Better Approximation Algorithms for the Graph Diameter

Shiri Chechik* Daniel H. Larkin† Liam Roditty‡ Grant Schoenebeck§
Robert E. Tarjan¶ Virginia Vassilevska Williams‖

Abstract

The diameter is a fundamental graph parameter and its computation is necessary in many applications. The fastest known way to compute the diameter exactly is to solve the All-Pairs Shortest Paths (APSP) problem.

In the absence of fast algorithms, attempts were made to seek fast algorithms that approximate the diameter. In a seminal result Aingworth, Chekuri, Indyk and Motwani [SODA'96 and SICOMP'99] designed an algorithm that computes in $\tilde{\mathcal{O}}\left(n^2 + m\sqrt{n}\right)$ time an estimate $\tilde{D}$ for the diameter D in directed graphs with nonnegative edge weights, such that $\lfloor 2/3 \cdot D \rfloor - (M-1) \leq \tilde{D} \leq D$, where M is the maximum edge weight in the graph. In recent work, Roditty and Vassilevska W. [STOC 13] gave a Las Vegas algorithm that has the same approximation guarantee but improves the (expected) runtime to $\tilde{\mathcal{O}}\left(m\sqrt{n}\right)$. Roditty and Vassilevska W. also showed that unless the Strong Exponential Time Hypothesis fails, no $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for sparse unweighted undirected graphs can achieve an approximation ratio better than $3/2$. Thus their algorithm is essentially tight for sparse unweighted graphs. For weighted graphs however, the approximation guarantee can be meaningless, as M can be arbitrarily large.

In this paper we exhibit two algorithms that achieve a genuine $3/2$-approximation for the diameter, one running in $\tilde{\mathcal{O}}\left(m^{3/2}\right)$ time, and one running in $\tilde{\mathcal{O}}\left(mn^{2/3}\right)$ time. Furthermore, our algorithms are *deterministic*, and thus we present the first deterministic $(2-\varepsilon)$-approximation algorithm for the diameter that takes subquadratic time in sparse graphs.

In addition, we address the question of obtaining an *additive* c-approximation for the diameter, i.e. an estimate $\tilde{D}$ such that $D - c \leq \tilde{D} \leq D$. An extremely simple $\tilde{\mathcal{O}}\left(mn^{1-\varepsilon}\right)$ time algorithm achieves an additive n^{ε}-approximation; no better results are known. We show that for any $\varepsilon > 0$, getting an additive n^{ε}-approximation algorithm for the diameter running in $\mathcal{O}\left(n^{2-\delta}\right)$ time for any $\delta > 2\varepsilon$ would falsify the Strong Exponential Time Hypothesis. Thus the simple algorithm is probably essentially tight for sparse graphs, and moreover, obtaining a subquadratic time additive c-approximation for any constant c is unlikely.

Finally, we consider the problem of computing the *eccentricities* of all vertices in an undirected graph, i.e. the largest distance from each vertex. Roditty and Vassilevska W. [STOC 13] show that in $\tilde{\mathcal{O}}\left(m\sqrt{n}\right)$ time, one can compute for each $v \in V$ in an undirected graph, an estimate $e(v)$ for the eccentricity $\epsilon(v)$ such that $\max\{R, 2/3 \cdot \epsilon(v)\} \leq e(v) \leq \min\{D, 3/2 \cdot \epsilon(v)\}$ where $R = \min_v \epsilon(v)$ is the radius of the graph. Here we improve the approximation guarantee by showing that a variant of the same algorithm can achieve estimates $\epsilon'(v)$ with $3/5 \cdot \epsilon(v) \leq \epsilon'(v) \leq \epsilon(v)$.

1 Introduction

The diameter, defined as the largest distance in a graph, is one of the most basic graph parameters. Its computation is necessary in many applications—in the analysis of social networks [35, 2], and even in linguistics where one may wish to find two synonyms furthest from one another.

Computing the diameter of the graph has been extensively studied. Fast algorithms are known for some special families of graphs, e.g. [27, 21, 13, 12, 7, 24, 30, 14, 19, 20, 39] (see also [26, 36] for diameter approximation in planar graphs). The fastest known algorithm that computes the diameter exactly in general graphs computes the distances between *every pair* of vertices in the graph, solving the all pairs shortest paths (APSP) problem. It is a longstanding open problem whether one can compute the diameter faster than APSP [15]. This problem is especially intriguing in the case of sparse graphs. Any algorithm for APSP must take quadratic time in the number of vertices, regardless of the graph sparsity, since this is the size of the output. In contrast, the output of the diameter problem is a single number, and it is unclear why one should spend quadratic time on its computation in sparse graphs.

A seemingly simpler problem is that of approximating the diameter. Indeed, there is a trivial *linear* time algorithm that returns a 2-approximation to the diameter D, i.e. an estimate $\tilde{D}$ such that $1/2 \cdot D \leq \tilde{D} \leq D$: run breadth first search (BFS) from and to an arbitrary vertex and return the maximum distance found. (The algorithm also works for weighted graphs by using Dijkstra's algorithm instead of BFS with a slight increase in running time.) Thus when it comes to 2-approximations, diameter compu-

*Microsoft Research, SVC
†Princeton University
‡Bar Ilan University
§University of Michigan
¶Princeton University and Microsoft Research, SVC
‖Stanford University

tation in sparse graphs is easier than APSP.

A seminal result of Aingworth et al. [1] showed that one can beat the approximation factor of 2 for the diameter with a running time of $\tilde{\mathcal{O}}\left(n^2 + m\sqrt{n}\right)$ [1] where n and m are the number of vertices and edges respectively. The algorithm guarantees that the estimate $\tilde{D}$ satisfies $\lfloor 2/3 \cdot D \rfloor - (M-1) \leq \tilde{D} \leq D$, where M is the maximum edge weight in the graph, thus it gives almost a $3/2$-approximation.

Unfortunately, for sparse graphs, Aingworth et al.'s algorithm still runs in $\Omega(n^2)$ time which suffices to compute APSP even exactly. In recent work, Roditty and Vassilevska W. [33] gave a Las Vegas algorithm running in expected $\tilde{\mathcal{O}}\left(m\sqrt{n}\right)$ time that has the same approximation guarantee as Aingworth et al., thus giving the first $O(n^{2-\delta})$ time (almost) $3/2$-approximation algorithm for sparse graphs. They also showed that obtaining a $(3/2-\varepsilon)$-approximation algorithm running in $\mathcal{O}\left(n^{2-\delta}\right)$ time in sparse undirected and unweighted graphs for constant $\varepsilon, \delta > 0$ would be difficult, as it would imply a fast algorithm for CNF Satisfiability, violating the widely believed Strong Exponential Time Hypothesis of Impagliazzo, Paturi and Zane [29].

The Roditty and Vassilevska W. result is essentially tight for undirected, unweighted graphs. Nevertheless, the approximation guarantee can be meaningless for arbitrary weighted graphs, as the additive error of M can be arbitrarily large.

A natural question is, can we remove the additive error while still keeping the running time (in terms of n) subquadratic for sparse graphs? As there are many cubic time algorithms that compute the diameter (an algorithm for APSP such as Floyd-Warshall would do), can the algorithm also run in truly subcubic running time, even for dense graphs?

In this paper we answer the above questions by exhibiting two algorithms that achieve a genuine $3/2$-approximation for the diameter, one running in $\tilde{O}(m^{3/2})$ time, and one running in $\tilde{O}(mn^{2/3})$ time. Our algorithms are purely combinatorial, do not use sophisticated data structures, and are likely to be practical. Furthermore, unlike the algorithm of [33], our algorithms are *deterministic*, and thus we present the first deterministic $(2-\varepsilon)$-approximation algorithm for the diameter that takes subquadratic time in sparse graphs.

THEOREM 1.1. *There is a deterministic algorithm that computes an estimate $\tilde{D}$ of the diameter D of a directed or undirected graph with nonnegative integer edge weights in time $\tilde{\mathcal{O}}\left(\min\left\{m^{3/2}, mn^{2/3}\right\}\right)$, so that $\lceil 2/3 \cdot D \rceil \leq \tilde{D} \leq D$.*

We note that Theorem 1.1 can be strengthened to hold also for graphs with arbitrary nonnegative *real* weights. This extension will appear in the full version of the paper.

