Subquadratic time approximation algorithms for the girth*

Liam Roditty † Virginia Vassilevska Williams ‡

Abstract

We study the problem of determining the girth of an unweighted undirected graph. We obtain several new efficient approximation algorithms for graphs with n nodes and m edges and unknown girth g. We consider additive and multiplicative approximations.

Additive Approximations. We present:

- an $\tilde{O}(n^3/m)$-time algorithm which returns a cycle of length at most $g+2$ if g is even and $g+3$ if g is odd. This complements the seminal work of Itai and Rodeh [SIAM J. Computing'78] who gave an algorithm that in $O(n^2)$ time finds a cycle of length g if g is even, and $g+1$ if g is odd.
- an $\tilde{O}(n^3/m)$-time algorithm which returns a cycle of length at most $g'+2$, where g' is the length of the shortest *even* cycle in G. This result complements the work of Yuster and Zwick [SIAM J. Discrete Math'97] who showed how to compute g' in $O(n^2)$ time.

Multiplicative Approximations. We present:

- an $\tilde{O}(n^{5/3})$-time algorithm which returns a cycle of length at most $3g/2 + z/2$ when g is even and $3g/2 + z/2 + 1$ when g is odd, where $z = -g \bmod 4$, $z \in \{0,1,2,3\}$. This gives an $\tilde{O}(n^{5/3})$-time 2-approximation for the girth, the first subquadratic 2-approximation algorithm, resolving an open question of Lingas and Lundell [IPL'09].
- an $O(n^{1.968})$-time (8/5)-approximation algorithm for the girth in graphs with girth at least 4 (i.e., triangle-free graphs). This is the first subquadratic time $(2-\varepsilon)$-approximation algorithm for the girth for triangle-free graphs, for any $\varepsilon > 0$. We prove that a deterministic algorithm of this kind is not possible for directed graphs, thus showing a strong separation between undirected and directed graphs for girth approximation.

1 Introduction

We consider the algorithmic problem of computing the length g of the shortest cycle in an unweighted undirected graph with n vertices and m edges. This natural graph parameter, called the *girth*, has many applications (e.g. in cycle packing [15] or computing a minimum cycle basis, which in itself has a myriad of applications [14]).

The problem of computing the girth has been studied since the 1970s. Itai and Rodeh [13] showed that the shortest cycle can be found in $O(nm)$ time by n applications of Breadth-First-Search (BFS), or in $O(n^\omega) \leq O(n^{2.376})$-time using fast matrix multiplication [8].

For more than 30 years these two algorithms have remained essentially the fastest ways to compute the girth. There has been an intuition that the problem is inherently tied to the all pairs shortest paths (APSP) and Boolean matrix multiplication (BMM) problems, the best algorithms for which have the same runtimes as those for the girth. Recently, Vassilevska W. and Williams [22] made this intuition formal by showing that any combinatorial algorithm computing the girth in truly subcubic runtime ($O(n^{3-\varepsilon})$-time for constant $\varepsilon > 0$) implies a truly subcubic time combinatorial algorithm for multiplying two Boolean matrices, and hence also for APSP in unweighted graphs.

Although not formally defined, combinatorial algorithms distinguish themselves from their algebraic counterparts (such as the bilinear algorithms for matrix multiplication) in that they are often less complex and have smaller constants in the big-O of the running time. Thus, although theoretically one can solve BMM, APSP and the girth problem using fast matrix multiplication, one often considers combinatorial algorithms instead, as they can be much more efficient in practice. The fastest combinatorial algorithm for BMM is a recent $O(n^3/\log^{2.25} n)$ time algorithm by Bansal and Williams [5] who improved on the 40-year record of $O(n^3/\log^2 n)$ by the Four-Russians Algorithm [4]. Obtaining a truly subcubic combinatorial algorithm for BMM is a longstanding open problem, and would be a significant breakthrough, as many computational problems can be reduced to BMM (see e.g. [9, 11, 21]). Hence the result of [22] implies a seeming cubic barrier for any practical algorithm which computes the girth exactly.

To circumvent this barrier, it is interesting to consider fast practical *approximation* algorithms for the girth. In the same paper [13] that contains their exact algorithm, Itai and Rodeh showed that with a simple BFS approach, in $O(n^2)$ time one can compute a cycle of length at most $g+1$. This is as good of an approximation

*The first author was supported by ISF grant no. 822/10. The second author was supported by NSF Grant #0963904 and NSF Grant #0937060 to the CRA for the CIFellows Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the CRA.

†Department of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel

‡Computer Science Division, University of California, Berkeley, Berkeley, CA 94720, USA

as one could hope for, short of computing g exactly. Furthermore, the quadratic runtime seems like it could be optimal since any $o(n^2)$ time algorithm which works for all graphs would run in *sublinear* time for dense enough graphs. It is an intriguing question, how good of an approximation to the girth can one achieve if one cannot even read the entire input?

We study this question in the context of two types of approximations: *additive* and *multiplicative*. For constant $c \geq 0$, an additive c-approximation for the girth g returns a cycle of length between g and $g + c$; as an example, Itai and Rodeh's $O(n^2)$ time algorithm returns an additive 1-approximation. For constant $c \geq 1$, a multiplicative c-approximation returns a cycle of length between g and $c \cdot g$.

Additive approximations. Itai and Rodeh's additive 1-approximation algorithm is the only additive constant-approximation algorithm for the girth to our knowledge. Even though it obtains the best possible approximation quality, short of returning an exact answer, it always runs in quadratic time, regardless of the graph density. We complement Itai and Rodeh's result by presenting an $O((n^3/m)\log^2 n)$-time algorithm that, with high probability, returns a cycle of length at most $g + 2$ if g is even and $g + 3$ if g is odd. We also show how to obtain a similar additive approximation for the *shortest even cycle* in the graph, complementing a result by Yuster and Zwick [23] who showed that a shortest even cycle can be found in $O(n^2)$ time. For any graph with $\omega(n \log^2 n)$ edges, we break the seeming quadratic time barriers of Itai and Rodeh (for the girth) and of Yuster and Zwick (for shortest even cycle), while keeping the additive error a small constant. In Section 3, we show how to make these algorithms deterministic, with a slight time overhead.

THEOREM 1.1. *Let $G(V, E)$ be an unweighted undirected graph. There is an $\tilde{O}(n^3/m)$ time algorithm that finds, with high probability, a cycle in G whose length is an additive 3-approximation for the girth of G. Moreover, there is an $\tilde{O}(n^3/m)$ time algorithm that finds, with high probability, an even cycle in G whose length is an additive 2-approximation for the length of the shortest even cycle in G.*

Multiplicative approximations. Any additive approximation can be considered a multiplicative one as well. Itai and Rodeh's additive 1-approximation, for instance, is at worst a multiplicative 4/3-approximation for the girth. More than 30 years after Itai and Rodeh's paper, Lingas and Lundell [16] presented the first algorithm that breaks the quadratic time bound, at the price of a much weaker approximation: their algorithm runs in $\tilde{O}(n^{3/2})$ time and returns a multiplicative 8/3-approximation. In fact, Lingas and Lundell's algorithm returns a cycle of length at most $2g + 2$: this is almost a 2-approximation, but not quite. At the end of their paper, they asked whether there exists a subquadratic time algorithm with a multiplicative approximation factor of 2 or less.

We are able to answer Lingas and Lundell's open question by giving the first subquadratic time, multiplicative 2-approximation for the girth in undirected unweighted graphs. Furthermore, unlike the previous subquadratic time approximation algorithm for the girth in [16], our algorithm is deterministic!

THEOREM 1.2. *There is an $O(n^{5/3}\log n)$-time deterministic multiplicative 2-approximation algorithm for the girth.*

Theorem 1.2 stems from a more general approximation result:

THEOREM 1.3. *Let G be a graph with unknown girth $g = 4c - z$ for some $c \geq 1$ and $z \in \{0, 1, 2, 3\}$. Then, there is an $O(n^{5/3}\log n)$-time deterministic algorithm which returns a $6c - z$-cycle if the girth is even, and a $6c - z + 1$-cycle if the girth is odd.*

Theorem 1.3 returns a cycle of length at most $3g/2 + z/2$ for even g and $3g/2 + z/2 + 1$ for odd g. This is almost a 1.5-approximation, but not quite. Theorem 1.3 implies an approximation algorithm for the shortest even cycle as well, giving the first subquadratic time $(2 - \varepsilon)$-approximation for the problem.

