arXiv:1207.3622v1 [cs.DS] 16 Jul 2012

Approximating the diameter of a graph

Liam Roditty[*] and Virginia Vassilevska Williams[†]

Abstract

In this paper we consider the fundamental problem of approximating the diameter D of directed or undirected graphs. In a seminal paper, Aingworth, Chekuri, Indyk and Motwani [SIAM J. Comput. 1999] presented an algorithm that computes in $\widetilde{O}(m\sqrt{n}+n^2)$ time an estimate $\hat{D}$ for the diameter of an n-node, m-edge graph, such that $\lfloor 2/3D \rfloor \leq \hat{D} \leq D$. In this paper we present an algorithm that produces the same estimate in $\widetilde{O}(m\sqrt{n})$ expected running time. We then provide strong evidence that a better approximation may be hard to obtain if we insist on an $O(m^{2-\varepsilon})$ running time. In particular, we show that if there is some constant $\varepsilon > 0$ so that there is an algorithm for undirected unweighted graphs that runs in $O(m^{2-\varepsilon})$ time and produces an approximation $\hat{D}$ such that $(2/3+\varepsilon)D \leq \hat{D} \leq D$, then SAT for CNF formulas on n variables can be solved in $O^*((2-\delta)^n)$ time for some constant $\delta > 0$, and the strong exponential time hypothesis of [Impagliazzo, Paturi, Zane JCSS'01] is false.

Motivated by this somewhat negative result, we study whether it is possible to obtain a better approximation for specific cases. For unweighted directed or undirected graphs, we show that if $D = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$, then it is possible to report in $\tilde{O}(\min\{m^{2/3}n^{4/3}, m^{2-1/(2h+3)}\})$ time an estimate $\hat{D}$ such that $2h + z \leq \hat{D} \leq D$, thus giving a better than $3/2$ approximation whenever $z \neq 0$. This is significant for constant values of D which is exactly when the diameter approximation problem is hardest to solve. For the case of unweighted undirected graphs we present an $\tilde{O}(m^{2/3}n^{4/3})$ time algorithm that reports an estimate $\hat{D}$ such that $\lfloor 4D/5 \rfloor \leq \hat{D} \leq D$.

[*]Bar Ilan University, liamr@cs.biu.ac.il. Work supported by the Israel Science Foundation (grant no. 822/10)

[†]UC Berkeley and Stanford University, virgi@eecs.berkeley.edu. Partially supported by NSF Grants CCF-0830797 and CCF-1118083 at UC Berkeley, and by NSF Grants IIS-0963478 and IIS-0904325, and an AFOSR MURI Grant, at Stanford University.

1 Introduction

The diameter of a graph is the longest of all distances between vertices in the graph. The diameter is a natural and fundamental graph parameter, and computing it efficiently has many applications (e.g. [3]). Essentially, the only known way to determine the diameter of a graph with arbitrary edge weights is to compute the distances between all pairs of vertices in the graph, that is, to solve the all-pairs shortest paths problem (APSP), and then to find the maximum distance. Because of this, some researchers have conjectured that APSP and diameter in weighted graphs may be equivalent in some sense (e.g. [21] and [5]). The fastest algorithms for computing APSP and hence for computing the diameter for directed or undirected graphs on n nodes and m edges with arbitrary edge weights and no negative cycles have a running time of $O(\min\{n^3 \log\log^3 n/\log^2 n, mn + n^2 \log\log n\})$ [4, 16].

For the special case of dense directed or undirected unweighted graphs, one can compute the diameter by reducing its computation to fast matrix multiplication, thus obtaining $\tilde{O}(n^\omega)$ time algorithms, where $\omega < 2.38$ is the matrix multiplication exponent [6, 19, 20]. In fact, any known algorithm for diameter in dense n-node unweighted graphs running in $T(n)$ time can also be used to compute the Boolean product of two $n \times n$ Boolean matrices in $O(T(n))$ time. This lead to conjectures [5, 1] that computing the diameter in dense unweighted graphs and Boolean matrix multiplication (BMM) may be equivalent.

For the special case of sparse directed or undirected unweighted graphs, the best known algorithm for both APSP and diameter does breadth-first search (BFS) from every node and hence runs in $O(mn)$ time. For sparse graphs with $m = O(n)$, the running time is $\Theta(n^2)$ which is natural for APSP since the algorithm needs to output n^2 distances. However, for the diameter the output is a single integer, so it is not immediately clear why one should spend $\Omega(n^2)$ time to compute it. In this paper, we show somewhat surprisingly, that breaking this seeming n^2 barrier would have major consequences for the complexity of NP-hard problems such as SAT.

A natural question is whether one can get substantially faster algorithms for the diameter by settling for an approximation. A c-approximation algorithm for the diameter D of a graph for $c \geq 1$ provides an estimate $\hat{D}$ such that $D/c \leq \hat{D} \leq D$. It is well known that a 2-approximation for the diameter in directed or undirected graphs with nonnegative weights is easy to achieve in $\tilde{O}(m)$ time using Dijkstra's algorithm from and to an arbitrary node. Dor, Halperin and Zwick [8] showed that any $(2-\varepsilon)$-approximation algorithm for APSP even in unweighted graphs running in $T(n)$ time would imply an $O(T(n))$ time for BMM, and hence apriori it could be that $(2-\varepsilon)$-approximating the diameter of a graph may also require solving BMM.

In their seminal paper, Aingworth, Chekuri, Indyk and Motwani [1] showed that it is in fact possible to get a subcubic $(2-\varepsilon)$-approximation algorithm for the diameter of graphs with nonnegative weights without resorting to fast matrix multiplication. In particular, they designed an $\tilde{O}(m\sqrt{n}+n^2)$ time algorithm computing an estimate $\hat{D}$ that satisfies $\lfloor 2D/3 \rfloor \leq \hat{D} \leq D$. Their algorithm has several important and interesting properties. It is the only known algorithm for approximating the diameter polynomially faster than $O(mn)$ for every m that is superlinear in n. It always runs in truly subcubic time even in dense graphs, and does not explicitly compute all-pairs approximate shortest paths.

A natural question is whether there is an almost linear time approximation scheme for the diameter problem: an algorithm that for any constant $\varepsilon > 0$ runs in $\tilde{O}(m)$ time and returns an estimate $\hat{D}$ such that $(1-\varepsilon)D \leq \hat{D} \leq D$. Such an algorithm would be of immense interest, and has not so far been explicitly ruled out, even conditionally. In this paper we give strong evidence that a fast $(3/2-\varepsilon)$-approximation algorithm for the diameter may be very hard to find, even for undirected unweighted graphs. We show:

Theorem 1 *Suppose there is a constant $\varepsilon > 0$ so that, there is a $(3/2-\varepsilon)$-approximation algorithm for the diameter in m-edge undirected unweighted graphs that runs in $O(m^{2-\varepsilon})$ time for every m. Then, SAT for*

CNF formulas on n variables can be solved in $O^*((2-\delta)^n)$ time for some constant $\delta > 0$.