So far we have only discussed multiplicative approximation guarantees. A stronger guarantee is to obtain an *additive* c-approximation, i.e. an estimate $\tilde{D}$ such that $D - c \leq \tilde{D} \leq D$, where c is hopefully a small quantity, typically unrelated to D. Good additive approximation algorithms have been obtained for APSP. For instance, Aingworth et al. [1] use the ideas behind their diameter algorithm to obtain an $\tilde{\mathcal{O}}\left(n^{5/2}\right)$ time algorithm that computes additive 2-approximations to all distances in the graph. This additive approximation for APSP of course also provides an additive approximation for the diameter. However, as before, the drawback is that the algorithm runs in $\Omega\left(n^2\right)$ time, even for sparse graphs.

A closely related notion is that of additive spanners. An additive spanner is a subgraph H such that for any pair of nodes s, t, the distance of s and t in H is at most their distance in G plus some additive term. A naive approach to approximate the diameter within an additive stretch is to first find a sparse additive spanner of the graph and then compute APSP on the spanner. However, one cannot beat quadratic time using this approach as APSP is computed. For related work on additive spanners see [5, 6, 31, 23, 10, 23, 34, 32, 38, 37, 11, 1, 18].

For sparse graphs, there are no known subquadratic time algorithms that provide constant or even logarithmic additive approximation guarantees for the diameter, even when the graphs are unweighted and undirected. The best additive approximation guarantee for sparse graphs is obtained by a simple random sampling argument that shows that for all $\varepsilon > 0$, there is an $\tilde{\mathcal{O}}\left(mn^{1-\varepsilon}\right)$ time randomized additive n^{ε}-approximation algorithm for the diameter. Boitmanis et al.[9] presented a *deterministic* algorithm with $\tilde{\mathcal{O}}\left(m\sqrt{n}\right)$ running time that computes the diameter with an additive error of $\sqrt{n}$. A natural question is, why is it so difficult to obtain good additive approximations for the diameter?

We give a partial answer to the above question by showing that for any $\varepsilon > 0$, getting an additive n^{ε}-approximation algorithm for the diameter running in $\mathcal{O}\left(n^{2-\delta}\right)$ time for any $\delta > 2\varepsilon$ would falsify the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi and Zane [29]. Before our hardness result, no lower bounds were known - not even for the case of computing an additive 2-approximation

[1] The $\tilde{O}$ notation suppresses polylogarithmic factors.

for the diameter. Our result is inspired by the hardness result presented in [33].

THEOREM 1.2. *If for some constants $\varepsilon > 0$ and $\delta > 2\varepsilon$ there is an $\mathcal{O}\left(n^{2-\delta}\right)$ time algorithm that achieves an additive n^{ε} approximation to the diameter of an unweighted, undirected graph, then there is a constant $\gamma > 0$ such that CNF-SAT on N variables can be solved in $\mathcal{O}^*\left((2-\gamma)^N\right)$ time*[2].

COROLLARY 1.1. *Either SETH is false, or there are no $O(n^{2-\delta})$ time additive $n^{o(1)}$-approximations for the diameter for any $\delta > 0$.*

Finally, we consider the problem of computing the *eccentricities* of all vertices in an undirected graph, i.e. the largest distance from each vertex. Roditty and Vassilevska W. [33] show that in $\tilde{\mathcal{O}}\left(m\sqrt{n}\right)$ time, one can compute for each $v \in V$, an estimate $e(v)$ for the eccentricity $\epsilon(v)$ such that $\max\{R, 2/3\epsilon(v)\} \leq e(v) \leq \min\{D, 3/2\epsilon(v)\}$ where $R = \min_v \epsilon(v)$ is tha radius of the graph. In the last section of this paper we improve the approximation guarantee by showing that in $\tilde{\mathcal{O}}\left(m^{3/2}\right)$ time, one can achieve estimates $\epsilon'(v)$ with $3/5 \cdot \epsilon(v) \leq \epsilon'(v) \leq \epsilon(v)$.

Related work. A related problem to the diameter is computing the *radius* of the graph, defined as the minimum eccentricity. Berman and Kasiviswanathan [8] showed that the approach of Aingworth et al. can be used to obtain in $\tilde{O}(m\sqrt{n}+n^2)$ time an estimate $\hat{r}$ of the radius r of an undirected graph that satisfies $r \leq \hat{r} \leq 3/2r$. Roditty and Vassilevska W. [33] also showed how to obtain an $\tilde{O}(m\sqrt{n})$ expected time, 3/2-approximation algorithm for the radius.

Aingworth et al.'s paper spawned a long line of research on APSP approximation. Dor, Halperin, and Zwick [18] presented an additive 2-approximation for APSP in unweighted undirected graphs with a running time of $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$, thus improving on Aingworth et al.'s APSP approximation algorithm. Baswana et al. [3] presented an algorithm for unweighted undirected graphs with an expected running time of $O(m^{2/3}n\log n + n^2)$ that computes an approximation of all distances with a multiplicative error of 2 and an additive error of 1. Elkin [22] presented an algorithm for unweighted undirected graphs with a running time of $O(mn^{\rho} + n^2\zeta)$ that approximates the distances with a multiplicative error of $(1+\varepsilon)$ and an additive error that is a function of ζ, ρ and ε. Cohen and Zwick [16] extended the results of [18] to weighted graphs. Baswana and Kavitha [4] presented an $\tilde{O}(m\sqrt{n}+n^2)$ time multiplicative 2-approximation algorithm and an $\tilde{O}(m^{2/3}n+n^2)$ time 7/3-approximation algorithm for APSP in weighted undirected graphs.

[2]Here $\mathcal{O}^*(f(n))$ means $O(f(n)\text{poly}(n))$.

2 Preliminaries

Unless otherwise noted, we consider graphs $G = (V, E)$ with $n = |V|$ and $m = |E|$. Each G is equipped with a nonnegative integer weight function, $w: E \to \mathbb{Z}^+ \cup \{0, \infty\}$. For edge $e = (u, v) \in E$, we will sometimes denote $w(e)$ by $w(u, v)$. The distance between two vertices u and v is denoted $d(u, v)$.

We will denote by G^R the graph obtained by reversing the directions of all edges of G; note, $d_G(v, w) = d_{G^R}(w, v)$.

Define by $\epsilon(v) = \max_{u \in V} d(v, u)$ be the *eccentricity* of vertex v. The *diameter* of a graph is defined as $D = \max_{v \in V} \epsilon(v)$. The *radius* of a graph is defined as $R = \min_{v \in V} \epsilon(v)$.

Let L be a positive integer parameter which will vary in our algorithms. The *L-nearest set* of a vertex v is denoted $N_L(v)$, and consists of the L closest vertices to v. More precisely, $|N_L(v)| = L$ and for all pairs of vertices $x \in N_L(v)$ and $y \notin N_L(v)$, $d(v, x) \leq d(v, y)$. The reverse nearest set of v, denoted by $N_L^R(v)$ is defined similarly as above but includes the L vertices that v is closest to: $\left|N_L^R(v)\right| = L$ and for all pairs of vertices $x \in N_L^R(v)$ and $y \notin N_L^R(v)$, $d(x, v) \leq d(y, v)$.

Let $S \subseteq V$. We denote by $d(v, S)$ the distance from v to its closest vertex in S, i.e. $d(v, S) = \min_{s \in S} d(v, s)$.

Given a distance r, the *ball* of radius r around v is defined as $B_r(v) = \{u : d(v, u) \leq r\}$. Such balls will never be used explicitly by our algorithms, but will be used in our analysis.

Dijkstra search. For $v \in V$, one can calculate $d(v, u)$ and $d(u, v)$ for every $u \in V$ using Dijkstra's algorithm [17] with Fibonacci heaps [25] in G and G^R respectively in $\mathcal{O}(m + n\log n)$ time. Without using sophisticated data structures, Dijkstra's algorithm runs in $\mathcal{O}(m\log n)$ time. We will use the term *Dijkstra search* to indicate running the shortest paths algorithm in *both directions*.

