) \cap S)$ to the right of u. We mark the pair (u, w). If such a w exists, then there is some $s \in N^{\text{out}}_{D'/2}(u) \cap S$ such that s is to the left of or is equal to w. By assumption, $d(s, w) \leq D'$, so $d(u, w) \leq d(u, s) + d(s, w) \leq D'/2 + D' = \frac{3D'}{2}$. By a symmetric argument, for any $w \in V$ and any $u \notin R(N^{\text{in}}_{D'/2}(w) \cap S)$ to the left of w, we have that $d(u, w) \leq \frac{3D'}{2}$, so again we mark any such pair (u, w). Thus, since we have assumed that $\epsilon(s) \leq \frac{3D'}{2}$ for all $s \in S$, the algorithm will mark all pairs of vertices $u, w \in V$ except those for which we have simultaneously that $u \in R(N^{\text{in}}_{D'/2}(w) \cap S)$ and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$.

Finally, check whether there exists an unmarked pair (u, w). If so, report that $D > D'$. Otherwise, report that $D \leq \frac{3D'}{2}$.

Runtime Analysis

Here we analyze the runtime of the algorithm of Proposition 18.

Lemma 6.3.3. *The algorithm of Proposition 18 runs in time* $\tilde{O}(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$.

Proof. Topologically sorting the graph takes $O(m \log n)$ time which is absorbed into the final runtime. Constructing for each vertex topologically ordered lists of its in-neighbors and out-neighbors can be done in time $\tilde{O}(n^2)$.

Computing the covering set S takes time $\tilde{O}(n^{1+\epsilon})$ and running BFS from its vertices takes time $O(n^{1-\epsilon} m \log n)$. Checking for each pair (u, w) whether $u \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ and $w \in L(N^{\text{out}}_{D'/2}(u) \cap S)$ can be done in $\tilde{O}(n^2)$ time.

For a fixed u, to compute X_u, we maintain a list of the at most n^ϵ left-most vertices we have found that are at distance $< D'/2$ from u. For each vertex, we store its left-most out-neighbor that

is not yet in our set. At each step, we find the left-most such out-neighbor of any vertex in the list; this takes time $O(n^\epsilon)$, and updating the list to reflect that this out-neighbor has been added to our set takes time $O(n^\epsilon)$. At each step we add a vertex to our set X_u, so there are at most $O(n^\epsilon)$ steps. Hence, constructing X_u for a fixed u takes $O(n^{2\epsilon})$ time. Then constructing all sets X_u, Y_w takes $O(n^{1+2\epsilon})$ time altogether.

Finally, note that there are at most n^ϵ 1s in each row of A, since we only set $A[u,t] = 1$ if $t \in X_u$. Thus, A contains at most $n^{1+\epsilon}$ 1s. By symmetry, the same holds for B. Then multiplying A and B can be done in time $O(n^{(1+\epsilon)\frac{2\beta}{\beta+1}+\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)})$, using Yuster and Zwick's fast sparse matrix multiplication (Theorem 6.3.10).

Then the total runtime is:

$$\tilde{O}(n^{1-\epsilon}m + n^{1+2\epsilon} + n^{(1+\epsilon)\frac{2\beta}{\beta+1}+\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)})$$

Let γ be the largest value such that $n^\gamma = O(m)$. Let $\epsilon = \frac{\alpha\beta+(\beta+1)(\gamma-1)}{3\beta+1}$; this value is chosen because it sets the first and third terms in the above runtime equal (up to $n^{o(1)}$ factors), hence asymptotically minimizing their sum. Substituting the value of ϵ and simplifying, the runtime of the algorithm is:

$$\tilde{O}(n^{\frac{2\beta}{3\beta+1}\gamma+\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{\frac{2\beta+2}{3\beta+1}\gamma+\frac{\beta-1+2\alpha\beta}{3\beta+1}} + n^{2+o(1)})$$