The fastest known algorithm for CNF-SAT is the exhaustive search algorithm that runs in $O^*(2^n)$ time by trying all possible 2^n assignments to the variables. It is a major open problem whether there is a faster algorithm. Several other NP-hard problems are known to be equivalent to CNF-SAT so that if one of these problems has a faster algorithm than exhaustive search, then all of them do [7]. Hence, our result also implies that if the diameter can be approximated fast enough, then also problems such as Hitting Set, Set Splitting, or NAE-SAT, all seemingly unrelated to the diameter, can be solved faster than exhaustive search. The strong exponential time hypothesis (SETH) of Impagliazzo, Paturi, and Zane [10, 11] implies that there is no improved $O^*((2-\delta)^n)$ time algorithm for CNF-SAT, and hence our result also implies that there is no $(3/2-\varepsilon)$-approximation algorithm for the diameter running in $O(m^{2-\varepsilon})$ time unless SETH fails. (We elaborate on this hypothesis later on in the paper.)

We prove Theorem 1 by showing that an $O(n^{2-\varepsilon})$ time, $(3/2-\varepsilon)$-approximation algorithm for the diameter in sparse graphs with $m = O(n)$ would imply an $O^*((2-\delta)^n)$ time CNF-SAT algorithm. This implies that unless SETH fails, $O(n^2)$ time is essentially required to get a $(3/2-\varepsilon)$-approximation algorithm for the diameter in sparse graphs, within $n^{o(1)}$ factors. Hence, within $n^{o(1)}$ factors, the time for $(3/2-\varepsilon)$-approximating the diameter in a sparse graph is the same as the time required for computing APSP exactly!

Even more concretely, we prove Theorem 1 by showing that distinguishing whether the diameter of a given undirected unweighted graph is 2 or at least 3 fast enough would imply an improved SAT algorithm. (Any $(3/2-\varepsilon)$-approximation algorithm for the diameter would be able to distinguish between graphs of diameter 2 and 3.) The fastest algorithms for this special case of the diameter problem still run in $\tilde{O}(\min\{mn, n^\omega\})$ time, and several papers have asked whether one can do better [5, 1]. In 1987, Chung [5] actually conjectured that this problem may be equivalent to BMM, so that any subcubic algorithm for it can be converted to a subcubic algorithm for BMM. Aingworth *et al.* [1] conjectured that if there is a polynomially faster than $O(mn)$ time algorithm for this problem, then one can use it to construct a fast algorithm that computes the diameter exactly. These conjectures remain open, but Theorem 1 shows that the 2 vs 3 diameter problem may be hard to solve very efficiently for a different reason.

Theorem 1 shows that unless SETH fails, the best one can do with an $O(m^{2-\varepsilon})$ time algorithm is a $3/2$-approximation. The Aingworth *et al.* $3/2$-approximation algorithm almost achieves an $O(m^{2-\varepsilon})$ runtime, except for very sparse graphs when it still runs in $\Omega(n^2)$ time. We notice that with a slight change in the parameters of the algorithm, the Aingworth *et al.* running time can be modified to be $\widetilde{O}(m^{2/3}n) \leq \widetilde{O}(m^{2-1/3})$. We then investigate whether we can obtain a $3/2$-approximation algorithm that improves upon these two runtimes of the Aingworth *et al.* algorithm. We give a new $3/2$-approximation algorithm with $\tilde{O}(m\sqrt{n})$ expected running time, thus removing the n^2 additive factor from the original Aingworth *et al.* runtime with some randomization, and also beating $\widetilde{O}(m^{2/3}n)$. Our algorithm is the first improvement over the Aingworth *et al.* diameter algorithm. The improvement is especially noticeable for sparse graphs (with $m = \widetilde{O}(n)$) in which our algorithm runs in $\tilde{O}(n^{1.5})$ time. Previously, such a result was known only for sparse *planar* graphs [2][1]. We also show that in some special cases our algorithm obtains an approximation that is better than $3/2$.

Theorem 2 *Let $G = (V, E)$ be a directed or an undirected graph with diameter $D = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$. In $\widetilde{O}(m\sqrt{n})$ expected time one can compute an estimate $\hat{D}$ of D such that $2h + z \leq \hat{D} \leq D$ for $z \in \{0, 1\}$ and $2h + 1 \leq \hat{D} \leq D$ for $z = 2$.*

For undirected or directed graphs with arbitrary nonnegative weights, we also obtain the following.

[1]disregarding polylogarithmic factors

Theorem 3 *Let $G = (V, E)$ be a directed or an undirected graph with nonnegative edge weights and diameter D. In $\widetilde{O}(m\sqrt{n})$ expected time one can compute an estimate $\hat{D}$ of D such that $\lfloor 2D/3 \rfloor \le \hat{D} \le D$.*

We further investigate whether one can improve the approximation for unweighted graphs obtained in Theorem 2 by possibly increasing the runtime, while still keeping it subcubic in n. Notice that in Theorem 2, the estimate $\hat{D}$ is at least $2h + z$ for $z \in \{0, 1\}$ and only at least $2h + 1$ for $z = 2$. This only guarantees that $\hat{D} \ge \lfloor 2D/3 \rfloor$. (This is also the case for the algorithm of Aingworth *et al.* [1].)

We show that with a slightly larger (but still subcubic) running time it is possible to get an estimate $\hat{D}$ of D such that $2h+z \le \hat{D}$ for any value $z \in \{0, 1, 2\}$, thus guaranteeing that $\hat{D} \ge \lceil 2D/3 \rceil$. This is significant when D is a constant, and also shows that when $z \ne 0$, the approximation factor is strictly better than $3/2$: $(3h + z)/(2h + z) = 3/2 - 1/(4h/z + 2) \le 3/2 - 1/(4h + 2) < 3/2$.

We note that approximating the diameter is most challenging when the diameter is small. When the input graph has diameter $D \ge n^{\varepsilon}$ for some $\varepsilon > 0$, one can efficiently find an arbitrarily good approximation by random sampling: if you randomly sample $Cn^{1-\varepsilon}/\delta \log n$ nodes, then with probability at least $1-1/n^C$, one of these nodes is at distance at least $(1-\delta)D$ from an endpoint of the diameter path; hence a $1/(1-\delta)$-approximation can be found in $\tilde{O}(mn^{1-\varepsilon}/\delta)$ time by BFS. For sparse enough graphs of diameter $n^{o(1)}$ however, the best known $(3/2 - \varepsilon)$-approximation algorithms still compute the diameter exactly in $\tilde{O}(mn)$ time. Hence, it is quite interesting that we can obtain $\tilde{O}(m\sqrt{n})$ time $(3/2 - \varepsilon)$-approximation algorithms for some constant values of the diameter.

