New Algorithms for Girth and Cycle Detection

Liam Roditty
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Plia Trabelsi
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Abstract

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. Let g be the girth of G, that is, the length of a shortest cycle in G. We present a randomized algorithm with a running time of $\tilde{O}\left(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}}\right)$ that returns a cycle of length at most $2\ell\left\lceil\frac{g}{2}\right\rceil - 2\left\lfloor\varepsilon\left\lceil\frac{g}{2}\right\rceil\right\rfloor$, where $\ell \geq 2$ is an integer and $\varepsilon \in [0, 1]$, for every graph with $g = \text{polylog}(n)$.

Our algorithm generalizes an algorithm of Kadria et al. [SODA'22] that computes a cycle of length at most $4\left\lceil\frac{g}{2}\right\rceil - 2\left\lfloor\varepsilon\left\lceil\frac{g}{2}\right\rceil\right\rfloor$ in $\tilde{O}\left(n^{1+\frac{1}{2-\varepsilon}}\right)$ time. Kadria et al. presented also an algorithm that finds a cycle of length at most $2\ell\left\lceil\frac{g}{2}\right\rceil$ in $\tilde{O}\left(n^{1+\frac{1}{\ell}}\right)$ time, where ℓ must be an integer. Our algorithm generalizes this algorithm, as well, by replacing the integer parameter ℓ in the running time exponent with a real-valued parameter $\ell - \varepsilon$, thereby offering greater flexibility in parameter selection and enabling a broader spectrum of combinations between running times and cycle lengths.

We also show that for sparse graphs a better tradeoff is possible, by presenting an $\tilde{O}(\ell \cdot m^{1+\frac{1}{\ell-\varepsilon}})$ time randomized algorithm that returns a cycle of length at most $2\ell(\lfloor\frac{g-1}{2}\rfloor) - 2(\lfloor\varepsilon\lfloor\frac{g-1}{2}\rfloor\rfloor + 1)$, where $\ell \geq 3$ is an integer and $\varepsilon \in [0, 1)$, for every graph with $g = \text{polylog}(n)$.

To obtain our algorithms we develop several techniques and introduce a formal definition of *hybrid* cycle detection algorithms. Both may prove useful in broader contexts, including other cycle detection and approximation problems. Among our techniques is a new cycle searching technique, in which we search for a cycle from a given vertex and possibly all its neighbors in linear time. Using this technique together with more ideas we develop two hybrid algorithms. The first allows us to obtain an $\tilde{O}(m^{2-\frac{2}{\lceil g/2\rceil+1}})$-time, $(+1)$-approximation of g. The second is used to obtain our $\tilde{O}(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}})$-time and $\tilde{O}(\ell \cdot m^{1+\frac{1}{\ell-\varepsilon}})$-time approximation algorithms.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph algorithms, All pairs shortest path, Girth, Cycle approximation

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.38

Related Version *Full Version*: https://arxiv.org/abs/2507.02061 [10]

Funding *Liam Roditty*: Supported in part by BSF grants 2016365 and 2020356.

1 Introduction

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. A set of vertices $C_\ell = \{v_1, v_2, \cdots, v_{\ell+1}\}$ in G, where $\ell \geq 2$, is a cycle of length ℓ if $v_1 = v_{\ell+1}$ and $(v_i, v_{i+1}) \in E$, for every $1 \leq i \leq \ell$. A $C_{\leq\ell}$ is a cycle of length at most ℓ. The *girth* g of G is the length of a shortest cycle in G. The girth of a graph has been studied extensively since the 1970s by researchers from both the graph theory and the algorithms communities.

Itai and Rodeh [6] showed that the girth can be computed in $O(mn)$ time or in $O(n^\omega)$ time, where $\omega < 2.371552$ [16], if Fast Matrix Multiplication (FMM) algorithms are used. They also proved that the problem of computing the girth is equivalent to the problem of deciding whether there is a C_3 (triangle) in a graph or not.

In practice, algorithms for FMM have very large constant factors in their running time. Combinatorial algorithms, informally, are algorithms which do not use algebraic methods that are being used by FMM algorithms, and consequently are often more practical. Vassilevska W.

and Williams [15] showed that if there exists a truly subcubic time combinatorial algorithm which detects if a graph has a triangle (and therefore also a subcubic time algorithm that computes the exact girth), then there exists a truly subcubic time combinatorial algorithm for Boolean Matrix Multiplication (BMM) (and therefore also for unweighted All Pairs Shortest Path (APSP), see [5], [13], [14]). Such an algorithm would be a major breakthrough. As a result, to get a faster running time for computing the girth, it is natural to settle for an *approximation* algorithm for the girth instead of an exact computation. An (α, β)-approximation $\hat{g}$ of g (where $\alpha \geq 1$ and $\beta \geq 0$), satisfies $g \leq \hat{g} \leq \alpha \cdot g + \beta$. We denote an approximation as an α-approximation if $\beta = 0$ and as a $(+\beta)$-approximation if $\alpha = 1$.

Itai and Rodeh [6] presented a $(+1)$-approximation algorithm that runs in $O(n^2)$ time. Notice that in contrast to the BMM or APSP problems, where a running time of $\Omega(n^2)$ is inevitable since the output size is $\Omega(n^2)$, in the girth problem the output is a single number, thus, there is no natural barrier for subquadratic time algorithms. Indeed, Lingas and Lundell [8] presented a $\frac{8}{3}$-approximation algorithm that runs in $\tilde{O}(n^{3/2})$ time, and Roditty and V. Williams [12] presented a 2-approximation algorithm that runs in $\tilde{O}(n^{5/3})$ time. Dahlgaard, Knudsen and Stöckel [4] presented two tradeoffs between running time and approximation. One generalizes the algorithms of [8, 12] and computes a cycle of length at most $2\lceil \frac{g}{2} \rceil + 2\lceil \frac{g}{2(\ell-1)} \rceil$ in $\tilde{O}(n^{2-1/\ell})$ time. The other computes, whp, a $C_{\leq 2^\ell g}$, for any integer $\ell \geq 2$, in $\tilde{O}(n^{1+1/\ell})$ time.

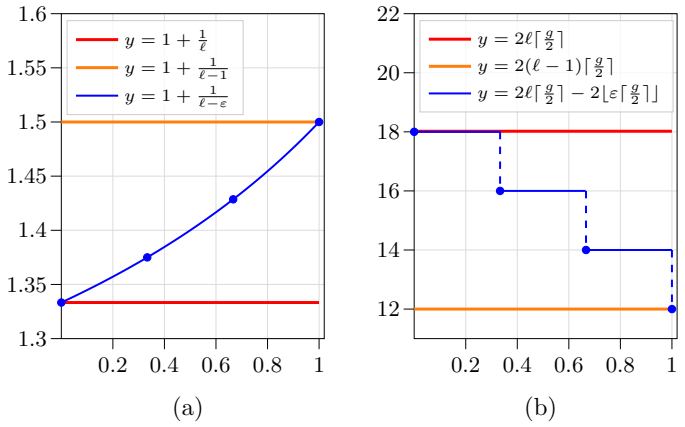
Kadria et al. [7] significantly improved upon the second algorithm of [4] and presented an algorithm, that for every integer $\ell \geq 1$, computes a $C_{\leq 2\ell\lceil g/2 \rceil}$ in $\tilde{O}(n^{1+1/\ell})$ time. They also presented an algorithm, that for every $\varepsilon \in (0, 1)$, computes a cycle of length at most $4\lceil \frac{g}{2} \rceil - 2\lfloor \varepsilon \lceil \frac{g}{2} \rceil \rfloor \leq (2-\varepsilon)g + 4$, in $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time, for every graph with $g = \text{polylog}(n)$.

These two algorithms of Kadria et al., as well as few other approximation algorithms (see for example, [8], [2], [9]), were obtained using a general framework for girth approximation in which a search is performed over the range of possible values of g, using some algorithm $\mathcal{A}$ that gets as an input an integer $\tilde{g}$ which is a guess for the value of g. In each step of the search $\mathcal{A}$ either returns a cycle $C_{\leq f(\tilde{g})}$, where f is a non decreasing function, or determines that $g > \tilde{g}$. The goal of the search is to find the smallest $\tilde{g}$, for which $\mathcal{A}$ returns a cycle, because for this value we have $g > \tilde{g} - 1$ (and thus $g \geq \tilde{g}$), and algorithm $\mathcal{A}$ returns a $C_{\leq f(\tilde{g})}$. This cycle is of length at most $f(g)$ since $g \geq \tilde{g}$ and f is a non decreasing function (f can represent the approximation, for example $f(\tilde{g}) = 2\tilde{g}$ yields a 2-approximation). The two possible outcomes of $\mathcal{A}$ and its usage in the general girth approximation framework inspired us to formally define the notion of a (γ, δ)-*hybrid* algorithm as follows:

▶ **Definition 1.** *A (γ, δ)-hybrid algorithm is an algorithm that either outputs a $C_{\leq \gamma}$ or determines that $g > \delta$.*

When $\gamma = \delta$, the algorithm is referred to as a γ-hybrid algorithm. The girth approximation framework described above suggests that a possible approach for developing efficient girth approximation algorithms is by developing efficient (γ, δ)-hybrid algorithms.