COROLLARY 1.1. *There is an $O(n^{5/3}\log n)$-time deterministic algorithm that returns an even cycle whose length is a multiplicative 5/3-approximation for the shortest even cycle.*

Multiplicative $(2 - \varepsilon)$-approximations. A natural question is, can one obtain a subquadratic time $(2-\varepsilon)$-approximation for the girth, or is there an $\Omega(n^2)$ time barrier?

For graphs of girth at least 6, Theorem 1.3 already gives a $(2 - \varepsilon)$-approximation for the girth in subquadratic time. Through a combination of Theorems 1.1 and 1.3, we obtain subquadratic time $(2 - \varepsilon)$-approximations for all graphs of girth at least 4 (also called triangle-free). Theorem 1.4 below is a special case of Theorems 5.1 and 5.2 in Section 5.

THEOREM 1.4. *There is a deterministic $O(n^{1.996})$-time multiplicative (8/5)-approximation algorithm for the girth in triangle-free graphs.*

Hence, the main difficulty in obtaining a subquadratic time $(2-\varepsilon)$-approximation for all undirected

graphs is the case of finding cycles of length less than 6 in graphs of girth 3. We conjecture that this problem requires essentially quadratic time. We are not able to prove this conjecture at this time. However, in Section 6 we explain why it is difficult to extend our particular approach to obtain a subquadratic-time $(2-\varepsilon)$-approximation algorithm for graphs with girth 3: extending the approach would require a truly subcubic time combinatorial algorithm for BMM.

Theorem 1.4 is interesting in itself, as it implies a concrete separation between the girth approximation problems in undirected and directed graphs.

THEOREM 1.5. *Let $\varepsilon > 0$. There is no deterministic $o(n^2)$-time multiplicative $(2-\varepsilon)$-approximation for the girth in triangle-free directed graphs, or in directed graphs with girth $\geq C$, for any constant C.*

There are well-known differences between directed and undirected graphs, when it comes to cycles. Bondy and Simonovits [7] showed that dense enough graphs contain short cycles of even length. This led to the $O(n^2)$-time algorithms of Yuster and Zwick [23] that can determine whether an undirected graph contains a $2k$-cycle for any constant k. The best known running time for finding fixed length cycles in directed graphs, on the other hand, is $O(n^\omega)$, where $\omega < 2.376$ is the matrix multiplication exponent [8].

It is also not hard to show (Lemma 5.2 in Section 5) that any combinatorial $O(n^{2-\varepsilon})$ time c-approximation for the girth in directed graphs for any constant $c \geq 1$ would imply a subcubic time combinatorial algorithm for BMM. Hence, even Lingas and Lundell's subquadratic time (8/3)-approximation for the girth in undirected graphs can be viewed as a separation between directed and undirected graphs.

However, both of these separations are conditional. The second separation is conditioned on there being no truly subcubic time combinatorial BMM algorithm. The first separation is conditioned on $\omega > 2$. Many researchers believe that $\omega = 2$, and hence even the $O(n^2)$ time algorithm for finding given length even cycles for undirected graphs may not present a concrete separation between the two girth problems. With Theorem 1.5 we show that a result such as Theorem 1.4 is *unconditionally impossible* for directed graphs, and hence give the first concrete separation between the girth problems in directed and undirected graphs.

Related work. *Cycles of a given fixed length.* Yuster and Zwick [23] showed that any even constant length cycle can be found in $O(n^2)$ time. Alon, Yuster and Zwick [3] showed that a k-cycle in a directed or undirected graph with m edges can be found in $O(m^{2-2/k})$ time when k is even, and $O(m^{2-2/(k+1)})$ time when k is odd.

Girth of weighted graphs. Recently, we have showed in [18] that computing a minimum weight cycle in an undirected n-node graph with edge weights in $\{1,\ldots,M\}$ or in a directed n-node graph with edge weights in $\{-M,\ldots,M\}$ and no negative cycles can be efficiently reduced to finding a minimum weight *triangle* in an $\Theta(n)-$node *undirected* graph with weights in $\{1,\ldots,O(M)\}$. This resolved a longstanding open problem posed by Itai and Rodeh [13]. A direct consequence of our reductions are $\tilde{O}(Mn^\omega) \leq \tilde{O}(Mn^{2.376})$-time algorithms for computing the girth in both undirected graphs with weights in $\{1,\ldots,M\}$ and directed graphs with weights in $\{-M,\ldots,M\}$.

Lingas and Lundell [16] also presented an algorithm for undirected graphs with integer edge weights in the range $\{1,\ldots,M\}$. Their algorithm computes in $O(n^2(\log n)\log nM)$ time a multiplicative 2-approximation. Recently, Roditty and Tov [17] improved the approximation factor to 4/3 for the weighted case while keeping the running time unchanged, and also presented an $O(\frac{1}{\varepsilon}n^2\log n(\log\log n))$ time $(4/3+\varepsilon)$-approximation algorithm for undirected graphs with nonnegative real edge weights, for any $\varepsilon > 0$. This is an essentially optimal combinatorial algorithm, as any subcubic time combinatorial $(4/3-\varepsilon)$-approximation for $0 < \varepsilon \leq 1/3$ implies a subcubic time combinatorial algorithm for BMM [22].

Shortest paths. The All Pairs Shortest Paths (APSP) problem is closely related to the problem of computing the girth both in weighted and unweighted graphs. Baswana and Kavitha [6] presented a general framework for approximating APSP in weighted undirected graphs. In particular, they presented an algorithm with a running time of $\tilde{O}(m\sqrt{n}+n^2)$ that computes a multiplicative 2-approximation, and an $\tilde{O}(n^2)$-time algorithm that for every $u,v \in V$ returns a path of length $2d(u,v)+w(w,u)$, where $d(u,v)$ is the uv-distance and $w(u,v)$ is the weight of the heaviest edge on the shortest uv-path. Dor, Halperin and Zwick [10] showed that using any subcubic time $(2-\varepsilon)$-approximation algorithm for APSP in an undirected graph it is possible to multiply two Boolean matrices in subcubic time.

In a seminal paper, Thorup and Zwick [20] constructed, for every integer $k \geq 1$, an $O(n^{1+1/k})$-space data structure which, when queried about two vertices at distance d, would report a distance of at most $(2k-1)\cdot d$, in constant time. This is called a $(2k-1)$-approximate distance oracle. Thorup and Zwick showed a $\Omega(n^2)$ space lower bound for 2-approximate distance oracles. This implies that no $o(n^2)$ time algorithm can construct a 2-approximate distance oracle. Our $o(n^2)$

time 2-approximation for the girth shows that the girth approximation problem is strictly easier than approximate distance oracle construction.

2 Preliminaries

Unless stated explicitly otherwise, all graphs in our paper are connected unweighted undirected graphs that contain at least one cycle. We denote with n the number of vertices, and with m the number of edges. The c-neighborhood $N^c(v)$ of a node v is defined as all vertices at distance c from v. The extended c-neighborhood $\tilde{N}^c(v)$ of a node v is defined as all vertices at distance at most c from v. A C_k is a simple cycle on k edges.

We denote with $P(u,v)$ the set of edges on a simple path between u and v. We denote with $P_s(u,v)$ the set of edges on the path between u and v in the BFS tree rooted at s. BFS-cycle(s) is the algorithm which runs BFS from vertex s until, by following some edge (u,v), a vertex v is visited a second time. In this case BFS-cycle returns the simple cycle enclosed by $P_s(x,u) \cup P_s(x,v) \cup \{(u,v)\}$, where x is the least common ancestor of u and v in the BFS tree rooted at s.

Below we present several techniques that will be used through the paper together with several new ideas to obtain our new results.

LEMMA 2.1. ([13], [16]) *For every vertex $u \in V$, BFS-cycle(u) runs in $O(n)$ time. If a vertex v is at distance ℓ from a vertex u, and there is a simple cycle of length k going through v then BFS-cycle(u) reports a simple cycle of length at most to $k+2\ell$ if k is even, and $k+2\ell+1$ if k is odd.*

The next theorem states a fundamental combinatorial property of even cycles in undirected graphs proven by Bondy and Simonovits [7].

THEOREM 2.1. ([7]) *Let $\ell \geq 2$ be an integer and let $G(V,E)$ be an undirected graph with n vertices and m edges, where $m \geq 100\ell n^{1+1/\ell}$. For every $k \in [\ell, n^{1/\ell}]$, there is a C_{2k} in G.*

Yuster and Zwick [23] presented an algorithm BFS-even-cycle(v) for finding shortest even cycles. BFS-even-cycle(v) is a clever modification of BFS that traverses the BFS tree rooted at v until one of 3 conditions is fulfilled by an edge which visits an already visited vertex. Each condition ensures that the edge which satisfies the condition closes an even cycle. Yuster and Zwick prove the following about their algorithm.