We note that $3\beta - 3\alpha\beta > 3(\omega - 2) \geq 0 > -1$, giving:

$$4\beta + 2 - \alpha\beta > 2 + (\beta - 1 + 2\alpha\beta) \geq 2\gamma + (\beta - 1 + 2\alpha\beta)$$

Thus, the first term of the above runtime dominates the second. Substituting $n^\gamma = O(m)$, and noting that the polylogarithmic factors in the runtime are of order $n^{o(1)}$, the runtime is $O(m^{\frac{2\beta}{3\beta+1}} n^{\frac{4\beta+2-\alpha\beta}{3\beta+1}+o(1)} + n^{2+o(1)})$, as desired.

□

Bibliography

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117, 2015.

[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.

[ADH+08] N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–i249, 2008.

[AESW91] Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(1):407–422, December 1991.

[AGM97] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

[AGV15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

[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 ICALP*, page to appear, 2019.

[AR18] Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 239–252, 2018.

[AV21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[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.

[AWW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua 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*, pages 377–391. SIAM, 2016.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[BCE80] B. Bollobás, P.A. Catlin, and P. Erdös. Hadwiger's conjecture is true for almost every graph. *European Journal of Combinatorics*, 1(3):195 – 199, 1980.

[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*, 2015. accepted.

[BGP+13] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14(7):1–13, 2013.

[BK07] P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. WADS*, pages 541–552, 2007.

[BKS18] Markus Bläser, Balagopal Komarath, and Karteek Sreenivasaiah. Graph pattern polynomials. *CoRR*, abs/1809.08858, 2018.

[BMBST07] B. Ben-Moshe, B. K. Bhattacharya, Q. Shi, and A. Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237 – 252, 2007.

[Bol88] B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[Bon21a] Édouard Bonnet. 4 vs 7 sparse undirected unweighted diameter is seth-hard at time $n^{4/3}$. In *Proc. ICALP*, pages 34:1–34:15, 2021.

[Bon21b] Édouard Bonnet. Inapproximability of diameter in super-linear time: Beyond the 5/3 ratio. In *38th International Symposium on Theoretical Aspects of Computer Science,*

STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference), volume 187 of *LIPIcs*, pages 17:1–17:13. 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.

[BS74] A. Bondy and M. Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory*, 16:97–105, 1974.

[BW17] Karl Bringmann and Philip Wellnitz. Clique-based lower bounds for parsing tree-adjoining grammars. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, pages 12:1–12:14, 2017.

[CCF+05] Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized np-hard problems. *Inf. Comput.*, 201(2):216–231, 2005.

[CD94] V. Chepoi and F. F. Dragan. A linear-time algorithm for finding a central vertex of a chordal graph. In *ESA*, pages 159–170, 1994.

[CD19] Ruoxu Cen and Ran Duan. Roundtrip spanners with $(2k-1)$ stretch. *CoRR*, abs/1911.12411, 2019.

[CDHP01] D.G. Corneil, F.F. Dragan, M. Habib, and C. Paul. Diameter determination on restricted graph families. *Discr. Appl. Math.*, 113:143 – 166, 2001.

[CDM17] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223, 2017.

[CDV02] V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *Proc. SODA*, pages 346–355, 2002.

[CE06] Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM Journal on Discrete Mathematics*, 20(2):463–501, 2006.

[CEGS94] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J Guibas, and Micha Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994.

[CGK13] Marek Cygan, Fabrizio Grandoni, and Telikepalli Kavitha. On pairwise spanners. *arXiv preprint arXiv:1301.1999*, 2013.

[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.

[Cha07] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. STOC*, pages 590–598, 2007.

[Cha12] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8(4):34, 2012.

[Chu87] F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congr. Numer.*, 60:295–317, 1987.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006.

[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.

[CIP10] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified k-cnf. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation - 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2010.

