Algorithmic trade-offs for girth approximation in undirected graphs

Avi Kadria* Liam Roditty† Aaron Sidford‡ Virginia Vassilevska Williams§
Uri Zwick¶

Abstract

We present several new efficient algorithms for approximating the girth, g, of weighted and unweighted n-vertex, m-edge undirected graphs. For undirected graphs with polynomially bounded, integer, non-negative edge weights, we provide an algorithm that for every integer $k \geq 1$, runs in $\widetilde{O}(m + n^{1+1/k} \log g)$ time and returns a cycle of length at most $2kg$. For unweighted, undirected graphs we present an algorithm that for every $k \geq 1$, runs in $\widetilde{O}(n^{1+1/k})$ time and returns a cycle of length at most $2k\lceil g/2 \rceil$, an almost k-approximation. Both algorithms provide trade-offs between the running time and the quality of the approximation. We also obtain faster algorithms for approximation factors better than 2, and improved approximations when the girth is odd or small (e.g., 3 and 4).

1 Introduction

The problem of computing the shortest cycle of a graph or its length, known as the *girth* of the graph, are fundamental problem in algorithmic graph theory that has been studied extensively since the 1970s.

Several results are known on the complexity of exactly solving these problems. Itai and Rodeh [IR78] proved that the girth g of an n-vertex, m-edge unweighted graph can be computed in $\min\{O(mn), O(n^\omega)\}$ time, where $\omega < 2.37286$ [AV21] is the matrix multiplication exponent. Further, Roditty and Vassilevska W. [RV12] extended Itai and Rodeh's result, showing that for directed or undirected graphs with positive integer weights bounded by M, the girth can be computed in $\tilde{O}(Mn^\omega)$ time. For graphs with arbitrary weights and no negative cycles, the fastest known algorithms for computing the shortest cycle also solve the All-Pairs Shortest Paths (APSP) problem. Despite more than seven decades of research, the fastest algorithms for APSP and for computing the girth of weighted graphs run in $O(\min\{mn + n^2 \log\log n, n^3/\exp(\sqrt{\log n})\})$ time [Pet04, Wil18].

Improving these running times further is a notoriously difficult problem and various attempts have been made to characterize the complexity of this problem and related problems. For example, computing the girth of an unweighted graph exactly is at least as hard as deciding whether a given graph contains a triangle. The latter problem is very well-studied and the fastest known algorithm for it runs in $O(n^\omega)$ time. It is conjectured that no faster algorithm exists (see e.g. [Vas19]).

Further, a very close formal connection between computing the value of the girth, triangle detection and computing APSP was proven by Vassilevska W. and Williams [VW18]. For weighted graphs, they proved that APSP and girth are subcubical equivalent, i.e. that a truly subcubic time ($O(n^{3-c})$-time for constant $c > 0$) algorithm for either problem implies one for the other. APSP is conjectured to require $n^{3-o(1)}$ time on a word-RAM with $O(\log n)$ bit words (see e.g. [Vas19]), and since the girth problem in weighted graph is equivalent to APSP, a truly subcubic time algorithm for it would be hard to obtain.

*Department of Computer Science, Bar Ilan University, Ramat Gan 5290002, Israel. E-mail avi.kadria3@gmail.com.

†Department of Computer Science, Bar Ilan University, Ramat Gan 5290002, Israel. E-mail liam.roditty@biu.ac.il. Supported in part by BSF grants 2016365 and 2020356.

‡Departments of Management Science and Engineering and Computer Science, Stanford University, Stanford, CA, 94305, USA. E-mail sidford@stanford.edu. Supported in part by BSF grant no. 2016365, a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship

§Department of Electrical Engineering and Computer Science and CSAIL, MIT, Cambridge, MA, USA. E-mail virg@mit.edu. Supported in part by NSF CAREER Award 1651838, NSF Grants CCF-1909429 and CCF- 2129139, BSF grants 2016365 and 2020356, a Google Research Fellowship and a Sloan Research Fellowship.

¶Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 6997801, Israel. E-mail zwick@tau.ac.il. Supported in part by BSF grants 2016365 and 2020356.

Vassilevska W. and Williams [VW18] also gave a connection between girth and APSP in unweighted graphs. They proved that any "combinatorial" algorithm[1] that computes the exact girth or can detect a triangle in truly subcubic runtime implies a truly subcubic time combinatorial algorithm for multiplying two Boolean matrices, and therefore also for APSP.

A natural approach to overcome these barriers for computing the girth, is to settle for an approximation of the girth. The girth can be approximated either by an additive approximation or by a multiplicative approximation. For $c \geq 1$, a multiplicative c-approximation algorithm returns a cycle of length between g and $c \cdot g$. For $c \geq 0$, an additive c-approximation for the girth g returns a cycle of length between g and $g + c$; additive approximation for small c are typically only meaningful for unweighted graphs.

1.1 Prior work on girth approximation While there has been some recent work on approximating the girth for directed graphs [PRS+18, CLRS20, DV20], in this paper we only focus on *undirected*, possibly weighted graphs, and we restrict our consideration to such graphs for the remainder of the paper.

Itai and Rodeh [IR78] gave the first girth approximation algorithm: an $O(n^2)$ time algorithm that in unweighted undirected graphs computes a cycle of length at most $g + 1$ if the girth g is odd and g if g is even. Their algorithm is based on a simple $O(n)$-time variant of the BFS algorithm.

While $\Omega(n^2)$ time is needed for problems like APSP whose output is of size n^2, the output for the girth problem is a single number, and it is unclear if $\Omega(n^2)$ time is needed. Indeed, Lingas and Lundell [LL09] presented a multiplicative approximation algorithm for the girth g in unweighted undirected graphs that breaks the quadratic time bound. Their algorithm runs in $\widetilde{O}(n^{3/2})$ time and returns a cycle that has a length of at most $2g + 2$, which gives a multiplicative 8/3-approximation.

Lingas and Lundell [LL09] also developed an $\widetilde{O}(n^2)$ time 2-approximation algorithm for undirected graphs with positive integer weights bounded by poly(n). Roditty and Tov [RT13] and Ducoffe [Duc19] considered the same regime. More specifically, Roditty and Tov [RT13] obtained a $\widetilde{O}(n^2)$ time 4/3-approximation algorithm. They also gave for $\varepsilon > 0$, an $\widetilde{O}(n^2/\varepsilon)$ time $(4/3 + \varepsilon)$-approximation algorithm for the girth of undirected graphs with non-negative real weights. Ducoffe [Duc19] developed the first subquadratic time approximation algorithms for weighted graphs, obtaining a 2-approximation running in $\tilde{O}(m + n^{5/3})$ time for graphs with polynomially bounded integer weights, almost matching the running time of an algorithm of [RV12] for unweighted graphs. Ducoffe also presents an $(2+\varepsilon)$-approximation running in $\tilde{O}(m+n^{5/3}\text{polylog}(1/\varepsilon))$ time for graphs with arbitrary real weights, and shows that the dependence on m can be removed if the edges in all adjacency lists are given sorted by weight.

Roditty and V. Williams [RV12] presented an $O(n^{5/3} \log n)$ time algorithm that computes for an *unweighted* undirected graph of girth $g = 4c - z$ for some $c \geq 1$ and $z \in \{0, 1, 2, 3\}$ a cycle of length at most $6c - z$ if g is even, and $6c - z + 1$ if g is odd. This is never worse than a 2-approximation, and it is almost a 3/2-approximation with a slight additive error. It was the first subquadratic time 2-approximation algorithm for the girth.

Dahlgaard, Knudsen and Stöckel [DKS17b] presented two trade–offs for unweighted undirected graphs. The first is an $O(n^{2-1/k}(\log n)^{1-1/k})$-time algorithm that computes a cycle of length at most $2\lceil \frac{g}{2} \rceil + 2\lceil \frac{g}{2(k-1)} \rceil$, for any integer $k \geq 2$. This generalized the results of [LL09] (who showed it for $k = 2$) and [RV12] (who showed it for $k = 3$). The second is an $O(n^{1+1/k} \log n)$-time algorithm that computes a cycle of length at most $2^k g$, for any integer $k \geq 2$, with probability $1 - 1/n$.

In this paper we present a variety of improved algorithms with different runtime versus approximation quality trade–offs for unweighted and weighted undirected graphs.

1.2 Our Results Our first result is a time-approximation trade-off for girth approximation in unweighted graphs. We essentially obtain for every integer $k \geq 1$ a k-approximation running in $\tilde{O}(n^{1+1/k})$ time. The previous best approximation algorithm running in $\tilde{O}(n^{1+1/k})$ time [DKS17a] achieved a 2^k approximation factor. We thus obtain an *exponential* improvement.

THEOREM 1.1. *For every integer $k \geq 1$, there is an algorithm that computes a cycle C in any n-vertex unweighted undirected graph G in $O(n^{1+1/k} \log n)$ time such that $wt(C) \leq 2k \cdot \lceil g/2 \rceil$, where g is the girth of G.*

[1] While the term "combinatorial" is not well-defined in this context, we use it to denote any algorithm that avoids the impracticalities of the Strassen-like algebraic algorithms for fast matrix multiplication.

The theorem appears as Theorem 4.3 in the body of the paper.

Theorem 1.1 generalizes a result of Itai and Rodeh [IR78] which proved it for $k = 1$. Further, the theorem for $k = 2$ is similar to the 2-approximation of Roditty and Vassilevska W. [RV12]; in comparison, Theorem 1.1 has a better running time but a slightly worse approximation if the girth is odd.

We note that for graphs with $\omega(n^{1+1/k} \log n)$ edges, our algorithm is *sublinear*. To obtain this, we develop a tool for girth approximation in unweighted undirected graphs that after running in $O(\min\{n^{1+1/k}, m\})$ time, allows us to assume that $m \leq O(n^{1+1/k})$.

For known odd girth and $k > 2$ we show how to extend the techniques that yield Theorem 1.1 to obtain alternative time approximation trade–offs of interest in Appendix B.

Our next result is a trade–off for weighted graphs, generalizing [LL09] which proved the case of $k = 1$. This is the first trade–off curve for weighted graphs.

THEOREM 1.2. *Let $G = (V, E)$ be a weighted undirected graph with $|V| = n, |E| = m$, with integral edge weights from the range $[1, M]$ and with girth g For every integer $k \geq 2$, there is an $O((n^{1+1/k} \log n + m) \log nM)$ time algorithm that computes a cycle C of weight $wt(C) \leq 2k \cdot g$.*

The theorem appears as Theorem 4.1 in the body of the paper.

We note that both the $\tilde{O}(n^2)$-time 4/3-approximation algorithm of Roditty and Tov [RT13] and the $\tilde{O}(m+n^{5/3})$-time 2-approximation algorithm of Ducoffe [Duc19], have better approximations than what our trade-off would be for $k = 1$. Our theorem gives improved results for $k \geq 2$, achieving, for instance, a 4-approximation faster than both [RT13] and [Duc19].

We also obtain improved algorithms for sparse graphs when the girth is a small constant.

THEOREM 1.3. *For every $k \geq 2$, there is an $O(\min\{m, n^{1+1/k}\})$ time algorithm that either succeeds in returning a $\leq 2k$-cycle or fails, in which case $m \leq O(n^{1+1/k})$ and there is an algorithm which can find a shortest cycle or determine that the girth is $> g$ in additional time*

- $O(\min\{n^{1+2/k}, m^{1+1/(k+1)}\})$ *if $g = 3$ or $g = 4$, and*
- $O(\min\{n^{1+3/k}, m^{1+2/(k+1)}\})$ *if $g = 5$.*

The theorem appears as Corollaries 6.1 and 6.2 in the body of the paper. Theorem 1.3 implies that there is an algorithm that either determines that the girth is more than g, or returns a cycle of length $\leq 2k$, and runs in time $O(\min\{n^{1+2/k}, m^{1+1/(k+1)}\})$ time if $g = 3$ or $g = 4$ and in time $O(\min\{n^{1+3/k}, m^{1+2/(k+1)}\})$ time if $g = 5$.

Let us compare Theorem 1.3 to Theorem 1.1. For any integer $\ell \geq 2$ and girth g, Theorem 1.1 returns in $\tilde{O}(n^{1+1/\ell})$ time a cycle of length at most $2\ell\lceil g/2 \rceil$. In order for the length of this cycle to be at most $2k$, we need $\ell = \frac{k}{\lceil g/2 \rceil}$.

Notice that ℓ needs to be an integer, so that one can only get exactly the same cycle length guarantee as Theorem 1.3 with Theorem 1.1 for values of k that are divisible by $\lceil g/2 \rceil$. For these values, the running time guaranteed by Theorem 1.1 is $\tilde{O}(n^{1+\frac{\lceil g/2 \rceil}{k}})$. (For other values, the running time is slightly higher, $\tilde{O}(n^{1+1/(\lfloor (k/\lceil g/2 \rceil) \rfloor)})$.)

Thus, for $g = 3$ and $g = 4$, the running time obtained by Theorem 1.1 (for k divisible by $\lceil g/2 \rceil$), is $\tilde{O}(n^{1+2/k})$, and that for $g = 5$ is $\tilde{O}(n^{1+3/k})$. Thus, the running time from Theorem 1.3 is always at least as good, and always better for $m = o(n^{1+1/k})$. In particular, for sparse graphs, the improvement is significant.