Nearest set computation. Given $v \in V$, we compute $N_L(v)$ using a modified version of Dijkstra's algorithm which runs in $\mathcal{O}\left(L^2\right)$ time. Assuming sorted adjacency lists, it is sufficient to relax at most the first L edges leaving each vertex, and only extract L vertices from the priority queue.

Hitting set selection. Given n sets of size L over a universe of size n, a set S of size $\mathcal{O}(n/L \log n)$ hitting all n sets in at least one element can be found deterministically in $\mathcal{O}(m + nL)$ time using a greedy

set cover algorithm (e.g. [1]).

Balls of radius r and working sets. In our algorithms it may be useful to compute the r-balls around all nodes, however, even for small r, the r-ball around v may be very large. Because of this, we do not compute balls explicitly, but rather concentrate on *working sets*.

Given a set $S \subseteq V$ and an integer r, the r-working set of v with respect to S, denoted by $W_{r,S}(v)$, is formed by adding vertices in order of increasing distance from v until the next vertex to be added either belongs to S or is at a distance greater than r from v. When S is clear from the context, we will just write $W_r(v)$. By construction, we have $W_r(v) \subseteq B_r(v)$. If $|S \cap N_L(v)| \geq 1$, then also $W_r(v) \subset N_L(v)$. In some cases we will construct working sets without radius restriction, and use the notation $W(v) = W_\infty(v)$.

Handling extremal diameter values. Since the edge weights are non-negative numbers or ∞, $D = 0$ if and only if when one considers the subgraph H containing only the edges of weight 0, and H is strongly connected. The other extremal case is when $D = \infty$, and this is if and only if the subgraph composed only of finite weight edges is not strongly connected. Since strong connectivity can be tested in linear time, we can check both these cases in a quick preprocessing phase. We hence concentrate on the case when the diameter is finite and positive.

3 Diameter Approximation

Let us first recall the algorithm of Roditty and Vassilevska W. [33]. The algorithm first picks a random sample S of vertices such that $|S| = \Theta(n/L \log n)$. With high probability, S hits the L-nearest sets $N_L(v)$ for all $v \in V$. Then, the algorithm does a Dijkstra search from every $s \in S$, and finds the furthest node w, i.e. the node maximizing $d(w, S)$. It then does a Dijkstra search from w, and thus explicitly finds $N_L(w)$. Finally, it does a Dijkstra search from each $t \in N_L(w)$ and returns the estimate $\tilde{D} = \max_{v \in V, t \in S \cup N_L(w)} \max\{d(t, v), d(v, t)\}$.

The running time of the algorithm comes from setting $L = \sqrt{n}$.

The correctness proof proceeds as follows. Let $D = 3q + z$ with $z \in \{0, 1, 2\}$, and let a and b be the endpoints of the diameter path so that $d(a, b) = D$. If $d(a, S) \leq q$, then $\tilde{D} \geq d(S, b) \geq 2q + z \geq 2D/3$. Thus assume that $d(a, S) > q$. By the choice of w, then $d(w, S) > q$. Furthermore, if $d(w, b) \geq 2q + z$, then again $\tilde{D} \geq 2q + z \geq 2D/3$. Thus assume $d(w, b) \leq 2q + z - 1$.

Now, since S hits $N_L(w)$ with high probability and since $d(w, S) > q$, we have that there is some node c on the shortest path P from w to b such that $c \in N_L(w)$ and $d(w, c) \leq q$ but the node c' after c on P has $d(w, c') > q$. Thus, $d(w, c) \geq q+1-w(c, c')$. But then, $d(c, b) = d(w, b)-d(w, c) \leq q + z - 2 + w(c, c')$, and $d(a, c) \geq 2q + 2 - w(c, c') \geq 2D/3 - w(c, c')$.

The estimate of the Roditty and Vassilevska W. algorithm thus satisfies $\lceil 2/3 \cdot D \rceil - w(c, c') \leq \tilde{D} \leq D$ for some edge (c, c'). For unweighted graphs this already provides a good approximation. For weighted graphs, however, the approximation can be arbitrarily bad since the weight of the edge (c, c') could be large.

We first provide a simple modification of the algorithm of Roditty and Vassilevska W. to remove the additive term in the approximation guarantee caused by the edge above, providing a genuine 3/2-approximation.

Suppose every vertex in the graph has constant out-degree. Then we could just expand $N_L(w)$ by one level, thus also running a Dijkstra search from all vertices with an inneighbor in $N_L(w)$. This extra computation does not increase the asymptotic running time since the number of edges incident to $N_L(w)$ is $O(|N_L(w)|)$.

Furthermore, this expansion of $N_L(w)$ will guarantee that we run a Dijkstra search from the vertex c' in the above argument, and will thus guarantee that in the problematic case, $\tilde{D} \geq d(a, c') \geq (3q + z) - (2q + z - 1 - (q + 1)) = 2q + 2 \geq \lceil 2/3 \cdot D \rceil$.

On graphs with nonconstant degree however, the above change in the algorithm can increase the running time to $O(mn)$ since $N_L(w)$ may have edges to all vertices, and we would run a Dijkstra search from all vertices. Fortunately, there is a simple transformation we can apply to the graph first to transform it into a bounded-degree graph. The transformation is as follows. Process each vertex one by one. If a node v has total degree $d(v)$ greater than 3, replace v by a directed cycle on $d(v)$ nodes. All cycle edges have weight 0. Associate each (in or out) neighbor u of v with a distinct cycle vertex c_u, and replace each edge (u, v) with (u, c_u) and each (v, u) with (c_u, u).

It is not hard to see that any distance calculation will label every vertex in each expanded cycle with the same distance, and that all distances are preserved by the transformation. The graph will have $\Theta(m)$ vertices and $\Theta(m)$ edges. This will have the effect of inflating the running time of our algorithm from $\tilde{\mathcal{O}}(m\sqrt{n})$ to $\tilde{\mathcal{O}}(m^{3/2})$.

An anonymous reviewer suggested a slight modification to the above approach that still guarantees an $\tilde{\mathcal{O}}(m^{3/2})$ running time, but keeps the number of

vertices in the instance $\mathcal{O}(n)$. Instead of reducing to the case of bounded degree, one applies the above cycle transformation, but instead of replacing each node with a cycle on $d(v)$ nodes, one replaces it with a cycle on $\lceil d(v)/(m/n) \rceil$ nodes, balancing the in- and out-edges so that no new node gets more than $\mathcal{O}(m/n)$ edges. The algorithm is the same as above. The running time becomes $\tilde{\mathcal{O}}(m \cdot (Lm/n + n/L))$ since the number of edges incident on $N_L(w)$ is $\mathcal{O}(Lm/n)$. We set $L = n/\sqrt{m}$ to obtain a running time of $\tilde{\mathcal{O}}(m^{3/2})$.

3.1 A New Approach In this section we describe our $\tilde{\mathcal{O}}(mn^{2/3})$ time algorithm. It relies on a new approach for diameter estimation. In order to give an informal description, we will begin with a crucial lemma.

Recall that for integer r and vertex v, the in- and out- balls of radius r around v, $B_r(v)$ and $B_r^R(v)$, are just the vertices that v is at distance at most r to or from. Given r and vertices x and y we define the *gap-distance* between x and y as

$$g_r(x,y) = \min_{\substack{(s,t)\in E \\ s\in B_r(x) \\ t\in B_r^R(y)}} d(x,s) + w(s,t) + d(t,y).$$

The lemma below says that the distance and gap-distance of two vertices are extremely related.

LEMMA 3.1. *Let $x, y \in V$ and let r be a nonnegative integer. Then $d(x,y) \leq 2r+1$ if and only if $g_r(x,y) \leq 2r+1$.*

Proof. Clearly, if $g_r(x,y) \leq 2r+1$, then as $d(x,y) \leq g_r(x,y)$, we also have $d(x,y) \leq 2r+1$.

Now suppose that $d(x,y) = \ell \leq 2r+1$. Then consider the shortest path P between x and y. If $y \in B_r(x)$, then we already have $g_r(x,y) \leq r \leq 2r+1$. Hence $y \notin B_r(x)$ and hence $\ell > r$.