[CL21] Shiri Chechik and Gur Lifshitz. Optimal girth approximation for dense directed graphs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 290–300. SIAM, 2021.

[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.

[CLRS19] Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Improved girth approximation and roundtrip spanners. *CoRR*, abs/1907.10779, 2019.

[CLRS20] Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Improved girth approximation and roundtrip spanners. In *Proceedings of STOC*, page to appear, 2020.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[CPS85] D. Corneil, Y. Perl, and L. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

[CW99] L. Cowen and C. Wagner. Compact roundtrip routing for digraphs. In *SODA*, pages 885–886, 1999.

[DDV19] Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of STOC 2019*, page to appear, 2019.

[DG04] Adrian Dumitrescu and Sumanta Guha. Extreme distances in multicolored point sets. *J. Graph Algorithms and Applications*, 8(1):27–38, 2004.

[DH04] D. Dvir and G. Handler. The absolute center of a network. *Networks*, 43:109 – 118, 2004.

[DK21] Mina Dalirrooyfard and Jenny Kaufmann. Approximation algorithms for min-distance problems in dags. *arXiv preprint arXiv:2106.02120*, 2021.

[DKS17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 112–120, 2017.

[DLW21] Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. *arXiv preprint arXiv:2106.06026*, 2021.

[DVW20] Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[DVW21] Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM Journal on Computing*, 50(5):1627–1662, 2021.

[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*, pages 1697–1710, 2021.

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

[DWVW19] Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, and Nicole Wein. Tight approximation algorithms for bichromatic graph diameter and related problems. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[EG04] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comp. Sci.*, 326(1-3):57–67, 2004.

[Epp99] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms and Applications*, 3(3):1–27, 1999.

[FHW12] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1150–1162. SIAM, 2012.

[FKLL13] Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 547–557, 2013.

[FKLL15] Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Induced subgraph isomorphism: Are some patterns substantially easier than others? *Theor. Comput. Sci.*, 605:119–128, 2015.

[FLR+12] Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012.

[GIKW17] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2162–2181, 2017.

[Gol93] A.V. Goldberg. Scaling algorithms for the shortest paths problem. In *Proc. SODA*, pages 222–231, 1993.

[GU17] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. *CoRR*, abs/1708.05622, 2017.

[GU18] Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046, 2018.

[Had57] H. Hadwiger. Ungelöste probleme nr. 20. *Elemente der Mathematik*, 12:121, 1957.

[Hak64] S.L. Hakimi. Optimum location of switching centers and absolute centers and medians of a graph. *Oper. Res.*, 12:450 – 459, 1964.

[Ind07] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *SODA*, volume 7, pages 39–42, 2007.

[IP01a] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IP01b] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[IP01c] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[IR78] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

[Kav17] Telikepalli Kavitha. New pairwise spanners. *Theory of Computing Systems*, 61(4):1011–1036, 2017.

[KI92] Naoki Katoh and Kazuo Iwano. Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, pages 320–329, 1992.

[KKM00] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Proc. Letters*, 74(3-4):115–121, 2000.

[KL17] Miroslaw Kowaluk and Andrzej Lingas. A fast deterministic detection of small pattern graphs in graphs without large cliques. In *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, pages 217–227, 2017.

[Kle06] Philip N Klein. A subset spanner for planar graphs, with application to subset tsp. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 749–756. ACM, 2006.

[KLL13] Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discrete Math.*, 27(2):892–909, 2013.

[KT05] Ken-ichi Kawarabayashi and Bjarne Toft. Any 7-chromatic graphs has K_7 or $K_{4,\,4}$ as A minor. *Combinatorica*, 25(3):327–353, 2005.

[Le 14] François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

[Li20] Ray Li. Improved seth-hardness of unweighted diameter. *CoRR*, abs/2008.05106v1, 2020.

[Li21] Ray Li. Settling seth vs. approximate sparse directed unweighted diameter (up to (nu)nseth). In *Proc. STOC*, STOC'2021, pages 1684–1696, 2021.