For the special case of 2-approximation of the girth, it can be returned via Theorem 1.3 in $\tilde{O}(\min\{n^{5/3}, m^{5/4}\})$ time whenever the girth is 3, in $\tilde{O}(\min\{n^{3/2}, m^{6/5}\})$ time whenever the girth is 4, and in $\tilde{O}(\min\{n^{8/5}, m^{4/3}\})$ time whenever the girth is 5. These are the best algorithms for 2-approximation of the girth for small values of g.

We obtain a slightly weaker improvement for approximating the girth of sparse graphs when the girth is 6.

THEOREM 1.4. *There is an $\tilde{O}(\min\{nm^{3/5}, n^{7/4}\})$ time algorithm that in m-edge n-vertex graphs, either returns a cycle of length at most 8, or determines that the girth is more than 6.*

There is an $\tilde{O}(\min\{n^{19/12}\})$ time algorithm that in $m = O(n)$-edge n-vertex graphs, either returns a cycle of length at most 10, or determines that the girth is more than 6.

The theorem appears as Theorem 6.3 in the body of the paper. The first algorithm is no worse than Theorem 1.1 for $k = 4$ and improves upon its running time for sparse graphs. The second algorithm offers an improvement over Theorem 1.1 for $k = 5$ and very sparse graphs.

Our final result is an algorithm that achieves a better than 2-approximation in subquadratic time, with a small additive error.

THEOREM 1.5. *Let $G = (V, E)$ be a given n-vertex, m-edge unweighted undirected graph with unknown girth g, and let $\epsilon \in (0, 1)$ be given. There is an algorithm that computes a cycle C in $\widetilde{O}(n^{1+1/(2-\epsilon)})$ time such that $wt(C) \leq 4\lceil \frac{g}{2} \rceil - 2\lfloor \epsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\epsilon)g + 4$ if $g \leq \log^2 n$ and $wt(C) \leq 4\lceil \frac{g}{2} \rceil - \lfloor \epsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\epsilon/2)g + 3$ if $g > \log^2 n$.*

Theorem 1.5 appears as Theorem 7.1 in the body of the paper.

Notice that when $g \leq \log^2 n$ if we set $\epsilon = 1 - 1/(k-1)$, we get $4\lceil \frac{g}{2} \rceil - 2\lfloor (1 - 1/(k-1))\lceil \frac{g}{2} \rceil \rfloor = 2\lceil \frac{g}{2} \rceil - 2\lfloor (-1/(k-1))\lceil \frac{g}{2} \rceil \rfloor$. This is bounded by $2\lceil \frac{g}{2} \rceil + 2\lceil (1/(k-1))\lceil \frac{g}{2} \rceil \rceil$.

This almost matches the first trade-off presented by Dahlgaard, Knudsen and Stöckel [DKS17b]. The running time is the same and the bound on the cycle length is slightly worse when g is odd, since they get a bound of $2\lceil \frac{g}{2} \rceil + 2\lceil \frac{g}{2(k-1)} \rceil$.

The main advantage of our algorithm is that we can set ϵ to be *any real value* between 0 and 1 and not only $1 - 1/k$ for integer values of k, as in the algorithm of Dahlgaard, Knudsen and Stöckel [DKS17b]. This allows us to obtain infinitely many new algorithms.

For example, for $\epsilon = 1/3$ we get a running time of $\widetilde{O}(n^{8/5})$ and a bound of $4\lceil \frac{g}{2} \rceil - 2\lfloor \lceil \frac{g}{6} \rceil \rfloor \sim \frac{5}{3}g$. For a very small value of ϵ, such as $\epsilon = 1/101$, we get a running time of $\widetilde{O}(n^{\frac{300}{201}})$ that is close to $\widetilde{O}(n^{\frac{3}{2}})$, while keeping the multiplicative approximation better than 2, for a large enough g. Notice that in the result of Dahlgaard, Knudsen and Stöckel [DKS17b] the best running time for a better than 2 multiplicative approximation is always $\Omega(n^{5/3})$. We present in Table 20 in Appendix A a comparison between the two results, for a multiplicative approximation better than 2.

Our better-than-2 approximation algorithms are based on the following result.

LEMMA 1.1. *Let G be a given n-vertex undirected unweighted graph and let t and c be integers. Then:*

- *There is an algorithm running in time $O(n^{1+\frac{t}{2t-c}})$ that either returns a cycle of length at most $4t - c$, or determines that the girth of G is strictly more than $2t$.*
- *There is an algorithm running in time $O((t-c)n^{1+\frac{t}{2t-c}})$ that either returns a cycle of length at most $4t-2c$, or determines that the girth of G is strictly more than $2t$.*

The Lemma appears as Lemmas 7.6 and 7.7 in the body of the paper.

The second result in the Lemma is better than the first whenever $t - c$ is small. We mostly use the second result, but the first is useful for graphs with large girth.

For large enough c, the results can be viewed as additive approximations. For instance, if g is even, for $t = g/2$ and $a = t - c$, we can return a cycle of length at most $g + a$ in time $O(an^{1+\frac{g}{g+2a}})$.

1.3 Additional Related Work Bondy and Simonovits [BS74] first showed that for any integer $k \geq 2$, any n-vertex graph that has at least $100kn^{1+1/k}$ edges must contain a $2k$-cycle. Yuster and Zwick [YZ97] made this result algorithmic, providing an $O(n^2)$ time algorithm for outputting a $2k$-cycle if one exists in a graph for any constant integer $k \geq 2$. Dahlgaard, Knudsen and Stöckel [DKS17a] extended this result further to sparse graphs, providing an $O(m^{2k/(k+1)})$ time algorithm for returning a $2k$-cycle in an m-edge graph, if one exists. As $\geq 100kn^{1+1/k}$-edge graphs must contain a $2k$-cycle, the algorithm of [DKS17a] always runs in no more than $O(n^2)$ time. Alon, Yuster and Zwick [AYZ97] provide algorithms for outputting a k-cycle in a directed or undirected graph, running in time $O(m^{2-2/k})$ time for even k and $O(m^{2-2/(k+1)})$ time for odd k. Yuster and Zwick [YZ04] and Dalirrooyfard, Duong Vuong and Vassilevska W. [DVV19] improve these running times using rectangular matrix multiplication.

Pachocki et al. [PRS+18], Chechik et al. [CLRS20], Dalirrooyfard and Vassilevska W. [DV20] and Chechik and Lifshitz [CL21] consider the girth approximation problem for directed graphs. Chechik et al. [CLRS20] obtain the first constant factor approximation algorithm running faster than APSP. They obtain a trade-off, for every k, an

Algorithm 1: `BallOrCycle`(v, R)

```
Q ← {v} with key 0;
while Q ≠ ∅ do
    u ← Extract-Min(Q), k(u) gets the key of u;
    V_v^R ← V_v^R ∪ {u};
    i ← 1;
    while (i ≤ |E(u)|) and k(u) + wt(E(u, i)) ≤ R do
        (u, w) ← E(u, i) ;
        if w ∈ Q then
            return null, P(LCA(u, w), u) ∪ {(u, w)} ∪ P(LCA(u, w), w);
        Q ← Q ∪ {w} with key k(u) + wt(u, w) ;
        E_v^R ← E_v^R ∪ {(u, w)} ;
        i ← i + 1 ;
return (V_v^R, E_v^R), null;
```

$O(k \log k)$-approximation in $\tilde{O}(mn^{1/k})$ time. Dalirrooyfard and Vassilevska W. [DV20] improve upon the constant factor in the big-O of the approximation factor of [CLRS20], obtaining a $(2+\varepsilon)$-approximation algorithm running in $\tilde{O}(m\sqrt{n})$ time and a 2-approximation running in $\tilde{O}(mn^{3/4})$ time. Chechik and Lifshitz [CL21] improve this to a 2-approximation in $\tilde{O}(\min\{n^2, m\sqrt{n}\})$ time.

2 Preliminaries

Let $G = (V, E)$ be a weighted undirected graph, where $n = |V|$ and $m = |E|$. Let $wt : E \to \{1, 2, 3, \ldots, M\}$ be a weight function on the edges of G. We assume that edges incident to every vertex are sorted in a non-decreasing order by weight and let $E(v, i)$ denote the i-th edge incident to v in this ordering. Further, we let $E(v)$ be the set of edges incident to v and for any $S \subseteq V$ we let $E(S)$ be the set of all edges with at least one endpoint in E. For every $u, v \in V$, let $d_G(u, v)$[2] be the distance between u and v in G, that is, the length of the shortest path between u and v. We use $P(u, v)$ to denote an ordered set of edges on a shortest path between u and v. For an edge $e = (u, v) \in E$ and a vertex $w \in V$ we define the distance between w and e to be $\min\{d_G(w, u), d_G(w, v)\} + wt(e)$ and denote this distance by $d_G(w, e)$.

Let the *girth* of G be the length of the shortest cycle in G. We denote the length of the girth by g.If $C = \{u_1, \ldots, u_t\}$ is a cycle in G then $(u_t, u_0) \in E$ and $(u_i, u_{i+1}) \in E$, for every $0 \leq i \leq t-1$. Let $wt(C)$ be the length of C.

For every $u \in V$ and a real number k we define the *ball graph* $H(u, k) = (V_u^k, E_u^k)$ of u as follows. The vertex set is $V_u^k = \{v \in V \mid d_G(u, v) \leq k\}$. The edge set is $E_u^k = \{e \in E \mid d_G(u, e) \leq k\}$. Let T_u^k be a shortest paths tree rooted at u whose vertex set is V_u^k.

Given a graph $G = (V, E)$ and a set of vertices $U \subseteq V$ we define $G - U$ to be $G[V \setminus U]$, that is, the graph obtained from G by deleting all the vertices of U together with their incident edges.

For graphs $H = (V', E')$ and $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$ then we say $H \subseteq G$.

3 Tools

In this section we present several tools we use to obtain our main results.

We proceed by proving an important property of the ball graph $H(v, k)$ for a vertex v (see the Section 2 for the definition).

LEMMA 3.1. *Let $G = (V, E)$ be a weighted undirected graph, let $v \in V$, and let k and R be real numbers. If there is no cycle in $H(v, k + R)$ then none of the vertices in V_v^k are part of any cycle of length at most R in G.*

Proof. We prove the claim by showing that if there is a vertex $u \in V_v^k$ that is part of a cycle of length at most R in G then there is a cycle in $H(v, k + R)$. Let $C = \{u_1, u_2, \ldots, u_t\}$ be a cycle such that $u_1 = u$ and $wt(C) \leq R$.

[2] When the graph G is clear from the context we omit the subscript.

We show that all the edges of C present in $H(v, k+R)$. Notice that from the definition of ball graphs it follows that for every $e \in E$ and $w \in V$ if $d_G(w,e) \leq \ell$ then e is in $H(w,\ell)$.

Consider now an edge $(x,y) \in C$. Since the entire length of C is at most R we have either $d_G(u,x)+wt(x,y) \leq R$ or $d_G(u,y)+wt(x,y) \leq R$. This implies that $d_G(u,(x,y)) \leq R$.

Next, we show that every edge $(x,y) \in C$ is part of $H(v,k+R)$. From the triangle inequality it follows that $d_G(v,(x,y)) \leq d_G(v,u) + d_G(u,(x,y))$. Since $u \in V_v^k$ it follows that $d_G(v,u) \leq k$ and we get that $d_G(v,(x,y)) \leq k+R$. This implies that every edge $(x,y) \in C$ is part of $H(v,k+R)$. ∎

In the case of unweighted graphs we prove the following:

LEMMA 3.2. *Let $G=(V,E)$ be an unweighted undirected graph and let $v \in V$. If there is no cycle in $H(v,k+\lceil R/2 \rceil)$ then all the vertices in V_v^k are not part of any cycle of length at most R in G.*

Proof. The proof follows the same structure as the proof of Lemma 3.1. Let $e=(x,y) \in E$ and $w \in V$. Notice that in unweighted graphs we have $d_G(w,e) = \min\{d_G(w,x), d_G(w,y)\}+1$.

Let $C=\{u_1,u_2,\ldots,u_t)$ be a cycle such that $u_1=u$ and $t \leq R$. Consider an edge $(x,y) \in C$. Since (x,y) is on a cycle of length at most R we have $d_G(u,x) \leq R/2$ and $d_G(u,y) \leq R/2$ when R is odd and either $d_G(u,x) \leq R/2$ and $d_G(u,y) < R/2$ or $d_G(u,x) < R/2$ and $d_G(u,y) \leq R/2$ when R is even. This implies that $d_G(u,(x,y)) \leq \lceil R/2 \rceil$. Similar to before, triangle inequality yields that (x,y) is an edge in $H(v,k+\lceil R/2 \rceil)$. ∎

Next, we present a procedure called `BallOrCycle` with pseudocode in Algorithm 1. This procedure is similar to the Bounded Dijkstra procedure presented in [LL09]. The input is a vertex v and parameter R. This procedure is a based on Dijkstra's algorithm in the case of weighted graphs and BFS in the case of unweighted graphs. We start to grow a shortest paths tree from v. We initiate a priority queue Q and add v to Q with key 0. As long as Q is not empty we extract the vertex with the minimum key from Q. Let this vertex be u. Next, we scan the set $E(u)$ in non decreasing order as follows. Let $(u,w) \in E(u)$. We check if $k(u)+wt(u,w) \leq R$. If this is the case and $w \notin Q$ we add w to Q with key equals to $k(u)+wt(u,w)$. If $w \in Q$ then a cycle is detected. Let z be the least common ancestor (LCA) of u and w in the shortest paths tree rooted at v. We return the edge (u,w) together with the path between u and z and the path between w and z, as the detected cycle.

We now analyse procedure `BallOrCycle`.