Kadria et al. [7] designed several algorithms that satisfy the definition of (γ, δ)-hybrid algorithms. Their girth approximation algorithms mentioned above were obtained using two different $(f(\tilde{g}), \tilde{g})$-hybrid algorithms. Additionally, for every integer $k \geq 2$, they presented a $(2k, 3)$-hybrid and a $(2k, 4)$-hybrid algorithms that run in $O(\min\{m^{1+1/(k+1)}, n^{1+2/k}\})$ time, and a $(\max\{2k, g\}, 5)$-hybrid algorithm that runs in $O(\min\{m^{1+2/(k+1)}, n^{1+3/k}\})$ time. Therefore, for $k \geq 2$, $\tilde{g} \in \{3, 4, 5\}$ and $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$, there is a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs in $O(\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time. A natural question is whether these three algorithms are only part of a general tradeoff between the running time, γ and δ.



ε	Time exp	Cycle bound
0	$1+\frac{3}{9}$	18
$\frac{1}{3}$	$1+\frac{3}{8}$	16
$\frac{2}{3}$	$1+\frac{3}{7}$	14
1	$1+\frac{3}{6}$	12

Figure 1 Our $\tilde{O}(\ell \cdot n^{1+\frac{1}{\ell-\varepsilon}})$-time girth approximation algorithm compared to the $\tilde{O}(n^{1+\frac{1}{\ell}})$-time algorithm of [7], for every $\varepsilon \in [0,1]$, choosing $\ell = 3$ and $g = 5$ or 6. (a) The y-axis is the exponent of n in the running time, and the x-axis is ε. (b) The y-axis is the upper bound on the length of the returned cycle, and the x-axis is ε. The blue points correspond to our algorithm at four specific choices of ε: $\varepsilon = 0, \frac{1}{3}, \frac{2}{3}, 1$ (see the table on the right).

▶ **Problem 1.1.** *Let $\tilde{g} \geq 6$ and $k \geq 2$ be two integers and let $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$. Is it possible to obtain a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs in $O(\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time?*

In this paper we present a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that runs, whp, in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time, for every $\tilde{g} \geq 3$. This algorithm provides an affirmative answer to Problem 1.1, albeit the $(\frac{k+1}{\alpha-1} + \alpha)$ factor in the running time.

Kadria et al. [7] presented also a $(2k, 6)$-hybrid algorithm whose running time is $\tilde{O}(\min\{nm^{\frac{1}{2}+\frac{1}{2(k+1)}}, n^{\frac{3}{2}+\frac{1}{k}}\})$, for every integer $k \geq 4$. The running time of our algorithm when $\tilde{g} = 6$ is $\tilde{O}(k \cdot \min\{m^{1+\frac{2}{k+1}}, n^{1+\frac{3}{k}}\})$. We improve the running time for every constant $k \geq 5$, in the case of $m < O(n^{1+1/k})$ (when $m \geq O(n^{1+1/k})$ the two algorithms run a similar procedure and thus have the same asymptotic running time).

Using our $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm we obtain a generalization of the $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time algorithm of Kadria et al. [7] that computes a cycle of length at most $4\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (2-\varepsilon)g + 4$, for every graph with $g = \text{polylog}(n)$. Our generalized algorithm runs in $\tilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$ time, whp, and returns a cycle of length at most $2\ell\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (\ell-\varepsilon)g + \ell + 2$, where $\ell \geq 2$ is an integer and $\varepsilon \in [0,1]$, for every graph with $g = \text{polylog}(n)$. We also show that if the graph is sparse then the approximation can be improved. More specifically, we present an algorithm that runs in $\tilde{O}(\ell \cdot m^{1+1/(\ell-\varepsilon)})$ time, whp, and returns a cycle of length at most $(\ell-\varepsilon)g - \ell + 2\varepsilon$, where $\ell \geq 3$ is an integer and $\varepsilon \in [0,1)$, for every graph with $g = \text{polylog}(n)$.

Our $\tilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$-time algorithm also generalizes the $\tilde{O}(n^{1+1/\ell})$-time algorithm of Kadria et al. [7], which computes a $C_{\leq 2\ell\lceil g/2\rceil}$ for every **integer** $\ell \geq 1$. In our algorithm, the integer parameter ℓ that appears in the exponent of the running time is replaced by a real-valued parameter $\ell - \varepsilon$. Thus, we introduce many new points on the tradeoff curve between running time and approximation ratio. Specifically, for every integer $\ell \leq \text{polylog}(n)$, up to $\lceil g/2 \rceil - 1$ additional tradeoff points are added.[1] For example, consider $\ell = 3$ and a

[1] Up to $\lceil g/2 \rceil - 1$ tradeoff points are added since for every such ℓ, when ε is a multiple of $\frac{1}{\lceil g/2 \rceil}$, we get an $\tilde{O}(n^{1+1/(\ell-\varepsilon)})$-time algorithm which computes a $C_{\leq 2(\ell-\varepsilon)\lceil g/2 \rceil}$.

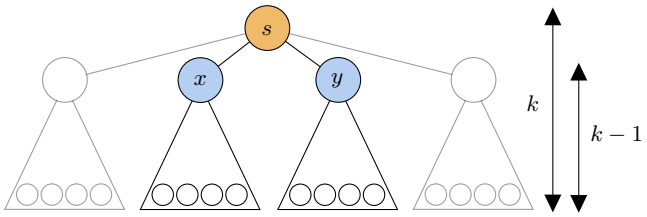


Figure 2 Disjoint sets of vertices at distance $k-1$ from x and y.

graph with girth $g = 5$ or $g = 6$. Our algorithm yields two additional points on the tradeoff curve, corresponding to $\varepsilon = \frac{1}{3}$ and $\varepsilon = \frac{2}{3}$. For $\varepsilon = \frac{1}{3}$, we compute a $C_{\leq 16}$ in $\tilde{O}(n^{1+\frac{3}{8}})$ time, and for $\varepsilon = \frac{2}{3}$, we compute a $C_{\leq 14}$ in $\tilde{O}(n^{1+\frac{3}{7}})$ time. These points lie between the two points on the tradeoff curve given by the algorithm of Kadria et al. [7], which computes either a $C_{\leq 12}$ in $\tilde{O}(n^{1+\frac{3}{6}})$ time or a $C_{\leq 18}$ in $\tilde{O}(n^{1+\frac{3}{9}})$ time. See Figure 1 for a comparison.

Together, the tradeoffs in the two algorithms of Kadria et al. [7] that we generalize encompass several other known results, including those of Itai and Rodeh [6], Lingas and Lundell [8], Roditty and V. Williams [12] and the first tradeoff of Dahlgaard et al. [4] (when $g = \text{polylog}(n)$). Notably, some of these algorithms have resisted improvement for many years. Therefore, the unification of these algorithms within a single tradeoff curve in our result, together with the addition of new valid points on this curve, reinforces the possibility that it captures a fundamental relationship between running time and approximation quality. This, in turn, motivates further investigation into whether a matching lower bound exists for this tradeoff.

The rest of this paper is organized as follows. In Section 2 we provide an overview. Preliminaries are in Section 3. In Section 4, we present a new cycle searching technique that is used by our algorithms. In Section 5 we present a $2k$-hybrid algorithm and then use it to obtain a $(+1)$-approximation algorithm for the girth. In Section 6 we generalize the $2k$-hybrid algorithm and present a $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm. In Section 7 we use the hybrid algorithm from Section 6 to obtain two more approximation algorithms for the girth.

2 Overview

Among the techniques that we develop to obtain our new algorithms, is a new cycle searching technique that might be of independent interest. Our new technique exploits the property that if $s \in V$ is not on a $C_{\leq 2k}$, then for any two neighbors x and y of s, the set of vertices at distance exactly $k-1$ from x and y that are also at distance k from s are disjoint (see Figure 2). This allows us to check efficiently for all the neighbors of s if they are on a $C_{\leq 2k}$. Using this technique, together with more tools that we develop, we obtain *two* hybrid algorithms.

The first is a relatively simple $O(m^{1+\frac{k-1}{k+1}})$-time, $2k$-hybrid algorithm. We use this hybrid algorithm in the girth approximation framework described earlier, to obtain an $\tilde{O}(m^{1+\frac{\ell-1}{\ell+1}})$-time, $(+1)$-approximation of the girth, where $g = 2\ell$ or $g = 2\ell - 1$ (the running time can also be written as $\tilde{O}(m^{2-\frac{2}{\lceil g/2 \rceil + 1}})$). We remark that using an algorithm of [3] for C_{2k} detection, it is possible to obtain a $(+1)$-approximation in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time (see Section 5.2 for more details). However, the additional $\ell^{O(\ell)}$ factor might be significant even for small values of ℓ.