LEMMA 2.2. ([23]) *There exists an $O(n)$-time algorithm BFS-even-cycle(v), such that*

- *if a cycle C is found while scanning the neighbors of a node at level ℓ of the BFS tree rooted at v, then $|C| \leq 2\ell + 2$, and*
- *if v is a vertex on a minimum even cycle, then BFS-even-cycle(v) finds a minimum even cycle.*

We prove the following additional lemma about BFS-even-cycle. For the details of Yuster and Zwick's algorithm, see [23], pages 10 and 11.

LEMMA 2.3. *If a vertex u is at distance ℓ from a vertex s, and there is a simple cycle of length $2k$ going through u then BFS-even-cycle(s) reports in $O(n)$ time a simple even cycle of length at most to $2k + 2\ell$.*

Proof. Let $\bar{u}$ be the node of C at largest distance from s. Clearly, $d(s,\bar{u}) \leq d(u,\bar{u}) + \ell$. Notice also that $d(u,\bar{u}) \leq |C|/2$. By the time all edges from level $d(s,\bar{u})$ of the BFS tree from s are relaxed by BFS-even-cycle, an even cycle will definitely be found as all edges from nodes of C will have been relaxed. Now, there are three cases:

1. If $d(s,\bar{u}) \leq d(u,\bar{u}) + (\ell - 1)$, then by Lemma 2.2, the length of the cycle found is at most $2 + 2d(s,\bar{u}) \leq 2 + 2\ell - 2 + 2d(u,\bar{u}) \leq 2\ell + |C|$.

2. If $d(s,\bar{u}) = d(u,\bar{u}) + \ell$ and $d(u,\bar{u}) \leq |C|/2 - 1$, then by Lemma 2.2, the length of the cycle found is at most $2 + 2d(s,\bar{u}) \leq 2\ell + |C|$.

3. If $d(s,\bar{u}) = d(u,\bar{u}) + \ell$ and $d(u,\bar{u}) = |C|/2$, then consider the two neighbors u_1, u_2 of $\bar{u}$ in C. We must have that $d(u,u_1) = |C|/2 - 1$ and $d(u,u_2) = |C|/2 - 1$ and hence $d(s,u_1) \leq |C|/2 + \ell - 1$ and $d(s,u_2) \leq |C|/2 + \ell - 1$. Therefore, an even cycle will in fact be found when scanning the neighbors of u_1 and u_2, i.e. before the first edge from a node at level $|C|/2 + \ell$ is relaxed. Thus by Lemma 2.2, the length of the cycle found is at most $2 + 2(|C|/2 + \ell - 1) = 2\ell + |C|$.

The following lemma is a standard greedy hitting set argument appearing e.g. in [1]. We need the more precise running time for our subquadratic time algorithms.

LEMMA 2.4. *Let $T_1, \ldots, T_t$ be t sets over n elements U, so that for every $i \leq t$, $|T_i| = R$. Then there is a $O(tR \log n + (n/R) \log t)$ time deterministic algorithm which produces a set S of size $O((n/R) \log n)$ such that for every $i \leq t$, $T_i \cap S \neq \emptyset$.*

Proof. Consider the following algorithm:

Let $S = \emptyset$. In $O(tR)$ time compute for every $u \in U$ the number $t(u) = |\{i \mid u \in T_i\}|$. Insert $(u, t(u))$ in a heap H, sorted by $t(u)$. Extract $(u, t(u))$ with maximum $t(u)$ and insert $u \in S$. For every T_i such that $u \in T_i$, and for every $x \in T_i$, decrease the key $t(x)$ by 1. Remove T_i. Remove u from U.

Let t_j be the number of sets left after j elements are placed in S. At this point, the number of elements left in U is $n-j$. Consider the maximum key element $(u, t(u))$ in H. Note that $t(u) \geq Rt_j/(n-j)$ as every remaining set T_i contains R elements. Hence, after removing the sets that u covers, we have $t_{j+1} \leq t_j(1 - R/(n-j))$.

Since for $j < j'$, $(1 - R/(n-j)) > (1 - R/(n-j'))$, we have that $t_j \leq t \cdot (1 - R/n)^j$. Hence when $(n/R) \ln t$ elements are placed in S, the number of sets left is $\leq t(1 - R/n)^{n/R \ln t} < t(1/e)^{\ln t} = 1$, and hence all sets must have been hit.

The running time is determined by the number of extractions from H, which is $O(n/R \ln t)$, and the runtime due to the decreaseKey operations. Notice that each decreaseKey operation is due to an element occurrence in a set, and each such occurrence causes at most one decreaseKey operation, since it is removed after it has been processed. Hence the number of decreaseKey operations is at most $O(tR)$, as $|T_i| = R$ for each i. Each decreaseKey operation costs $O(\log n)$ time and hence the entire runtime is $O(tR \log n)$.

Finally, the following is a simple lemma which says that finding two distinct paths between a pair of vertices is sufficient to find a cycle.

LEMMA 2.5. *Let $x, y \in V$. If $P(x, y)$ and $P'(x, y)$ are two different simple paths between x and y then $P(x, y) \cup P'(x, y)$ contains a simple cycle of length at most $|P(x, y)| + |P'(x, y)|$.*

Proof. Let $(a, b) \in P(x, y)$ and $(a, b') \in P'(x, y)$ be two distinct edges such that the subpaths $P(x, a)$ and $P'(x, a)$ are identical. Such edges must exist as $P(x, y) \neq P'(x, y)$. Let c be the first vertex which is an endpoint of an edge both in $P(b, y)$ and in $P'(b', y)$. Such a vertex must exist since both paths end at y. The subpaths $P(a, c)$ and $P'(a, c)$ are internally vertex disjoint. Moreover, since $b \neq b'$ at least one of these paths is of length at least 2. Hence the two paths enclose a simple cycle of length at least 3 and at most $|P(x, y)| + |P'(x, y)|$.

3 Additive approximation

Reminder of Theorem 1.1. *Let $G(V, E)$ be an unweighted undirected graph and let $|V| = n$ and $|E| = m$. There is an $\tilde{O}(n^3/m)$ time algorithm that finds, with high probability*[1]*, a cycle in G whose length is an additive 3-approximation for the girth of G. Moreover, there is an $\tilde{O}(n^3/m)$ time algorithm that finds, with high probability, an even cycle in G whose length is an additive 2-approximation of the length of the shortest even cycle in G.*

Proof. Suppose first that $m < 101(1 + 1/\varepsilon)n^{1+\varepsilon}$ for $\varepsilon \geq \log\log n / \log n$. This implies that $m \leq O(n \log^2 n / \log\log n)$. For the minimum length cycle we use Itai and Rodeh's [13] $O(n^2)$ time algorithm to find an additive 1-approximation of the girth. For the shortest even cycle we use Yuster and Zwick's [23] $O(n^2)$ time algorithm to find the exact shortest even cycle. Both algorithms runs in $O((n^3/m) \log^2 n / \log\log n)$ time.

Suppose now that $m \geq 101(1 + 1/\varepsilon)n^{1+\varepsilon}$ for $\varepsilon \geq \log\log n / \log n$. Let $k' = \lceil \varepsilon^{-1} \rceil$. Since $\varepsilon^{-1} \in [k'-1, k']$ it follows that $m \geq 101k' n^{1+1/k'}$. Moreover, there exists a $k \leq O(\log n / \log\log n)$ such that $101(k+1)n^{1+1/(k+1)} \leq m < 101kn^{1+1/k}$. It follows from Theorem 2.1 that any graph with at least $100(k+1)n^{1+1/(k+1)}$ edges contains a C_{2k+2}. Hence, if we pick a random edge $(u, v) \in E$ the probability that it is part of a C_{2k+2} is at least $1/100$. If we independently pick $O(\log n)$ random edges, the probability that none of them is part of a C_{2k+2} is $1/\text{ poly}(n)$.

Now, for each picked edge (u, v), using Algorithm BFS-cycle either from u or from v we can find a cycle of length at most $2k+2$, in $O(n)$ time per edge, $O(n \log n)$ time overall. If this is not an additive 3-approximation of the girth, then the girth is at most $2k - 2$.

In the case of shortest even cycle we use Lemma 2.2 to find an even cycle of length at most $2k + 2$ in $O(n)$ time per edge. If this is not an additive 2-approximation for the shortest even cycle, then the shortest even cycle has length of at most $2k - 2$.