LEMMA 3.3. *Let $v \in V$. If $H(v,R)$ is not a tree, procedure* `BallOrCycle`(v,R) *reports a cycle of length at most $2R$ in $H(v,R)$. If $H(v,R)$ is a tree then* `BallOrCycle`(v,R) *returns $H(v,R)$. The running time of* `BallOrCycle`(v,R) *is $O(|V_v^R| \log n)$. For unweighted graphs the running time of* `BallOrCycle`(v,R) *is $O(|V_v^R|)$.*

Proof. If $H(v,R)$ is not a tree then there is at least one edge (u,w) such that $d(v,(u,w)) \leq R$ and $(u,w) \notin T_v^R$. This implies that $d(v,u) \leq R$ and $d(v,w) \leq R$. Since at any stage the maximum key in Q is at most R and $d(v,(u,w)) \leq R$ it follows from the correctness of Dijkstra's algorithm that at some stage we extract a vertex x from Q for which we encounter an edge (x,y) while scanning $E(x)$ such that $y \in Q$ and $d(v,(x,y)) \leq R$. This results in a cycle of length at most $2R$. If $H(v,R)$ is a tree it follows from the correctness of Dijkstra's algorithm that we will scan all the vertices at distance at most R from v. Since we add a condition that we only scan edges at distance at most R from v the algorithm stops once all the edges of $H(v,R)$ are scanned and since there is no cycle in $H(v,R)$ it cannot stop before that. If $H(v,R)$ is a tree the cost of `BallOrCycle` is $O(|V_v^R| \log n)$. If $H(v,R)$ is not a tree then a cycle is reported at the first time a vertex is updated in Q for the second time. The cost of `BallOrCycle` in this case is also $O(|V_v^R| \log n)$. If the graph is unweighted it is easy to implement the algorithm in $O(|V_v^R|)$ time, using BFS implementation. ∎

For unweighted undirected graphs, prior work [IR78, RV12, LL09] considered an algorithm `BFS-Cycle` with input a vertex v that would be identical to our algorithm `BallOrCycle`(v,∞) run with parameter $R=\infty$ on the unweighted graph. The following is a useful Lemma from prior work.

LEMMA 3.4. ([IR78, RV12, LL09]) *Let v be a vertex in an n-vertex undirected unweighted graph such that v is at distance at most d from a cycle of length g. Then* `BFS-Cycle` *runs in $O(n)$ time and outputs a cycle of length at most $2d+g$, if g is even, and at most $2d+g+1$, if g is odd.*

Algorithm 2: `Cycle`(G, R, k)

```
1 if G is empty then return No;
2 Let v ∈ V be an arbitrary vertex;
3 H(v, 0) ← ({v}, ∅);
4 for i ← 1 to k do
5 |   (H(v, iR), C) ← BallOrCycle(v, i · R) ;
6 |   if C ≠ null then return C;
  |   /* n is a global variable denoting the size of the original graph   */
7 |   if |V_v^{iR}| ≤ n^{1/k} · |V_v^{(i−1)R}| then
8 |   |   return Cycle(G \ V_v^{(i−1)R}, R, k)
```

4 A general scheme for girth approximation

In this section we preset an algorithm called `Cycle` with pseudocode in Algorithm 2. The algorithm gets as an input a weighted undirected graph $G = (V, E)$ and two parameters R and k. The algorithm either returns a cycle of length at most $2kR$ or reports that there is no cycle of length at most R in G. The algorithm works as follows. We pick an arbitrary vertex $v \in V$ as a source and iterate over i, from 1 until k. In the ith iteration, we call `BallOrCycle`$(v, i \cdot R)$. If `BallOrCycle`$(v, i \cdot R)$ reports a cycle, we stop the loop and report a cycle. If not, we check if the size of $V_v^{i \cdot R}$ is larger than the size of $V_v^{(i-1) \cdot R}$ by a factor that is at most $n^{1/k}$. If this is the case then we return the result of a recursive call to the algorithm with the graph $G \setminus V_v^{(i-1) \cdot R}$. If this is not the case, we proceed to the next iteration. As we will show in the analysis, there must be an iteration i, where $1 \leq i \leq k$, in which $V_v^{i \cdot R}$ is larger than the size of $V_v^{(i-1) \cdot R}$ by a factor that is at most $n^{1/k}$. Notice that in order to obtain a slightly more efficient implementation we do not need to compute $H(v, i \cdot R)$ from scratch in the ith iteration, instead we can use $H(v, (i-1) \cdot R)$ as the starting point for the run of `BallOrCycle`$(v, i \cdot R)$. This allows us to avoid a factor of k in the running time. Moreover, to implement efficiently the deletion of the vertex set $V_v^{(i-1) \cdot R}$ from the graph, we assume that every edge has pointers to its endpoints, so when a vertex $w \in V_v^{(i-1) \cdot R}$ is removed from G, we can delete every edge $(w, w') \in E(w)$ also from the adjacency list of w' in constant time.

Next, we prove that Algorithm `Cycle`(G, R, k) detects a cycle of length at most $2kR$ if G has a cycle of length at most R .

LEMMA 4.1. *If G has a cycle C such that $wt(C) \leq R$ then Algorithm* `Cycle`(G, R, k) *returns a cycle of length at most $2kR$.*

Proof. If a cycle is reported then it must be of length at most $2kR$ since for every source v for which we call `BallOrCycle`$(v, i \cdot R)$, we have $i \leq k$. Let $C = \{u_1, \ldots, u_t\}$ be a cycle in G such that $wt(C) \leq R$. Assume, towards a contradiction, that `Cycle`(G, R, k) does not report a cycle. Notice that this cannot happen if no vertex of C is removed from G because at some stage a vertex w from C will be picked as a source and in the call `BallOrCycle`(w, R), for $i = 1$, we will detect a cycle of length at most $2R$.

Thus, there must be a stage in which a vertex of C is removed from G. Let v be the first source that while performing its for loop at least one vertex of C is removed from G and let i be the iteration in which this happened. This implies that $V_v^{(i-1)R}$ contains a vertex of C. It follows from Lemma 3.1 and from the fact that $wt(C) \leq R$ that in such a case `BallOrCycle`$(v, i \cdot R)$ should have returned a cycle, thus we reach a contradiction. ∎

For the case of unweighted graphs we can prove the following:

LEMMA 4.2. *Let $G = (V, E)$ be an unweighted undirected graph. If G has a cycle C such that $wt(C) \leq R$ then Algorithm* `Cycle`$(G, \lceil R/2 \rceil, k)$ *returns a cycle of length at most $2k\lceil R/2 \rceil$.*

Proof. If a cycle is reported then it must be of length at most $2k\lceil R/2 \rceil$ since for every source v for which we call `BallOrCycle`$(v, \lceil i \cdot R/2 \rceil)$, we have $i \leq k$. Let $C = \{u_1, \ldots, u_t\}$ be a t-cycle in G such that $t \leq R$.

Assume, towards a contradiction, that `Cycle`$(G, \lceil R/2 \rceil, k)$ does not report a cycle. Notice that this cannot happen if no vertex of C is removed from G because at some stage a vertex w from C will be picked as a source and in the call `BallOrCycle`$(w, \lceil R/2 \rceil)$, for $i = 1$, we will detect a cycle of length at most $2\lceil R/2 \rceil$.

Thus, there must be a stage in which a vertex of C is removed from G. Let v be the first source that while performing its for loop at least one vertex of C is removed from G and let i be the iteration in which this happened. This implies that $V_v^{(i-1)\lceil R/2 \rceil}$ contains a vertex of C. However, it follows from Lemma 3.2 and from the fact that the length of C, t is $\leq R$ that in such a case $\texttt{BallOrCycle}(v, i \cdot \lceil R/2 \rceil)$ should have returned a cycle, thus we reach a contradiction. ∎

We now turn to analyze the running time of Algorithm $\texttt{Cycle}(G, R, k)$.

LEMMA 4.3. *Algorithm* $\texttt{Cycle}(G, R, k)$ *runs in* $O(m + n^{1+1/k} \log n)$ *time.*

Proof. When a vertex $v \in V$ is picked as a source in line 2 we iterate in a for loop over i in lines 4-8 from 1 to k. We refer to the for loop performed right after v is picked to be a source, as the for loop of v. We denote by $G(v)$ the input graph to the run of `Cycle` in which v is picked to be the source.

Notice first that if a cycle is never found in line 6, while iterating from 1 to k, then there must be an iteration in which $|V_v^i| \leq n^{1/k} \cdot |V_v^{i-1}|$. To see this assume, for the sake of contradiction, that this is not the case, and in every iteration $|V_v^i| > n^{1/k} \cdot |V_v^{i-1}|$, for every $1 \leq i \leq k$. This implies that $|V_v^i| > n^{i/k}$, for every $1 \leq i \leq k$, and in particular, $|V_v^k| > n$, a contradiction.

Thus, the loop ends either in line 6 or line 7. If the loop ends in line 6 then a cycle is detected. In such a case the total cost of the for loop of v is $O(n \log n)$, assuming the computation of $\texttt{BallOrCycle}(v, i \cdot R \cdot i)$ uses the output of $\texttt{BallOrCycle}(v, (i-1) \cdot R)$.

If the loop ends in line 7 then a recursive call to `Cycle` is initiated with the graph $G(v) \setminus V_v^{i-1}$. Consider the vertices of V_v^{i-1}. Each one of these vertices, except v, was not a source up to this stage. This follows from the fact that when a vertex is a source it is either removed or a cycle is detected. The vertices of V_v^{i-1} are now removed and hence each one of them will not be a source from this stage onward. Since $|V_v^i| \leq n^{1/k} \cdot |V_v^{i-1}|$, the cost of computing $\texttt{BallOrCycle}(v, R \cdot j)$, for every $1 \leq j \leq i$, is $O(n^{1/k} |V_v^{i-1}| \log n)$, assuming that $\texttt{BallOrCycle}(v, R \cdot j)$ uses the output of $\texttt{BallOrCycle}(v, R \cdot (j-1))$. We charge every vertex of V_v^{i-1} with a cost of $n^{1/k} \log n$ to cover the cost of the call to $\texttt{BallOrCycle}(v, R \cdot j)$, for every $1 \leq j \leq i$. We charge every vertex u that is removed with $deg(u)$ to pay the cost of deleting its edges from the graph.

We can now analyze the total cost of $\texttt{Cycle}(G, R, k)$. There is at most one source for which a cycle is detected. The cost of the for loop for this vertex is $O(n \log n)$. The cost of the for loop of a source w from which a cycle is not found is paid by charging the vertices that are removed at the for loop of w. Since a vertex can be removed at most once, every $w \in V$ is charged with a cost of $O(n^{1/k} \log n + \deg(w))$. Therefore, the total running time is $O(m + n^{1+1/k} \log n)$. ∎

We can now use Algorithm $\texttt{Cycle}(G, R, k)$ in order to obtain the main result of this section.

THEOREM 4.1. *Let* $G = (V, E)$ *be a weighted undirected graph with integral edge weight from the range* $[1, M]$ *and girth* g. *There is an algorithm that computes a cycle* C *in* $O((n^{1+1/k} \log n + m) \log nM)$ *time such that* $wt(C) \leq 2k \cdot g$

Proof. We perform a binary search over the interval $[1, nM]$. Let R^{YES} be a value for which $\texttt{Cycle}(G, R^{\text{YES}}, k)$ returns a cycle and let R^{NO} be a value for which $\texttt{Cycle}(G, R^{\text{NO}}, k)$ does not returns a cycle. At the beginning of the search we set R^{YES} to Mn and R^{NO} to 1. In each step if $R^{\text{YES}} - R^{\text{NO}} > 1$ we set R to $R^{\text{NO}} + \lfloor (R^{\text{YES}} - R^{\text{NO}})/2 \rfloor$, otherwise we stop and the result is the output of $\texttt{Cycle}(G, R^{\text{YES}}, k)$. If $\texttt{Cycle}(G, R, k)$ returns a cycle we set R^{YES} to R. If $\texttt{Cycle}(G, R, k)$ does not return a cycle we set R^{NO} to R. It follows from Lemma 4.3 that the cost of each step in the search $O(n^{1+1/k} \log n + m)$, thus the total cost of the search is $O((n^{1+1/k} \log n + m) \log nM)$. From Lemma 4.1 it follows that $R^{\text{NO}} < g$ since $\texttt{Cycle}(G, R^{\text{NO}}, k)$ does not returns a cycle. Since $R^{\text{YES}} - R^{\text{NO}} = 1$ we get that $R^{\text{YES}} \leq g$ and since the call to $\texttt{Cycle}(G, R^{\text{YES}}, k)$ return a cycle of length at most $2kR^{\text{YES}}$ we get that $2kR^{\text{YES}} \leq 2k \cdot g$. ∎

For unweighted graphs we prove a slightly better result.

THEOREM 4.2. *Given any* m*-edge,* n*-vertex unweighted undirected graph, there is an algorithm that computes a cycle* C *in* G *in* $O((n^{1+1/k} + m) \log n)$ *time such that* $wt(C) \leq 2k \cdot \lceil g/2 \rceil$ *and* g *is the girth of* G.