The second is the $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm that solves Problem 1.1. Its main component is an $(2k, 2\alpha)$-hybrid algorithm that runs in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$-time, whp, and generalizes the first $O(m^{1+\frac{k-1}{k+1}})$-time $2k$-hybrid algorithm, by introducing an additional parameter $\alpha \leq k$. Using α we can tradeoff between the running time and the lower bound on g and obtain a faster running time at the price of a worse lower bound.

We compare our $(2k, 2\alpha)$-hybrid algorithm to algorithm `Cycle` of Kadria et al. [7], an $O(m + n^{1+\frac{1}{\ell}})$-time,[2] $(2\ell\alpha, \tilde{g})$-hybrid algorithm, where $\alpha = \lceil \tilde{g}/2 \rceil$, that they used to obtain the $\tilde{O}(n^{1+\frac{1}{\ell}})$-time, $2\ell\lceil g/2 \rceil$-approximation algorithm. As we show later, the running time of our $(2k, 2\alpha)$-hybrid algorithm can be bounded by $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot n^{1+\frac{\alpha}{k}})$. Since in our algorithm k is not necessarily a multiple of α (compared to the $\ell\alpha$ of `Cycle`), our algorithm allows more flexibility, and we achieve many more possible tradeoffs between the running time and the output cycle length. For example, if we consider a multiplicative approximation better than 3, when the value of g is a constant known in advance, our algorithm can return longer cycles that are still shorter than $3g$, in a faster running time.[3]

The flexibility of our algorithm is also demonstrated as follows. For a given constant value of k, if our $(2k, 2\alpha)$-hybrid algorithm returns a cycle then its length is at most $2k$. If we want algorithm `Cycle` to output a $C_{\leq 2k}$, then $\lfloor \frac{k}{\alpha} \rfloor$ is the largest ℓ that we can choose, since ℓ must be an integer. The running time is $O(n^{1+1/\ell}) = O(n^{1+1/\lfloor \frac{k}{\alpha} \rfloor})$. Our algorithm achieves a better running time if k is not divisible by α. (See the relevant figures in the full version of this paper [10] for a comparison).

Next, we overview our $(2k, 2\alpha)$-hybrid algorithm that either finds a $C_{\leq 2k}$ or determines that $g > 2\alpha$ in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time. In order to determine that $g \geq 2\alpha$, we can check for every $v \in V$ if v is on a $C_{\leq 2\alpha}$ (and either find a $C_{\leq 2k}$ since $\alpha \leq k$, or return that $g > 2\alpha$). If v is on a $C_{\leq 2\alpha}$, then all the vertices and edges of this $C_{\leq 2\alpha}$ are at distance at most α from v. If, for every $v \in V$, the number of edges at distance at most α is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$ then using standard techniques we can check for every $v \in V$ if v is on a $C_{\leq 2\alpha}$ in $O(m^{1+\frac{\alpha-1}{k+1}})$ time. However, this is not necessarily the case, and the region at distance at most α from some vertices might be dense. To deal with dense regions within the promised running time we develop an iterative sampling procedure (see `BfsSample` in Section 6), whose goal is to sparsify the graph, or to return a $C_{\leq 2k}$. One component of the iterative sampling procedure is a generalization of our new cycle searching technique mentioned above. In the generalization instead of checking whether a vertex s and its neighbors are on a $C_{\leq 2k}$, we check whether all the vertices up to a possibly further distance from s are on a $C_{\leq 2\alpha}$, for $\alpha \leq k$, and if not we mark them so that they can be removed later.

If the iterative sampling procedure ends without finding a $C_{\leq 2k}$ then there are two possibilities. Let $r = (k+1) \bmod (\alpha - 1)$. If $r = 0$ then it holds that the number of edges at distance at most α from every $v \in V$ is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$, whp, as required. If $r > 0$ then it holds that the number of edges at distance at most r from every $v \in V$ is $O(m^{\frac{r}{k+1}})$, whp. This does not necessarily imply that the graph is sparse enough for checking whether $g > 2\alpha$. In this case, we run another algorithm (see `HandleRemainder` in Section 6) that continues

[2] `Cycle` runs in $O(n^{1+\frac{1}{\ell}} + m)$ time, which can be reduced to $O(n^{1+\frac{1}{\ell}})$ time, as shown in [7].

[3] [7] presented also an $O((\alpha - c) \cdot n^{1+\frac{\alpha}{2\alpha-c}})$-time, $(4\alpha - 2c, 2\alpha)$-hybrid algorithm, where $0 < c \leq \alpha$ are integers. For $c = 2\alpha - k$, this is an $O((k - \alpha) \cdot n^{1+\frac{\alpha}{k}})$-time, $(2k, 2\alpha)$-hybrid algorithm, similar to our $(2k, 2\alpha)$-hybrid algorithm. However, since $0 < c \leq \alpha$, the possible values of k are restricted and must satisfy $\alpha \leq k < 2\alpha$. By choosing $\alpha = \lceil \frac{g}{2} \rceil$ and an appropriate k the two algorithms have a similar flexibility for a 2-approximation, but since in our algorithm also larger values of k are allowed, we can achieve a faster running time for a t-approximation where $t > 2$.

to sparsify the graph until the number of edges at distance at most α from every $v \in V$ is $O(deg(v) \cdot m^{\frac{\alpha-1}{k+1}})$ and checking whether $g > 2\alpha$ is possible within the required running time of $O(m^{1+\frac{\alpha-1}{k+1}})$.

3 Preliminaries

Let $G = (V, E)$ be an unweighted undirected graph with n vertices and m edges. Let $U \subseteq V$ be a set of vertices and let $G \setminus U$ be the graph obtained from G by deleting all the vertices of U together with their incident edges. For two graphs $G = (V, E)$ and $G' = (V', E')$, let $G \setminus G'$ be $G \setminus V'$. We say that $G \subseteq G'$ if $V \subseteq V'$ and $E \subseteq E'$. For convenience, we use both $u \in V$ and $u \in G$ to say that $u \in V$. For every $u, v \in V$, let $d_G(u, v)$ be the length of a shortest path between u and v in G. The *girth* g of G is the length of a shortest cycle in G. Let $wt(C)$ be the length of a cycle C. For an integer ℓ, we denote a cycle of length (at most) ℓ by $(C_{\leq \ell})$ C_ℓ.[4] Let $E(v)$ be the edges incident to v and $E(v, i)$ the ith edge in $E(v)$. Let $deg_G(v)$ be the degree of v in G. Let $N(v)$ be the set of neighbors of v, namely $N(v) = \{w \mid (v, w) \in E\}$. For an edge set S, let $V(S)$ be the endpoints of S's edges, that is, $V(S) = \{u \in V \mid \exists (u, v) \in S\}$. Let $e = (u, v) \in E$ and $w \in V$. The distance $d_G(w, e)$ between w and e is $\min\{d_G(w, u), d_G(w, v)\} + 1$. For every $u \in V$ and a real number k let $B(G, u, k) = (V_u^k(G), E_u^k(G))$ be the *ball graph* of u, where $V_u^k(G) = \{v \in V \mid d_G(u, v) \leq k\}$ and $E_u^k(G) = \{e \in E \mid d_G(u, e) \leq k\}$ [7]. For an integer $\ell \geq 0$ and a vertex $u \in V$, Let $L_u^\ell(G) = \{w \mid d_G(u, w) = \ell\}$.[5]

We now turn to present several essential tools that are required in order to obtain our new algorithms. We first restate an important property of the ball graph $B(v, R)$.

▶ **Lemma 2** ([7]). *Let $0 \leq t \leq R$ be two integers and let $v \in V$. If $B(v, R)$ is a tree then no vertex in V_v^{R-t} is part of a cycle of length at most $2t$ in G.*

We use procedure `BallOrCycle`(G, v, R) [7, 8] (pseudo-code in [10]) that searches for a $C_{\leq 2R}$ in the ball graph $B(v, R)$. We summarize `BallOrCycle`'s properties in the next lemma.

▶ **Lemma 3** ([7]). *Let $v \in V$. If the ball graph $B(v, R)$ is not a tree then* `BallOrCycle`(G, v, R) *returns a $C_{\leq 2R}$ from $B(v, R)$. If $B(v, R)$ is a tree then* `BallOrCycle`(G, v, R) *returns V_v^R.*[6] *The running time of* `BallOrCycle`(G, v, R) *is $O(|V_v^R|)$.*

Next, we obtain a simple t-hybrid algorithm, called `AllVtxBallOrCycle`, using `BallOrCycle`. `AllVtxBallOrCycle` (pseudo-code in [10]) gets a graph G and an integer $t \geq 2$, and runs `BallOrCycle`(v, t) from every $v \in G$ as long as no cycle is found by `BallOrCycle`. If `BallOrCycle` finds a cycle then `AllVtxBallOrCycle` stops and returns that cycle. If no cycle is found then `AllVtxBallOrCycle` returns `null`. We prove the next lemma in the full version of this paper [10].