Now, since $d(x,y) > r$ but $d(x,x) = 0$, P has a *middle* edge $e = (u,v)$ such that $d(x,u) \leq r$ but $d(x,v) > r$.

As $d(x,u) \leq r$, $u \in B_r(x)$. Similarly, $d(x,v) \geq r+1$, so $d(v,y) \leq r$ and $v \in B_r^R(y)$. Therefore,

$$g_r(x,y) \leq d(x,u) + w(u,v) + d(v,y) = \ell \leq 2r+1.$$

□

Intuition behind the algorithm. Recall that $D = 3q + z$ where $z \in \{0,1,2\}$ and a and b are vertices that satisfy $d(a,b) = D$. Suppose q is given to us, a possible strategy to detect a pair of vertices at distance at least $2q+z$ is as follows.

Let L be a parameter. We first compute $N_L(v)$ and $N_L^R(v)$ for all vertices v. We then compute a hitting set S of size $\mathcal{O}(n/L \log n)$ as before, guaranteeing that all $N_L(v)$ and $N_L^R(v)$ sets are hit. Now, since we have q, we compute $W_q(v)$ and $W_q^R(v)$ for all v.

Recall $W_q(v)$ is formed by adding the closest vertices reachable from v before a node of S or a node at distance $> q$ is discovered. $W_q^R(v)$ is the same but when the edges are traversed in the opposite direction.

Now, we can run a Dijkstra search for each node in S. There are two cases. If any of the vertices of S is contained in $B_q(a)$ or $B_q^R(b)$, then we are guaranteed to find distance of at least $2q+z$. Otherwise, by the construction of the working sets, $B_q(a) = W_q(a)$ and $B_q^R(b) = W_q^R(b)$. To handle this case, we can take all pairs of vertices x and y such that $B_q(x) \cap S = \emptyset$ and $B_q^R(y) \cap S = \emptyset$ and compute $g_q(x,y)$. If the maximum gap-distance found is at least $2q+2$, return $2q+2$. (Alternatively, if we take a pair with gap-distance at least $2q+2$ and run Dijkstra for an arbitrary vertex from this pair we will get an actual distance as the desired approximation.)

Suppose that the maximum gap-distance found is $\leq 2q+1$. Then $d(x,y) \leq 2q+1$ for each of the pairs considered and in particular $d(a,b) \leq 2q+1$. However, $d(a,b) = 3q+z$, and hence $q \leq 1-z$. Except in the special cases $D = 0, 1$ and 3 that can be handled separately, this is impossible. Hence, except for some special cases, the value returned will be $2q+2 \geq 2q+z$.

This idea has two problems. The first is that we do not know q ahead of time, and guaranteeing that we could find it exactly in the time we have allotted would imply a contradiction to the Strong Exponential Time Hypothesis [33]. To overcome this problem we will use a binary search technique which, while not always explicitly identifying q, still always guarantees a good estimate for the diameter. The idea behind the binary search is as follows. Suppose we have some guess r for q and the maximum gap-distance computed is at least $2r+2$, then either $2r+2$ is a good estimate already, or we must have $q > r$, so we can try to increase our guess. Otherwise, if the maximum gap-distance is at most $2r+1$, then either the Dijkstra searches have already computed a good estimate, or $q < r$ and we can thus reduce r. A simple way to implement this is to search over the possible values of the diameter, the integers between 1 and $M(n-1)$, however this would result in a $\log Mn$ factor in the running time. We instead do a binary search over a carefully selected set of $\tilde{\mathcal{O}}(n^{1.5})$ integers, and thus obtain only an $\mathcal{O}(\log n)$ overhead.

The second problem is that it is not immediately clear how to compute the gap-distances efficiently,

since we might have to consider many vertex pairs and then to scan all edges between the working sets of each pair. To overcome this issue, we carefully pick more vertices to run a Dijkstra search from until the number of vertex pairs and edges between working sets is small.

The algorithm. Our algorithm works as follows. First, as in the preliminaries, check whether the diameter is 0, and if so, return 0. From now on, assume that $D \geq 1$.

For a parameter L, for every $v \in V$ we compute the set $N_L(v)$ and $N_L^R(v)$. We add all the distinct distances found during this computation to a list denoted $\mathcal{L}$. Add -1 and Mn to this list as well and sort it. The list $\mathcal{L}$ will represent all the values over which we will binary search for q, except for -1 and Mn which we include for technical reasons. Notice that $N = |\mathcal{L}| = \mathcal{O}(nL)$ so that creating $\mathcal{L}$ and sorting it is efficient, and the number of search steps over the binary search is $\mathcal{O}(\log n)$. Furthermore, if $\mathcal{L} = \{-1 = r_0, r_1, \ldots, r_N = Mn\}$, we have that $r_0 < q < r_N$.

We pick a set S that hits all $N_L(v)$ and $N_L^R(v)$ and run Dijkstra for every $v \in S$.

We then start a binary search over $\mathcal{L}$. An iteration of the binary search for a value $r \in \mathcal{L}$ is composed of the following steps. First, for each vertex $v \in V$ we compute $W_r(v)$ and $W_r^R(v)$. For every vertex v either $W_r(v) \subset B_r(v)$, or $B_r(v) = W_r(v)$. In the former case $S \cap B_r(v) \neq \emptyset$ and in the latter case we add v to a set U. U contains all nodes that we have not processed by running Dijkstra on some nearby node. Similarly, we create a set U^R. On the fly we also compute for every vertex u a list $\ell(u)$ (respectively $\ell^R(u)$) of all vertices v such that $u \in W_r(v)$ and $v \in U$ (respectively $u \in W_r^R(v)$ and $v \in U^R$).

Next, our goal is to find all pairs of vertices u and v such that $u \in U$ and $v \in U^R$ and $g_r(u,v) \leq 2r+1$. A straightforward way to do it is by scanning the graph edges and for each edge (x,y) and for each pair of vertices $u \in \ell(x)$ and $v \in \ell^R(y)$, to compute $d(u,x) + w(x,y) + d(y,v)$. If it happens that for all edges (x,y) we have $|\ell(x)| \leq t$ and $|\ell^R(y)| \leq t$ for some small t, then one can find all pairs of $u \in U, v \in U^R$ with small gap-distance in $\mathcal{O}(mt^2)$ time. However, apriori, the lists $\ell(x)$ and $\ell^R(y)$ can have $\Omega(n)$ vertices.

To overcome this problem we have an additional pruning step in which we run a Dijkstra search from each vertex v such that $|\ell(v)| \geq t$ or $|\ell^R(v)| \geq t$. We go through each v for which $|\ell(v)| \geq t$. We remove from U, v and each vertex u such that $u \in \ell(v)$. We remove u from $\ell(w)$ if $w \in W_r(u)$. This changes the sizes of some sets $\ell(\cdot)$. We continue to the next vertex with large $\ell(\cdot)$. The same is done for the reverse sets for which $|\ell^R(v)| \geq t$.

The transformation above can run at most $\mathcal{O}(n/t)$ Dijkstra searches since each Dijkstra search is associated with at least t vertices being removed from U or U^R. The removal of vertices from $\ell(w)$ can be amortized to their insertion, and hence the entire pruning step costs $\tilde{\mathcal{O}}(mn/t)$ operations.

After the transformation is completed, we are guaranteed that every edge connects at most t^2 working sets. Now we can compute gap-distances by scanning the edges in $\mathcal{O}(mt^2)$ time overall. For a vertex $u \in U$ we store all vertices from U^R at gap-distance at most $2r+1$ in a set $C(u)$. Notice that this way we can quickly detect whether there are some $u \in U$, $v \in U^R$ with $g_r(u,v) \geq 2r+2$, just by checking whether $|C(u)| < |U^R|$ for some $u \in U$.

Finally, we are ready to make the decision of how to proceed with the binary search. We increase the value of r if we detect a pair of vertices at gap-distance at least $2r+2$ and decrease it otherwise. In the final stage of the algorithm, we need to do a little bit more work to ensure a good approximation, due to the fact that our binary search is not over all possible values for r. The steps of the algorithm are summarized below.