Algorithm 3: `DegenerateOrCycle`(G, k)

Let $m' = 1 + \lceil nD \rceil$ for $D := 1 + \min\{n^{1/k}, 2m^{1/(k+1)}\}$;
if $m \geq m'$ **then** $G \leftarrow$ the edge induced subgraph from an arbitrary subset of $1 + \lceil nD \rceil$ edges of G ;
$d_a \leftarrow$ the degree of a for all $a \in V$;
$S \leftarrow \{a \in V | d_a \leq D\}$;
while $S \neq \emptyset$ **do**
    Pick $a \in S$ and let $S \leftarrow S \setminus \{a\}$;
    For all $\{a, b\} \in E(a)$ let $d_b \leftarrow d_b - 1$ and if as a result $d_b \leq D$ add b to S ;
    Remove a and all of incident edges from G ;
if $G \neq (\emptyset, \emptyset)$ **then**
    $(B, C) \leftarrow$ `BallOrCycle`(a, k) for arbitrary $a \in V$;
    return C ; `// C is a cycle of length` $\leq 2k$
return $\emptyset$; `// G is` D`-degenerate`

Proof. The proof is almost identical to the proof of Theorem 4.1. There are two changes. The first is that we do a binary search over the interval $[1, n]$. The second is that any usage of `Cycle`(G, X, k) is replaced with `Cycle`$(G, \lceil X/2 \rceil, k)$, where $X \in \{R^{\text{NO}}, R^{\text{YES}}, R\}$. The rest of the proof is identical. ∎

In the next section, in Theorem 5.1 we present a tool for removing the dependence on m in girth approximation algorithms in unweighted graphs. The algorithm of Theorem 5.1 runs in $O(\min\{m, n^{1+1/k}\})$ time, and either returns a $\leq 2k$-cycle, or determines that $m \leq n^{1+1/k}$.

This allows us to obtain the following improvement to Theorem 4.2 by first running the algorithm of Theorem 5.1, and if it does not return a cycle, running the algorithm of Theorem 4.2.

THEOREM 4.3. *For every integer $k \geq 1$, there is an algorithm that computes a cycle C in any n-vertex unweighted undirected graph G in $O(n^{1+1/k} \log n)$ time such that $wt(C) \leq 2k \cdot \lceil g/2 \rceil$, where g is the girth of G.*

5 Sublinear approximation algorithms for dense graphs

In this section we show how to remove the additive m in the algorithms of the previous section and achieve sublinear runtimes. To do this, we provide a more general result that given a graph it is possible to efficiently either find a cycle or conclude that the graph is degenerate. The algorithm for this procedure, `DegenerateOrCycle`, is given as Algorithm 3 and the main result of this section is the following theorem bounding its performance.

THEOREM 5.1. (DEGENERATE OR CYCLE) *Given unweighted n-vertex m-edge $G = (V, E)$ and integer $k \geq 1$,* `DegenerateOrCycle`(G, k) *(Algorithm 3) in $O(\min\{m, nD\})$ time for $D := 1 + \min\{n^{1/k}, 2m^{1/(k+1)}\}$ either outputs a cycle of length at most $2k$ or $\emptyset$ in which case G is D-degenerate and hence $m \leq nD$.*

To prove this theorem we first give the following lemma regarding the performance of `BallOrCycle` on graphs of large degree.

LEMMA 5.1. (`BallOrCycle` IN LARGE-DEGREE GRAPHS) *If $G = (V, E)$ is an n-vertex undirected graph where every vertex has degree at least $1 + n^{1/k}$ for integer $k \geq 1$ then* `BallOrCycle`(a, k) *returns a cycle of length at most $2k$ in $O(n)$-time.*

Proof. By Lemma 3.3 it suffices to show that `BallOrCycle`(a, k) does not output a ball. Proceed by contradiction and suppose that `BallOrCycle` outputs a ball. In this case the edges inside that ball constitute a tree rooted at v where every vertex of depth $< k$ has degree at least $1 + n^{1/k}$ and therefore at least $n^{1/k}$ children. Consequently, for all integers $i \leq k$ this implies that the tree has at least $n^{i/k}$ vertexs at depth i. Therefore the tree has at least n vertices at depth k and since $k \geq 1$, the root, v, is not one of these k vertices. Consequently, the tree has $n + 1$ vertices contradicting that G is a n-vertex graph. ∎

Leveraging Lemma 5.1 we prove Theorem 5.1.

Proof. [Proof of Theorem 5.1] The algorithm simply picks an arbitrary set of at most m' edges, where $m' = 1 + \lceil nD \rceil$, in $O(\min\{m, m'\}) = O(\min\{m, nD\})$ time and then repeatedly removes vertices of degree at most D in $O(nD)$ time. If after the removal the resulting graph is nonempty (Line 9) then a cycle found by `BallOrCycle` is returned, otherwise the algorithm returns that the graph is degenerate. By Lemma 3.3, which bounds the runtime of `BallOrCycle` with $O(n)$ we get that the runtime of the algorithm is $O(\min\{m, nD\})$, as desired.

To prove that the algorithm has the desired output, first note that if $m \leq m'$ (so the subset of at most m' edges is the entire graph) and $G = (\emptyset, \emptyset)$ in Line 9, then the graph is D-degenerate (every vertex was removed from the original graph and it had degree at most D when it was removed. On the other hand if $m \geq m'$ then since every vertex removal in an iteration of the while loop removes $\leq D$ edges at most $nD < m'$ edges are removed and $G \neq (\emptyset, \emptyset)$ in Line 9. Consequently, it only remains to show that whenever $G \neq (\emptyset, \emptyset)$ in Line 9 then the returned C is a cycle of length $2k$.

For this last case, suppose $G \neq (\emptyset, \emptyset)$ in Line 9. Let N and M denote the number of vertices and edges respectively in the graph at this time. If $D = 1 + n^{1/k}$ then every vertex in G at Line 9 has degree at least $1+n^{1/k} \geq 1+N^{1/k}$ (since vertices were only removed from the original graph). On the other hand, if $D = 1+2m^{1/k}$ then since every vertex of G at Line 9 has degree at least $1+2m^{1/(k+1)}$ it is the case that $N(1+2m^{1/(k+1)}) \leq 2M$. Further, since $M \leq m$ (since edges were only removed from the original graph) this implies that $N \leq M^{k/(k+1)}$ and $m^{1/(k+1)} \geq M^{1/(k+1)} \geq N^{1/k}$. In either case every vertex of G at Line 9 has degree at least $1 + N^{1/k}$ and a cycle of length at most $2k$ is output by Lemma 5.1. ∎

6 Faster algorithms for small girth and sparse graphs

In this section we will use the following theorem of Alon, Yuster and Zwick [AYZ97].

THEOREM 6.1. ([AYZ97]) *Given an m-edge graph of degeneracy at most D, for any integer $\ell \geq 1$,*

- *a $4\ell - 2$-cycle if one exists can be found in $O(m^{2-1/\ell} D^{1-1/\ell})$ time,*
- *a $4\ell - 1$ or a 4ℓ-cycle if one exists can be found in $O(m^{2-1/\ell} D)$ time, and*
- *a $4\ell + 1$-cycle if one exists can be found in $O(m^{2-1/\ell} D^{1+1/\ell})$ time.*

We begin by proving the following general theorem:

THEOREM 6.2. *For every $k \geq 2$, there is an $O(\min\{m, n^{1+1/k}\})$ time algorithm that either returns a $\leq 2k$-cycle, and if it fails then $m \leq O(n^{1+1/k})$ and then one can find a shortest cycle or determine that the girth is $> g$ in additional time*

- *$O(m^{2-1/\ell+1/(k+1)-1/(\ell(k+1))})$ if $g = 4\ell - 2$,*
- *$O(m^{2-1/\ell+1/(k+1)})$ if $g = 4\ell - 1$ or $g = 4\ell$, and*
- *$O(m^{2-1/\ell+1/(k+1)+1/(\ell(k+1))})$ if $g = 4\ell + 2$.*

As corollaries we obtain the following results for small girth.

COROLLARY 6.1. *For every $k \geq 2$, there is an $O(\min\{m, n^{1+1/k}\})$ time algorithm that either returns a $\leq 2k$-cycle, and if it fails then $m \leq O(n^{1+1/k})$ and then one can find a shortest cycle or determine that the girth is $> g$ in additional time*

- *$O(\min\{n^{1+2/k}, m^{1+1/(k+1)}\})$ if $g = 3$ or $g = 4$, and*
- *$O(\min\{n^{1+3/k}, m^{1+2/(k+1)}\})$ if $g = 5$.*

COROLLARY 6.2. *Let $k \geq 2$ be an integer. There is an algorithm that given an n-, m-edge graph either determines that the girth is more than g, or returns a cycle of length $\leq 2k$, and runs in time $O(\min\{n^{1+2/k}, m^{1+1/(k+1)}\})$ time if $g = 3$ or $g = 4$ and in time $O(\min\{n^{1+3/k}, m^{1+2/(k+1)}\})$ time if $g = 5$.*

Now we can prove Theorem 6.2.

Proof. First, we can assume that the girth is at least g, as we can first run the algorithm for the setting $g-1$.

If the number of edges m of G is $\geq 1+n^{1+1/k}$, then by Theorem 5.1, running Algorithm 3 on G and k will return a cycle of length $\leq 2k$ in $O(n^{1+1/k})$ time and we do not need to run the rest of the algorithm below.

Hence we can assume that $m \leq \lceil n^{1+1/k} \rceil$. Here we can afford to go through all edges and compute the degrees in $O(m)$ time. Then we also run Algorithm 3. If the algorithm returns a $\leq 2k$-cycle, then we are done. Hence assume otherwise.

In this case, from Theorem 5.1 we know that G is D-degenerate, where $D \leq 2m^{1/(k+1)}$. We can now run the cycle-finding algorithm of Alon, Yuster and Zwick on the D-degenerate graph. The running time is as follows:

- For $g = 4\ell - 2$ the running time is $O(m^{2-1/\ell}D^{1-1/\ell})$ time, which for our setting of $D \leq 2m^{1/(k+1)}$ is $O(m^{2-1/\ell+1/(k+1)-1/(\ell(k+1))})$.
- For $g = 4\ell - 1$ or $g = 4\ell$ the running time is $O(m^{2-1/\ell}D)$, which for our setting of $D \leq 2m^{1/(k+1)}$ is $O(m^{2-1/\ell+1/(k+1)})$, and
- For $g = 4\ell + 1$ the running time is $O(m^{2-1/\ell}D^{1+1/\ell})$, which for our setting of $D \leq 2m^{1/(k+1)}$ is $O(m^{2-1/\ell+1/(k+1)+1/(\ell(k+1))})$.

∎

Finally, we present an algorithm for $g = 6$ that beats the $\tilde{O}(n^{1+3/k})$ runtime of Theorem 4.2 when the girth is 6, in the special case when k is either 4 or 5 and m is small.

THEOREM 6.3. *For $k = 4$ and $k = 5$, there is an $\tilde{O}(\min\{nm^{1/2+1/(2k+2)}, n^{3/2+1/k}\})$ time algorithm that in m-edge n-vertex graphs, either returns a cycle of length at most $2k$, or determines that the girth is more than 6.*

Proof. We run Algorithm 3 on G and k. We either get a $2k$-cycle and are done, or we have degeneracy at most $D = 2m^{1/(k+1)}$ and $m \leq n^{1+1/k}$ edges.

Then, we want to handle the case that the shortest cycle has a vertex of degree Δ, for a parameter Δ. We sample $\tilde{O}(n/\Delta)$ vertices S, and for every $s \in S$ we run `BFS-Cycle`(s).

The sample S can either be obtained in a randomized way, or can be done deterministically by the greedy algorithm for hitting set, where we are trying to hit the neighborhoods of all vertices of degree at least Δ. This would take $O(n\Delta)$ time, and is subsumed by the rest of the running time of the algorithm.

If the shortest cycle has a high degree vertex, some $s \in S$ is at distance at most 1 from the cycle, by Lemma 3.4, `BFS-Cycle` will return a cycle of length no more than 8 if the girth is at most 6. If $8 \leq 2k$, we are done, so this will work as long as $k \geq 4$.

The greedy degeneracy assignment of edges to one of their end-points given by Theorem 5.1 is an acyclic orientation of the edges. Thus any cycle in G has a vertex d whose edges in the cycle are not assigned to it, i.e. d is a sink for the degeneracy orientation. Thus, every 6-cycle of G can be decomposed into two length 3 paths $a-b-c-d$ and $a-b'-c'-d$ so that $c-d$ is assigned to c and $c'-d$ is assigned to c'. Moreover, we can assume that all vertices on these 3-paths have degree at most Δ, as otherwise we would have found a short cycle already.

We enumerate all such paths as follows: go over all edges (a,b) where b has degree at most Δ, go over all the at most Δ neighbors c of b and the at most D edges (c,d) assigned to c. We enumerate these paths in $O(mD\Delta)$ time and try to find two that share their endpoints, in the same time. If we find two such paths, we have found a ≤ 6-cycle.

We thus find a $2k$-cycle, a $g+2 = 8$-cycle, a ≤ 6-cycle, or determine that the girth is more than 6. The running time is minimized for when $n^2/\Delta = mD\Delta$, so $\Delta^2 = n^2/(mD)$.

Notice that since $m \leq n^{1+1/k}$, we have that $\Delta^2 = n^2/(mD) = n^2/(2m^{(k+2)/(k+1)}) \geq n^2/(2n^{1+2/k}) = n^{1-2/k}/2$, which is at least 1 for $k > 2$ and large enough n.

Thus, we can set $\Delta = \Theta(n/\sqrt{mD})$, and this will be more than 1 for our choice of $k = 4, 5$. The running time is $\tilde{O}(\sqrt{mD}n) = \tilde{O}(nm^{1/2+1/(2k+2)})$.