▶ **Lemma 4.** `AllVtxBallOrCycle`(G, t) *either finds a $C_{\leq 2t}$ or determines that $g > 2t$, in $O(\sum_{v \in V} |V_v^t|)$ time.*

We now show that if the input graph satisfies a certain sparsity property then the running time of `AllVtxBallOrCycle`(G, t) can be bounded as follows.

---

[4] Both C_ℓ and $C_{\leq \ell}$ might not be simple cycles. However, the cycles that our algorithms return are simple.
[5] When the graph is clear from the context, we sometimes omit G from the notations.
[6] If V_v^R is returned then we assume that V_v^R is ordered by the distance from v, and for every $u \in V_v^R$ we store $d(u, v)$ with u. Thus, given the set V_v^R, we can find $V_v^{R'}$ for every $R' < R$ in $O(|V_v^{R'}|)$ time.

Algorithm 1 `SparseOrCycle`(G, D, x, y).

```
foreach w ∈ V do
    if IsDense(G, w, D^x, x) = Yes then
        (V_w^{x-1+y}, C) ← BallOrCycle(w, x − 1 + y);
        if C ≠ null then return C;
        G ← G \ V_w^{x-1};
return null;
```

▶ **Corollary 5.** *If $|E_u^{t-1}| < D^{t-1}$ for every $u \in V$ then* `AllVtxBallOrCycle`(G, t) *runs in $O(mD^{t-1})$ time.*

Proof. For every $v \in V$, we have $O(|V_v^t|) \leq O(|E_v^t|) \leq O(\sum_{u \in N(v)} |E_u^{t-1}|)$, which is at most $O(\sum_{u \in N(v)} D^{t-1}) = O(deg(v) \cdot D^{t-1})$. Thus, by Lemma 4, the running time of `AllVtxBallOrCycle`(G, t) is $O(\sum_{v \in V} |V_v^t|) \leq O(\sum_{v \in V} deg(v) \cdot D^{t-1}) = O(mD^{t-1})$. ◀

Next, we present procedure `IsDense`(G, w, T, r) from [7]. `IsDense` (pseudo-code in [10]) gets a graph G, a vertex w, a budget $T \geq 1$ (real) and a distance $r \geq 0$ (integer). In the procedure a BFS is executed from w. The BFS counts the edges that are scanned as long as their total number is less than T and the farthest vertex from w is at distance at most r.

▶ **Lemma 6** ([7])**.** *Procedure* `IsDense`(G, w, T, r) *runs in $O(\lceil T \rceil) = O(T)$ time. If* `IsDense`$(G, w, T, r) =$ `No` *then $|E_w^r| < T$. If* `IsDense`$(G, w, T, r) =$ `Yes` *then $|E_w^r| \geq T$.*

Given a vertex v and a distance R we sometimes want to bound $|E_v^R|$. Therefore, we adapt a lemma and a corollary of [7] from vertices to edges. The proofs are in the full version of this paper [10].

▶ **Lemma 7.** *Let x, y be positive integers, let $D \geq 1$ be a real number, and let $w \in V$. If $|E_w^x| < D^x$, and $|E_u^y| < D^y$ for every $u \in V$, then $|E_w^{x+y}| < D^{x+y}$.*

▶ **Corollary 8.** *Let x be a positive integer and let $D \geq 1$ be a real number. If $|E_w^x| < D^x$ for every $w \in V$, then $|E_w^{ix}| < D^{ix}$, for every $w \in V$ and $i \geq 1$.*

We also adapt procedure `SparseOrCycle`(G, D, x, y) of [7] to our needs. `SparseOrCycle` (see Algorithm 1) gets a graph G, a parameter $D \geq 1$, and two integers $x, y > 0$, and iterates over vertices using a for-each loop. Let w be the vertex currently considered and $G(w)$ the current graph. If `IsDense`$(G(w), w, D^x, x) =$ `Yes` then `BallOrCycle`$(G(w), w, x - 1 + y)$ is called. If `BallOrCycle` returns a cycle C then C is returned by `SparseOrCycle`. Otherwise, the vertex set V_w^{x-1} is removed from $G(w)$ along with the edge set E_w^x. After the loop ends, if no cycle was found, we return `null`. Let $W \subseteq V$ be the set of vertices for which `BallOrCycle` was called and no cycle was found, and $\hat{G} = (\hat{V}, \hat{E})$ the graph after `SparseOrCycle` ends. The following lemma is similar to the corresponding lemmas from [7]. For completeness, the proof is provided in the full version of this paper [10].

▶ **Lemma 9.** `SparseOrCycle`(G, D, x, y) *satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2(x - 1 + y)$*
- **(ii)** *If a cycle is not returned then $|E_u^x| < D^x$, for every $u \in \hat{G}$*
- **(iii)** *If $u \in G \setminus \hat{G}$ then u is not part of a $C_{\leq 2y}$ in G*
- **(iv)** `SparseOrCycle`(G, D, x, y) *runs in $O(nD^x + \sum_{w \in W}(|V_w^{x-1+y}|))$ time.*

Similarly to `AllVtxBallOrCycle`, we show for `SparseOrCycle` that if G satisfies a certain sparsity property, the running time can be bounded as follows. The proof is in the full version [10].

▶ **Corollary 10.** *If* $|E_u^{x-1+y}| < D^{x-1+y}$ *for every vertex* $u \in V$ *then* `SparseOrCycle`(G, D, x, y) *runs in* $O(nD^x + mD^{y-1})$ *time.*

Finally, we include a standard lemma about sampling a hitting set (see, e.g., [1], [8], [11]).

▶ **Lemma 11.** *It is possible to obtain in* $O(m)$ *time, using sampling, a set of edges* S *of size* $\tilde{\Theta}(\frac{m}{s})$*, that hits, whp, the* s *closest edges of every* $v \in V$*.*

We remark that some of our algorithms get a graph G that is being updated during their run. Within their scope, G denotes the current graph that includes all updates done so far.

4 A new cycle searching technique

In this section we introduce a new cycle searching technique implemented in algorithm `NbrBallOrCycle`. This technique exploits the property that a vertex $s \in V$ is not on a $C_{\leq 2k}$, to check efficiently whether any neighbor of s lies on a $C_{\leq 2k}$.

Consider a vertex $s \in V$. It is straightforward to check whether s is on a $C_{\leq 2k}$, for every integer k, using `BallOrCycle`(G, s, k) in $O(n)$ time. If `BallOrCycle`(G, s, k) does not return a $C_{\leq 2k}$ then for every $x, y \in N(s)$ it holds that $V_x^{k-1}(G') \cap V_y^{k-1}(G') = \emptyset$, where $G' = G \setminus \{s\}$, as otherwise there was a $C_{\leq 2k}$ passing through s and `BallOrCycle`(G, s, k) would have returned a $C_{\leq 2k}$ (see Figure 2). We show that it is possible to exploit this property to check for every $v \in N(s)$ whether v is on a $C_{\leq 2k}$, using `BallOrCycle`(G', v, k), in $O(n+m)$ time instead of $O(deg(s) \cdot n)$. More specifically, we present algorithm `NbrBallOrCycle`(G, s, k) (see Algorithm 2) that gets a graph G, a vertex s, and an integer $k \geq 2$. We first initialize $\hat{U}$ to $\emptyset$. Then, we run `BallOrCycle`(G, s, k). If a cycle C is found by `BallOrCycle`(G, s, k) then C is returned by `NbrBallOrCycle`. Otherwise, we add the vertex s to $\hat{U}$, keep the neighbors of s in N_s, and then remove s from G. Recall that G' equals $G \setminus \{s\}$. Next, for every $v \in N_s$ we run `BallOrCycle`(G', v, k), as long as a cycle is not found. If a cycle C is returned by `BallOrCycle`(G', v, k) then C is returned by `NbrBallOrCycle`.[7] Otherwise, v is added to $\hat{U}$. After the loop ends, the vertex s and its adjacent edges are added back to the graph, and the set $\hat{U}$ is returned by `NbrBallOrCycle`. We prove the following lemma in the full version of this paper [10].

▶ **Lemma 12.** *If algorithm* `NbrBallOrCycle`(G, s, k) *finds a cycle* C *then* $wt(C) \leq 2k$*. Otherwise, no vertex in* V_s^1 *is part of a* $C_{\leq 2k}$ *in* G*, and the set* $\hat{U} = V_s^1$ *is returned.*

To bound the running time of `NbrBallOrCycle`, we show how to use the fact that no $C_{\leq 2k}$ was found by `BallOrCycle`(G, s, k), to efficiently run `BallOrCycle`(G', v, k) for every $v \in N_s$.