Algorithm 1

1. Check whether the graph on only the finite edges is strongly connected. If not, return $\tilde{D} = \infty$, together with two vertices in different strongly connected components.
2. Check whether the graph on only the 0 edges is strongly connected. If so, return $\tilde{D} = 0$, together with two arbitrary vertices.
3. Otherwise, initialize $\tilde{D}$ to equal the minimum positive edge weight. (Hence $\tilde{D} \geq 1$.) Let $\tilde{a}$ and $\tilde{b}$ be arbitrary vertices with $w\left(\tilde{a}, \tilde{b}\right) = \tilde{D}$.
4. Compute $N_L(v)$ and $N_L^R(v)$ for each v. Create a list $\mathcal{L}$ of all distinct distances found during the nearest set computations, together with -1 and Mn. Sort $\mathcal{L}$ in increasing order: $\mathcal{L} = \{-1 = r_0, r_1, \ldots, r_N = Mn\}$.
5. Pick S hitting all $N_L(v)$ and $N_L^R(v)$.
6. Run a Dijkstra search from each vertex $v \in S$. Update $\tilde{D}$ to be the largest distance found in any of the searches so far, and let $\tilde{a}$ and $\tilde{b}$ be the corresponding vertices such that $\tilde{D} = d\left(\tilde{a}, \tilde{b}\right)$.
7. Initialize $i = 0$, $k = N$, and $j = \left\lceil \frac{i+k}{2} \right\rceil$.
8. Repeat while $k > i + 1$:

(a) Set $r = r_j$.

(b) Initialize sets $U = U^R = \emptyset$. For each $v \in V$, initialize lists $\ell(v)$ and $\ell^R(v)$ to be empty.

(c) For each vertex $v \in V$, build $W_r(v)$ and $W_r^R(v)$. If $B_r(v) = W_r(v)$, add v to U, and for each $u \in W_r(v)$ add v to $\ell(u)$. Similarly if $B_r^R(v) = W_r^R(v)$, add v to U^R and for each $u \in W_r^R(v)$ add v to $\ell^R(u)$.

(d) For each vertex $v \in U$, if $|\ell(v)| > t$, do the following:

i. Run a Dijkstra search from v. Update $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$ appropriately.

ii. For each $u \in \ell(v)$, if $u \in U$, remove u from U. For every $w \in W_r(u)$ remove u from $\ell(w)$.

(e) For each vertex $v \in U^R$, if $\left|\ell^R(v)\right| > t$, do the following:

i. Run a Dijkstra search from v. Update $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$ appropriately.

ii. For each $u \in \ell^R(v)$, if $u \in U^R$, remove u from U^R. For every $w \in W_r(u)$ remove u from $\ell^R(w)$.

(f) For each $v \in U$, initialize a dictionary $C(v)$ to contain v if $v \in U^R$ and to be empty otherwise.

(g) For each $(x, y) \in E$ do the following. For each $u \in \ell(x)$ and $v \in \ell^R(y)$, compute $d(u, x) + w(x, y) + d(y, v)$. If it is at most $2r + 1$ and $v \notin C(u)$ add v to $C(u)$.

(h) If there exists a vertex $v \in U$ such that $U^R \setminus C(v) \neq \emptyset$ (i.e. if $|C(v)| < |U^R|$ for some v), set $i = j$, run a Dijkstra search from v, and update $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$ appropriately. Otherwise set $k = j$.

(i) Set $j = \left\lceil \frac{i+k}{2} \right\rceil$.

9. Run steps 8b through 8g for $r = r_k - 1$. If there exists a vertex $u \in U$ such that $U^R \setminus C(u) \neq \emptyset$, then run a Dijkstra search from u and update $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$. Otherwise, let $g' = \max_{x \in U, y \in U^R} g_{r_k - 1}(x, y)$ and let u be the node such that for some u, $g_r(u, v) = g'$. Set $r' = \lfloor g'/2 \rfloor - 1$ and if $2r' + 2 > \tilde{D}$ run a Dijkstra search from u and update $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$.

10. Return $\left(\tilde{D}, \tilde{a}, \tilde{b}\right)$.

Before proving the correctness, let us briefly discuss the running time. Since we do not want to use sophisticated data structures, we will assume that each Dijkstra search takes $\mathcal{O}(m \log n)$ time. (Using Fibonacci heaps in Dijkstra's algorithm one can improve the asymptotic running time by a logarithmic factor.)

Computing nearest sets takes $\mathcal{O}\left(nL^2\right)$ time. Computing the Dijkstra searches for the hitting set S takes $\mathcal{O}\left(mn/L \log^2 n\right)$ time. The additional Dijkstra searches and gap-distance computation over all iterations of the loop take $\mathcal{O}\left(mn/t \log^2 n + mt^2 \log n\right)$ time: as argued earlier, each of the $\mathcal{O}(\log n)$ iterations calls at most $\mathcal{O}(n/t)$ searches, and the remaining gap-distance computation takes $\mathcal{O}\left(mt^2\right)$ time. Setting $L = m^{1/3} \log^{2/3} n$ and $t = (n \log n)^{1/3}$ we get a runtime of $\mathcal{O}\left(mn^{2/3} \log^{5/3} n\right)$.

THEOREM 3.1. *Algorithm 1 returns an estimate $\tilde{D}$ to the diameter such that $\lceil 2/3 \cdot D \rceil \leq \tilde{D} \leq D$.*

Proof. In the special case when $D = 3$, notice that the algorithm runs at least one Dijkstra search from a node v. The longest distance found from or to v must be at least $\lceil D/2 \rceil = 2$ as otherwise the diameter would be 2.

The algorithm checks whether the special case $D = 0$ happens, and if it doesn't, it always returns a positive integer estimate, via step 3. Thus, the special case $D = 1$ is covered as well. From now on we can assume that $q > (1 - z)$, hence $2q + 2 \leq 3q + z = D$, and in particular $D \geq 2$.

Let a and b be the endpoints of a diameter path such that $d(a, b) = D$.

We consider an iteration of the binary search and analyze the behavior of the algorithm for the different values of $r = r_j$.

1. $r \leq q$.

(a) Suppose for all pairs of vertices x and y such that $d(x, y) \geq 2r + 2$, either $x \notin U$ or $y \notin U^R$. In particular, we have done a Dijkstra search from either $u \in B_r(a)$ or $v \in B_r^R(b)$, and will set $\tilde{D}$ to a value $\geq 2q + z$. Notice that in this case we set $k = j$.

(b) There exists some pair of vertices x and y such that $d(x, y) \geq 2r + 2$ and both $x \in U$ and $y \in U^R$. Thus we set $i = j$ and search for a larger value of r. We also run a Dijkstra search from an arbitrary vertex that belongs to some pair. Notice that if we happen to have $r = q$ we will set $\tilde{D}$ to a value $\geq 2q + 2 \geq 2q + z$.

2. $q < r$.

(a) Suppose for all pairs of vertices x and y such that $d(x,y) \geq 2r+2$, either $x \notin U$ or $y \notin U^R$. Thus we set $k=j$ and search for a smaller value of r.

(b) There exists some pair of vertices x and y such that $d(x,y) \geq 2r+2$ and both $x \in U$ and $y \in U^R$. Thus we run a Dijkstra search from such a vertex x and obtain an estimate $\tilde{D} \geq 2r+2 \geq 2q+z$. Furthermore we update $i=j$.

Now consider the value of r_i as the algorithm progresses. From the manner in which we construct $\mathcal{L}$, it is clear that either the algorithm returns early, or $r_0 = -1 < q$ and $r_N = Mn > q$. When it is initialized, $r_i = -1 < q$ and r_i never decreases and increases at least once since the distance 0 is in $\mathcal{L}$ and $q \geq 0$ and $D \geq 2q+2$. Whenever r_i increases, it is to the value currently held by $r = r_j$. Similarly whenever r_k decreases, it is to the value held by $r = r_j$. Upon termination of the loop, $j = k = i+1$.

First, suppose that at some point in the execution of the algorithm $r_i \geq q$. Then, at some previous point r_j was $\geq q$ and after that iteration, i was set to j. But this means, that either Case 2b occurred and hence the estimate is at least $2r_i + 2 \geq 2q+2 \geq 2q+z$, or Case 1b occurred and $r_j = q$ which also means the returned estimate is at least $2q+z$.