As $m \leq n^{1+1/k}$, we obtain the following running time upper bound purely in terms of n:

$$\tilde{O}(nn^{(k+1)/k\cdot(k+2)/(2(k+1))}) \leq \tilde{O}(n^{3/2+1/k}).$$

∎

The algorithm is never worse than the $\tilde{O}(n^{1+3/k})$ runtime of Theorem 4.2 if $k = 4$, and it gets faster for lower sparsity.

Let's consider the special case when $m = O(n)$. Then the algorithm runs in $\tilde{O}(n^{3/2+1/(2k+2)})$ time. The exponent is better than the $1+3/k$ exponent of Theorem 4.2 whenever $k \leq 5$. Thus, we obtain a faster algorithm for $g = 6$ for very sparse graphs and $k = 4$ or 5.

7 Additive approximation scheme

In this section we focus on additive approximation in unweighted graphs. Our goal is to get an approximation that is better than a multiplicative approximation of 2 at the price of a bit worse running time. The main technical contribution in this section is an algorithm that given two parameters R and a, where $a < \lceil R/2 \rceil$, returns a cycle of length at most $2\lceil R/2 \rceil + 2a$ if there is a cycle of length at most R in the graph.

The algorithm is composed of a couple of building blocks. Each building block on its own is relatively simple. The main technical challenge is in combining these building blocks carefully into an efficient algorithm.

We start with a simple procedure called `IsDense`. The input to `IsDense` is a graph $G = (V, E)$, a vertex w in V, and two integers, D and r. The procedure `IsDense` is a simple variant of a BFS from w. The search is performed as long as the total number of edges that were scanned during the search, D' is less than D and the farthest vertex from w is at distance at most r. Recall that $H(w, r) = (V_w^r, E_w^r)$ is the ball graph of w at distance r. Procedure `IsDense` returns No if $|E_w^r| < D$ and Yes if $|E_w^r| \geq D$.

The procedure is implemented in a similar way to the BFS algorithm. We keep in j the current distance between w and the next vertex to be dequeue from a queue Q. We have two counters ℓ_c and ℓ_n, where ℓ_c is updated to reflect the number of vertices that are left in Q at distance j, and ℓ_n is updated to reflect the number of vertices that are added to Q at distance $j + 1$. We initialize ℓ_c to 1, ℓ_n to 0 and j to 0. When ℓ_c reaches 0 we set ℓ_c to ℓ_n, ℓ_n to 0 and increment j by one. When a vertex u is dequeue from Q we scan $E(u)$ if $j < r$ and as long as $D' < D$. We increment ℓ_n by one for each vertex that is added to Q. Procedure `IsDense` is presented in in Algorithm 4. In the next Lemma we prove the running time and the correctness of `IsDense`.

LEMMA 7.1. *Procedure* `IsDense`(G, w, D, r) *runs in* $O(D)$ *time. If* `IsDense`$(G, w, D, r) = No$ *then* $|E_w^r| < D$ *and* $|V_w^r| \leq D$. *If* `IsDense`$(G, w, D, r) = Yes$ *then* $|E_w^r| \geq D$ *and if* $H(w, r)$ *is also a tree then* $|V_w^r| > D$.

Proof. The value of D' is incremented by one in each iteration of the inner while loop. The outer while loop ends immediately if the inner loop ends because $D' = D$, thus the running time is $O(D)$.

As long as $D' < D$, the algorithm increments D' by one for every edge $(x, y) \in E_w^r$[3]. Therefore, $|E_w^r| \geq D'$.

If the main while loop ends and $D' < D$ then all the edges of E_w^r were scanned in the inner while loop and hence $|E_w^r| = D' < D$. Thus, if `IsDense`$(G, w, D, r) = No$ then $|E_w^r| < D$. Since $|V_w^r| \leq |E_w^r| + 1$ we get that $|V_w^r| \leq D$.

If the main while loop ends and $D' = D$ then the set of edges that were scanned by the inner while loop is contained in E_w^r and hence $|E_w^r| \geq D$. Thus, if `IsDense`$(G, w, D, r) = Yes$ then $|kE_w^r| \geq D$. If we also have that $H(w, r)$ is a tree then $|V_v^r| = |E_v^r| + 1$ and we get that $|V_v^r| = |E_v^r| + 1 \geq D + 1 > D$. ∎

We proceed with a simple but yet crucial property of procedure `IsDense` that allows us use information obtained on using `IsDense` on one graph also on other versions of that graph.

LEMMA 7.2. *Let* x *and* D *be positive integers. If* `IsDense`$(G, w, D, x) = No$ *and* $G' \subseteq G$ *then* `IsDense`$(G', w, D, x) = No$.

Proof. The fact that `IsDense`$(G, w, D, x) = No$ implies that $|E_w^x| < D$. Since $G' \subseteq G$ we have $|E'^x_w| \leq |E_w^x| < D$ and from Lemma 7.1 it follows that `IsDense`$(G', w, D, x) = No$. ∎

We now proceed with proving a property of `IsDense` that allows us to aggregate information gained by applying `IsDense` to different vertices of the graph.

LEMMA 7.3. *Let* x, y *and* D *be positive integers and let* $w \in V$. *If* `IsDense`$(G, w, D^x, x) = No$ *and* `IsDense`$(G, u, D^y, y) = No$, *for every* $u \in V$, *then* $|V_w^{x+y}| \leq D^{x+y}$.

[3]Notice that once an edge (u, v) is scanned by u we remove it from $E(v)$ to avoid double counting (u, v)

Algorithm 4: `IsDense`(H, w, D, r)

```
1  D' ← 0, ℓ_c ← 1, ℓ_n ← 0, j ← 0 ;
2  Q ← {w} ;
3  while (Q is not empty) and (j < r) and (D' < D) do
4      u ← dequeue(Q);
5      ℓ_c ← ℓ_c − 1, i ← 0;
6      while (i ≤ |E(u)|) and (j < r) and (D' < D) do
7          (u, v) ← E(u, i) ;
8          remove (u, v) from E(v);
9          D' ← D' + 1 ;
10         if v is not marked then
11             Q ← Q ∪ {v}, mark v, ℓ_n ← ℓ_n + 1 ;
12         i ← i + 1;
13     if ℓ_c = 0 then
14         ℓ_c ← ℓ_n, ℓ_n ← 0 ;
15         j ← j + 1 ;
16 if (D' = D) or (D = 1 and r = 0) then return Yes;
17 else return No;
```

Proof. Let $w \in V$ and assume that `IsDense`$(G, w, D^x, x) = No$. We also know that `IsDense`$(G, u, D^y, y) = No$, for every $u \in V$. Notice that $V_w^{x+y} = \bigcup_{u \in V_w^x} V_u^y$. Therefore, $|V_w^{x+y}| = |\bigcup_{u \in V_w^x} V_u^y|$.

It follows from Lemma 7.1 that $|V_w^x| \leq D^x$ since `IsDense`$(G, w, D^x, x) = No$. It also follows from Lemma 7.1 that $|V_u^y| \leq D^y$, for every $u \in V_w^x$ since `IsDense`$(G, u, D^y, y) = No$, for every $u \in V$. Thus, we get:

$$|\bigcup_{u \in V_w^x} V_u^y| \leq \sum_{u \in V_w^x} |V_u^y| \leq \sum_{u \in V_w^x} D^y \leq D^{x+y}$$

We conclude that $|V_w^{x+y}| \leq D^{x+y}$. ∎

We use this Lemma to prove the following Corollary:

COROLLARY 7.1. *Let x and D be positive integers. If* `IsDense`$(G, w, D^x, x) = No$ *for every $w \in V$, then $|V_w^{ix}| \leq D^{ix}$, for every $w \in V$ and $i > 1$.*

Proof. The proof is by induction on i. For $i = 2$ it follows from Lemma 7.3 that $|V_w^{2x}| \leq D^{2x}$. Assume now the claim holds for $j \leq i - 1$. This implies that $|V_w^{(i-1)x}| \leq D^{(i-1)x}$, for every $w \in V$. Combining this with the fact that $|V_w^x| \leq D^x$, for every $w \in V$, we get that $|V_w^{ix}| \leq D^{ix}$, for every $w \in V$ and $i > 1$. ∎

We now present the second building block, a procedure called `SparseOrCycle`, that gets as an input a graph $G = (V, E)$ and three parameters D, x and y. In `SparseOrCycle` we scan the vertices of G. For each vertex w we call `IsDense`(G, w, D^x, x). If `IsDense`(G, w, D^x, x) returns Yes we call `BallOrCycle`$(w, x + y)$. The call to `BallOrCycle` either returns a cycle of length at most $2(x+y)$ or the ball graph $H(w, x+y)$. If a cycle is returned then we return the cycle as the output of `SparseOrCycle`. In case that the ball graph $H(w, x + y)$ is returned then we remove the set of vertices V_w^x from G and repeat the same process with a new vertex. If the loop ends without finding a cycle then we return null. Procedure `SparseOrCycle` is presented in Algorithm 5.

In the analysis we denote with $G(w)$ the current version of the input graph G during the execution of `SparseOrCycle`(G, D, x, y) when the vertex considered by the for-all loop is w. We also denote with $\hat{G} = (\hat{V}, \hat{E})$ the graph $G = (V, E)$ after the execution of `SparseOrCycle`(G, D, x, y) ends. We denote with $W \subseteq V$ the set of vertices for which `BallOrCycle` was called during the execution of `SparseOrCycle`(G, D, x, y) and a cycle was not found. We assume that $y > 0$. Consider a vertex $u \in V \setminus \hat{V}$. This means that there is a vertex $w \in W$ such that when w was considered by the for-all loop $u \in V_w^x$ and thus removed from H. We denote with $\rho(u)$ the vertex w that is responsible for removing u from G. Notice that for $w \in W$ we have $\rho(w) = w$.

Algorithm 5: `SparseOrCycle`(G, D, x, y)

```
for all w ∈ V do
    if IsDense(G, w, D^x, x) = Yes then
        (H(w, x + y), C) ← BallOrCycle(w, x + y) ;
        if C ≠ null then return C;
        else G ← G \ V_w^x;
return null;
```

In the next lemma we analyze the running time of `SparseOrCycle`.

LEMMA 7.4. *The running time of* `SparseOrCycle`(G, D, x, y) *is* $O(|V|D^x + \sum_{w \in W}(|V_w^{x+y}|))$.

Proof. The cost of each call to `IsDense` is $O(D^x)$. In the worst case we call $O(|V|)$ times to `IsDense`. The total cost of this step is $O(|V|D^x)$. Each call to `BallOrCycle`$(w, x + y)$ that does not return a cycle costs $O(|V_w^{x+y}|)$. At most one call to `BallOrCycle` returns a cycle. We can bound this call with $O(n)$. The cost of removing the set V_w^x from G is bounded by $O(|E_w^{x+1}|)$. However, since we are in the case that `BallOrCycle`$(w, x + y)$ does not return a cycle it follows that $H(w, x + y)$ is a tree and $|E_w^{x+1}| = O(|V_w^{x+1}|)$. ∎

We now turn to prove several useful properties of `SparseOrCycle`.

LEMMA 7.5. *If* `SparseOrCycle`(G, D, x, y) *returns a cycle* C *then* $wt(C) \leq 2(x + y)$.
If `SparseOrCycle`(G, D, x, y) *does not return a cycle then:*

(i) If $u \in V \setminus \hat{V}$ *then* u *is not part of a cycle of length at most* $2y$ *in* $G(\rho(u))$.

(ii) If $u \in \hat{V}$ *then* `IsDense`$(\hat{G}, w, D^x, x) = No$.

(iii) If $x = 0$ *then* $\hat{G}$ *is empty.*

Proof. (i) Let $u \in V \setminus \hat{V}$. Let $w = \rho(u)$. Thus, $u \in V_w^x$ and `BallOrCycle`$(w, x + y)$ returned null. It follows from Lemma 3.2 that u is not on a cycle of length $2y$ in $G(w)$.

(ii) Let $u \in \hat{V}$. This implies that u was not removed by `SparseOrCycle`(G, D, x, y) and thus was considered at the for-all loop at some stage of the execution of `SparseOrCycle`(G, D, x, y). At this stage `IsDense`$(G(u), u, D^x, x)$ was No, as otherwise u would have been removed. Since $\hat{G} \subseteq G(u)$, it follows from Lemma 7.2 that `IsDense`$(\hat{G}, w, D^x, x) = No$.

(iii) If $x = 0$ then `IsDense`$(G(u), u, 1, 0) = Yes$ for every $u \in V$. Thus, a vertex $u \in V$ is removed during the execution of `SparseOrCycle`$(G, D, 0, y)$ either because the for-all loop reached to u and `IsDense`$(G(u), u, 1, 0) = Yes$ or because u is in V_w^y, where w is a vertex reached by the for-all loop.
∎

We are now ready to present the main algorithm of this section. The input is an unweighted undirected graph $G = (V, E)$ and two parameters R and c. If R is odd then $R = 2t - 1$ and if R is even then $R = 2t$. The algorithm either returns a cycle of length at most $4t - 2c$ or reports that the girth of G is strictly more than $2t$. The algorithm starts by calling to `SparseOrCycle`$(G, D, t - c, t)$. If a cycle is not found the algorithm proceeds by setting d to c, q to $\lfloor \frac{t-c}{d} \rfloor$ and running a repeat-until loop. In each iteration of the repeat-until loop we execute an inner loop implemented as a for loop. In the for loop the index i runs from 1 to q. In each iteration we call to `SparseOrCycle`$(G, D, t - c - i \cdot d, t)$. If the inner loop ends without finding a cycle we set r to $(t - c) \mod d$, d to $t \mod r$ and q to $\lfloor \frac{t-c}{d} \rfloor$. We stop when $r = 0$. Algorithm `CycleAdditive` is presented in Algorithm 6.