In Section 5 we prove the following Theorem.

Theorem 4 *Let $G = (V, E)$ be a directed or undirected unweighted graph with diameter $D = 3h + z$, where $h \ge 0$ and $z \in \{0, 1, 2\}$. There is an $\tilde{O}(m^{2/3}n^{4/3})$ time algorithm that reports an estimate $\hat{D}$ such that $2h + z \le \hat{D} \le D$.*

Marginally, we show how to get a better estimate for undirected graphs in the same running time.

Theorem 5 *Let $G = (V, E)$ be an undirected unweighted graph with diameter D. There is an $\tilde{O}(m^{2/3}n^{4/3})$ time algorithm that reports an estimate $\hat{D}$ such that $\lfloor 4D/5 \rfloor \le \hat{D} \le D$.*

The running time in Theorem 4 however is $\tilde{\Theta}(n^2)$ for sparse graphs. We hence investigate whether one can get an estimate $\lceil 2D/3 \rceil \le \hat{D} \le D$ in $O(m^{2-\varepsilon})$ time. We show:

Theorem 6 *There is an $\tilde{O}(m^{2-1/(2h+3)})$ time deterministic algorithm that computes an estimate $\hat{D}$ with $\lceil 2D/3 \rceil \le \hat{D} \le D$ for all m-edge unweighted graphs of diameter $D = 3h+z$ with $h \ge 0$ and $z \in \{0, 1, 2\}$. In particular, $\hat{D} \ge 2h + z$.*

Notation. Let $G = (V, E)$ denote a graph. It can be directed or undirected; this will be specified in each context. If the graph is weighted, then there is a function on the edges $w : E \to \mathbb{Q}^+ \cup \{0\}$. Unless explicitly specified, the graphs we consider are unweighted.

For any $u, v \in V$, let $d(u, v)$ denote the distance from u to v in G. Let $BFS^{\text{in}}(v)$ and $BFS^{\text{out}}(v)$ be the incoming and outgoing breadth-first search (BFS) trees of v, respectively, that is the BFS trees in G starting at v and in G with the edges reversed starting at v. Let $d^{\text{in}}(v)$ be the depth of $BFS^{\text{in}}(v)$, i.e. the largest distance from a vertex of $BFS^{\text{in}}(v)$ to v. Similarly, let $d^{\text{out}}(v)$ be the depth of $BFS^{\text{out}}(v)$.

For $h \le d^{\text{in}}(v)$, let $BFS^{\text{in}}(v, h)$ be the vertices in the first h levels of $BFS^{\text{in}}(v)$. Similarly, for $h \le d^{\text{out}}(v)$, let $BFS^{\text{out}}(v, h)$ be the vertices in the first h levels of $BFS^{\text{out}}(v)$.

Let $N_s^{\text{in}}(v)$ ($N_s^{\text{out}}(v)$) be the set of the s closest incoming (outgoing) vertices of v, where ties are broken by taking the vertex with the smaller id. We assume throughout the paper that for each v and each $s \le n$, $|N_s^{\text{in}}(v)| = |N_s^{\text{out}}(v)| = s$, as otherwise the diameter of the graph would be ∞, and this can be checked with two BFS runs from and to an arbitrary node.

Let $d_s^{\text{in}}(v)$ be the largest distance from a vertex of $N_s^{\text{in}}(v)$ to v, and $d_s^{\text{out}}(v)$ be the largest distance from v to a vertex of $N_s^{\text{out}}(v)$. Let $d_s^{\text{in}} = \max_{v \in V} d_s^{\text{in}}(v)$ and $d_s^{\text{out}} = \max_{v \in V} d_s^{\text{out}}(v)$.

For a set $S \subseteq V$ and a vertex $v \in V$ we define $p_S(v)$ to be a vertex of S such that $d(v, p_S(v)) \le d(v, w)$ for every $w \in S$, i.e. the closest vertex of S to v.

For a degree Δ we define $p_\Delta(v)$ to be the closest vertex to v of degree at least Δ, that is, $d(v, p_\Delta(v)) \le d(v, w)$ for every $w \in V$ of degree at least Δ.

We use the following standard notation for running times. For a function of n, $f(n)$, $\tilde{O}(f(n))$ denotes $O(f(n)\text{poly}\log n)$ and $O^*(f(n))$ denotes $O(f(n)\text{poly } n)$.

2 Diameter approximation and the Strong Exponential Time Hypothesis

Impagliazzo, Paturi, and Zane [10, 11] introduced the Exponential Time Hypothesis (ETH) and its stronger variant, the Strong Exponential Time Hypothesis (SETH). These two complexity hypotheses assume lower bounds on how fast satisfiability problems can be solved. They have frequently been used as a basis for conditional lower bounds for other concrete computational problems.

Hypothesis 1 ([10, 11]) ETH*: There exists a real constant $\delta > 0$ such that 3-SAT instances on n variables and m clauses cannot be solved in $2^{\delta n}\text{poly}(m, n)$ time.*

A natural question is how fast can one solve r-SAT as r grows. Impagliazzo, Paturi, and Zane define

$$s_r = \inf\{\delta \mid \exists\, O^*(2^{\delta n}) \text{ time algorithm solving } r\text{-SAT instances with } n \text{ variables}\}, \text{ and } s_\infty = \lim_{r\to\infty} s_r.$$

Clearly $s_r \le s_{r+1}$ so that the sequence is nondecreasing. Impagliazzo, Paturi, and Zane show that if ETH holds, then s_r also increases infinitely often. Furthermore, all known algorithms for r-SAT nowadays take time $O(2^{n(1-c/r)})$ for some constant c independent of n and r (e.g. [9, 12, 15, 14, 17, 18]). Because of this, it seems plausible that $s_\infty = 1$, and this is exactly the strong exponential time hypothesis.

Hypothesis 2 ([10, 11]) SETH*: $s_\infty = 1$.*

One immediate consequence of SETH is that CNF-SAT on n variables cannot be solved in $2^{n(1-\varepsilon)}\text{poly}(n)$ time for any $\varepsilon > 0$. The best known algorithm for CNF-SAT is the $O^*(2^n)$ time exhaustive search algorithm which tries all possible 2^n assignments to the variables, and it has been a major open problem to obtain an improvement. Cygan et al. [7] showed that SETH is also equivalent to the assumption that several other NP-hard problems cannot be solved faster than by exhaustive search, and the best algorithms for these problems are the exhaustive search ones.