▶ **Lemma 13.** *Let* $s \in V$*. If the ball graph* $B(G, s, k)$ *is a tree then the total cost of running* `BallOrCycle`(G', v, k) *for every* $v \in N_s$ *is* $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w))$*.*

[7] For our needs it suffices to stop and return a cycle passing through an s neighbor once we find one, though `BallOrCycle` can be run from all the neighbors of s in the same running time bound of $O(n+m)$.

Algorithm 2 `NbrBallOrCycle`(G, s, k).

```
1  Û ← ∅;
2  (V_s^k, C) ← BallOrCycle(s, k);
3  if C ≠ null then return (∅, C);
4  Û ← Û ∪ {s}, N_s ← N(s), G ← G \ {s};
5  foreach v ∈ N_s do
6      (V_v^k, C) ← BallOrCycle(v, k);
7      if C ≠ null then return (∅, C);
8      Û ← Û ∪ {v} ;
9  add s and the edge set {(s, v) | v ∈ N_s} back to G;
10 return (Û, null);
```

Proof. By the definitions of V_s^k, L_s^k and G', we know that $V_s^k(G) = \cup_{v \in N_s} V_v^{k-1}(G') \cup \{s\}$ and $L_s^k(G) = \cup_{v \in N_s} L_v^{k-1}(G')$. Since $B(G, s, k)$ contains no cycles it follows that $V_x^{k-1}(G') \cap V_y^{k-1}(G') = \emptyset$ and $L_x^{k-1}(G') \cap L_y^{k-1}(G') = \emptyset$, for any two distinct vertices $x, y \in N_s$. Therefore, **(i)** $|V_s^k(G)| = \sum_{v \in N_s} |V_v^{k-1}(G')| + 1$, and **(ii)** $L_s^k(G) = \cup_{v \in N_s} L_v^{k-1}(G')$. From Lemma 3 it follows that the total cost of the calls to `BallOrCycle`(G', v, k) for every $v \in N_s$ is $O(\sum_{v \in N_s} |V_v^k(G')|)$. It holds that $|V_v^k(G')| \leq |V_v^{k-1}(G')| + \sum_{w \in L_v^{k-1}(G')} deg(w)$, for every $v \in N_s$. Thus, we get that the total cost is $O(\sum_{v \in N_s} |V_v^k(G')|) = O(\sum_{v \in N_s} (|V_v^{k-1}(G')| + \sum_{w \in L_v^{k-1}(G')} deg(w)))$. This equals to $O(\sum_{v \in N_s} |V_v^{k-1}(G')| + \sum_{v \in N_s} \sum_{w \in L_v^{k-1}(G')} deg(w)))$, and it follows from **(i)** and **(ii)** that this is at most $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w))$. ◀

We use Lemma 13 to bound the running time of `NbrBallOrCycle`.

▶ **Lemma 14.** *Algorithm `NbrBallOrCycle` runs in $O(n + m)$ time.*

Proof. Running `BallOrCycle`(G, s, k) costs $O(n)$. If a cycle is found, it is returned and the running time is $O(n) = O(n+m)$. If no cycle is found, then removing (and later adding back) s and its edges costs $O(deg(s))$. By Lemma 13, the cost of running `BallOrCycle`(G', v, k) for every $v \in N_s$ is $O(|V_s^k(G)| + \sum_{w \in L_s^k(G)} deg(w)) = O(n + m)$. Adding s and N_s to $\hat{U}$ takes $O(V_s^1) = O(n)$ time. Thus, the total running time of `NbrBallOrCycle` is $O(n + m)$. ◀

5 A $2k$-hybrid algorithm and a $(+1)$-approximation of the girth

In this section we first show how to use algorithm `NbrBallOrCycle` from the previous section to obtain a $2k$-hybrid algorithm that in $O(m^{1+\frac{k-1}{k+1}})$ time, either returns a $C_{\leq 2k}$ or determines that $g > 2k$. Then, we use the $2k$-hybrid algorithm to compute a $(+1)$-approximation of g.

5.1 A $2k$-hybrid algorithm

We first present algorithm `2-SparseOrCycle`(G, k) that gets a graph G and an integer $k \geq 2$. Let G_0 (G_1) be G before (after) running `2-SparseOrCycle`. `2-SparseOrCycle`(G, k) either finds a $C_{\leq 2k}$ or removes vertices that are not on a $C_{\leq 2k}$, such that for every $u \in G_1$, the ball graph $B(G_1, u, 2)$ is relatively sparse, that is, $|E_u^2(G_1)| < m^{\frac{2}{k+1}}$. `2-SparseOrCycle` (pseudo-code in [10]) iterates over vertices using a for-each loop. Let s be the vertex currently considered. If $\sum_{v \in N(s)} deg(v) \geq m^{\frac{2}{k+1}}$ then `NbrBallOrCycle`(G, s, k) is called. If `NbrBallOrCycle` returns a cycle C then `2-SparseOrCycle` returns C. If `NbrBallOrCycle` returns a vertex set $\hat{U}$ then $\hat{U}$ is removed from G. After the loop ends, if no cycle was found, we return `null`.

▶ Remark. Notice that `SparseOrCycle`$(G, m^{\frac{2}{k+1}}, 2, k)$ either finds a $C_{\leq 2k+2}$ or removes vertices that are not on a $C_{\leq 2k}$, such that for every $u \in G_1$ it holds that $|E^2_u(G_1)| < m^{\frac{2}{k+1}}$. Using `NbrBallOrCycle` instead of `BallOrCycle` in 2-`SparseOrCycle` enables us in the case that a cycle is found to bound the cycle length with $2k$ rather than $2k+2$, while still maintaining the property that $|E^2_u(G_1)| < m^{\frac{2}{k+1}}$, for every $u \in G_1$, in the case that no cycle is found.

We prove the following lemma in the full version of this paper [10].

▶ **Lemma 15.** *2-`SparseOrCycle`(G, k) satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2k$*
- **(ii)** *If a cycle is not returned then $|E^2_u| < m^{\frac{2}{k+1}}$, for every $u \in G_1$*
- **(iii)** *If $u \in G_0 \setminus G_1$ then u is not part of a $C_{\leq 2k}$ in G_0*
- **(iv)** *2-`SparseOrCycle`(G, k) runs in $O(m^{1+\frac{k-1}{k+1}})$ time.*

Next, we use 2-`SparseOrCycle` to design a $2k$-hybrid algorithm called $2k$-`Hybrid`. Notice first that if $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$ for every $u \in V$, then it is straightforward to obtain an $O(m^{1+\frac{k-1}{k+1}})$-time $2k$-hybrid algorithm, by running `AllVtxBallOrCycle`(G, k). Thus, in $2k$-`Hybrid` we ensure that if we call `AllVtxBallOrCycle` then it holds for every $u \in V$ that $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$. To do so, we run 2-`SparseOrCycle` and possibly `SparseOrCycle`. If no cycle was returned then we have $|E^{k-1}_u| < m^{\frac{k-1}{k+1}}$ for every $u \in V$, and we can safely run `AllVtxBallOrCycle`.

$2k$-`Hybrid` (pseudo-code in [10]) gets a graph G and an integer $k \geq 2$. $2k$-`Hybrid` is composed of three stages. In the first stage we call 2-`SparseOrCycle`(G, k). If 2-`SparseOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise we proceed to the second stage. In the second stage, if $(k-1) \bmod 2 \neq 0$, we call `SparseOrCycle`$(G, m^{\frac{1}{k+1}}, 1, k)$. If `SparseOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise we proceed to the last stage. In the last stage, we call `AllVtxBallOrCycle`(G, k). If `AllVtxBallOrCycle` returns a cycle C then $2k$-`Hybrid` stops and returns C, otherwise $2k$-`Hybrid` returns `null`. We prove the next lemma in the full version of this paper [10].

▶ **Lemma 16.** *$2k$-`Hybrid`(G, k) either returns a $C_{\leq 2k}$ or determines that $g > 2k$, in $O(m^{1+\frac{k-1}{k+1}})$ time.*

5.2 A $(+1)$-approximation of the girth

Next, we describe algorithm `AdtvGirthApprox`, which uses $2k$-`Hybrid` and the framework described in Section 1, to obtain a $(+1)$-approximation of g, when $g \leq \log n$. `AdtvGirthApprox` (pseudo-code in [10]) gets a graph G. In `AdtvGirthApprox`, we set k to 2 and start a while loop. In each iteration, we create a copy G' of G and call $2k$-`Hybrid`(G', k). If $2k$-`Hybrid` finds a cycle C then `AdtvGirthApprox` stops and returns C, otherwise we increment k by 1 and continue to the next iteration. We prove the following theorem in the full version of this paper [10].

▶ **Theorem 17.** *Algorithm `AdtvGirthApprox`(G) returns either a C_g or a C_{g+1}, and runs in $\tilde{O}(m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$ and $2 \leq \ell < \log n$.*[8]

[8] We note that when $\ell \geq \log n$, the $O(n^2)$-time, $(+1)$-approximation of Itai and Rodeh [6] can be used.