Before we turn to analyse the algorithm we introduce some notation to ease the presentation. We denote with ℓ be the number of iterations of the outer loop (the repeat-until loop). Let $1 \leq j \leq \ell$. We denote the values of r, d and q in the beginning of the jth iteration of the outer loop with r_j, d_j and q_j. Let $1 \leq u \leq q_j$. We denote with G_j^i ($\hat{G}_j^i$) the current version of the input graph G at the beginning (end) of the ith iteration of the inner

loop right before (after) the call to $\texttt{SparseOrCycle}(G_i^j, D, t-c-id_j, t)$. We denote with W_i^j the set of vertices for which $\texttt{SparseOrCycle}(G_i^j, D, t-c-id_j, t)$ initiated a call to `BallOrCycle` in the ith iteration of the inner loop during the jth iteration of the outer loop.

LEMMA 7.6. *Let $G=(V,E)$. Let R and c be two parameters and let $t=\lceil R/2\rceil$. If $\texttt{CycleAdditive}(G,R,c)$ returns a cycle C then $wt(C)\leq 4t-2c$. If $\texttt{CycleAdditive}(G,R,c)$ returns No then the girth of G is strictly more than $2t$. The running time of $\texttt{CycleAdditive}(G,R,c)$ is $O((t-c)\cdot n^{1+\frac{t}{2t-c}})$.*

Proof. We first show that if $\texttt{CycleAdditive}(G,R,c)$ returns a cycle C then $wt(C)\leq 4t-2c$. During the run of the algorithm every call to $\texttt{SparseOrCycle}(G,D,x,y)$ is with $x\leq t-c$ and $y=t$. Thus, when `SparseOrCycle` returns a cycle C it follows from Lemma 7.5 that $wt(C)\leq 2(x+y)\leq 4t-2c$.

We now turn to show that if $\texttt{CycleAdditive}(G,R,c)$ returns No then the girth of G is strictly more than $2t$. Notice that in this case every call to `SparseOrCycle` returns No.

We start by showing that if the algorithm ends without returning a cycle then $\hat{G}_\ell^{q_\ell}$ is empty. Since ℓ is the last iteration of the outer loop r is updated to 0 after the inner loop ends, therefore, $(t-c) \mod d_\ell$ is 0. Let $x=t-c-q_\ell d_\ell$. Since $(t-c) \mod d_\ell=0$ and $q_\ell=\lfloor\frac{t-c}{d_\ell}\rfloor$ we have $x=0$. Thus, in the q_ℓ iteration of the inner loop during the ℓ iteration of the outer loop we call to $\texttt{SparseOrCycle}(G_\ell^{q_\ell}, D, 0, t)$. It follows from Lemma 7.5(iii) that $\hat{G}_\ell^{q_\ell}$ is empty.

Next, we use the fact that $\hat{G}_\ell^{q_\ell}$ is empty to show that there is no cycle of length at most $2t$ in G. Let $C=\{u_1,\ldots,u_{t'}\}$ be a cycle in the input graph G. Assume towards a contradiction that $wt(C)\leq 2t$. Since $\hat{G}_\ell^{q_\ell}$ is empty every vertex of C is removed at some stage during the execution of $\texttt{CycleAdditive}(G,R,c)$. Let $u\in C$ be the first vertex that is being removed. This event either happens during the execution of $\texttt{SparseOrCycle}(G,D,t-c,t)$ before the outer loop starts or during the execution of $\texttt{SparseOrCycle}(G_j^i, D, t-c-i\cdot d_j, t)$, where $1\leq j\leq \ell$ and $1\leq i\leq q_j$.

In case that $\texttt{SparseOrCycle}(G,D,t-c,t)$ removed u then C is in $G(\rho(u))$. In case that $\texttt{SparseOrCycle}(G_j^i, D, t-c-i\cdot d_j, t)$ removed u then C is in $G_j^i(\rho(u))$. However, since u is removed it follows from Lemma 7.5(i) that u is not on a cycle of length at most $2t$ in $G(\rho(u))$ in the former case and in $G_j^i(\rho(u))$ in the later case, a contradiction.

Finally, we turn to prove that the running time of $\texttt{CycleAdditive}(G,R,c)$ is $O((t-c)n^{1+\frac{t}{2t-c}})$. Let $D=n^{\frac{1}{2t-c}}$. First, recall that from Lemma 7.4 it follows that the running time of $\texttt{SparseOrCycle}(G,D,x,y)$ is $O(|V|D^x+\sum_{w\in W}|V_w^{x+y}|)$.

In every call to $\texttt{SparseOrCycle}(H,D,x,y)$, during the execution of $\texttt{CycleAdditive}(G,R,c)$, we have $H\subseteq G$ and $x\leq t-c$, hence, the term $O(|V|D^x)$ in the running time is bounded by $O(n^{1+\frac{t-c}{2t-c}})$. There are at most $t-c$ calls to `SparseOrCycle` thus, total contribution to the cost of this term is bounded by $O((t-c)n^{1+\frac{t-c}{2t-c}})$.

Consider the first call to $\texttt{SparseOrCycle}(G,D,t-c,t)$ right before the outer loop starts. For every $w\in W$ the call to `IsDense` with w returned Yes and since no cycle was found, it follows from Lemma 7.1 that $|V_w^{t-c}|\geq D^{t-c}$. The call to $\texttt{BallOrCycle}(w,t-c+t)$ costs $O(n)=O(D^{2t-c})$, since $D=n^{1/(2t-c)}$. Since $|V_w^{t-c}|\geq D^{t-c}$ and the vertices of V_w^{t-c} are removed from the graph and will not be considered any more, we can charge every vertex of V_w^{t-c} with a cost of D^t to cover the $O(D^{2t-c})$ cost of $\texttt{BallOrCycle}(w,t-c+t)$.

Consider now the first iteration of the outer loop. In this iteration $d_1=c$. From Lemma 7.5(ii) it follows that if $w\in\hat{V}$ after the execution of $\texttt{SparseOrCycle}(G,D,t-c,t)$ then $\texttt{IsDense}(\hat{G},w,D^{t-c},t-c)=No$.

We analyze the cost of $\texttt{SparseOrCycle}(G_1^i,D,t-c-id_1,t)$, where $1\leq i\leq q_1$. For every $w\in W_1^i$ the call to `IsDense` with w returned Yes and since no cycle was found, it follows from Lemma 7.1 that $|V_w^{t-c-id_1}|\geq D^{t-c-id_1}$. The cost of a call to `BallOrCycle` with w is $O(|V_w^{t-c-id_1+t}|)$.

Since w survived all the calls to $\texttt{SparseOrCycle}(G_1^{i'},D,t-c-i'd_1,t)$, where $i'<i$, it follows from Lemma 7.5(ii) that $\texttt{IsDense}(\hat{G}_1^{i'},w,D^{t-c-i'd_1},t-c-i'd_1)=No$. Since $G_1^i(w)\subseteq\hat{G}_1^{i'}$, it follows from Lemma 7.2 that $\texttt{IsDense}(G_1^i(w),w,D^{t-c-i'd_1},t-c-i'd_1)=No$. In particular, for $i'=i-1$ we have $\texttt{IsDense}(G_1^i(w),w,D^{t-c-(i-1)d_1},t-c-(i-1)d_1)=No$.

Notice that $d_1=c$, thus, $t-c-(i-1)d_1+t-c=t-c-id_1+t$. Since for w we have $\texttt{IsDense}(G_1^i(w),w,D^{t-c-(i-1)d_1},t-c-(i-1)d_1)=No$ and for every u in $G_1^i(w)$ we have $\texttt{IsDense}(G_1^i(w),u,D^{t-c},t-c)=No$ we can apply Lemma 7.3 to bound the $O(|V_w^{t-c-id_1+t}|)$ cost of `BallOrCycle` with D^{t-c-id_1+t}. We can charge every vertex of $V_w^{t-c-id_1}$ with D^t to cover this cost, since $|V_w^{t-c-id_1}|\geq D^{t-c-id_1}$.

Algorithm 6: `CycleAdditive`(G, R, c)

$t \leftarrow \lceil \frac{R}{2} \rceil$, $D \leftarrow n^{\frac{1}{2t-c}}$;
$C \leftarrow$ `SparseOrCycle`$(G, D, t-c, t)$;
if $C \neq$ *null* **then return** C;
$d \leftarrow c$;
$q \leftarrow \lfloor \frac{t-c}{d} \rfloor$;
repeat
  for $i \leftarrow 1$ *to* q **do**
    $C \leftarrow$ `SparseOrCycle`$(G, D, t-c-i \cdot d, t)$;
    if $C \neq$ *null* **then return** C;
  $r \leftarrow (t-c) \mod d$;
  $d \leftarrow t \mod r$;
  $q \leftarrow \lfloor \frac{t-c}{d} \rfloor$;
until $r = 0$;
return No ;

Consider now the case of an iteration $j > 1$ of the outer loop. Recall that the values of r, d and q in the beginning of the jth iteration are denoted with r_j, d_j and q_j.

Notice that $r_j = (t-c) \mod d_{j-1} = (t-c) - q_{j-1}d_{j-1}$, thus, when the $(j-1)$th iteration of the outer loop ends it follows from Lemma 7.5(ii) that `IsDense`$(\hat{G}_{j-1}^{q_{j-1}}, u, D^{r_j}, r_j) = No$ for every u in $\hat{G}_{j-1}^{q_{j-1}}$. From Corollary 7.1 it follows that `IsDense`$(\hat{G}_{j-1}^{q_{j-1}}, u, D^{z \cdot r_j}, z \cdot r_j) = No$, for every integer z. Since $t - d_j$ is a multiple of r_j and $G_j^i \subseteq \hat{G}_{j-1}^{q_{j-1}}$, where $1 \leq i \leq q_j$, it follows from Lemma 7.2 that `IsDense`$(G_j^i, u, D^{t-d_j}, t-d_j) = No$ for every u in G_i^j.

Consider the call to `SparseOrCycle`$(G_j^i, D, t-c-id_j, t)$. For every $w \in W_j^i$ the call to `IsDense` returned Yes and since no cycle was found, it follows from Lemma 7.1 that $|V_w^{t-c-id_j}| \geq D^{t-c-id_j}$. The cost of a call to `BallOrCycle` with w is $O(|V_w^{t-c-id_j+t}|)$. Since w survived the $(i-1)$th iteration of the inner loop and is in $\hat{G}_j^{i-1}$ and since $G_j^i \subseteq \hat{G}_j^{i-1}$ it follows from Lemma 7.5(ii) and Lemma 7.2 that `IsDense`$(G_j^i, w, D^{t-c-(i-1)d_j}, t-c-(i-1)d_j) = No$.

As $G_j^i(w) \subseteq G_j^i$, it follows from Lemma 7.2 that `IsDense`$(G_j^i(w), w, D^{t-c-(i-1)d_j}, t-c-(i-1)d_j) = No$ and `IsDense`$(G_j^i(w), u, D^{t-d_j}, t-d_j) = No$ for every u in $G_j^i(w)$. We use this to apply Lemma 7.3 and bound the $O(|V_w^{t-c-id_j+t}|)$ cost of `BallOrCycle` with D^{t-c-id_j+t}. We can charge every vertex of $V_w^{t-c-id_j}$ with D^t to cover the cost of the call to `BallOrCycle` with w since $|V_w^{t-c-id_j}| \geq D^{t-c-id_j}$. ∎

Notice that using Lemma 7.6 we can compute efficiently an approximation whenever the girth is bounded by $O(\log^x n)$, for some constant x. There are, however, degenerated cases of very sparse graphs with very large girth. In such a case the factor of $(t-c)$ in the $O((t-c) \cdot n^{1+\frac{t}{2t-c}})$ running time becomes dominant. We can avoid the $(t-c)$ factor at a price of slightly worse approximation.

LEMMA 7.7. *Let $G = (V, E)$. Let R and c be two parameters and let $t = \lceil R/2 \rceil$. If* `CycleAdditive'`(G, R, c) *returns a cycle C then $wt(C) \leq 4t - c$. If* `CycleAdditive'`(G, R, c) *returns No then the girth of G is strictly more than $2t$. The running time of* `CycleAdditive'`(G, R, c) *is $O(n^{1+\frac{t}{2t-c}})$.*

We defer the details regarding the implementation of `CycleAdditive'`(G, R, c) and the proof of Lemma 7.7 to the Appendix.

We can now use Lemma 7.6 and Lemma 7.7 to prove the following result.

THEOREM 7.1. *Let $G = (V, E)$ be a given n-vertex, m-edge unweighted undirected graph with unknown girth g, and let $\epsilon \in (0, 1)$ be given. There is an algorithm that computes a cycle C in $\widetilde{O}(n^{1+1/(2-\epsilon)})$ time such that $wt(C) \leq 4\lceil \frac{g}{2} \rceil - 2\lfloor \epsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\epsilon)g + 4$ if $g \leq \log^2 n$ and $wt(C) \leq 4\lceil \frac{g}{2} \rceil - \lfloor \epsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\epsilon/2)g + 3$ if $g > \log^2 n$.*

Proof. Assume that $g < \log^2 n$. Let $t = \lceil g/2 \rceil$. Let $c(x, \epsilon) = \lfloor x\epsilon \rfloor$, where x is an integer.