We now handle the case that the shortest (even) cycle is of length at most $2k - 2$. Let Δ be a degree parameter. We refer to vertices of degree at most Δ as low degree vertices and vertices of degree strictly more than Δ as high degree vertices. For every low degree vertex v we compute the first $k-1$ levels of the BFS tree rooted at v in the induced graph of low degree vertices. This search runs in $O(n\Delta^{k-1})$ time and either finds a (even) cycle of length at most $2k - 2$, or determines that the shortest (even) length cycle through low degree vertices is of length at least $2k$.

The only remaining case is when the shortest (even) cycle is of length at most $2k - 2$ and at least one of its vertices is a high degree vertex. We sample a set S of $O(n/\Delta \log n)$ vertices uniformly at random. Let v be a high degree vertex on the minimum length cycle. With

[1] *High probability* means probability at least $1 - 1/\text{ poly}(n)$.

high probability, S contains a neighbor of v. (The set S can also be found deterministically in $O(n\Delta \log n)$ time by Lemma 2.4 by letting the element set be V and the sets be Δ-size subsets of the neighborhoods of all high degree vertices in the graph. However, since the first part of the algorithm is randomized, the full algorithm remains randomized.)

For the minimum length cycle we run BFS-cycle from each vertex of S. If there is a minimum length cycle of length at most $2k-2$ with at least one high degree vertex, then Lemma 2.1 implies that the shortest among all cycles that the algorithm finds is an additive 3-approximation for the shortest cycle. The running time is $O(n^2 \log n/\Delta)$.

For the shortest even length cycle we run BFS-even-cycle from each vertex of S. If there is a shortest even length cycle of length at most $2k-2$ with at least one high degree vertex, then Lemma 2.3 implies that the shortest among all cycles that the algorithm finds is an additive 2-approximation for the shortest even length cycle. The running time is $O(n^2 \log n/\Delta)$.

In both cases, the runtime of the algorithm is $O(n^2 \log n/\Delta + n\Delta^{k-1})$. To minimize the runtime, we set $\Delta = (n \log n)^{1/k}$ and obtain a running time of

$$O(n^3(\log n)^{1-1/k}/n^{1+1/k}) \leq O(n^3 k(\log n)^{1-1/k}/m) <$$

$$O((n^3/m)\log^2 n/\log\log n).$$

Notice that the only randomized part of the algorithm is to find a $2(k+1)$-cycle in a graph with at least $\Omega(n^{1+1/(k+1)})$ edges. Here we show that for graphs with a superlinear number of edges, one can obtain a deterministic additive approximation algorithm. We utilize an algorithm by Alon, Yuster and Zwick [3] which can find any fixed length $2k$-cycle in $O(m^{2-2/k})$ time in a graph with m edges, if such a cycle exists.

THEOREM 3.1. *There is an $O(n^{2-2/(k+1)^2})$ time deterministic algorithm which computes an additive 3-approximation for the girth in graphs with at least $100(k+1)n^{1+1/(k+1)}$ edges, for any integer $k \geq 1$.*

We note that a slightly better running time dependence can be obtained by using Alon, Yuster and Zwick's algorithms [3] for finding even cycles in sparse undirected graphs which run in $O(m^{2+1/(2t^2)-1/(2t)})$ time for $(4t-2)$-cycles and in $O(m^{2+1/(2t+1)-1/t})$ time for $4t$-cycles. However, the runtime dependence is not as clean. Furthermore, our main use of Theorem 3.1 is in the proof of Theorem 5.2, the runtime of which would not be affected if we use the even cycle algorithms of [3], as it involves finding odd cycles as well.

Proof. We show how to find a $2(k+1)$ cycle in a graph with at least $100(k+1)n^{1+1/(k+1)}$ edges deterministically. Pick $100(k+1)n^{1+1/(k+1)}$ arbitrary edges. By Theorem 2.1 the induced subgraph contains a $2(k+1)$-cycle. Now, we use the deterministic algorithm of Alon, Yuster and Zwick [3] to find a $2(k+1)$ cycle in $O((n^{1+1/(k+1)})^{2-2/(k+1)}) = O(n^{2((k+1)^2-1)/(k+1)^2}) = O(n^{2-2/(k+1)^2})$ time. The rest of the algorithm in Theorem 1.1 runs in $O(n^3(\log n)^{1-1/k}/n^{1+1/k}) = \tilde{O}(n^{2-1/k})$ time. Since $2-1/k \leq 2-2/(k+1)^2$ for any value of k, the running time is dominated by the time to find the $2(k+1)$-cycle, and is $O(n^{2-2/(k+1)^2})$.

4 Multiplicative approximation for general graphs

Reminder of Theorem 1.3. *Let G be a graph with unknown girth $g = 4c - z$ for $c \geq 1$ and $z \in \{0,1,2,3\}$. Then, there is an $O(n^{5/3}\log n)$-time deterministic algorithm which returns a $6c-z$-cycle if the girth is even, and a $6c-z+1$-cycle if the girth is odd.*

Proof. Let C be the shortest cycle of G and recall that $g = |C|$. Let $g = 4c - z$ for $c \geq 1$ and $z \in \{0,1,2,3\}$, i.e. $z = -g \bmod 4$. In the proof below we will use c, however, the algorithm itself does not need to know c.

Let R be a parameter to be fixed later. For every node $v \in V$, run the regular BFS algorithm from v until the BFS tree T_v has R vertices.

First, use Lemma 2.4 to deterministically, in $O(nR\log n)$ time, find a set S of $O((n/R)\log n)$ vertices which hit every T_v with $|T_v| = R$. For every $s \in S$, run BFS-cycle(s), and let C_s be the cycle reported from s, if any. This takes $O((n^2/R)\log n)$ time.

Suppose that there exists a node v on the shortest cycle C, so that there are more than R vertices at distance at most c from v, that is, $|\tilde{N}^c(v)| > R$. Since, $|T_v| = R$ it follows that $T_v \subseteq \tilde{N}^c(v)$. Then S hits this c-neighborhood in some node s, and, by Lemma 2.1, $|C_s| \leq |C| + 2c$ when $|C|$ is even, and $|C_s| \leq |C| + 2c + 1$ if $|C|$ is odd.

Now suppose that every vertex v on C has $|\tilde{N}^c(v)| \leq R$. Then for every vertex v on C, $\tilde{N}^c(v)$ is completely contained in T_v. As a first stage, check for each $v \in V$ whether any two neighbors of v in T_v share an edge. If this is the case, a triangle has been found, and it must be the shortest cycle. If a triangle has not been found, then in the remainder of the proof we assume that $|C| \geq 4$. This step takes $O(nR^2)$ time.

We now show how to find C when $|C| \geq 4$. For each $v \in V$, we compute the tree distances $d_v(x,y)$ in T_v between all pairs of nodes (x,y) of T_v, and determine the pairs (x,y) for which v is the least common ancestor in T_v. This entire step takes $O(nR^2)$ time since each T_v

has at most R vertices. For every vertex $x \in T_v \setminus \{v\}$, let $pr_v(x)$ be the predecessor of x in T_v.

Consider the pairs of vertices (x, y) for which $x, y \in T_v \setminus \{v\}$ for some v; there are at most $O(nR^2)$ such pairs. For each such pair, create a list Q_{xy}. For every v and $x, y \in T_v \setminus \{v\}$ for which v is the least common ancestor of x and y in T_v, add a 4-tuple $\langle v, d_v(x, y), pr_v(x), pr_v(y)\rangle$ to Q_{xy}.

Sort each Q_{xy} in nondecreasing order of $d_v(x, y)$. Notice that since there are at most R vertices in each tree, the size of $\cup_{(x,y)\in V\times V} Q_{xy}$ is $O(nR^2)$. Thus, the total time required for building and sorting all Q_{xy} is $O(nR^2 \log n)$.

We now color each vertex in the graph independently uniformly at random red or blue; this coloring will later be derandomized without much runtime loss.

Split Q_{xy}, while keeping its relative order, into the lists Q_{xy}^{RED} and Q_{xy}^{BLUE}, where Q_{xy}^{RED} (Q_{xy}^{BLUE}) includes all 4-tuples $\langle a, d_a(x, y), pr_a(x), pr_a(y)\rangle$ of Q_{xy} for which both $pr_a(x)$ and $pr_a(y)$ are colored red (blue). (We ignore all 4-tuples for which $pr_a(x)$ and $pr_a(y)$ are colored differently.)

Any two 4-tuples, $\langle a, d_a(x, y), pr_a(x), pr_a(y)\rangle \in Q_{xy}^{\text{RED}}$ and $\langle b, d_b(x, y), pr_b(x), pr_b(y)\rangle \in Q_{xy}^{\text{BLUE}}$, correspond to two simple paths P_a and P_b between x and y, going through a and b, respectively. Since $pr_a(x) \neq pr_b(x)$, P_a and P_b must differ in at least one edge. Lemma 2.5 implies that $P_a \cup P_b$ contains a simple cycle of length $\leq d_a(x, y) + d_b(x, y)$ which can be found greedily.