Assuming SETH, one can prove tight conditional lower bounds on the complexity of some problems in P as well. The problem that we will look at is k-dominating set for constant k: given an undirected graph $G = (V, E)$, is there a set S of k vertices so that every vertex $v \in V$ is either in S or has an edge to some vertex in S? The best known algorithm for k-dominating set for $k \ge 7$ runs in $n^{k+o(1)}$ time and uses rectangular matrix multiplication [13]. Pătraşcu and Williams [13] showed that improving on this runtime may be hard as it would imply faster algorithms for CNF-SAT.

Theorem 7 ([13]) *Suppose there is a $k \geq 3$ and function f such that k-Dominating Set in an N-node graph is in $O(N^{f(k)})$ time. Then CNF-SAT on n variables and m clauses is in $O^*((m + k2^{n/k})^{f(k)})$ time.*

If $f(k) = k - \varepsilon$ for some constant $\varepsilon > 0$, then the above implies that SETH is false.

We show a strong relationship between the diameter problem in undirected unweighted graphs and k-dominating set.

Theorem 8 *Suppose one can distinguish between diameter 2 and 3 in an m-edge undirected unweighted graph in time $O(m^{2-\varepsilon})$ for some constant $\varepsilon > 0$. Then for all integers $k \geq 2/\varepsilon$, $2k$-dominating set can be solved in $O^*(n^{2k-\varepsilon})$ time. Moreover, CNF-SAT on n variables and m clauses is in $O^*(2^{n(1-\varepsilon^2/4)})$ time, and SETH is false.*

Theorem 8 immediately implies Theorem 1 in the introduction, as any $(3/2 - \varepsilon)$-approximation algorithm can distinguish between diameter 2 and 3.

Proof. Given an instance $G = (V, E)$ of $2k$-Dominating set for constant k, we construct an instance of the 2 vs 3 diameter problem and we show that $2k$-Dominating set in n-node graphs can be solved in $O^*(n^{2k-\delta})$ time for some constant $\delta > 0$ depending on ε.

Take all k-subsets of the vertices in V and add a node for each of them to the 2 vs 3 instance G'. Add a node for every vertex in V – call this set of nodes V' and make V' into a clique.

For every k-subset S of vertices of V, connect S to $v \in V'$ in G' iff S *does not dominate* v in G. While we do this we check whether each S is a k-dominating set in G, and if so, we stop. From now on we can assume that none of the k-subsets S are dominating sets in G.

Now, notice that if S and T are two k-subsets so that their union is not a $(\leq 2k)$-dominating set in G, then the distance in G' between S and T is 2: there is some u that is dominated by neither S nor T and so $S - u - T$ is a path of length 2. If, on the other hand, $S \cup T$ is a dominating set in G, then there is no such path and the shortest path between S and T in G' is to go from S to some v that S doesn't dominate, then to some u that T doesn't dominate (V' is a clique) and then from u to T.

The distance between any u and v in V' is 1, and the distance between any u and any S is at most 2: go from u to some node v that S doesn't dominate and then to S.

Hence, if there is no $2k$-dominating set in G, then the diameter of G' is 2, and if there is one, then the diameter of G' is 3. G' has $\binom{n}{k} + n$ nodes and at most $O(n \cdot \binom{n}{k}) \leq O(n^{k+1})$ edges.

Since we can solve the diameter problem in $O(m^{2-\varepsilon})$ time, applying that algorithm to G' solves $2k$-dominating set in G for any $k \geq 2$ in time $O(n^{2k+2-\varepsilon k-\varepsilon})$.

We want this to be $O(n^{2k-\delta})$ for some $\delta > 0$, so it suffices to pick k so that $-\delta \geq 2 - \varepsilon(k+1)$. If we want $\delta = \varepsilon$, then $k \geq 2/\varepsilon$ suffices. $\square$

3 The algorithm of Aingworth *et al.*

In this section we revisit the algorithm of Aingworth, Chekuri, Indyk and Motwani [1], that computes a $3/2$-approximation of the diameter of a directed (or undirected) graph in $\widetilde{O}(m\sqrt{n} + n^2)$ time. (The algorithm can also be made to work for graphs with nonnegative weights with roughly the same running time and approximation factor. In this section we only focus on the algorithm for unweighted graphs.)

Let s be a given parameter in $[1, n]$. The algorithm works as follows. First, it computes $N_s^{\text{out}}(v)$ for every $v \in V$. Then, for a vertex w, where $d_s^{\text{out}}(w) = d_s^{\text{out}}$ it computes $BFS^{\text{out}}(w)$ and for every $u \in N_s^{\text{out}}(w)$ it computes $BFS^{\text{in}}(u)$. Next, it computes a set S that hits $N_s^{\text{out}}(v)$ for every $v \in V$ and for every $u \in S$ it computes $BFS^{\text{out}}(u)$. As an estimate, the algorithm returns the depth of the deepest computed BFS tree.

The next lemma appears in [1]. We state it for completeness.

Lemma 1 *The running time of the algorithm is* $\widetilde{O}(ns^2 + (n/s + s)m)$.

Aingworth et al. set $s = \sqrt{n}$ and obtain their running time. We note that if one sets $s = m^{1/3}$ instead, one can get a runtime of $\widetilde{O}(m^{2/3}n)$ that is better for sparse graphs; we later show that both of these runtimes can be improved with randomization.

We now analyze the quality of the estimate returned by the algorithm. Aingworth *et al.* [1] proved that this estimate is at least $\lfloor 2D/3 \rfloor$ in graphs of diameter D. Here we present a tighter analysis.

Lemma 2 *Let* $G = (V, E)$ *be a directed graph with diameter* $D = 3h + z$*, where* $h \geq 0$ *and* $z \in \{0, 1, 2\}$*. Let* $\hat{D}$ *be the estimate returned by the algorithm. For* $z \in \{0, 1\}$*, we have* $2h + z \leq \hat{D} \leq D$*. For* $z = 2$*, we have that* $2h + 1 \leq \hat{D} \leq D$*.*

Proof. Let $a, b \in V$ such that $d(a, b) = D$. First notice that the algorithm always returns a depth of some shortest paths tree and hence $\hat{D} \leq D$.