Thus, let's assume that the algorithm terminates with $r_i < q$. Then, either some iteration falls into Case 1a and we are guaranteed a good estimate, or *every* iteration satisfying Case 1 fell into Case 1b. Suppose the latter is true and consider the final iteration of the loop. First, it is possible that $r_j = r_k = q$ and again we are guaranteed a good estimate by Case 1b. Alternatively, it could be that $r_k < q$. Then during a previous iteration, k was updated to have the value of j in that iteration satisfying $r_j < q$, which implies that Case 1a was satisfied in that iteration, a contradiction to our assumption that this case was never satisfied. The remaining option is that $r_k > q$.

Consider the end of the loop where $k = i+1$ and $r_i < q < r_k = r_{i+1}$. First notice that $i > 0$ since if $i = 0$, then $k = 1$, and hence $r_k = r_1 = 0$ (as 0 is in $\mathcal{L}$), giving a contradiction to $r_k > q \geq 0$. Since $i > 0$, we must have had an iteration where r_j had the value r_i. For that iteration there are some x, y with $g_{r_i}(x,y) \geq 2r_i + 2$ since i was set at Case 1b.

Let $\hat{r}$ be any integer that satisfies $r_i \leq \hat{r} < r_{i+1}$. Then $W_{\hat{r}}(x) = W_{r_i}(x)$ for every $x \in V$ since for all edges (u,v) with $u \in W_{r_i}(x), v \notin W_{r_i}(x)$, it holds that $d(x,u) + w(u,v) \geq r_{i+1}$ as otherwise there was another value in $\mathcal{L}$ between r_i and r_{i+1}. Thus, for every $x, y \in V$ and $r_i \leq \hat{r} < r_{i+1}$ we have $g_{r_i}(x,y) = g_{\hat{r}}(x,y)$.

Recall that a and b are endpoints of the diameter. Notice that either we have a Dijkstra search from a vertex s such that $d(a,s) \leq q$ or $d(s,b) \leq q$, or we have that $d(a,s), d(s,b) > q$ for all nodes s that we have run Dijkstra through. By our assumption that Case 1a was never satisfied, this means that in every iteration with $r_j \leq q$ we have $a \in U$ and $b \in U^R$ and since $i > 0$ we have at least one such iteration.

Consider the iteration with $r = r_k - 1$ initiated at step 9 and the last iteration with $r = r_i$. As $W_{r_i}(u) = W_{r_k-1}(u)$ for every $u \in V$, it follows that the sets U and U^R of both iterations are the same (they start the same and we can also prune them in the same order). Thus, we have $a \in U$ and $b \in U^R$ in the iteration $r = r_k - 1$. Now consider the outcome of this iteration. There are two possible cases. The first case is that there exists a vertex $u \in U$ such that $U^R \setminus C(u) \neq \emptyset$. Let $v \in U^R \setminus C(u)$. By the way $C(u)$ is defined it follows that $g_{r_k-1}(u,v) \geq 2(r_k-1)+2$. By Lemma 3.1 it follows that $d(u,v) \geq 2(r_k-1)+2$. As $r_k - 1 \geq q$ we get that $g_{r_k-1}(u,v) \geq 2q+2$, thus $d(u,v) \geq 2q+2$ and since we run Dijkstra from u we get the required estimate.

We now turn to the second case which is a bit more subtle to analyze. In this case there is no vertex $u \in U$ such that $U^R \setminus C(u) \neq \emptyset$, thus, $g_{r_k-1}(u,v) \leq 2(r_k-1)+1$ for every $u \in U$ and $v \in U^R$. Let $g' = \max_{x \in U, y \in U^R} g_{r_k-1}(x,y)$ and let $r' = \lfloor g'/2 \rfloor - 1$. So $2r'+2 \leq g' \leq 2r'+3$.

Consider a pair u, v such that $g_{r_k-1}(u,v) = g' \geq 2r'+2$. By Lemma 3.1 it follows that $d(u,v) \geq 2r'+2$ so if $r' \geq r_k - 1 \geq q$ then we have the required estimate as we run Dijkstra from u. So we can assume that $r' < r_k - 1$. Now recall that for every $x, y \in V$ and $r_i \leq \hat{r} < r_{i+1}$ we have $g_{r_i}(x,y) = g_{\hat{r}}(x,y)$. In particular, since $g_{r_k-1}(x,y) = g_{r_i}(x,y)$ and since in the iteration for $r = r_i$, r_i was set as a lower bound, we must have that $g_{r_i}(x,y) \geq 2r_i + 2$ for some x, y and hence $g' \geq 2r_i + 2$. This implies that $r' \geq r_i$ and since $r' < r_k - 1$ we can conclude that $g_{r_k-1}(x,y) = g_{r'}(x,y) = g_{r'+1}(x,y)$ for every $x \in U$ and $y \in U^R$.

Consider now a and b. As $a \in U$ and $b \in U^R$, we have $g_{r'+1}(a,b) \leq 2r'+3 = 2(r'+1)+1$. This follows from the fact that $g_{r'+1}(a,b) = g_{r'}(a,b) \leq g' \leq 2r'+3$. Using Lemma 3.1 we get that $d(a,b) \leq 2r'+3$. Combining this with the fact that $d(u,v) \geq 2r'+2$ we get that $d(u,v) \geq D-1$. We also know that $r' \geq r_i \geq 0$, thus, $d(u,v) \geq 2$. So for $D = 2$ we have the exact diameter and for $D > 2$ we have the desired estimate.

Thus we have proven that the algorithm returns

an estimate $\tilde{D} \geq 2q + z \geq \lceil 2/3 \cdot D \rceil$ as well as two vertices $\tilde{a}$ and $\tilde{b}$ such that $\tilde{D} = d\left(\tilde{a}, \tilde{b}\right)$. □

4 Additive Diameter Approximation

In this section we consider additive approximations to the diameter. We show that a simple approximation algorithm is essentially tight, assuming a well-known hypothesis about the complexity of CNF Satisfiability.

Consider a graph G on n vertices and m edges. Given an arbitrary constant $\varepsilon > 0$, consider the following almost trivial algorithm: sample $c \cdot n^{1-\varepsilon} \log n$ nodes S uniformly at random. Run Dijkstra's algorithm from and to each vertex from S. Return the maximum distance found.

PROPOSITION 4.1. *The above algorithm runs in $\tilde{O}(mn^{1-\varepsilon})$ time and returns an estimate $\tilde{D}$ such that with high probability, $D - n^{\varepsilon} \leq \tilde{D} \leq D$.*

Proof. The running time is clear. Let us discuss the approximation guarantee.

Let $\tilde{D} = \max_{s \in S, v \in V} \max\{d(s,v), d(v,s)\}$ be the estimate that the algorithm returns. Let a and b be the end points of the diameter path.

Consider first the case when the diameter D is at most $2n^{\varepsilon}$. Then, consider an arbitrary vertex $s \in S$. $D \leq d(a,s) + d(s,b) \leq 2\tilde{D}$, and hence $D/2 \leq \tilde{D} \leq D$. However, $D/2 = D - D/2 \geq D - n^{\varepsilon}$, and hence $D - n^{\varepsilon} \leq \tilde{D} \leq D$.

Now suppose that $D > 2n^{\varepsilon}$. Then with high probability there is a node $s \in S$ such that $d(a,s) \leq n^{\varepsilon}$. Thus, $d(s,b) \geq D - d(a,s) \geq D - n^{\varepsilon}$, and hence again $D - n^{\varepsilon} \leq \tilde{D} \leq D$.

A popular conjecture, the *Strong Exponential Time Hypothesis* (SETH) [28, 29] states that for every $\gamma > 0$ there exists an integer k such that no algorithm running in time $2^{(1-\gamma)n}$ solves k-SAT in the worst case when n is the number of variables. We now show that the above simple algorithm is almost tight for sparse graphs, assuming the SETH.

THEOREM 4.1. *Assuming the SETH, approximating the diameter of a graph with N edges to within an additive error of N^{δ} requires time $\Omega(N^{2-\epsilon})$ for all $\epsilon > 2\delta$.*

The reduction is similar in spirit to the reduction of Roditty and Vassilevska W. [33] but requires a nontrivial change in order to be able to handle additive errors.