We now show that if C is the shortest cycle and $|C| \geq 4$, then there are two nodes x, y on C and two 4-tuples $\langle a, d_a(x, y), pr_a(x), pr_a(y)\rangle$ and $\langle b, d_b(x, y), pr_b(x), pr_b(y)\rangle$ in Q_{xy} for which $|C| = d_a(x, y) + d_b(x, y)$, and $\{pr_a(x), pr_a(y)\} \cap \{pr_b(x), pr_b(y)\} = \emptyset$, so that, with constant probability, one tuple is in Q_{xy}^{RED}, and the other in Q_{xy}^{BLUE}.

To prove this, consider $a, b, x, y \in C$, which appear in the order a, x, b, y, a on C, so that

- x is $c - \lceil z/3\rceil \geq 1$ cycle edges after a,
- b is $c \geq 1$ cycle edges after x,
- y is $c - \lfloor z/3\rfloor \geq 1$ cycle edges after b, and
- a is $c - \lfloor z/2\rfloor \geq 1$ cycle edges after y.

Notice that since $|C| \geq 4$ these vertices exist and are distinct. We denote the paths along C between these vertices with $C(a, x)$, $C(x, b)$, $C(b, y)$ and $C(y, a)$, respectively. Figure 1 shows the choice of x, y, a, b both in general and in the special cases of C_7, C_9 and C_{10}.

Consider the paths $P_a(a, x)$ and $P_a(a, y)$ in T_a between a and x and a and y. Suppose that $C(a, x) \neq P_a(a, x)$. Then by Lemma 2.5, G has a simple cycle C'

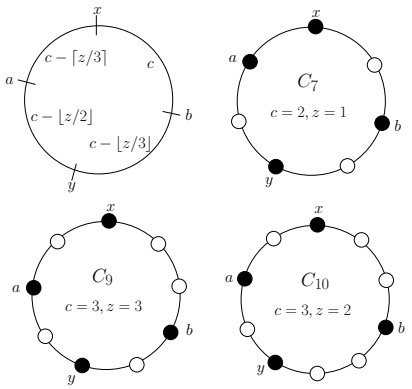

Figure 1: Example of 4 nodes on a cycle.

of length at most $|C(a, x)| + |P_a(a, x)|$. Since $P_a(a, x)$ is a shortest path, $|P_a(a, x)| \leq |C(a, x)| \leq c$ and we get that $|C'| \leq 2c$. Moreover, since $C(a, x) \neq P_a(a, x)$, $C(a, x)$ must be of length at least 2 and hence $c \geq 2$. Thus, $|C| = 4c - z \geq 2c + (2c - 3) \geq 2c + 1 > |C'|$. This is a contradiction to C being a shortest cycle. Hence, $C(a, x) = P_a(a, x)$. An analogous argument shows that $P_a(a, y) = C(y, a), P_b(b, x) = C(x, b), P_b(b, y) = C(b, y)$. This implies that $d_a(x, y) = |C(a, x)| + |C(y, a)|$ and $d_b(x, y) = |C(x, b)| + |C(b, y)|$. Moreover, since $C(y, a) \cap C(a, x) = \{a\}$ and $C(x, b) \cap C(b, y) = \{b\}$, a and b are the least common ancestors of x and y in T_a and T_b, respectively. We conclude that the two 4-tuples $\langle a, d_a(x, y), pr_a(x), pr_a(y)\rangle$ and $\langle b, d_b(x, y), pr_b(x), pr_b(y)\rangle$ are added to Q_{xy}.

We now show that $\{pr_a(x), pr_a(y)\} \cap \{pr_b(x), pr_b(y)\} = \emptyset$. We have that $pr_a(x) \in C(a, x) \setminus \{x\}$, and $pr_b(x) \in C(x, b) \setminus \{x\}$, $pr_b(y) \in C(b, y) \setminus \{y\}$ and $pr_a(y) \in C(y, a) \setminus \{y\}$. Because of our choice of a, b, x, y, we have that $C(a, x) \cap C(x, b) = \{x\}$ and hence $pr_a(x) \neq pr_b(x)$. We also have that $C(a, x) \cap C(b, y) = \emptyset$ since a, b, x, y are distinct vertices. Thus, $pr_a(x) \neq pr_b(y)$. Similarly, $C(y, a) \cap C(b, y) = \{y\}$ implies that $pr_a(y) \neq pr_b(y)$, and $C(y, a) \cap C(x, b) = \emptyset$ implies that $pr_a(y) \neq pr_b(x)$.

Hence,there is a probability of 1/8 that one of these tuples will be in Q_{xy}^{RED} and the other will be Q_{xy}^{BLUE}.

The actual cycle will be found as follows. After the coloring and the splitting, take the first element of Q_{xy}^{RED}, $\langle u, d_u(x, y), pr_u(x), pr_u(y)\rangle$, and the first element $\langle w, d_w(x, y), pr_w(x), pr_w(y)\rangle$ of Q_{xy}^{BLUE}. Since any pair of such tuples corresponds to a cycle whose length is in the range $[|C|, d_u(x, y) + d_w(x, y)]$ when the tuples from the shortest cycle $\langle a, d_a(x, y), pr_a(x), pr_a(y)\rangle$ and $\langle b, d_b(x, y), pr_b(x), pr_b(y)\rangle$ are in Q_{xy}^{RED} and Q_{xy}^{BLUE}, respectively, they will be first in both lists as otherwise the minimality of C is contradicted. Hence, with constant

probability, we can find the shortest cycle after considering, in $O(nR^2)$ time, all pairs of minimum elements of Q_{xy}^{RED} and Q_{xy}^{BLUE}, over all x, y, finding the pair which minimizes $d_u(x,y)+d_w(x,y)$ over all x, y and recovering the cycle from it.

The color-coding portion of the algorithm can be efficiently derandomized, using the ideas of Alon, Yuster and Zwick [2]. Let $F = \{f_1, \ldots, f_{|F|}\}$ a k-perfect hash family of hash functions from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$ so that for every $V' \subset V$ with $|V'| = k$, there exists some i so that f_i maps the elements of V' to distinct colors. In our case, it suffices to map $pr_a(x)$ and $pr_b(x)$ to different colors and $pr_a(y)$ and $pr_b(y)$ to different colors, and so $k = 2$. By enumerating through the functions of F, and using each f_i in place of the random coloring, our algorithm runs in $O(nR^2(\log n + |F|))$ time, provided each f_i can be evaluated in constant time.

Schmidt and Siegel [19] (following Fredman, Komlos and Szemeredi [12]) gave an explicit construction of a k-perfect family in which each function is specified using $O(k) + 2\log\log n$ bits. For our case of $k = 2$, the size of the family is therefore $O(\log^2 n)$. The value of each one of the hash functions on each specified element of V can be evaluated in $O(1)$ time. Alon, Yuster and Zwick [2], reduced the size of the hash family to $O(\log n)$. Using this family we can derandomize this part of our algorithm so that it runs in deterministic $O(nR^2 \log n)$ time.

The overall running time becomes $O((nR^2 + n^2/R)\log n)$ which is minimized for $R = n^{1/3}$ and is $O(n^{5/3}\log n)$.

Notice that for all cycle lengths $4c - z$, the above algorithm returns a cycle of length $\leq 6c-z+1 \leq 8c-2z$ since $2c \geq z+1$ whenever $c \geq 2$, and when $c = 1$, z must be ≤ 1, and so again $2c \geq z + 1$. Thus we have:

Reminder of Theorem 1.2. *There is an $O(n^{5/3}\log n)$ time deterministic multiplicative 2-approximation algorithm for the girth.*

This resolves the open problem raised by Lingas and Lundell [16].

5 Multiplicative approximation algorithms for triangle-free graphs

Theorem 1.3 already gives a better-than-2-approximation for graphs whose girth is of even length, or of odd length at least 7. Theorem 1.1, on the other hand, gives a better-than-2-approximation for all graphs with girth at least 4 (i.e. triangle-free graphs). However, its running time is subquadratic only for dense enough graphs. In this section we show how to obtain the first subquadratic, better-than-2-approximation algorithm for triangle-free graphs. We show how to prove Theorem 5.1, by combining Theorem 1.3 and Theorem 1.1, together with the algorithm of Alon, Yuster and Zwick [3] for finding given length cycles in sparse graphs.