▶ Remark. Dahlgaard, Knudsen and Stöckel [3] showed that a C_{2k}, if exists, can be found in $O(k^{O(k)} \cdot m^{\frac{2k}{k+1}})$ time. It is possible to use their C_{2k} detection algorithm to obtain a $(+1)$-approximation for the girth in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$, as mentioned in Section 2. This $(+1)$-approximation is obtained as follows. We run the detection algorithm with increasing values of k, starting with $k = 2$. If a C_{2k} is detected we stop and return the detected cycle. In such a case $2k$ is a $(+1)$-approximation for the girth. If there is no C_{2k} then it follows from the analysis of [3] that a C_{2k-1}, if exists, can be detected within the same running time using a slight modification of their algorithm. If a C_{2k-1} is detected we stop and return the detected cycle. In such a case $2k-1$ is the girth. If neither a C_{2k} nor a C_{2k-1} is found we increment k by 1. Since a cycle must be found when $k = \ell$, we obtain a $(+1)$-approximation for the girth in $\tilde{O}(\ell^{O(\ell)} \cdot m^{1+\frac{\ell-1}{\ell+1}})$ time, where $g = 2\ell$ or $g = 2\ell - 1$.

A disadvantage of the algorithm described above is the $\ell^{O(\ell)}$ factor which in our new algorithm does not exist. To better appreciate our contribution we compare our $2k$-`Hybrid` algorithm (for $C_{\leq 2k}$ detection) and the algorithm of [3] (for C_{2k} detection), that are used for obtaining the approximation.

Both algorithms first sparsify the graph (or find a cycle). The algorithm of [3] first handles high degree vertices, by running an $O(m)$-time C_{2k} detection algorithm from each high degree vertex (vertex of degree $> m^{2/(k+1)}$). If a C_{2k} is found then a C_{2k} is returned, otherwise the high degree vertices (and their adjacent edges) are removed. This is possible within the running time since there cannot be too many high degree vertices. In our algorithm, we handle first vertices with at least $m^{2/(k+1)}$ edges up to distance 2 rather than 1. This is possible using our `NbrBallOrCycle` algorithm, since we can search for a $C_{\leq 2k}$ from a vertex and also its neighbors in $O(m)$ time. If a C_{2k} is found then a C_{2k} is returned, otherwise the vertex and its neighbors are removed (therefore also the edges up to distance 2 from the vertex). Then, if required, we handle also high degree vertices and remove them.

After the sparsification stage, in [3] they prove that when the maximum degree is bounded, if there is no C_{2k} then the number of *capped k-walks* (walks of length k that visit only nodes according to a fixed ordering) in the graph is bounded by $O(k^{O(k)} m^{1+(k-1)/(k+1)})$, where the $k^{O(k)}$ factor follows from their analysis. They use this property to build a series of subgraphs in which the total number of edges is bounded, and therefore it is possible to search for a C_{2k} in these graphs efficiently. In our algorithm, after the sparsification stage it holds that the total number of edges up to distance k from all the vertices is bounded by $O(m^{1+(k-1)/(k+1)})$ (avoiding the $k^{O(k)}$ factor), and we can search efficiently for a $C_{\leq 2k}$.

6 A general hybrid algorithm

Algorithm $2k$-`Hybrid`(G, k), presented in the previous section, either returns a $C_{\leq 2k}$ or determines that $g > 2k$, in $O(m^{1+\frac{k-1}{k+1}})$ time. In this section we introduce an additional parameter $2 \leq \alpha \leq k$ and present a $(2k, 2\alpha)$-hybrid algorithm that either returns a $C_{\leq 2k}$ or determines that $g > 2\alpha$, in $O((\frac{k+1}{\alpha-1} + \alpha) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time. In Section 7 we use the $(2k, 2\alpha)$-hybrid algorithm to present two tradeoffs for girth approximation.

To obtain the $(2k, 2\alpha)$-hybrid algorithm we first extend algorithm `NbrBallOrCycle`. Then, we use the extended `NbrBallOrCycle` together with additional tools that we develop to either return a $C_{\leq 2k}$ or sparsify dense regions of the graph, so that we can check whether $g > 2\alpha$ (or return a $C_{\leq 2k}$) in $O(m^{1+\frac{\alpha-1}{k+1}})$ time, by running `AllVtxBallOrCycle`(G, α).

6.1 Extending NbrBallOrCycle

In algorithm NbrBallOrCycle we mark vertices that can be removed from the graph, by using the property that if BallOrCycle(v,k) did not return a $C_{\leq 2k}$ then v is not on a $C_{\leq 2k}$. In [7], they introduce an additional parameter α and used the following extended version of this property: If BallOrCycle(v,k) did not return a $C_{\leq 2k}$ then no vertex of $V_v^{k-\alpha}$ is on a $C_{\leq 2\alpha}$. We use the same approach and modify NbrBallOrCycle to get an additional integer parameter α such that $2 \leq \alpha \leq k$. After each call to BallOrCycle(G', v, k), where $v \in N_s$, if no cycle was found we add $V_v^{k-\alpha}$, instead of v, to $\hat{U}$. The modified pseudo-code appears in the full version of this paper [10]. We rephrase Lemma 12 to suit this modification. The proof is also given in [10].

▶ **Lemma 18.** *Let $2 \leq \alpha \leq k$. If NbrBallOrCycle(G, s, k, α) finds a cycle C then $wt(C) \leq 2k$. Otherwise, no vertex in $V_s^{k-\alpha+1}$ is part of a $C_{\leq 2\alpha}$ in G, and the set $\hat{U} = V_s^{k-\alpha+1}$ is returned.*

For the running time, we note that the sets V_v^k are computed during the execution of BallOrCycle(G', v, k), for every $(s, v) \in N_s$ for which no cycle was found. Their total size is also $O(n+m)$, and we can obtain from them the sets $V_v^{k-\alpha}$ and add these sets to $\hat{U}$ in $O(n+m)$ time. Therefore, the modified NbrBallOrCycle also runs in $O(n+m)$ time.

6.2 A $(\max\{2k, g\}, \tilde{g})$-hybrid algorithm

In this section we present a $(\max\{2k, g\}, 2\alpha)$-hybrid algorithm called ShortCycle, where $\alpha = \lceil \frac{\tilde{g}}{2} \rceil$. ShortCycle (pseudo-code in [10]) gets a graph G and two integers $\alpha, k \geq 2$. If $m \geq 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then we run algorithm ShortCycleDense(G, k) (pseudo-code in [10]), which is based on algorithm DegenerateOrCycle of [7].

If $m < 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then the main challenge is when $\alpha \leq k < \frac{n}{2}$. In this case we run algorithm ShortCycleSparse(G, k, α) (described later). The cases that $k \geq \frac{n}{2}$ or $k < \alpha$ are relatively simple and are treated by algorithm SpecialCases(G, k, α) (see the full version of this paper [10]). We summarize the properties of ShortCycle(G, k, α) in the next theorem.

▶ **Theorem 19.** *Let $\alpha, k \geq 2$ be integers. ShortCycle(G, k, α) runs whp in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time and either returns a $C_{\leq \max\{2k,g\}}$, or determines that $g > 2\alpha$.*

The next corollary follows from Theorem 19, when $\alpha \leq k$.

▶ **Corollary 20.** *Let $2 \leq \alpha \leq k$. Algorithm ShortCycle(G, k, α) runs whp in $\tilde{O}((\frac{k+1}{\alpha-1} + \alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$ time and either returns a $C_{\leq 2k}$, or determines that $g > 2\alpha$.*

In the rest of this section, we present the proof of Theorem 19. As follows from [7], if $m \geq 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$ then ShortCycleDense(G, k) returns in $O(\min\{m, n^{1+1/k}\})$ time a $C_{\leq 2k}$. We now consider the case in which $m < 1 + \lceil n \cdot (1 + n^{1/k}) \rceil$. We prove in the full version of this paper [10] that SpecialCases(G, k, α) satisfies the claim of Theorem 19, when $k \geq \frac{n}{2}$ or $k < \alpha$.

Our main technical contribution is algorithm ShortCycleSparse that handles the case of $\alpha \leq k < \frac{n}{2}$. Notice that if $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$ for every $u \in V$, then AllVtxBallOrCycle(G, α) is a $(2k, 2\alpha)$-hybrid algorithm that either finds a $C_{\leq 2\alpha}$ (which is also a $C_{\leq 2k}$ as $\alpha \leq k$) or determines that $g > 2\alpha$, in $O(m^{1+\frac{\alpha-1}{k+1}})$ time. Thus, in ShortCycleSparse we ensure that if we call AllVtxBallOrCycle, the property that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$, for every $u \in V$ (whp), holds. To do so, we run BfsSample and possibly HandleRemainder. If no cycle was returned, the property holds, and we can safely run AllVtxBallOrCycle.