Now, if $d_s^{\text{out}}(w) \leq h$ then also $d_s^{\text{out}}(a) \leq h$ and as S hits $N_s^{\text{out}}(a)$, one of the BFS trees computed for vertices of S has depth at least $2h + z$. Hence, assume that $d_s^{\text{out}}(w) > h$. We can also assume that $d^{\text{out}}(w) < 2h + z$ as otherwise when we compute $BFS^{\text{out}}(w)$, we'd return a depth at least $2h + z$.

As $d^{\text{out}}(w) < 2h + z$, also $d(w, b) < 2h + z$. Since $d_s^{\text{out}}(w) > h$, we have that $BFS^{\text{out}}(w, h) \subseteq N_s^{\text{out}}(w)$. Hence there is a vertex $w' \in N_s^{\text{out}}(w)$ on the path from w to b such that $d(w, w') = h$ and hence $d(w', b) < h + z$. Since $d(a, b) = 3h + z$, we must have that $d(a, w') \geq 2h + 1$. As the algorithm computes $BFS^{\text{in}}(u)$ for every $u \in N_s^{\text{out}}(w)$, in particular, it computes $BFS^{\text{in}}(w')$, and returns an estimate $\geq 2h + 1$. For $z \in \{0, 1\}$, $d(a, w') \geq 2h + 1 \geq 2h + z$ and hence the final estimate returned is always at least $2h + z$. For $z = 2$ we only have that $d(a, w') \geq 2h + 1$ and if the algorithm returns $d(a, w')$ as an estimate, it may return $2h + 1$ instead of $2h + z$. □

4 Improving the running time

The algorithm of Aingworth *et al.* [1] runs in $\widetilde{O}(ns^2 + (n/s + s)m)$. In this section we show that it is possible to get rid of the ns^2 term, while keeping the quality of the estimate unchanged. By choosing $s = \sqrt{n}$, we get an algorithm running in $\widetilde{O}(m\sqrt{n})$ time.

The term of ns^2 in the running time comes from the computation of $N_s^{\text{out}}(v)$ for every $v \in V$. This computation is done to accomplish two tasks. One task is to obtain $d_s^{\text{out}}(v)$ for every $v \in V$ and then to use it to find a vertex w such that $d_s^{\text{out}}(w) = d_s^{\text{out}}$. A second task is to obtain, deterministically, a hitting set S of size $\widetilde{O}(n/s)$ that hits the set $N_s^{\text{out}}(v)$ of every $v \in V$.

Our main idea is to accomplish these two tasks without explicitly computing $N_s^{\text{out}}(v)$ for every $v \in V$. The major step in our approach is to completely modify the first task above by picking a different type of vertex to play the role of w. Making the second task above fast can be accomplished easily with randomization. We elaborate on this below.

Our algorithm works as follows. First, it computes a hitting set by using randomization, that is, it picks a random sample S of the vertices of size $\Theta(n/s \log n)$. This guarantees that with high probability (at least $1 - n^{-c}$, for some constant c), $S \cap N_s^{\text{out}}(v) \neq \emptyset$, for every $v \in V$. This accomplishes the second task above in $\tilde{O}(n)$ time, with high probability. Similarly to the algorithm of Aingworth *et al.* [1], our algorithm computes $BFS^{\text{out}}(v)$, for every $v \in S$.

We now explain the main idea of our algorithm, i.e. how we are able replace the first task from before with a much faster step. First, for every $v \in V$ our algorithm computes the closest node of S, $p_S(v)$, to v, by creating a new graph as follows. It adds an additional vertex r with edges (u, r), for every $u \in S$. It computes $BFS^{\text{in}}(r)$ in this graph. It is easy to see that for every $v \in V$ the last vertex before r on the shortest path from v to r is $p_S(v)$. This step takes $O(m)$ time.

Now, as opposed to the algorithm of Aingworth *et al.* that picks a vertex w such that $d_s^{\text{out}}(w) = d_s^{\text{out}}$, our algorithm finds a vertex $w \in V$ that is furthest away from S: i.e. such that $d(w, p_S(w)) \geq d(u, p_S(u))$, for every $u \in V$. The vertex w plays the same role as its counterpart in [1]: Our algorithm computes $BFS^{\text{out}}(w)$ and obtains $N_s^{\text{out}}(w)$ from it. Finally, it computes $BFS^{\text{in}}(u)$ for every $u \in N_s^{\text{out}}(w)$. As an estimate, the algorithm returns the depth of the deepest BFS tree that it has computed.

In the next Lemma we analyze the running time of the algorithm.

Lemma 3 *The running time of the algorithm is $\widetilde{O}((n/s + s)m)$.*

Proof. A hitting set S is formed in $O(n)$ time. With a single BFS computation, in $O(m)$ time, we find $p_S(v)$ for every $v \in V$, and hence also find w. The cost of computing a BFS tree for every $v \in S \cup N_s^{\text{out}}(w)$ is $\widetilde{O}((n/s + s)m)$. □

Next, we show that the estimate produced by our algorithm is of the same quality as the estimate produced by Aingworth *et al.* algorithm, with high probability.

Lemma 4 *Let $G = (V, E)$ be a directed (or undirected) graph with diameter $D = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$. Let $\hat{D}$ be the estimate returned by the above algorithm. With high probability, $2h + z \leq \hat{D} \leq D$ whenever $z \in \{0, 1\}$, and $2h + 1 \leq \hat{D} \leq D$ whenever $z = 2$.*

Proof. Let $a, b \in V$ such that $d(a, b) = D$. Let w be a vertex that satisfies $d(w, p_S(w)) \geq d(u, p_S(u))$, for every $u \in V$.

If $d(w, p_S(w)) \leq h$ then also $d(a, p_S(a)) \leq h$. As the algorithm computes $BFS^{\text{out}}(v)$ for every $v \in S$, it follows that $BFS^{\text{out}}(p_S(a))$ is computed as well and its depth is at least $2h + z$ as required. Hence, assume that $d(w, p_S(w)) > h$. We can assume also that $d^{\text{out}}(w) < 2h + z$ since the algorithm computes $BFS^{\text{out}}(w)$ and if $d^{\text{out}}(w) \geq 2h + z$ then it computes a BFS tree of depth at least $2h + z$ as required.

Since $d^{\text{out}}(w) < 2h + z$ it follows that $d(w, b) < 2h + z$. Moreover, since $d(w, p_S(w)) > h$ and S hits $N_s^{\text{out}}(w)$ whp, we must have that $N_s^{\text{out}}(w)$ contains a node at distance $> h$ from w, and hence $BFS^{\text{out}}(w, h) \subseteq N_s^{\text{out}}(w)$. This implies that there is a vertex $w' \in N_s^{\text{out}}(w)$ on the path from w to b such that $d(w, w') = h$ and hence $d(w', b) < h + z$. Since $d(a, b) = 3h + z$, we also have that $d(a, w') \geq 2h + 1$.