THEOREM 5.1. *For every integer $c \in [3, 7]$, there is an $\tilde{O}(n^{24(c-2)/(12c-23)} + n^{5/3})$-time randomized algorithm which returns, with high probability, a multiplicative $\max\{8/5, (6c-2)/(4c-3)\}$-approximation of the girth in triangle-free graphs.*

Hence, the following randomized multiplicative approximation algorithms exist for triangle-free graphs:

- *an $O(n^{1.875})$-time,* 1.778*-approximation,*
- *an $O(n^{1.929})$-time,* 1.693*-approximation,*
- *an $O(n^{1.95})$-time,* 1.648*-approximation,*
- *an $O(n^{1.962})$-time,* 1.62*-approximation,*
- *an $O(n^{1.969})$-time,* 1.6*-approximation,*

In order to prove Theorem 5.1 we first need several facts about the approximation ratios that Theorem 1.3 gives us.

CLAIM 1. *The approximation ratios of Theorem 1.3 decrease with increasing c while keeping z fixed, or with decreasing z while keeping $c \geq 2$ fixed.*

Proof. If z is odd, the approximation ratio is $(6c - z + 1)/(4c - z) = (6c - 1.5z + 0.5z + 1)/(4c - z) = (3/2)+(z+2)/(2(4c-z))$. If z is even, the approximation ratio is $(6c-z)/(4c-z) = (6c-1.5z+0.5z)/(4c-z) = (3/2) + z/(2(4c-z))$. Hence, in both cases increasing c while keeping z fixed decreases the ratio.

Consider decreasing z by 1 while keeping $c \geq 2$ fixed. If z is originally odd, then the ratio is reduced from $(3/2)+(z+2)/(2(4c-z))$ to $(3/2)+(z-1)/(2(4c-z+1))$. If z is originally even, the ratio is changed from $(3/2) + z/(2(4c-z))$ to $(3/2) + (z+1)/(2(4c-z+1))$. Since $c \geq 2$ and $z \leq 3$, we have that $z \leq (4c - z)$, $z(4c - z + 1) \leq (z + 1)(4c - z)$, and hence $(3/2) + z/(2(4c-z)) \leq (3/2) + (z+1)/(2(4c-z+1))$.

CLAIM 2. *Theorem 1.3 returns an $(6c - 2)/(4c - 3)$-approximation when the girth is $4c - 6$.*

Proof. When the girth is $4c-6$, the approximation ratio is $(6c - 8)/(4c - 6)$. Consider any $c \geq 2$. Then, $4 \leq 2c$, and so, $24c^2+24-(32+18)c \leq 24c^2+12-(36+8)c$, and so $(6c - 8)(4c - 3) \leq (6c - 2)(4c - 6)$. Therefore, $(6c - 8)/(4c - 6) \leq (6c - 2)/(4c - 3)$.

CLAIM 3. *If the girth of the graph is at least $4c - 6$ for some $c \geq 2$, then Theorem 1.3 guarantees an approximation ratio of at most $(6c - 2)/(4c - 3)$.*

Proof. By Claim 2, when the girth is $4c-6 = 4(c-1)-2$, the ratio is at most $(6c-2)/(4c-3)$. By Claim 1, when we fix $c' = (c-1)$ and decrease z from 2 to 0, we get that for any girth $4(c-1)-z$ for $z \in \{0,1,2\}$, the ratio is at most $(6c-2)/(4c-3)$. Theorem 1.3 guarantees an approximation ratio of at most $(6c-2)/(4c-3)$ for girth $(4c-3)$ as well. Now, fix any $z \in \{0,1,2,3\}$. By Claim 1, for any $c' \geq c$, when the girth is $4c'-z$, the approximation ratio is still at most $(6c-2)/(4c-3)$. This covers all values of the girth that are at least $4c-6$.

Now we are ready to prove Theorem 5.1.

Proof of Theorem 5.1. We will show that for every integer $c \geq 2$, there is an $\tilde{O}(n^{24(c-2)/(12c-23)} + n^{5/3})$-time multiplicative $\max\{8/5, (6c-2)/(4c-3)\}$-approximation for the girth in triangle-free graphs. The running times in the theorem statement follow by evaluating the approximation ratios and running times for $3 \leq c \leq 7$.

Use the $\tilde{O}(n^{5/3})$-time algorithm from Theorem 1.3. If the minimum cycle happens to have length at least $4c-6$, by Claim 3 we get that this step guarantees a $(6c-2)/(4c-3)$-approximation.

Now suppose that the minimum cycle has length $\leq 4c-7$. We handle this case in two different ways. Pick a parameter M to be set later. Let m denote the number of edges in the graph. If $m \geq M$, then use the $\tilde{O}(n^3/m)$-time algorithm from Theorem 1.1 to find an additive 3-approximation if the girth is odd and an additive 2-approximation if the girth is even. When the graph is triangle-free, this is at worst an 8/5-approximation.

If $m < M$, then for each value $k \leq 4c-7$, try to find a C_k explicitly using the $O(m^{2-2/(k+1)})$-time algorithm from Alon, Yuster and Zwick [3]. This takes $O(m^{2-2/(4c-6)})$ time. If a cycle is found explicitly, the shortest cycle found has length exactly the girth. If a cycle is not found, then the step which uses Theorem 1.3 is a $(6c-2)/(4c-3)$-approximation.

We conclude that the algorithm finds a cycle whose length is a multiplicative $\max\{8/5, (6c-2)/(4c-3)\}$-approximation of the girth.

The runtime is $\tilde{O}(n^{5/3} + n^3/M + M^{2-1/(2c-3)})$. To minimize it, we set $n^3/M = M^{2-1/(2c-3)}$, and hence $M = n^{3(2c-3)/(6c-10)}$. The runtime becomes $\tilde{O}(n^{3(4c-7)/(6c-10)} + n^{5/3})$.

The only part of the algorithm in Theorem 5.1 which is randomized is in the use of Theorem 1.1 when the graph has $\geq M$ edges. Here we give a deterministic version, using Theorem 3.1 instead.

THEOREM 5.2. *For every integer $c \in [3,7]$, there is an $\tilde{O}(n^{2-2/(4c-6)^2} + n^{5/3})$-time deterministic algorithm which returns a multiplicative $\max\{8/5, (6c-2)/(4c-3)\}$-approximation of the girth in triangle-free graphs.*

Hence, the following deterministic multiplicative approximation algorithms exist for triangle-free graphs:

- *an $O(n^{1.945})$-time, 1.778-approximation,*
- *an $O(n^{1.98})$-time, 1.693-approximation,*
- *an $O(n^{1.99})$-time, 1.648-approximation,*
- *an $O(n^{1.994})$-time, 1.62-approximation,*
- *an $O(n^{1.996})$-time, 1.6-approximation.*

Proof. We show that for any $c \geq 3$, there is an $\tilde{O}(n^{2-2/(4c-6)^2} + n^{5/3})$ time deterministic algorithm which achieves a multiplicative $\max\{8/5, (6c-2)/(4c-3)\}$-approximation for the girth in triangle-free graphs.

As in Theorem 5.1 first use the algorithm from Theorem 1.3 and compute a cycle which is guaranteed to be a $(6c-2)/(4c-3)$-approximation if the girth is at least $4c-6$. Assume that the girth is at most $4c-7$, and pick $M = 100(k+1)n^{1+1/(k+1)}$ for $k = 4c-7$. If the graph has at least M edges, then use Theorem 3.1 to find an additive 3-approximation for the girth which for triangle-free graphs is at worst a multiplicative 8/5-approximation, in $O(n^{2-2/(k+1)^2}) = O(n^{2-2/(4c-6)^2})$ time. If the graph has less than M edges, then use Alon, Yuster and Zwick's algorithm to attempt to find j-cycles for each $j \in \{5, \ldots, 4c-7\}$. The running time of this algorithm is $O(M^{2-2/(4c-6)}) = O(n^{2-2/(4c-6)^2})$. Then the final runtime becomes $\tilde{O}(n^{5/3} + n^{2-2/(4c-6)^2})$. To obtain the algorithms in the statement of the theorem, we just evaluate the runtime and approximation ratios for $c = 3, \ldots, 7$.

Now we show that a result such as Theorem 5.2 is not possible for directed graphs, thus showing that girth approximation in directed graphs is truly more difficult than in undirected graphs.

Reminder of Theorem 1.5. *Let $\varepsilon > 0$. There is no deterministic $o(n^2)$-time multiplicative $(2-\varepsilon)$-approximation for the girth in triangle-free directed graphs, or in any directed graphs with girth $\geq C$, for any constant C.*

The proof of Theorem 1.5 proceeds in two steps: first show that any subquadratic time $(2-\varepsilon)$-approximation implies a subquadratic time triangle algorithm (Theorem 5.3), and then show that a deterministic subquadratic time algorithm for triangle finding cannot exist (Lemma 5.1).