[Loo11] Andy Loo. On the primes in the interval [3n, 4n]. *arXiv preprint arXiv:1110.2377*, 2011.

[LU18] Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018.

[LVW18] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1236–1252, 2018.

[Mar07] Dániel Marx. Can you beat treewidth? In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 169–179. IEEE, 2007.

[MLH09] Clémence Magnien, Matthieu Latapy, and Michel Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *J. Exp. Algorithmics*, 13:10:1.10–10:1.9, February 2009.

[MRS21] Pasin Manurangsi, Aviad Rubinstein, and Tselil Schramm. The strongish planted clique hypothesis and its consequences. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Nag52] Jitsuro Nagura. On the interval containing at least one prime number. *Proceedings of the Japan Academy*, 28(4):177–181, 1952.

[NP85] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Math. Universitatis Carolinae*, 26(2):415–419, 1985.

[Ola90] Stephan Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34:121–128, 1990.

[PCJ06] N. Przulj, D. G. Corneil, and I. Jurisica. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics*, 22(8):974–980, 2006.

[Pet02] Seth Pettie. A faster all-pairs shortest path algorithm for real-weighted sparse graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 85–97. Springer, 2002.

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

[PR02] Seth Pettie and Vijaya Ramachandran. Computing shortest paths with comparisons and additions. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 267–276, 2002.

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

[PRS+18] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1374–1392. SIAM, 2018.

[PRT12] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 660–672, 2012.

[PVW19] Maximilian Probst, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *unpublished manuscript*, page submitted, 2019.

[PVW20] Maximilian Probst, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of STOC*, page to appear, 2020.

[Ros08] Benjamin Rossman. On the constant-depth complexity of k-clique. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 721–730, 2008.

[RSST97] Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2 – 44, 1997.

[RST93] Neil Robertson, Paul Seymour, and Robin Thomas. Hadwiger's conjecture for k6-free graphs. *Combinatorica*, 13(3):279–361, 1993.

[RV11] Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189. IEEE Computer Society, 2011.

[RV12] Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 833–845. SIAM, 2012.

[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.

[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.

[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*, CIKM '11, pages 1191–1196, 2011.

[Vas08a] V. Vassilevska. Efficient algorithms for path problems. *Ph.D. Thesis in Computer Science, Carnegie Mellon University*, 2008.

[Vas08b] Virginia Vassilevska Williams. Efficient algorithms for path problems in weighted graphs. *Ph.D. Thesis, Carnegie Mellon University*, 2008.

[Vas09] Virginia Vassilevska. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4):254–257, 2009.

[Vas12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

[Vas15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015.

[Vas18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

[VW10] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.

[VWSK16] Virginia Vassilevksa Williams, Nike Sun, and Nishith Khandwala. Lecture notes in graph algorithms (hitting sets, APSP), October 2016.

[Wag37] K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

[Wil04] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. ICALP*, pages 1227–1237, 2004.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

[Wil14] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014.

[Wil18] Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1207–1215, 2018.

[Wul08] C. Wulff-Nilsen. Wiener index, diameter, and stretch factor of a weighted planar graph in subquadratic time. *Technical report, University of Copenhagen*, 2008.

[WW13] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.

[WWWY15] Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680, 2015.

[WY13] O. Weimann and R. Yuster. Approximating the diameter of planar graphs in near linear time. In *Proc. ICALP*, 2013.

[Yao82] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[Yus10] Raphael Yuster. Computing the diameter polynomially faster than apsp. *arXiv preprint arXiv:1011.6181*, 2010.

[YZ94] R. Yuster and U. Zwick. Finding even cycles even faster. In *Proc. ICALP*, pages 532–543, 1994.

[YZ04] R. Yuster and U. Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proc. SODA*, pages 247–253, 2004.

[YZ05] R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. on Algorithms*, 1(1):2–13, 2005.

[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.