Our construction proceeds in two phases. In the first, we create a graph G^{φ} for which the diameter is 3 if φ is unsatisfiable and 4 if it is satisfiable. Then, for any integer ℓ, we construct a graph G^{φ}_{ℓ} by subdividing the edges of G^{φ}. G^{φ}_{ℓ} has the property that if φ is unsatisfiable, then G^{φ}_{ℓ} has diameter $3(\ell+1)$ and if φ is satisfiable, then G^{φ}_{ℓ} has diameter $4(\ell+1)$.

Notice that if we were only after a lower bound for multiplicative approximation as in [33], we would only prove a lower bound for $(4/3-\varepsilon)$-approximations which is weaker than the Roditty-Vassilevska W. result. However, this seemingly weaker construction shows that no additive ℓ-approximation is doable in truly subquadratic time, assuming the SETH. In the additive regime, the previous construction only applied to additive 1-approximations.

Construction of G^{φ} Given k-SAT formula φ with m clauses C and n variables D we partition the variables into two sets D_1 and D_2, each of size $n/2$. Let X_1 be the $2^{n/2}$ partial assignments on the variables in D_1 and let X_2 be the $2^{n/2}$ partial assignments on the variables in D_2.

We construct the *bipartite* graph $G^{\varphi} = (V \cup U, E)$ on partitions U and V as follows:

$V = C \cup \{a_1\} \cup \{a_2\}$, i.e. the clauses plus two additional vertices.

$U = X_1 \cup X_2 \cup \{x_*\}$, i.e. the partial assignments plus an additional variable.

E contains the following edges:

- (c, x): for all $c \in C$, and $x \in X_1 \cup X_2$ where the partial assignment x is consistent with the clause c being violated. (That is, x does not satisfy any of the literals in c).
- (a_i, x); for all $i \in \{1, 2\}$ and $x \in X_i$.
- (v, x_*): for all $v \in V$.

LEMMA 4.1. *For all $v, v' \in V$, v and v' share a common neighbor. φ is unsatisfiable if and only if for all $u, u' \in U$, u and u' share a common neighbor.*

Proof. Let $v, v' \in V$. Then they share the neighbor x_*.

Consider $u, u' \in U$. Suppose that $u \in X_i \cup \{x_*\}$. If $u' \in X_i \cup \{x_*\}$ as well, then a_i is a common neighbor of u and u'. Hence assume that $u \in X_i$ and $v \in X_j$ for $j \neq i$. Then we'll show that u and u' have a common neighbor if and only if u, u' corresponds to an unsatisfying assignment: If u and u' do not satisfy φ then there exists a clause $c \in C$ that neither u nor u' satisfy. This clause will be a common neighbor. If u and u' satisfy φ, then they satisfy every clause. Thus for every clause $c \in C$ either u or u' set a literal in the clause to true and so is *not* connected to this clause c in the graph. However, the only possible common neighbors correspond to clauses in C, and thus u and u' have no common neighbor.

Construction of G_ℓ^φ We define a *ℓ-subdivision* of an arbitrary graph as follows.

DEFINITION 1. (ℓ-SUBDIVISION G_ℓ OF G) *Given a graph $G = (V, E)$, its ℓ-subdivision $G_\ell(V_\ell, E_\ell)$ is defined as follows:*

$V_\ell = V \cup (E \times \{1, \ldots, \ell\})$

Assume that each $e \in E$ has some arbitrarily chosen "direction" (i, j). Then $E_\ell = \{(i, e_1), (e_1, e_2), \ldots, (e_{\ell-1}, e_\ell), (e_\ell, j) : e = (i, j) \in E\}$.

We say that a vertex $v \in G_\ell$ is an original *vertex if it is also a vertex in G.*

LEMMA 4.2. *If φ is unsatisfiable, then G_ℓ^φ has diameter at most $3(\ell+1)$, and if φ is satisfiable, then G_ℓ^φ has diameter at least $4(\ell+1)$.*

Proof. Assume that φ is unsatisfiable. Let v, v' be two arbitrary vertices in G_ℓ^φ. Consider how far v and v' are away from vertices from the original graph G^φ. Without loss of generality assume that v is closer and is distance $r \leq \frac{\ell+1}{2}$ away from some original vertex $\bar{v}$. (If v is an original vertex, then $\bar{v} = v$ and $r = 0$.) Now v' is on an edge e' from G^φ (if v' is also an original vertex and is on several edges, choose one arbitrarily). Let $\bar{v'}$ be the end point of e' that is in the same partition as $\bar{v}$, i.e. either both $\bar{v}$ and $\bar{v'}$ are in U, or both are in V.

Now $d(\bar{v'}, v') \leq (\ell+1) - r$ because otherwise v' would be closer to an original vertex than v. Also $d(\bar{v}, \bar{v'}) \leq 2(\ell+1)$ because $\bar{v}$ and $\bar{v'}$ share a common neighbor in G^φ by Lemma 4.1. Thus there is a path from v to v' of length at most $d(v, \bar{v}) + d(\bar{v}, \bar{v'}) + d(\bar{v'}, v') \leq r + 2(\ell+1) + (\ell+1) - r = 3(\ell+1)$.

Assume that φ is satisfiable. Every simple path connecting original vertices in G_ℓ^φ induces a path in G^φ. Let $u \in X_1, u' \in X_2$ be two vertices corresponding to a satisfying assignment. By Lemma 4.1, u and u' are on the same side of the partition in G^φ but have no common neighbor. Thus the distance between u and u' in G^φ is at least 4, and so any path connecting them in G_ℓ^φ has distance at least $4(\ell+1)$.

Proof. [Proof of Theorem 4.1] For the sake of contradiction, let $\epsilon > 2\delta$ such that there exists an $N^{2-\epsilon}$ time algorithm which approximates the diameter of a graph with N edges to within additive error N^δ.

Let

$$\gamma = \frac{2 - (1 + \frac{\delta}{1-\delta})(2-\epsilon)}{5}.$$

Note that $\gamma > 0$ because $(1 + \frac{\delta}{1-\delta})(2-\epsilon) < (1 + \frac{\delta}{1-\delta})(2-2\delta) = 2$. Now by SETH, there exists a k such that solving k-SAT requires time $2^{(1-\gamma)n}$. We will use our reduction and the assumed algorithm to violate this.

Given a k-SAT instance φ on m clauses and n variables, let $\ell = (4m2^{n/2})^{\frac{\delta}{1-\delta}}$, and create the graph G_ℓ^φ. The number of edges N is at most $2\ell m 2^{n/2} + 2 + 2n^{n/2} < 4\ell m 2^{n/2}$. Thus the error our algorithm makes is at most

$$N^\delta < (4m\ell 2^{n/2})^\delta = (4m(4m2^{n/2})^{\frac{\delta}{1-\delta}} 2^{n/2})^\delta$$

$$= (4m2^{n/2})^{\frac{\delta}{1-\delta}} = \ell.$$

Thus computing the diameter within an additive error of N^δ will solve the k-SAT instance φ.

The time it will take to solve the instance is $N^{2-\epsilon}$ as creating the graph G_ℓ^φ takes only $\mathcal{O}(N)$ time. However,

$$N^{2-\epsilon} = (4\ell m 2^{n/2})^{2-\epsilon} = (4m(4m2^{n/2})^{\frac{\delta}{1-\delta}})2^{n/2})^{2-\epsilon}$$

$$= (4m2^{n/2})^{(1+\frac{\delta}{1-\delta})(2-\epsilon)} \leq (4m2^{n/2})^{2-5\gamma}.$$

Now we know that $m \leq (2n)^k$ (otherwise we can remove the duplicate clauses before the reduction) and so for large enough n, we have that $(4m)^2 < 2^{\gamma n}$. Thus, $(4m2^{\frac{n}{2}})^{2-5\gamma} < 2^{(1-2\gamma)n}$ giving us our contradiction.

5 Approximating the Eccentricities

In this section we give a 5/3-approximation to the eccentricities of all vertices in a given undirected weighted graph. The algorithm runs in $\tilde{\mathcal{O}}(m^{3/2})$ time. It is presented below.