`ShortCycleSparse` (pseudo-code in [10]) gets a graph G and two integers $\alpha, k \geq 2$ such that $\alpha \leq k < \frac{n}{2}$, and is composed of three stages. In the first stage we call `BfsSample`(G, k, α) (described later). If `BfsSample` returns a cycle C then `ShortCycleSparse` stops and returns C, otherwise we proceed to the second stage. In the second stage, if $(k+1) \bmod (\alpha - 1) \neq 0$, we call `HandleRemainder`$(G, k, \alpha)$ (also described later). If `HandleRemainder` returns a cycle C then `ShortCycleSparse` stops and returns C, otherwise we proceed to the last stage. In the last stage, we call `AllVtxBallOrCycle`(G, α). If `AllVtxBallOrCycle` returns a cycle C then $2k$`-Hybrid` stops and returns C, otherwise `ShortCycleSparse` returns `null`.

Next, we give a high level description of `BfsSample`. The goal of `BfsSample` is to either sparsify the graph without removing any $C_{\leq 2\alpha}$, or to report a $C_{\leq 2k}$. For simplicity assume that $(k+1) \bmod (\alpha - 1) = 0$. In such a case, if `BfsSample` does not report a $C_{\leq 2k}$, then the graph after `BfsSample` ends contains every $C_{\leq 2\alpha}$ that was in the original graph, and satisfies, whp, the following sparsity property: For every $u \in V$ it holds that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$.

This implies that in `BfsSample` we need to find every $u \in V$ which is in a dense region with $|E_u^{\alpha-1}| \geq m^{\frac{\alpha-1}{k+1}}$, and to check if u is in a $C_{\leq 2\alpha}$, so that if not we can remove u. Finding every such u is possible within the time limit by running `IsDense`$(G, u, m^{\frac{\alpha-1}{k+1}}, \alpha - 1)$ for every $u \in V$. The problem is that checking whether u is on a $C_{\leq 2\alpha}$ for every such u is too costly since there might be n such vertices, and this check costs $O(n)$ using `BallOrCycle`(G, u, α).

One way to overcome this problem is to sample an edge set S of size $\tilde{\Theta}(m^{1-\frac{\alpha-1}{k+1}})$ that hits the $m^{\frac{\alpha-1}{k+1}}$ closest edges of each vertex, and then use S to detect the vertices in the dense regions that are not on a $C_{\leq 2\alpha}$. In `BfsSample` we use a *detection process* in which we call `BallOrCycle` or `NbrBallOrCycle` from the endpoints of S's edges, and then, if no cycle was found, we use the information obtained from this call to identify vertices that are not on a $C_{\leq 2\alpha}$. The detection process either detects vertices that are not on a $C_{\leq 2\alpha}$ and can be removed, or reports a $C_{\leq 2k}$. However, it is not clear how to implement this detection process efficiently, since just running `BallOrCycle` from the endpoints of S's edges takes $O(nm^{1-\frac{\alpha-1}{k+1}})$ time which might be too much. Our solution is an *iterative* sampling procedure that starts with a smaller hitting set of edges, of size $\tilde{\Theta}(m^{\frac{\alpha-1}{k+1}})$. For such a hitting set we can run our detection process. If a $C_{\leq 2k}$ was not reported, then we remove the appropriate vertices and sparsify the graph without removing any $C_{\leq 2\alpha}$. When the graph is sparser, the running time of our detection process becomes faster. Thus, in the following iteration we can sample a larger hitting set for which we run this process, and either return a $C_{\leq 2k}$ or sparsify the graph further for the next iteration. We continue the iterative sampling procedure until we get to the required sparsity property in which $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$ for every $u \in V$ (whp).

We remark that in the first iteration of `BfsSample`, the detection process calls `NbrBallOrCycle`, while in the rest of the iterations `BallOrCycle` is called. The use of `NbrBallOrCycle` allows us, in the case that no $C_{\leq 2k}$ is reported, to bound $|E_v^d|$ with $m^{\frac{d}{k+1}}$ for every $v \in V$, rather than $|E_v^{d-1}|$. This is used to achieve the required sparsity property. Since `NbrBallOrCycle` runs in $O(m)$ time we can only use it in the first iteration when the sampled set is small enough. In the rest of the iterations we use `BallOrCycle` instead.

We now formally describe `BfsSample`. `BfsSample` (see Algorithm 3) gets a graph G and two integers $\alpha, k \geq 2$ such that $\alpha \leq k < \frac{n}{2}$. We first set y to $\lceil \frac{k+1}{\alpha-1} \rceil - 1$. Then, we start the main for loop that has at most y iterations. In the ith iteration, we initialize $\hat{V}$ to $\emptyset$ and sample a set $S_i \subseteq E$ of size $\tilde{\Theta}(m^{\frac{i\cdot(\alpha-1)}{k+1}})$. Next, we scan the endpoints in $V(S_i)$ using an inner for-each loop.

If $i = 1$, we call `NbrBallOrCycle`(G, s, k, α) from every endpoint $s \in V(S_1)$. `NbrBallOrCycle` returns either a cycle or a set of vertices $\hat{U}$. If `NbrBallOrCycle` returns a cycle then the cycle is returned by `BfsSample`. Otherwise, we add $\hat{U}$ to $\hat{V}$.

Algorithm 3 `BfsSample`(G, k, α).

```
1  y ← ⌈(k+1)/(α−1)⌉ − 1;
2  for i ← 1 to y do
3      sample a set S_i of Θ̃(m^{i·(α−1)/(k+1)}) edges;
4      V̂ ← ∅;
5      foreach s ∈ V(S_i) do
6          if i = 1 then
7              (Û, C) ← NbrBallOrCycle(G, s, k, α);
8              if C ≠ null then return C;
9              V̂ ← V̂ ∪ Û;
10         else
11             k_i ← (k + 1) − (i − 1) · (α − 1);
12             (V_s^{k_i}, C) ← BallOrCycle(s, k_i);
13             if C ≠ null then return C;
14             V̂ ← V̂ ∪ V_s^{k_i−α};
15     G ← G \ V̂;
16 return null;
```

If $i > 1$ then we call `BallOrCycle`(s, k_i), where $k_i = (k+1) - (i-1)\cdot(\alpha-1)$, from every endpoint $s \in V(S_i)$. If a cycle is found by `BallOrCycle` then the cycle is returned by `BfsSample`. If `BallOrCycle` does not return a cycle then we add $V_{u_j}^{k_i-\alpha}$ to $\hat{V}$.

Right after the inner for-each loop ends, we remove $\hat{V}$ from G, and continue to the next iteration of the main for loop. If no cycle was found after y iterations, we return `null`. Let $\ell \leq y$ be the last iteration in which vertices were removed. Let $\hat{V}_i$ be the set of vertices that were removed during the ith iteration, where $\hat{V}_i = \emptyset$ for $i > \ell$. Let G_0 (G_1) be G before (after) running `BfsSample`. The relevant figures in the full version of this paper [10] illustrate the key steps of the first and the following iterations of `BfsSample`. We summarize the properties of `BfsSample` in the next lemma. The proof is also given in the full version [10].

▶ **Lemma 21.** *`BfsSample`(G, k, α) satisfies the following:*

- **(i)** *If a cycle C is returned then $wt(C) \leq 2k$*
- **(ii)** *If a cycle is not returned then $|E_u^{(k+1)-y\cdot(\alpha-1)}| < m^{1-\frac{y\cdot(\alpha-1)}{k+1}}$, for every $u \in G_1$, whp*
- **(iii)** *If $u \in G_0 \setminus G_1$ then u is not part of a $C_{\leq 2\alpha}$ in G_0*
- **(iv)** *`BfsSample`(G, k, α) runs in $\tilde{O}(\lfloor \frac{k+1}{\alpha-1} \rfloor \cdot m^{1+\frac{\alpha-1}{k+1}})$ time, whp.*

Recall that our goal is to obtain the sparsity property that $|E_u^{\alpha-1}| < m^{\frac{\alpha-1}{k+1}}$, for every $u \in V$, so that we can run `AllVtxBallOrCycle`(G, α). However, after running `BfsSample` the required sparsity property is guaranteed to hold (whp) only if $(k+1) \bmod (\alpha-1) = 0$. In the case that $(k+1) \bmod (\alpha-1) \neq 0$ we need an additional step which is implemented in `HandleRemainder`, to guarantee that the required sparsity property holds.

Next, we formally describe `HandleRemainder`. `HandleRemainder` (see Algorithm 4) gets a graph G and two integers $\alpha, k \geq 2$ such that $k+1 = q(\alpha-1)+r$ where $q \geq 1$ and $r > 0$. We set D to $m^{\frac{1}{k+1}}$ and r to $(k+1) \bmod (\alpha-1)$. Then, a while loop runs as long as $(\alpha-1) \bmod r \neq 0$. Let r_i be the value of r when the ith iteration begins, so that r_1 is $(k+1) \bmod (\alpha-1)$. Let ℓ be the total number of iterations and $r_{\ell+1}$ the value of r after the ℓth iteration. During the

Algorithm 4 `HandleRemainder`(G, k, α).