We do a binary search in the range $[1, \log^2 n]$. We denote with R^{YES} a value for which a call to $\texttt{CycleAdditive}(G, R^{\text{YES}}, c(\lceil R^{\text{YES}}/2 \rceil, \epsilon))$ returns a cycle. We denote with R^{NO} a value for which a call to $\texttt{CycleAdditive}(G, R^{\text{NO}}, c(\lceil R^{\text{NO}}/2 \rceil, \epsilon))$ does not return a cycle. At the beginning of the search we set R^{YES} to $\lceil \log^2 n \rceil$ and R^{NO} to 1. As long as $R^{\text{YES}} - R^{\text{NO}} > 1$ we set R to $R^{\text{NO}} + \lfloor (R^{\text{YES}} - R^{\text{NO}})/2 \rfloor$ and call to $\texttt{CycleAdditive}(G, R, c(\lceil R/2 \rceil, \epsilon))$. If it returns a cycle we set R^{YES} to R and if not we set R^{NO} to R. Once $R^{\text{YES}} - R^{\text{NO}} = 1$ we output the result of $\texttt{CycleAdditive}(G, R^{\text{YES}}, c(\lceil R^{\text{YES}}/2 \rceil, \epsilon))$.

From Lemma 7.6 it follows that $R^{\text{NO}} < g$ since $\texttt{CycleAdditive}(G, R^{\text{NO}}, c(\lceil R^{\text{NO}}/2 \rceil, \epsilon))$ does not returns a cycle. Since $R^{\text{YES}} - R^{\text{NO}} = 1$ we get that $R^{\text{YES}} \leq g$ and since the call to $\texttt{CycleAdditive}(G, R^{\text{YES}}, c(\lceil R^{\text{YES}}/2 \rceil, \epsilon))$ returns a cycle C such that $wt(C) \leq 4(\lceil R^{\text{YES}}/2 \rceil) - 2c(\lceil R^{\text{YES}}/2 \rceil, \epsilon)$, we get that $wt(C) \leq 4\lceil \frac{g}{2} \rceil - 2c(\lceil \frac{g}{2} \rceil, \epsilon)$.

We now get $4\lceil \frac{g}{2} \rceil - 2c(\lceil \frac{g}{2} \rceil, \epsilon) \leq 2g + 2 - 2\lfloor \epsilon g/2 \rfloor$, since $c(\lceil \frac{g}{2} \rceil, \epsilon) = \lfloor \lceil \frac{g}{2} \rceil \epsilon \rfloor \leq \lfloor \epsilon g/2 \rfloor$. Now $2g + 2 - 2\lfloor \epsilon g/2 \rfloor \leq 2g + 2 - 2(\epsilon g/2 - 1) = 2g + 4 - \epsilon g = (2 - \epsilon)g + 4$.

When $g \geq \log^2 n$ we simply switch the call to $\texttt{CycleAdditive}$ with a call to $\texttt{CycleAdditive}'$. In this case $\texttt{CycleAdditive}'(G, R^{\text{YES}}, c(\lceil R^{\text{YES}}/2 \rceil, \epsilon))$ returns a cycle of length at most $4(\lceil R^{\text{YES}}/2 \rceil) - c(\lceil R^{\text{YES}}/2 \rceil, \epsilon)$. This is bounded by $4\lceil \frac{g}{2} \rceil - c(\lceil \frac{g}{2} \rceil, \epsilon)$. We now get $4\lceil \frac{g}{2} \rceil - c(\lceil \frac{g}{2} \rceil, \epsilon) \leq 2g + 2 - \lfloor \epsilon g/2 \rfloor$, since $c(\lceil \frac{g}{2} \rceil, \epsilon) = \lfloor \lceil \frac{g}{2} \rceil \epsilon \rfloor \leq \lfloor \epsilon g/2 \rfloor$. Now $2g + 2 - \lfloor \epsilon g/2 \rfloor \leq 2g + 2 - (\epsilon g/2 - 1) = 2g + 3 - \epsilon g/2 = (2 - \epsilon/2)g + 3$.

Since we do not know in advance the value of g we run first the algorithm for the case that $g < \log^2 n$. If no cycle is returned then we run the algorithm for $g \geq \log^2 n$. ∎

References

[AV21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021.

[AYZ97] Noga Alon, Raphy Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[BS74] John A. Bondy and Miklós Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory*, pages 16:97–16:105, 1974.

[CL21] Shiri Chechik and Gur Lifshitz. Optimal girth approximation for dense directed graphs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 290–300. SIAM, 2021.

[CLRS20] Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1010–1023. ACM, 2020.

[DKS17a] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In *Proc. of 49th STOC*, pages 112–120, 2017.

[DKS17b] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. New subquadratic approximation algorithms for the girth. *CoRR*, abs/1704.02178, 2017.

[Duc19] Guillaume Ducoffe. Faster approximation algorithms for computing shortest cycles on weighted graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[DV20] Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. In *Proceedings of the 47th ICALP*, volume 168 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[DVV19] Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: hardness for all induced patterns and faster non-induced cycles. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1167–1178. ACM, 2019.

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[LL09] Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009.

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

[PRS+18] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1374–1392. SIAM, 2018.

[RT13] Liam Roditty and Roei Tov. Approximating the girth. *ACM Trans. Algorithms*, 9(2):15:1–15:13, 2013.

[RV12] Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 833–845. SIAM, 2012.

[Vas19] Virginia Vassilevska Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians ICM 2018*, volume 3, pages 3447–3487, 2019.

[VW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

[Wil18] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

[YZ97] Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2):209–222, 1997.

[YZ04] Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 254–260. SIAM, 2004.

A An additive approximation for very sparse graphs

Algorithm 7: `CycleAdditive'`(G, R, c)

$t \leftarrow \lceil \frac{R}{2} \rceil$, $D \leftarrow n^{\frac{1}{2t-c}}$;
$q \leftarrow \lfloor \frac{t-c}{c} \rfloor$;
$r \leftarrow (t-c) \mod c$;
$C \leftarrow$ `SparseOrCycle`$(G, D, t-c, t)$;
if $C \neq$ *null* **then return** C;
for *all* $w \in V$ **do**
  for $i \leftarrow 1$ *to* $q-1$ **do**
    if *IsDense*$(G, w, D^{ic}, ic) = Yes$ and *IsDense*$(G, w, D^{(i+1)c}, (i+1)c) = No$ **then**
      $(H(w, ic+t), C) \leftarrow$ `BallOrCycle`$(w, ic+t)$;
      if $C \neq$ *null* **then return** C;
      else
        $G \leftarrow G \setminus V_w^{ic-1}$;
        exit the inner for loop and proceed with a new vertex ;
$C \leftarrow$ `SparseOrCycle`$(G, D, t - \lfloor (c+r)/2 \rfloor, t)$;
if $C \neq$ *null* **then return** C;
$C \leftarrow$ `SparseOrCycle`$(G, D, t - \lceil (c+r)/2 \rceil, t)$;
if $C \neq$ *null* **then return** C;
$C \leftarrow$ `SparseOrCycle`$(G, D, t-c-r, t)$;
if $C \neq$ *null* **then return** C;
return No;

For the case of $g > \log^2 n$ we use a different approach that allows us to avoid the factor of $t-c$ in the running time of $O((t-c) \cdot n^{1+\frac{t}{2t-c}})$ of `CycleAdditive`.

Let $q = \lfloor \frac{t-c}{c} \rfloor$ and $r = (t-c) \mod c$. We start by running `SparseOrCycle`$(G, D, t-c, t)$. If a cycle is not returned we proceed and scan the set V using a for-all loop. For every $w \in V$ we execute an inner loop implemented using a for loop with index i running from 1 to $q-1$. If we encounter $1 \leq i \leq q-1$ such that `IsDense`$(G, w, D^{ic}, ic) = Yes$ and `IsDense`$(G, w, D^{(i+1)c}, (i+1)c) = No$ we run `BallOrCycle`$(w, ic+t)$. If a cycle is returned we return the cycle and stop, if not we remove V_w^{ic-1} from G and exit from the inner for loop and proceed to the next vertex in the outer for-all loop. Notice that it is easy to implement `IsDense` such that `IsDense`$(G, w, D^{(i+1)c}, (i+1)c)$ uses the information computed by `IsDense`(G, w, D^{ic}, ic) to avoid an extra factor

of q in the running time.

If a cycle is not reported during the entire execution of the for-all loop then we make three additional calls to `SparseOrCycle`, where each subsequent call is executed only if the previous call did not report a cycle. We first call to `SparseOrCycle`$(G, D, t - \lfloor (c+r)/2 \rfloor, t)$ then if needed we call to `SparseOrCycle`$(G, D, t - \lceil (c+r)/2 \rceil, t)$ and then, again only if needed, we call to `SparseOrCycle`$(G, D, t - c - r, t)$.

We denote this version of `CycleAdditive` with `CycleAdditive`$'$. The algorithm is presented in Algorithm 7. Next, we prove the correctness and the running time of `CycleAdditive`$'$.

Reminder of Lemma 7.7. *Let $G = (V, E)$. Let R and c be two parameters and let $t = \lceil R/2 \rceil$. If `CycleAdditive`$'(G, R, c)$ returns a cycle C then $wt(C) \leq 4t - c$. If `CycleAdditive`$'(G, R, c)$ returns No then the girth of G is strictly more than $2t$. The running time of `CycleAdditive`$'(G, R, c)$ is $O(n^{1+\frac{t}{2t-c}})$.*

Proof. We first show that if `CycleAdditive`$'(G, R, c)$ returns a cycle C then $wt(C) \leq 4t - c$. During the run of the algorithm every call to `SparseOrCycle`(G, D, x, y) is with $x \leq t - \lfloor (c+r)/2 \rfloor$ and $y = t$. Since $1 \leq r < c$, we have $t - \lfloor (c+r)/2 \rfloor \leq t - \lceil c/2 \rceil$. Thus, when `SparseOrCycle` returns a cycle C it follows from Lemma 7.5 that $wt(C) \leq 2(x + y) \leq 4t - c$.

If `BallOrCycle`$(w, ic + t)$ returns a cycle C for some i and w, then since $i \leq q - 1$ we have $ic \leq (q-1)c \leq t - 2c - r$, and from Lemma 3.3 we get $wt(C) \leq 2(2t - 2c - r) \leq 4t - 4c - 2r$.

We now turn to show that if `CycleAdditive`$'(G, R, c)$ returns No then the girth of G is strictly more than $2t$. Notice that in this case every call to `SparseOrCycle` and `BallOrCycle` returns No.

Assume the algorithm returns No and assume also for the sake of contradiction that $C = \{u_1, \ldots, u_{t'}\}$ is a cycle in the input graph G such that $wt(C) \leq 2t$.

We first show that if a vertex of C is being removed at some stage of the run of the algorithm we reach a contradiction. Let $u \in C$ be the first vertex of C that is being removed. When u is being removed it is because $u \in V_w^x$, for some $x < t$ right after a call to `BallOrCycle`$(w, x + t)$ returned No (either initiated directly by the algorithm or by one of the calls to `SparseOrCycle`). However, since u is the first to be removed C is in the graph when `BallOrCycle`$(w, x + t)$ is called, and from Lemma 3.3 it follows that `BallOrCycle`$(w, x + t)$ should have report a cycle, a contradiction.

Thus, it cannot be that a vertex of C is removed without a cycle is being reported. But if no vertex of C is being removed there will be a call to `BallOrCycle`$(u, x + t)$, where $x \leq t - \lceil c/2 \rceil$ for some $u \in C$, while C is in the graph and a cycle must be reported since $wt(C) \leq 2t$, again a contradiction to the fact that the algorithm returns No.

We now turn to analyze the running time. Let $D = n^{\frac{1}{2t-c}}$. As before we analyze the running time as long as no cycle is reported. The cost of reporting a cycle is at most $O(n)$ and this can only happen once. The first call to `SparseOrCycle` is to `SparseOrCycle`$(G, D, t - c, t)$. The analysis of the running time in this case is the same as in Lemma 7.6.

Notice that from Lemma 7.5(ii) it follows that if `SparseOrCycle`$(G, D, t - c, t)$ terminated without reporting a cycle then `IsDense`$(\hat{G}, w, D^{t-c}, t - c) = No$, for every $w \in \hat{V}$. From Lemma 7.2 it follows that any future call to `IsDense`$(G', w, D^{t-c}, t - c)$, where $G' \subseteq \hat{G}$, returns No.

We now analyze the cost and the effect on the graph of the for-all loop. Let $w \in V$ be a vertex considered by the for-all loop. First, the total cost of calling `IsDense`(G, w, D^{ic}, ic), for all $i \in [1, q]$ is only $O(D^{t-c-r})$ and not $O(qD^{t-c-r})$ since we assume `IsDense` is implemented such that the call to `IsDense`$(G, w, D^{(i+1)c}, (i+1)c)$, where $i \in [1, q-1]$, uses the information computed by `IsDense`(G, w, D^{ic}, ic). Consider now the cost of `BallOrCycle`. We call `BallOrCycle`$(w, ic + t)$ if we encounter for w an i such that `IsDense`$(G, w, D^{ic}, ic) = Yes$ and `IsDense`$(G, w, D^{(i+1)c}, (i+1)c) = No$, where $1 \leq i \leq q - 1$.

The cost of `BallOrCycle`$(w, ic + t)$ is $O(|V_w^{ic+t}|)$. Since `IsDense`$(G, w, D^{ic}, ic) = Yes$ and there is no cycle it follows from Lemma 7.1 that $|V_w^{ic}| \geq D^{ic}$.