Algorithm 2

1. Transform the graph to have bounded degree (as in Section 3). $G = (V, E)$ is the new graph.
2. Sort the adjacency lists in ascending order of weight.
3. Let $L = \sqrt{m \log m}$. For all $v \in V$, compute $N_L(v)$.
4. Pick a set $S \subset V$ of size $O((m/L)\log m)$ hitting $N_L(v)$ for all $v \in V$.
5. Run a Dijkstra search from every $s \in S$.
6. Determine the farthest vertex from S, and call it w.
7. Run a Dijkstra search from w.
8. $T := \{w\} \cup \{x \in V \mid \exists y \in N_L(w) \wedge (y, x) \in E\}$. Notice that $N_L(w) \subseteq T$.
9. Run a Dijkstra search from each vertex in T.

10. Set $\epsilon'(v) := \epsilon(v)$ for each $v \in T \cup S$.

11. For each $v \notin T \cup S$, set $\epsilon'(v) = \max_{u \in T \cup S} \{\max(d(u,v), \epsilon(u) - d(u,v))\}$.

12. Return the list of $\epsilon'(v)$ values.

THEOREM 5.1. *Algorithm 2 returns an estimate $\epsilon'(v)$ to the eccentricity $\epsilon(v)$ for each vertex v such that $3\epsilon(v)/5 \leq \epsilon'(v) \leq \epsilon(v)$.*

Proof. First, note that in an undirected graph, for any $x, y \in V$ we have that $\epsilon(x) \geq \epsilon(y) - d(x,y)$. Indeed, let $\epsilon(x) = d(x,t_x)$ and $\epsilon(y) = d(y,t_y)$. Then $d(x,t_x) \geq d(x,t_y) + d(x,y) - d(x,y) \geq d(y,t_y) - d(x,y)$.

Because of this, for any $v \in V$, the estimate $\epsilon'(v)$ returned is always at most $\epsilon(v)$: either it is a real distance to v, or it is $\epsilon(u) - d(u,v)$ for some u.

It remains to show that for any $v \in V$, $\epsilon'(v) \geq 3\epsilon(v)/5$. Consider any $u \in T \cup S$. From the argument above, we have that $\epsilon(u) \geq \epsilon(v) - d(u,v)$, and so $\epsilon'(v) \geq \epsilon(u) - d(u,v) \geq \epsilon(v) - 2d(u,v)$. We will show that either for some $u \in T \cup S$, $d(u,v) \geq 3\epsilon(v)/5$, or for some $u \in T \cup S$, $d(u,v) < \epsilon(v)/5$ so that $\epsilon(v) - 2d(u,v) \geq 3\epsilon(v)/5$, and thus the estimate is always good.

First, if $d(w,v) \geq 3\epsilon(v)/5$, then the estimate is already good, so assume that $d(w,v) < 3\epsilon(v)/5$.

Let t be the farthest vertex from v. If we run a Dijkstra search from a vertex in $B_{2\epsilon(v)/5}(t)$, then we are guaranteed a good estimate, so assume otherwise. This means that $d(w,S) \geq d(t,S) > 2\epsilon(v)/5$ since w is the farthest vertex from S. Since S hits $N_L(w)$, we also have that $N_L(w)$ contains all vertices of distance $\leq 2\epsilon(v)/5$ from w.

Consider the shortest path P from w to v. Let x be the last vertex of $N_L(w)$ on P, and let w' be the vertex after x on P. Since $v \notin N_L(w)$, w' must exist. Since $N_L(w)$ contains all vertices of distance at most $2\epsilon(v)/5$ from w, we must have $d(w,w') > 2\epsilon(v)/5$, and since $d(w,v) < 3\epsilon(v)/5$, we must have $d(w',v) < \epsilon(v)/5$. Since $w' \in T$, we have run a Dijkstra search from w' and $\epsilon'(v) \geq \epsilon(v) - 2d(w',v) \geq 3\epsilon(v)/5$, completing the proof. □

THEOREM 5.2. *Algorithm 2 runs in time $\mathcal{O}\left((m \log n)^{3/2}\right)$.*

Proof. Because we make the graph have constant degree, computing $N_L(v)$ for all $v \in V$ takes $O(mL)$ time. The Dijkstra searches take $\mathcal{O}(((m/L)\log m + L)m \log m) = \mathcal{O}\left((m \log m)^{3/2}\right)$ time.

Acknowledgments. This paper was written while the last author was a research associate at Stanford and a research engineer at UC Berkeley supported by NSF Grants CCF-0830797, CCF-1118083, IIS-0963478 and IIS-0904325, and an AFOSR MURI Grant. The third author is supported by ISF grant grant no. 822/10. The second and fifth authors' work at Princeton University was partially supported by NSF grant CCF-0832797.

References

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

[2] R. Albert, H. Jeong, and A.L. Barabasi. Diameter of the world wide web. *Nature*, 401:130 – 131, 1999.

[3] S. Baswana, V. Goyal, and S. Sen. All-pairs nearly 2-approximate shortest paths in $o(n^2 \text{poly} \log n)$ time. *Theor. Comput. Sci.*, 410(1):84–93, 2009.

[4] S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM J. Comput.*, 39(7):2865–2896, 2010.

[5] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (alpha, beta)-spanners and purely additive spanners. In *SODA*, pages 672–681, 2005.

[6] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Transactions on Algorithms*, 7(1):5, 2010.

[7] 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.

[8] P. Berman and S. P. Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. WADS*, pages 541–552, 2007.

[9] K. Boitmanis, K. Freivalds, P. Ledins, and R. Opmanis. Fast and simple approximation of the diameter and radius of a graph. In *WEA*, pages 98–108, 2006.

[10] Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discrete Math.*, 19(4):1029–1055, 2005.

[11] Shiri Chechik. New additive spanners. In *SODA*, pages 498–512, 2013.

[12] V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *Proc. SODA*, pages 346–355, 2002.

[13] 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.

[14] Victor Chepoi, Feodor F. Dragan, Bertrand Estellon, Michel Habib, and Yann Vaxès. Diameters, centers, and approximating trees of delta-

hyperbolicgeodesic spaces and graphs. In *Symposium on Computational Geometry*, pages 59–68, 2008.

[15] F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congr. Numer.*, 60:295–317, 1987.

[16] E. Cohen and U. Zwick. All-pairs small-stretch paths. *J. Algorithms*, 38(2):335–353, 2001.

[17] E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, pages 269–271, 1959.

[18] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

[19] Feodor F. Dragan. Almost diameter of a house-hole-free graph in linear time via lexbfs. *Discrete Applied Mathematics*, 95(1-3):223–239, 1999.

[20] Feodor F. Dragan and Falk Nicolai. Lexbfs-orderings of distance-hereditary graphs with application to the diametral pair problem. *Discrete Applied Mathematics*, 98(3):191–207, 2000.

[21] D. Dvir and G. Handler. The absolute center of a network. *Networks*, 43:109 – 118, 2004.

[22] M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.

[23] Michael Elkin and David Peleg. (1+epsilon, beta)-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004.

[24] Arthur M. Farley and Andrzej Proskurowski. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics*, 2:185–191, 1980.

[25] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[26] Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for geometric intersection graphs. *CoRR*, abs/cs/0605029, 2006.

[27] S.L. Hakimi. Optimum location of switching centers and absolute centers and medians of a graph. *Oper. Res.*, 12:450 – 459, 1964.

[28] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

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

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

[31] Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009.

[32] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*, pages 261–272, 2005.

[33] 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.

[34] Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *SODA*, pages 802–809, 2006.

[35] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[36] O. Weimann and R. Yuster. Approximating the diameter of planar graphs in near linear time. In *Proc. ICALP*, 2013.

[37] David P. Woodruff. Lower bounds for additive spanners, emulators, and more. In *FOCS*, pages 389–398, 2006.

[38] David P. Woodruff. Additive spanners in nearly quadratic time. In *ICALP (1)*, pages 463–474, 2010.

[39] C. Wulff-Nilsen. Wiener index, diameter, and stretch factor of a weighted planar graph in subquadratic time. *Technical report, University of Copenhagen*, 2008.