The algorithm computes $BFS^{\text{in}}(u)$ for every $u \in N_s^{\text{out}}(w)$, and in particular, it computes $BFS^{\text{in}}(w')$, thus returning an estimate at least $d(a, w') \geq 2h + 1$. Hence for $z \in \{0, 1\}$ the final estimate is always $\geq 2h + z$, and for $z = 2$ the estimate could be $2h + 1$ but no less. □

We now turn to prove Theorem 2 from the introduction.

Reminder of Theorem 2 *Let $G = (V, E)$ be a directed or an undirected graph with diameter $D = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$. In $\widetilde{O}(m\sqrt{n})$ expected time one can compute an estimate $\hat{D}$ of D such that $2h + z \leq \hat{D} \leq D$ for $z \in \{0, 1\}$ and $2h + 1 \leq \hat{D} \leq D$ for $z = 2$.*

Proof. From Lemma 3 we have that if we set $s = \sqrt{n}$ the algorithm runs in $\widetilde{O}(m\sqrt{n})$ worst case time. From Lemma 4 we have that with high probability, that is $1 - n^{-c}$ for some constant c, the algorithm returns an estimate of the desired quality. We now show how to convert the algorithm into a Las-Vegas one so that it always returns an estimate of the desired quality but the running time is $\widetilde{O}(m\sqrt{n})$ in expectation.

Randomization is used only in order to obtain a set that hits $N_s^{\text{out}}(v)$ for every $v \in V$. The only place that the hitting set affects the quality of the approximation is in Lemma 4 where we used the fact that, whp, S contains a node of $N_s^{\text{out}}(w)$, so that if $d(w, S) > h$, $N_s^{\text{out}}(w)$ contains a node at distance $> h$ from w.

Note that we compute $N_s^{\text{out}}(w)$ and we can check whether S intersects it in $\widetilde{O}(s)$ time. If it doesn't, we can rerun the algorithm until we have verified that $S \cap N_s^{\text{out}}(w) \neq \emptyset$. Since $S \cap N_s^{\text{out}}(w) = \emptyset$ holds with very small probability, the expected running time of the algorithm is $\widetilde{O}(m\sqrt{n})$ and its estimate is guaranteed to have the required quality. $\square$

Just as in [1], we can make our algorithm work for graphs with nonnegative weights as well by replacing every use of BFS with Dijkstra's algorithm. The proofs are analogous, and the running time is increased by at most a $\log n$ factor. We obtain

Reminder of Theorem 3 *Let $G = (V, E)$ be a directed or an undirected graph with nonnegative edge weights and diameter D. In $\widetilde{O}(m\sqrt{n})$ expected time one can compute an estimate $\hat{D}$ of D such that $\lfloor 2D/3 \rfloor \leq \hat{D} \leq D$.*

5 Improving the approximation for unweighted graphs

In this section we show that in some cases it is possible to improve the approximation of the algorithm of Aingworth *et al.* for unweighted graphs. Recall that for a graph with diameter $D = 3h + 2$ their algorithm returns an estimate $\hat{D}$ such that $2h + 1 \leq \hat{D} \leq D$. We show that for such a case it is possible to return an estimate $\hat{D}$ such that $2h + 2 \leq \hat{D} \leq D$. This is significant for small diameter values. For example, for a graph of diameter 5 our estimate is at least 4, while the previous estimate was at least 3.

We present two algorithms that obtain this improved approximation, one works well for dense graphs and the other for sparse graphs.

5.1 Dense graphs

Our algorithm Approx-Diam(G) works as follows. (The pseudocode is in the appendix.) First, it runs the Aingworth *et al.* algorithm both on the input graph G and on the input graph with the edge directions reversed, G^R. Let $\hat{D}$ be the maximum value returned by these two runs. A byproduct of this step is that for every $v \in V$ we have computed $BFS^{\text{out}}(v, d_s^{\text{out}}(v) - 1)$ and $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$. Next, our algorithm scans all pairs of vertices u and v and checks whether the following condition holds: $BFS^{\text{out}}(u, d_s^{\text{out}}(u) - 1)$ and $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$ are disjoint and there is no edge between $BFS^{\text{out}}(u, d_s^{\text{out}}(u) - 1)$ and $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$. Given a pair of vertices u and v for which the condition holds, the algorithm updates $\hat{D}$ to be the maximum between its current value and $d_s^{\text{out}}(u) + d_s^{\text{in}}(v)$.

We start by showing that the estimate reported by the algorithm is upper-bounded by the graph diameter.

Lemma 5 *Let $G = (V, E)$ be a graph of diameter D. If $\hat{D} =$ Approx-Diam(G), then $\hat{D} \leq D$.*

Proof. If Approx-Diam(G) returns the value that it gets from one of the runs of Aingworth *et al.* algorithm then the claim follows from Lemma 2. If the algorithm reports $d_s^{\text{out}}(u) + d_s^{\text{in}}(v)$ for some pair of vertices $u, v \in V$ it is because there is no edge from $BFS^{\text{out}}(u, d_s^{\text{out}}(u) - 1)$ to $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$, and no vertex in common between the two trees. This means that there is no path of length at most $d_s^{\text{out}}(u) + d_s^{\text{in}}(v) - 1$ from u to v, and hence, any path from u to v, and in particular the shortest one, is of length at least $d_s^{\text{out}}(u) + d_s^{\text{in}}(v) \leq D$ as required. $\square$

Next, we lower-bound the estimate reported by the algorithm.

Lemma 6 *Let $G = (V, E)$ be a graph of diameter $D = 3h + z$, where $h \geq 1$ and $z \in \{0, 1, 2\}$. If $\hat{D} =$ Approx-Diam(G) then $2h + z \leq \hat{D} \leq 3h + z$.*

Proof. Let $a, b \in V$ such that $d(a, b) = D$. Running the algorithm of Aingworth *et al.* for G and the reverse G^R of G implies that we get an approximation of $2h + z$ in the following cases.

Case 1: [$z \neq 2$]. From Lemma 2, we have that the estimate is at least $2h + z$.

Case 2: [$d_s^{\text{out}}(a) \leq h$ or $d_s^{\text{in}}(b) \leq h$]. If $d_s^{\text{out}}(a) \leq h$ then the hitting set computed by the Aingworth *et al.* algorithm contains a vertex at distance at most h from a and hence one of the BFS trees that it computes has depth at least $2h + z$. Running the algorithm on G^R guarantees that the same holds when $d_s^{\text{in}}(b) \leq h$.