Now since `IsDense`$(G, w, D^{(i+1)c}, (i+1)c) = No$ and `IsDense`$(G, u, D^{t-c}, t - c) = No$, for every $u \in G$ we can use Lemma 7.3 to bound the $O(|V_w^{ic+t}|)$ cost of `BallOrCycle` with $D^{(i+1)c+t-c}$. We can charge every vertex of V_w^{ic} with D^t to cover this cost, since $|V_w^{ic}| \geq D^{ic}$.

We now analyze the cost of the three additional calls to `SparseOrCycle`. Let G_ℓ be the input graph to the ℓth call to `SparseOrCycle` after the for-all loop ends, where $\ell \in \{1, 2, 3\}$. Similarly, let W_ℓ be the set W of the ℓth call. Recall that from Lemma 7.4 it follows that the running time of `SparseOrCycle`(G, D, x, y) is $O(|V|D^x + \sum_{w \in W} |V_w^{x+y}|)$. In each of the three calls to `SparseOrCycle` we have $x \leq t$, thus the term $O(|V|D^x)$

Girth	[DKS17b]		Lemma 7.6	
	Running Time	$wt(C) \leq$	Running Time	$wt(C) \leq$
4	$O(n^{1.666})$	6	$O(n^{1.666})$	6
5	$O(n^{1.75})$	8	$O(n^{1.75})$	8
6	$O(n^{1.666})$	10	$O(n^{1.6})$	10
7	$O(n^{1.666})$	12	$O(n^{1.666})$	12
8	$O(n^{1.666})$	12	$O(n^{1.571})$	14
9	$O(n^{1.666})$	16	$O(n^{1.625})$	16
10	$O(n^{1.666})$	16	$O(n^{1.555})$	18
11	$O(n^{1.666})$	18	$O(n^{1.6})$	20
12	$O(n^{1.666})$	18	$O(n^{1.538})$	22
13	$O(n^{1.666})$	22	$O(n^{1.583})$	24
14	$O(n^{1.666})$	22	$O(n^{1.533})$	26
15	$O(n^{1.666})$	24	$O(n^{1.571})$	28
16	$O(n^{1.666})$	24	$O(n^{1.529})$	30
17	$O(n^{1.666})$	28	$O(n^{1.5625})$	32
18	$O(n^{1.666})$	28	$O(n^{1.526})$	34
19	$O(n^{1.666})$	30	$O(n^{1.555})$	36
20	$O(n^{1.666})$	30	$O(n^{1.526})$	38

Table 1: Comparison between Lemma 7.6 and [DKS17b]

is at most $O(n^{1+t/(2t-c)})$ in each call.

We now focus on the term $O(\sum_{w \in W_\ell} |V_w^{x+y}|)$, for $\ell \in \{1,2\}$. Let $c_1 = \lfloor (c+r)/2 \rfloor$ and let $c_2 = \lceil (c+r)/2 \rceil$. We have $t - c \leq t - c_2 \leq t - c_1$.

For every $w \in W_\ell$, where $\ell \in \{1,2\}$ the call to `IsDense` with w returned Yes and since no cycle was found, it follows from Lemma 7.1 that $|V_w^{t-c_\ell}| \geq D^{t-c_\ell} \geq D^{t-c}$. The call to `BallOrCycle`$(w, t - c_\ell + t)$ costs $O(n) = O(D^{2t-c})$, since $D = n^{1/(2t-c)}$. Since $|V_w^{t-c_\ell}| \geq D^{t-c}$ and the vertices of $V_w^{t-c_\ell}$ are removed from the graph and will not be considered any more, we can charge every vertex of $V_w^{t-c_\ell}$ with a cost of D^t to cover the $O(D^{2t-c})$ cost of `BallOrCycle`$(w, t - c_\ell + t)$. Thus, $O(\sum_{w \in W_1 \cup W_2} |V_w^{x+y}|) = O(n^{1+t/(2t-c)})$.

We now turn to bound the $O(\sum_{w \in W_3} |V_w^{x+y}|)$ cost of the last call to `SparseOrCycle`. For every $w \in W_3$ the call to `IsDense` with w returned Yes and since no cycle was found, it follows from Lemma 7.1 that $|V_w^{t-c-r}| \geq D^{t-c-r}$.

The call to `BallOrCycle`$(w, t - c - r + t)$ costs $O(V_w^{2t-c-r})$. Since every vertex in G_3 survived the calls to `SparseOrCycle`$(G_1, D, t - c_1, t)$ and `SparseOrCycle`$(G_2, D, t - c_2, t)$ it follows from Lemma 7.5(ii) and Lemma 7.2 that `IsDense`$(G_3(w), u, D^{t-c_1}, t - c_1) = No$ and `IsDense`$(G_3(w), u, D^{t-c_2}, t - c_2) = No$, for every $u \in G_3$. We can use Lemma 7.3 together with the fact that $c_1 + c_2 = c + r$ to bound the $O(|V_w^{2t-c-r}|)$ cost of `BallOrCycle` with $O(D^{2t-c-r})$. We can charge every vertex of V_w^{t-c-r} with D^t to cover this cost, since $|V_w^{t-c-r}| \geq D^{V_w^{t-c-r}}$. ∎

B Odd girth approximation

In this section we present additional algorithms for approximating the girth of an unweighted undirected graph of odd girth g. The main result of this section is the following theorem.

Theorem B.1. (Odd Girth Approximation) *There is an algorithm which given any n-vertex m-edge undirected graph, integer $k \geq 2$, and integer odd g that is at least the girth of g, finds a cycle of length $2k\lfloor \frac{g}{2} \rfloor + 2$ in time $O(m(n^2/m)^{1/k})$.*

To prove this theorem we provide a variant of our general girth approximation scheme, `Cycle` (Algorithm 2 from Section 4). The two key differences is that in this new algorithm, `CycleOdd` (Algorithm 8), `BallOrCycle` is always invoked from the vertex of largest degree and that rather than stopping `BallOrCycle` when the number of vertices in the ball graph doesn't increase too much, we instead consider the number of edges incident to the ball graph and carefully search for cycles in these edges.

Algorithm 8: `CycleOdd`(G, R, k)

1 **if** G *is empty* **then return** No;
2 Let $v \in V$ be the vertex of maximum degree in G;
3 $H(v, 0) \leftarrow (\{v\}, \emptyset)$;
4 **for** $i \leftarrow 1$ *to* k **do**
5 $\quad (H(v, iR), C) \leftarrow$ `BallOrCycle`$(v, i \cdot R)$;
6 $\quad$ **if** $C \neq$ `null` **then return** C ;
7 $\quad$ **if** $|E(V_v^{iR})| \geq n$ **then**
8 $\quad\quad (H(v, iR+1), C) \leftarrow$ `BallOrCycle`$(v, i \cdot R)$;
9 $\quad\quad$ **return** C
10 $\quad$ **if** $|E(V_v^{iR})| \leq (n/\deg(v))^{1/k} \cdot |E(V_v^{(i-1)R})|$ **then**
11 $\quad\quad$ **if** *there is* $e \in E(V_v^{iR})$ *with* $e \subseteq V_v^{iR}$ **then**
12 $\quad\quad\quad$ **return** the cycle consisting of e and the path from each endpoint of e to the LCA in $H(v, iR)$
13 $\quad\quad$ **else return** `CycleOdd`$(G \setminus E(V_v^{(i-1)R}), R, k)$;

To analyze this algorithm, we first provide the following variant of Lemma 3.1 and Lemma 3.2 that allows us to rule out what edges can be in cycle of length at most R for odd R.

LEMMA B.1. *Let $G = (V, E)$ be an unweighted undirected graph and let $v \in V$. Further, let E_k denote all the edges of $E(V_v^k)$ with both endpoints in V_v^k. If there is no cycle in $E_{k+\lfloor R/2 \rfloor}$ for odd integer R then all edges in $E(V_v^k)$ are not part of any cycle of length at most R in G.*

Proof. Proceed by contradiction and suppose that there is $\{a, b\} \in E(V_v^k)$ that is part of a cycle, C, of length at most R. Note that since $\{a, b\} \in E(V_u^k)$ either $d_G(v, a) \leq k$ or $d_G(v, b) \leq k$ and both $d_G(v, a) \leq k+1$ and $d_G(v, b) \leq k+1$. Further, since C has length at most R for odd R it is the case that for every vertex z in C either both $d_C(a, z) \leq \lfloor R/2 \rfloor$ and $d_C(b, z) \leq \lfloor R/2 \rfloor$, in which case either $d_G(v, z) \leq k + \lfloor R/2 \rfloor$ or $d_G(v, z) \leq k + \lfloor R/2 \rfloor$ (by triangle inequality), or either $d_C(a, z) \leq \lfloor R/2 \rfloor - 1$ or $d_C(b, z) \leq \lfloor R/2 \rfloor - 1$ in which case one of $d_G(v, z) \leq k + \lfloor R/2 \rfloor$ or $d_G(v, z) \leq k + \lfloor R/2 \rfloor$ (again by triangle inequality). In either case, this implies that $z \in V_v^{k+\lfloor R/2 \rfloor}$ and since every vertex of C is in $V_v^{k+\lfloor R/2 \rfloor}$ we have that C is in $E_{k+\lfloor R/2 \rfloor}$. ∎

Leveraging this we now prove that Algorithm `CycleOdd`(G, R, k) detects a cycle of length at most $2k\lfloor R/2 \rfloor + 1$ if G has a cycle of length at most R for odd R.

LEMMA B.2. *Let $G = (V, E)$ be an unweighted undirected graph. If G has a cycle C of length $\leq R$ for odd, integer R then Algorithm `CycleOdd`$(G, \lfloor R/2 \rfloor, k)$ returns a cycle of length at most $2k\lfloor R/2 \rfloor + 2$.*

Proof. Not that $|E(V_v^{0R})| = \deg(v)$ and that by Line 10 if the algorithm doesn't output a cycle on an iteration of the for loop for a given value of i, it must be the case that $|E(V_v^{iR})| > \deg(v)(n/\deg(v))^{i/k}$. However, since $\deg(v)(n/\deg(v))^{k/k} = n$, by Line 7 we see that thee algorithm must return on on some iteration $i \in [k]$.

Now, consider each case where a cycle is returned without a recursive call on Line 13. If a cycle is returned on Line 6 then by Lemma 3.3 it has length at most $2iR \leq 2kR$. On the other hand, if the algorithm returns on Line 9, then since $|E(V_v^{iR})| \geq n$ it must be the case that $H(v, iR+1)$ contains a cycle and therefore by Lemma 3.3, a cycle of length at most $2iR + 2 \leq 2kR + 2$ is returned. Finally, if a cycle is returned on Line 13 then it has length at most $2iR + 1$.

Consequently, it remains to how that on recursive calls the resulting graph is never empty. However, suppose G has a cycle C of length at most R. Note that before we remove $E(V_v^{(i-1)R})$ in a recursive call the method checks if there is a cycle within the edges that have both endpoints in V_v^{iR}. By Lemma B.1 this guarantees that C is not in $E(V_v^{(i-1)R}$. Consequently, C is never removed from the graph and since edges are deleted every recursive call the procedure must output a cycle of the desired length.

If a cycle is reported then it must be of length at most $2k\lfloor R/2 \rfloor + 1$ since for every source v for which we call `BallOrCycle`$(v, \lfloor R/2 \rfloor \cdot i)$, we have $i \leq k$ (since otherwise we would have $|E(V_v^{kR})| > \deg(v)(m/\deg(v))^{k/k}$ which is impossible). ∎

We now turn to analyze the running time of Algorithm `CycleOdd`(G, R, k).

LEMMA B.3. *Algorithm* `CycleOdd`(G, R, k) *runs in* $O(m(n^2/m)^{1/k})$ *time.*

Proof. First, note that in $O(m+n)$ time we can compute the degree of every vertex and create an array where element i of the array contains a list of all vertices of degree i. As edges are deleted this array and which is the largest degree can be maintained in amortized $O(1)$ per deletion and additional $O(n)$ time. Consequently, maintaining the largest degree vertex can be done in total $O(m+n)$ time.

Next, note that whenever a cycle is returned, the cost of that recursive execution is at most $O(m+n)$. Further, whenever `CycleOdd` is invoked recursively note that the runtime is at most $O((n/\deg(v))^{1/k})$ more than the number of edges removed (for $\deg(v)$ at the time the vertex was removed). To analyze this total cost. Let $e_1, ..., e_\ell$ denote an ordering of the edges in the order in which they were removed. Further, let d_i be the degree of the maximum degree vertex, m_i be the number of edges in the graph, and n_i be the number of vertices in the graph all at the time that edge i was removed. Note, that $m_i \geq m+1-i$ and $n_i \leq n$. Consequently, $d_i \geq m_i/n_i \geq 2(m+1-i)/n$ (since the maximum degree vertex is at least the average degree. From this, we see that the total cost for processing these edges is

$$O\left(\sum_{i\in\ell}(n/d_i)^{1/k}\right) \leq O\left(\sum_{i\in[m]}\left(\frac{n^2}{2(m+1-i)}\right)^{1/k}\right) = O\left(m(n^2/m)^{1/k}\right)$$

Where in the last step we used that $\int_{i=1}^{m} x^{-1/k}dx = \frac{1}{1-\frac{1}{k}}[m^{1-1/k}-1] = O(m^{1-1/k})$ for $k \geq 2$. ∎

Combining yields the main theorem of this section

Proof. [Proof of Theorem B.1] Correctness follows from Lemma B.2 and runtime follows from Lemma B.3 ∎