THEOREM 5.3. *If for any constant $g \geq 3$, and any $\varepsilon > 0$, there is an $o(n^2)$-time multiplicative $(2-\varepsilon)$-approximation for the girth in directed graphs of girth at least g, then there is an $o(n^2)$-time algorithm which finds a triangle in an n-node graph.*

Proof of Theorem 5.3. Let $G = (V, E)$ be a directed graph on n vertices and m edges in which we want to find a triangle. Let $g \geq 3$ be any integer constant. We actually prove that any $T(m, n)$ time algorithm for $(2-\varepsilon)$-approximation of the girth implies that a triangle in G can be found in $T(O(m), O(n))$ time.

Consider the following new graph G'. Create g copies of V, $V_1, \ldots, V_g$. For every node $v \in V$, let v_i be its copy in V_i. If $g \geq 4$, add edges (v_i, v_{i+1}) for every $v \in V$ and $i \in \{2, \ldots, g-2\}$. For every $(u, v) \in E$, add edges (u_1, v_2), (u_{g-1}, v_g) and (u_g, v_1).

Now, G' has gn vertices and $3m + (g-3)n$ edges. Since G' is directed and the only edges go between V_i and V_{i+1} (where the indexing is modulo g), the girth of G' is always a multiple of g. Furthermore, G' has girth g if and only if G has a triangle. Suppose that there is an algorithm A running in subquadratic time which can find a cycle of length at most $(2-\varepsilon)g'$ where g' is the girth of G'. Run A on G' as follows. Do not form G' explicitly, but whenever A needs to access an edge in G', compute whether the edge is there, accessing G instead. (In the adjacency list representation, if A wants the k-th neighbor of v_i, just return v_{i+1} if $i \in \{2, \ldots, g-2\}$ and $k = 1$, u_{i+1} if $i \in \{1, g-1, g\}$ and u is the k-th neighbor of v in G, and no node otherwise.)

If G has a triangle, then $g' = g$, and otherwise, $g' \geq 2g$, and hence A will be able to determine which is the case.

The following lemma is a most likely folklore result. We include it for completeness.

LEMMA 5.1. *Any deterministic algorithm for triangle finding must take at least $\Omega(n^2)$ time on at least one n-node graph, for all sufficiently large n.*

Proof. Suppose that A is a deterministic algorithm which runs in $o(n^2)$ time and determines whether any n node graph contains a triangle.

Let G be the complete bipartite graph with n nodes in each partition (i.e. $K_{n,n}$). A accesses at most $o(n^2)$ edges or nonedges of G. Hence there exists a node x such that at least one incident edge to x, (x, y') is not touched, and such that there is a nonedge (x, y) which is not touched (y is in the same partition of G as x).

Since A is deterministic, A would run in the same way on the graph G' with edge set $E(G) \cup \{(x, y)\} \setminus \{(x, y')\}$, and hence would give the same answer. However G contains no triangles since it is bipartite, and G' contains at least one triangle, and A must be wrong on at least one of these graphs.

The lower bound holds for both the adjacency matrix and adjacency list representations. In the first, we assume that A has not accessed the (x, y) and (x, y') adjacency matrix entries, and in the second we assume that x is a node for which $o(n)$ neighbors were accessed and so A has not determined which one of (x, y) and (x, y') is an edge.

Finally, we show that any subquadratic time multiplicative constant approximation for the girth in directed graphs implies a subcubic time algorithm for BMM.

LEMMA 5.2. *Suppose that for some constants $c \geq 1$ and $\varepsilon > 0$, there is an $O(n^{2-\varepsilon})$ time (combinatorial) algorithm for c-approximating the girth of an n-node directed graph. Then there is an $O(n^{3-\varepsilon})$ time (combinatorial) algorithm for finding a triangle in an n-node graph, and hence an $O(n^{3-\delta})$ time (combinatorial) algorithm for BMM for some $\delta > 0$.*

Proof. Let G be an n-node directed graph in which we want to find a triangle. Without loss of generality, G is tripartite with partitions I, J, K where the edges are directed from I to J, from J to K and from K to I. For every node $i \in I$, consider the subgraph G_i of G induced by $\{i\} \cup J \cup K$. The girth of G_i is 3 if there is a triangle in G through i, and is ∞ otherwise.

Suppose that there is an $O(n^{2-\varepsilon})$ time algorithm A which can c-approximate the girth of any graph. Suppose that we can run A on G_i, for each $i \in I$. Since A computes a c-approximation, it can distinguish between girth 3 and ∞ in $O(n^{2-\varepsilon})$ time in each G_i, and hence A can compute whether G contains a triangle in $O(n^{3-\varepsilon})$ time.

We do not want to build each G_i separately, however, since that may take cubic time in itself. We show how to avoid this by reusing parts of G. Build the subgraph H of G induced by $J \cup K$ in $O(n^2)$ time. To build G_i, add a node i and edges to i from the inneighbors of i in H, and with edges from i to all outneighbors of i in H. After running A on G_i, if a triangle is not found, remove i and its incident edges from G_i so that only H remains and build G_{i+1}, etc. Constructing the graphs G_i only takes additional $O(n^2)$ time, $O(n)$ per G_i.

6 Discussion

An natural question is why our techniques in Theorem 1.3 do not allow us to get a multiplicative $(2-\varepsilon)$-approximation of the girth (for any $\varepsilon > 0$) in subquadratic time in general graphs, and in particular in graphs with girth 3.

Notice that a subquadratic time multiplicative $(2-\varepsilon)$-approximation algorithm for the girth would imply that it is possible in subquadratic time to determine whether a graph with no C_4 or C_5 contains a triangle. We conjecture that this problem requires essentially quadratic time, and is the main obstacle to obtaining a subquadratic $(2-\varepsilon)$-approximation for the girth.

CONJECTURE 1. *Determining whether a graph which does not contain a C_4 or C_5 contains a triangle requires $\Omega(n^{2-\varepsilon})$ time for all $\varepsilon > 0$.*

In the absence of a proof of the conjecture, we consider a natural extension of the approach of Theorem 1.3 and show that obtaining a subquadratic time $(2-\varepsilon)$-approximation using this approach may be difficult.

Consider the girth approximation problem in a graph G which may contain triangles. As in Theorem 1.3 we can argue that if the graph contains at least $202n^{3/2}$ edges, Bondy and Simonovits [7] (Theorem 2.1 in Preliminaries) implies that a C_4 exists, and we can find it in expected $O(n)$ time. Hence we can assume that G has at most $202n^{3/2}$ edges. On the other hand, one can also assume that every vertex has degree $\Omega(n^{1/2-\varepsilon})$ for some $\varepsilon > 0$, as if a vertex has degree $O(n^{1/2-\varepsilon})$, its neighborhood can be searched for a triangle in $O(n^{1-2\varepsilon})$ time, and so all low degree vertices can be processed and removed in $O(n^{2-2\varepsilon})$ time.

Moreover, if a triangle of G contains a vertex with degree $\Omega(n^{1/2+\varepsilon})$, since G has at most $O(n^{3/2})$ edges, there are at most $O(n^{1-\varepsilon})$ such high degree vertices, and in $O(n^{2-\varepsilon})$ time by running BFS-cycle from each of them we can find at worst a C_4, thus obtaining a 4/3-approximation.

Hence, the hard instance is an m-edge graph with $\Omega(n^{3/2-\varepsilon}) \leq m \leq 202n^{3/2}$ for any $\varepsilon > 0$ in which all node degrees are between $\Omega(n^{1/2-\varepsilon})$ and $O(n^{1/2+\varepsilon})$.

An important ingredient in our framework for approximating the girth in Theorem 1.3 (and throughout the paper) has been to find a set S of nodes of size $O((n/\Delta)\log n)$ which hits the neighborhoods of all nodes of degree at least Δ, either via random sampling, or by Lemma 2.4. Then, our approach has been to run a search for a cycle from each node of S. In the case of our hard instance, we can find a set S of $O(n^{1/2+\varepsilon}\log n)$ nodes which hits all node neighborhoods. Consider the natural extension of our approach which attempts to return a C_3, C_4 or C_5 through some node of S. We show that this problem would be difficult to solve in subquadratic time.