Case 3: [$\exists w \in V$ s.t. $d_s^{\text{out}}(w) \geq h + 2$]. In this case let w be the vertex with the largest $d_s^{\text{out}}(w)$ value. The Aingworth *et al.* algorithm computes $BFS^{\text{out}}(w)$. If $d^{\text{out}}(w) \geq 2h + 2$ then the claim holds so assume that $d^{\text{out}}(w) \leq 2h + 1$. The algorithm computes $BFS^{\text{in}}(v)$ for every $v \in BFS^{\text{out}}(w, h + 1)$ and since $d(w, b) \leq 2h + 1$ there is a vertex $w' \in BFS^{\text{out}}(w, h + 1)$ such that $d(w', b) \leq h$. As the algorithm computes $BFS^{\text{in}}(w')$ and $d(a, w') \geq 2h + z$ the claim holds.

For the rest of the proof we assume that the three cases above do not hold, hence, $z = 2$, $d_s^{\text{out}}(a) = h+1$ and $d_s^{\text{in}}(b) = h + 1$. The second part of our algorithm searches for a pair of vertices $u, v \in V$ such that there is no edge from $BFS^{\text{out}}(u, d_s^{\text{out}}(u) - 1)$ to $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$ (and no vertex in common between the two trees). As $D = d(a, b) = 3h + 2 > 2h + 1$, and $d_s^{\text{out}}(a) - 1 = h$ and $d_s^{\text{in}}(b) - 1 = h$, we have that there is no edge from $BFS^{\text{out}}(a, d_s^{\text{out}}(a) - 1)$ to $BFS^{\text{in}}(b, d_s^{\text{in}}(b) - 1)$ (and no vertex in common between the two trees). Since the estimate reported by the algorithm is the maximum among values that also include $d_s^{\text{out}}(a) + d_s^{\text{in}}(b) = 2h + 2$, we get that $\hat{D} \geq 2h + 2$, as required. □

Reminder of Theorem 4 *Let $G = (V, E)$ be a directed or undirected unweighted graph with diameter $D = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$. There is an $\tilde{O}(m^{2/3}n^{4/3})$ time algorithm that reports an estimate $\hat{D}$ such that $2h + z \leq \hat{D} \leq D$.*

Proof. The bounds on the estimate follow from Lemma 6 and Lemma 5. Running the algorithm of Aingworth *et al.* takes $\widetilde{O}(m(s+n/s)+ns^2)$ time. Searching for a pair of vertices $u, v \in V$ such that there is no edge from $BFS^{\text{out}}(u, d_s^{\text{out}}(u) - 1)$ to $BFS^{\text{in}}(v, d_s^{\text{in}}(v) - 1)$ takes $O(n^2 s^2)$ time. Setting $s = (m/n)^{1/3}$ gives us the running time. □

We can use Theorem 4 to obtain an even better approximation for undirected graphs.

Reminder of Theorem 5 *Let $G = (V, E)$ be an undirected unweighted graph with diameter D. There is an $\tilde{O}(m^{2/3}n^{4/3})$ time algorithm that reports an estimate $\hat{D}$ such that $\lfloor 4D/5 \rfloor \leq \hat{D} \leq D$.*

Proof. Using [8] we compute the distances between every pair of vertices in the graph, with an additive error of 2 in $O(\min(n^{3/2}\sqrt{m}, n^{7/3}))$ time. If $\hat{D}$ is the maximum distance minus 2 then $D - 2 \leq \hat{D} \leq D$. For every $D \geq 6$ we have that $D - 2 \geq \lfloor 4/5D \rfloor$. Thus, when $\hat{D} \geq 4$ we get an estimate of at least $\lfloor 4D/5 \rfloor$. If $\hat{D} = 3$ then D might be either 3, 4 or 5, that is, $D = 3 + z$, where $z \in \{0, 1, 2\}$. If $D = 5$, an estimate of 3 is not good enough, thus we run Approx-Diam(G). Let D' be the estimate reported by Approx-Diam(G). From Lemma 6 it follows that if $D = 5$ then $D' \geq 4$ and we have the required approximation. If $\hat{D} = 2$ then D might be either 2, 3 or 4, and for this case we can just use the Aingworth *et al.* algorithm to get an estimate of 3 whenever $D = 4$ which gives the desired approximation. □

5.2 Sparse graphs

We now show that it is possible to obtain the better approximation also in $\widetilde{O}(m^{2-\varepsilon})$ time for constant $\varepsilon > 0$ when the diameter of the given graph is constant.

Our algorithm, Approx-Diam-Sparse($G, \tilde{h}$) is given an estimate $\tilde{h}$ of h so that $\tilde{h} \geq h$ and works as follows. (The pseudocode can be found in the appendix.) Let Δ be a parameter and let H be the set of vertices of outdegree at least Δ. For every vertex of H, the algorithm computes an outgoing BFS tree. Then, it computes the distance from every node in $V \setminus H$ to H. This is done by adding an extra node r to the graph with edges from each node of H to r and then computing an incoming BFS to r in $O(m)$ time. The distance of a node v to H is its distance to r, minus 1. The algorithm then picks the vertex w that is furthest from H and computes $BFS^{\text{out}}(w)$. Let $h' = \min\{\tilde{h}+1, d(w,H)\}$. The algorithm computes $BFS^{\text{in}}(v)$ for every $v \in BFS^{\text{out}}(w,h')$. Finally, it returns the maximum depth of all computed BFS trees.

We now analyze the quality of the approximation.

Lemma 7 *Let $G=(V,E)$ be a graph of constant diameter $D=3h+z$, where $h \geq 0$ and $z \in \{0,1,2\}$. If $\hat{D}$ = Approx-Diam-Sparse($G,\tilde{h}$) for $\tilde{h} \geq h$, then $2h+z \leq \hat{D} \leq D$.*

Proof. First notice, that in any case the algorithm returns a depth of some BFS tree in the graph, thus $\hat{D} \leq D$.

Now, let $a,b \in V$ such that $d(a,b)=D$ and let $H \subseteq V$ be the set of vertices of outdegree at least Δ. Let $y^o \in H$ be the vertex with the deepest outgoing BFS in H. Let y^i be the vertex with the deepest incoming BFS among the vertices of $BFS^{\text{out}}(w,h')$, where $h' = \min\{\tilde{h}+1, d(w,H)\}$. The algorithm returns as an estimate $\max(d^{\text{out}}(y^o), d^{\text{out}}(w), d^{\text{in}}(y^i))$.

If $d(a,H) \leq h$, then $d^{\text{out}}(y^o)$ is at least $2h+z$ and the estimate is of the desired quality. So assume that $d(a,H) > h$, and hence $d(w,H) \geq d(a,H) \geq h+1$. Thus $h' \geq h+1$, as we also have $\tilde{h} \geq h$ by assumption. Assume also that $BFS^{\text{out}}(w)$ is of depth at most $2h+z-1$ as if it is of depth at least $2h+z$ then the estimate is of the desired quality. Then, there is a vertex $w' \in BFS^{\text{out}}(w,h')$ on the shortest path from w to b with $d(w,w')=h+1$ and hence $d(w',b) \leq h+z-2$. As $d(a,b)=3h+z$, we must also have $d(a,w') \geq 2h+2$ and as $d^{\text{in}}(y^i) \geq d(a,w')$, the estimate is of the desired quality. □

Next, we analyze the running time of the algorithm.

Lemma 8 *Let $G=(V,E)$ be a graph of diameter $D=3h+z$, where $h \geq 0$ and $z \in \{0,1,2\}$. If $\tilde{h} \geq h$, Approx-Diam-Sparse($G,\tilde{h}$) runs in $O(m^2/\Delta + \Delta^{\tilde{h}+1}m)$ time.*

Proof. The algorithm computes a BFS tree for every vertex of H. $|H| = O(m/\Delta)$ since there are at most that many vertices of outdegree at least Δ. Hence the BFS computation from H takes $O(m^2/\Delta)$ time.

Computing the distances of the nodes in $V \setminus H$ to H takes only $O(m)$ time. Picking the node w at largest distance to H takes $O(n)$ time. The algorithm computes $BFS^{\text{out}}(w)$ in $O(m)$ time. It then computes $BFS^{\text{in}}(v)$ for every $v \in BFS^{\text{out}}(w,h')$ where $h' \leq \tilde{h}+1$. Since we also have that $h' \leq d(w,H)$, every $v \in BFS^{\text{out}}(w,h'-1)$ has outdegree at most Δ. Thus, $|BFS^{\text{out}}(w,h')| \leq \Delta^{h'} \leq \Delta^{\tilde{h}+1}$. The running time of computing $BFS^{\text{in}}(v)$ for every $v \in BFS^{\text{out}}(w,h')$ is hence at most $O(m\Delta^{\tilde{h}+1})$. □

We now prove Theorem 6 from the introduction.

Reminder of Theorem 6 *There is an $\tilde{O}(m^{2-1/(2h+3)})$ time deterministic algorithm that computes an estimate $\hat{D}$ with $\lceil 2D/3 \rceil \leq \hat{D} \leq D$ for all m-edge unweighted graphs of diameter $D=3h+z$ with $h \geq 0$ and $z \in \{0,1,2\}$. In particular, $\hat{D} \geq 2h+z$.*

Proof. In $O(m)$ time we can get a 2-approximation to the diameter, i.e. an estimate E with $D/2 \leq E \leq D$. Since $D=3h+z$, we have that $(E-2)/3 \leq h \leq 2E/3$. Setting $\tilde{h} = 2E/3$ guarantees that $h \leq \tilde{h} \leq 2h+4/3 < 2h+2$, and hence $h \leq \tilde{h} \leq 2h+1$.

The quality of the estimate follows from Lemma 7 and by Lemma 8, the runtime is $O(m^2/\Delta + m\Delta^{2h+2})$. Picking $\Delta = m^{1/(2h+3)}$ minimizes the running time at $O(m^{2-1/(2h+3)})$. □

Acknowledgements The first author wants to thank Edith Cohen, Haim Kaplan and Yahav Nussbaum for fruitful discussions on the problem. The second author wants to thank Bob Tarjan for asking whether there is an almost linear time approximation scheme for the diameter.

References

[1] 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.

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

[3] D. Chakrabarti, C. Faloutsos, and M. McGlohon. Graph mining: Laws and generators. In *Managing and Mining Graph Data*, pages 69–123. 2010.

[4] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. STOC*, pages 590–598, 2007.

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

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

[7] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom. On problems as hard as CNFSAT. In *Proc. CCC*, page to appear, 2012.

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

[9] E. A. Hirsch. Two new upper bounds for SAT. In *Proc. SODA*, pages 521–530, 1998.

[10] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62:367–375, 2001.

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

[12] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287 – 295, 1985.

[13] M. Patrascu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. SODA*, pages 1065–1075, 2010.

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

[15] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999, 1999.

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

[17] I. Schiermeyer. Solving 3-satisfiability in less then 1.579^n steps. In *CSL*, pages 379–394, 1992.

[18] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. FOCS*, pages 410–414, 1999.

[19] A. Stothers. *Ph.D. Thesis, U. Edinburgh*, 2010.

[20] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. STOC*, 2012. To appear.

[21] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.

6 Appendix

Algorithm 1: Approx-Diam(G)

$X_1 \leftarrow$ Aingworth(G);
$X_2 \leftarrow$ Aingworth(G^R);
$\hat{D} \leftarrow \max(X_1, X_2)$;
foreach $v \in V$ **do**
  foreach $u \in V \setminus \{v\}$ **do**
    if $BFS^{out}(u, d_s^{out}(u) - 1) \cap BFS^{in}(v, d_s^{in}(v) - 1) = \emptyset \wedge \nexists (u', v') \in E$ *s.t.* $u' \in BFS^{out}(u, d_s^{out}(u) - 1) \wedge v' \in BFS^{in}(v, d_s^{in}(v) - 1)$ **then**
      $\hat{D} \leftarrow \max(\hat{D}, d_s^{\text{out}}(u) + d_s^{\text{in}}(v))$
return $\hat{D}$;

Algorithm 2: Approx-Diam-Sparse($G, \tilde{h}$)

$H \leftarrow \{v \mid deg(v) \geq \Delta\}$;
foreach $v \in H$ **do** Compute $BFS^{\text{out}}(v)$;
$y^o \leftarrow \arg\max_{x \in H} d^{\text{out}}(x)$;
$\hat{D} \leftarrow d^{\text{out}}(y^o)$;
Compute $d(v, H)$ for all $v \in V$ with a single BFS;
$w \leftarrow$ vertex of largest $d(w, H)$;
Compute $BFS^{\text{out}}(w)$;
$\hat{D} \leftarrow \max\{\hat{D}, d^{\text{out}}(y^o)\}$;
$h' \leftarrow \min\{\tilde{h} + 1, d(w, H)\}$;
foreach $v \in BFS^{out}(w, h')$ **do** Compute $BFS^{\text{in}}(v)$;
$y^i \leftarrow \arg\max_{x \in BFS^{\text{out}}(w,h')} d^{\text{in}}(x)$;
$\hat{D} \leftarrow \max\{\hat{D}, d^{\text{in}}(y^i)\}$;
return $\hat{D}$;