Consider running BFS from $s \in S$ until either a cycle is found, or all edges out of all neighbors of s are scanned. That is, either a C_4 or a C_3 is found and the algorithm can stop, or for all $s \in S$, we have computed $N^2(s)$ in overall time $O(|S|n) = O(n^{3/2+\varepsilon}\log n)$. If a C_3 or a C_4 is not found through S, then our approach would dictate to attempt to return a C_5 through a node of S. This is the same as determining whether some $N^2(s)$ contains an edge.

We have arrived at the following problem.

DEFINITION 1. **Empty-$N^2(s)$**: *Let $G = (V, E)$ be a graph with $O(n^{3/2})$ edges, $S \subseteq V$ with $|S| = L\sqrt{n}$ for some $L \geq 1$, $L \leq O(n^{\varepsilon})$ for all constant $\varepsilon > 0$, and such that there is no C_3 or C_4 through any node of S. Determine whether $\cup_{s\in S} E(N^2(s)) = \emptyset$.*

Herein lies the main obstacle. We show that solving Empty-$N^2(s)$ in truly subquadratic time using a combinatorial algorithm implies a truly subcubic combinatorial algorithm for BMM, a longstanding open problem. This seems to suggest that it may be difficult to $(2-\varepsilon)$-approximate the girth in general graphs in subquadratic time by solving Empty-$N^2(s)$.

THEOREM 6.1. *If Empty-$N^2(s)$ can be solved in $O(n^{2-\delta})$ time for some $\delta > 0$ by a combinatorial algorithm, then for some $\delta' > 0$ there is an $O(n^{3-\delta'})$-time combinatorial algorithm for $n \times n$ BMM.*

Theorem 6.1 does not rule out solving Empty-$N^2(s)$ by using fast matrix multiplication. However, the proof of Theorem 6.1 shows that even such algebraic solutions would be interesting. We show that subquadratic time Empty-$N^2(s)$ is *equivalent* to solving the following triangle detection problem in truly subquadratic time: given a tripartite graph G' with $O(n^{1.5})$ edges and partitions I_1, I_2, I_3 such that $|I_1| = |I_3| = n$, and $|I_2| = \sqrt{n}$, determine whether G' contains a triangle. We call this problem $(\sqrt{n}, n^{1.5})$-Triangle.

It is not known whether $(\sqrt{n}, n^{1.5})$-Triangle can be solved in truly subquadratic time, even by using fast matrix multiplication.

LEMMA 6.1. *If $(\sqrt{n}, n^{1.5})$-Triangle can be solved in $O(n^{2-\delta})$ time for some $\delta > 0$, then Empty-$N^2(s)$ can be solved in $O(n^{2-\delta'})$ time for some $\delta' > 0$.*

Furthermore, if one can solve Empty-$N^2(s)$ in $O(n^{2-\delta})$ time for some $\delta > 0$, then $(\sqrt{n}, n^{1.5})$-Triangle can be solved in $O(n^{2-\delta})$ time as well.

Proof. Suppose first that $(\sqrt{n}, n^{1.5})$-Triangle can be solved in $O(n^{2-\delta})$ time. Let $G = (V, E)$ be a graph with $O(n^{1.5})$ edges and let $S \subseteq V$ with $|S| \leq L\sqrt{n}$. Let's assume that $L = 1$. Otherwise, we can split the problem int L instances of the case when $|S| \leq \sqrt{n}$ by partitioning S into L parts $S_1, \ldots, S_L$ on $\sqrt{n}$ nodes and looking at the graph induced by $S_i \cup (V \setminus S)$. Then the running time will just be multiplied by L, and since $L \leq O(n^{\varepsilon})$ for all $\varepsilon > 0$, we will still get a truly subquadratic time algorithm.

Create a tripartite graph G' with partitions I_1, I_2, I_3 where $I_2 = S$ and I_1 and I_3 are copies of V. For every s and $v \in N^2(s)$, place an edge between s and each copy of v in I_1 and I_3. For every edge $(u, v) \in G$ add an edge between the copy of u in I_1 and the copy of v

in I_3 and an edge between the copy of u in I_3 and the copy of v in I_1. Now, the only triangles that G' has are of the form u, v, s where $u \in I_1$, $s \in I_2$ and $v \in I_3$. Any such triangle implies that (u, v) is an edge within $N^2(s)$. Moreover, if any $N^2(s)$ contains an edge (u, v), then s, u, v is a triangle in G'. This concludes the proof of the first part of the lemma.

Suppose now that one can determine whether $E(N^2(s)) \neq \emptyset$ for some $s \in S$ in $O(n^{2-\delta})$ time for a graph G with $O(n^{1.5})$ edges and for $|S| = O(\sqrt{n})$. Let G be an instance of $(\sqrt{n}, n^{1.5})$-Triangle. Let I_1, I_2, I_3 be the partitions of the nodes of G with $|I_2| = \sqrt{n}$, $|I_1| = |I_3| = n$. Create a new graph G' as follows. Include in G' the subgraph of G induced by $I_1 \cup I_3$. Add a set S of $\sqrt{n}$ nodes, one for each node of I_2. For every node $s \in S$ add $\sqrt{n}$ nodes $s_1, \ldots, s_{\sqrt{n}}$ and add edges between s and s_i for each $i \in \{1, \ldots, \sqrt{n}\}$.

Now consider any node $u \in I_2$. Let $v_1, \ldots, v_{deg(u)}$ be the neighbors of u in G. These neighbors can be partitioned into at most $\sqrt{n}$ disjoint groups $V_{uj} = \{v_{j\sqrt{n}+k} \mid k \in \{1, \ldots, \sqrt{n}\}\}$ for $j \geq 0$ and $j < deg(u)/\sqrt{n} \leq \sqrt{n}$.

Let s be the copy of u in S. Recall that s currently has neighbors $s_1, \ldots, s_{\sqrt{n}}$ in G'. For every $j \in \{0, \ldots, \sqrt{n} - 1\}$, add an edge between s_{j+1} and the copy in G' of every node (if any) in V_{uj}. This completes the construction of G'. Notice that G' does not contain any C_3 or C_4 through S. G' has $O(n^{1.5})$ edges and $O(n)$ nodes. $|S| = O(\sqrt{n})$, and there is an edge $(u, v) \in N^2(s)$ iff u, v, s is a triangle in G. Hence one can find a triangle in G in $O(n^{2-\delta})$ time.

Now we show that any subquadratic time *combinatorial* algorithm solving Empty-$N^2(s)$ implies a subcubic time combinatorial algorithm for BMM.

Proof of Theorem 6.1. Suppose that one can solve Empty-$N^2(s)$ in truly subquadratic time using a combinatorial algorithm. Then by Lemma 6.1 one can solve $(\sqrt{n}, n^{1.5})$-Triangle in truly subquadratic time using a combinatorial algorithm. Then take any tripartite graph G' with n nodes in each partition I_1, I_2, I_3. Partition the edge set in $I_1 \times I_3$ into $K = O(\sqrt{n})$ pieces $E_1, \ldots, E_K$ on $O(n^{1.5})$ edges each. Partition the nodes in I_2 into $K = O(\sqrt{n})$ parts $X_1, \ldots, X_K$ on $O(\sqrt{n})$ nodes each. Then for every pair (E_j, X_k), detect whether the graph formed by $X_k \times I_1$, $X_k \times I_3$ and the edges in E_j contains a triangle in $O(n^{2-\delta})$ time. Since the number of pairs (E_j, X_k) is $K^2 = O(n)$, a triangle in G' can be found in $O(n^{3-\delta})$ time. By [22], this implies that there is a combinatorial $O(n^{3-\delta'})$ time algorithm for BMM for some constant $\delta' > 0$.

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] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[3] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[4] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.

[5] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. FOCS*, pages 745–754, 2009.

[6] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Proc. FOCS*, pages 591–602, 2006.

[7] A. Bondy and M. Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory*, 16:97–105, 1974.

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

[9] A. Czumaj, M. Kowaluk, and A. Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1–2):37–46, 2007.

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

[11] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proc. FOCS*, pages 129–131, 1971.

[12] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31:538–544, 1984.

[13] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

[14] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199–243, 2009.

[15] M. Krivelevich, Z. Nutov, and R. Yuster. Approximation algorithms for cycle packing problems. In *Proc. SODA*, pages 556–561, 2005.

[16] A. Lingas and E-M. Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009.

[17] L. Roditty and R. Tov. Approximating the girth. In *Proc. SODA*, pages 1446–1454, 2011.

[18] L. Roditty and V. Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proc. FOCS*, 2011.

[19] J.P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*,

19(5):775–786, 1990.

[20] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. STOC*, pages 183–192, 2001.

[21] L. G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308–315, 1975.

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

[23] R. Yuster and U. Zwick. Finding even cycles even faster. In *Proc. ICALP*, pages 532–543, 1994.