$D \leftarrow m^{\frac{1}{k+1}}$;
$r \leftarrow (k+1) \bmod (\alpha - 1)$;
while $(\alpha - 1) \bmod r \neq 0$ **do**
  $r \leftarrow (\lceil \frac{\alpha-1}{r} \rceil \cdot r) - (\alpha - 1)$;
  $C \leftarrow$ `SparseOrCycle`(G, D, r, α);
  if $C \neq$ *null* **then return** C;
return `null`;

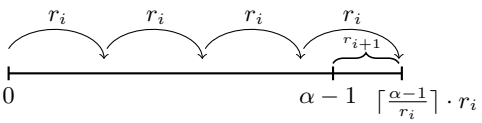


Figure 3 The relation between r_i, r_{i+1} and $\alpha - 1$.

ith iteration, we set r_{i+1} to $(\lceil \frac{\alpha-1}{r_i} \rceil \cdot r_i) - (\alpha - 1)$, where $\lceil \frac{\alpha-1}{r_i} \rceil \cdot r_i$ is the smallest multiple of r_i that is at least $\alpha - 1$ (see Figure 3). Then, we call `SparseOrCycle`(G, D, r_{i+1}, α). If `SparseOrCycle` returns a cycle C then `HandleRemainder` returns C. If `SparseOrCycle` does not return a cycle then it might be that some vertices were removed from G, and we continue to the next iteration. If the while loop ends without returning a cycle then we return `null`. Let G_0 (G_1) be G before (after) running `HandleRemainder`. Next, we prove two properties on the value of r during the run of `HandleRemainder` (see the full version of this paper [10] for the proof).

▷ Claim 22. Let $k+1 = q(\alpha-1)+r_1$ and assume $r_1 > 0$. **(i)** $0 < r_{\ell+1} < r_\ell < \cdots < r_1 < \alpha-1$. **(ii)** $(r_{i+1} + \alpha - 1) \bmod r_i = 0$, for every $1 \leq i \leq \ell$.

We also prove in the full version [10] the main lemma regarding `HandleRemainder`.

▶ **Lemma 23.** *Let $k+1 = q(\alpha-1)+r_1$ and assume that $r_1 > 0$, $q \geq 1$, and that $|E_u^{r_1}| < D^{r_1}$, for every $u \in V$. `HandleRemainder`(G, k, α) satisfies the following:*

(i) *If a cycle C is returned then $wt(C) \leq 2k$*
(ii) *If a cycle is not returned then $|E_u^{\alpha-1}| < D^{\alpha-1}$, for every vertex $u \in G_2$*
(iii) *If $u \in G_1 \setminus G_2$ then u is not on a $C_{\leq 2\alpha}$ in G_1*
(iv) *`HandleRemainder`(G, k, α) runs in $O(r_1 \cdot m^{1+\frac{\alpha-1}{k+1}})$ time.*

Now we are ready to prove the correctness and running time of `ShortCycleSparse`. The proof is in the full version [10].

▶ **Lemma 24.** *Let $2 \leq \alpha \leq k < \frac{n}{2}$ such that $k+1 = q(\alpha-1)+r$, where $q \geq 1$ and $0 \leq r < \alpha-1$ are integers. Algorithm `ShortCycleSparse`(G, k, α) runs whp in $\tilde{O}((q+r) \cdot m^{1+\frac{\alpha-1}{k+1}})$ time and either returns a $C_{\leq 2k}$, or determines that $g > 2\alpha$.*

Since `ShortCycleSparse` is run by `ShortCycle` when $m \leq O(n^{1+\frac{1}{k}})$ and when $\alpha \leq k$ and so $1 + \frac{\alpha-1}{k+1} \leq 2$, we have $m^{1+\frac{\alpha-1}{k+1}} \leq O(n^{1+\frac{\alpha}{k}})$. Thus, the running time of `ShortCycleSparse` is whp $\tilde{O}((q+r) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$, which is at most $\tilde{O}((\frac{k+1}{\alpha-1}+\alpha) \cdot \min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$. Together with the relatively simple special cases handled in the full version of this paper [10], this completes the proof of Theorem 19.

7 Approximation of the girth

In this section we present two new tradeoffs for girth approximation that follow from Corollary 20. In these tradeoffs we use `ShortCycle` with $2 \leq \alpha \leq k$, so by Corollary 20, `ShortCycle` is an $\tilde{O}((\frac{k+1}{\alpha-1}+\alpha)\cdot\min\{m^{1+\frac{\alpha-1}{k+1}}, n^{1+\frac{\alpha}{k}}\})$-time, $(2k, 2\alpha)$-hybrid algorithm.

7.1 Dense graphs

Kadria et al. [7] presented an $O((\alpha-c)\cdot n^{1+\frac{\alpha}{2\alpha-c}})$-time, $(4\alpha-2c, 2\alpha)$-hybrid algorithm, where $0 < c \leq \alpha$ are two integers (see footnote 3 in Section 2). This algorithm, combined with a binary search, was used by [7] to compute for every $\varepsilon \in (0,1]$ a cycle C such that $wt(C) \leq 4\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (2-\varepsilon)g+4$, in $\widetilde{O}(n^{1+1/(2-\varepsilon)})$ time, if $g \leq \log^2 n$. We use `ShortCycle` in a similar way and prove:

▶ **Theorem 25.** *Let $\ell \geq 2$ be an integer, $\varepsilon \in [0,1]$ and $g \leq \log^2 n$. It is possible to compute, whp, in $\widetilde{O}(\ell \cdot n^{1+1/(\ell-\varepsilon)})$ time, a cycle C such that $wt(C) \leq 2\ell\lceil\frac{g}{2}\rceil - 2\lfloor\varepsilon\lceil\frac{g}{2}\rceil\rfloor \leq (\ell-\varepsilon)g+\ell+2$.*[9]

The proof is in the full version of this paper [10]. Our algorithm can be viewed as a natural generalization of a couple of algorithms from [7]. By setting $\ell = 2$ in Theorem 25 we get the $(2-\varepsilon)g+4$ approximation of [7]. By setting $\varepsilon = 0$ we get an $\widetilde{O}(\ell \cdot n^{1+1/\ell})$ time algorithm that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$, which is similar to the $\widetilde{O}(n^{1+1/\ell})$ time algorithm of [7] that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$.

7.2 Sparse graphs

We use a similar approach to obtain a tradeoff for girth approximation in sparse graphs. We prove the following theorem in the full version of this paper [10].

▶ **Theorem 26.** *Let $\ell \geq 3$ be an integer, $\varepsilon \in [0,1)$ and $g \leq \log^2 n$. It is possible to compute, whp, in $\widetilde{O}(\ell \cdot m^{1+1/(\ell-\varepsilon)})$ time, a cycle C such that $wt(C) \leq 2\ell(\lceil\frac{g}{2}\rceil-1) - 2\lfloor\varepsilon(\lceil\frac{g}{2}\rceil-1)\rfloor - 2 \leq (\ell-\varepsilon)g - \ell + 2\varepsilon$.*

By setting $\varepsilon = 0$ we get an $\widetilde{O}(\ell \cdot m^{1+1/\ell})$-time algorithm that computes a $C_{\leq 2\ell(\lceil\frac{g}{2}\rceil-1)-2}$, as opposed to the $\widetilde{O}(\ell \cdot n^{1+1/\ell})$ time algorithm that computes a $C_{\leq 2\ell\lceil\frac{g}{2}\rceil}$.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 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. doi:10.1145/3357713.3384330.

3 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Finding even cycles faster via capped k-walks. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 112–120. ACM, 2017. doi:10.1145/3055399.3055459.

[9] We note that when $\ell > \log n$, we can run the $\tilde{O}(n^{1+1/\ell'})$-time algorithm of [7], which computes a $C_{\leq 2\ell'\lceil g/2\rceil}$, with $\ell' = \ell - 1$, to get the required approximation. Since $\ell > \log n$, its running time is $\tilde{O}(n)$. Thus, our running time becomes $\tilde{O}(n^{1+1/(\ell-\varepsilon)})$, where the ℓ factor is absorbed into the $\tilde{O}$ notation.

4 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. New subquadratic approximation algorithms for the girth. *arXiv preprint arXiv:1704.02178*, 2017. arXiv:1704.02178.

5 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997. doi:10.1006/INCO.1997.2620.

6 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.

7 Avi Kadria, Liam Roditty, Aaron Sidford, Virginia Vassilevska Williams, and Uri Zwick. Algorithmic trade-offs for girth approximation in undirected graphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1471–1492. SIAM, 2022. doi:10.1137/1.9781611977073.62.

8 Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Information Processing Letters*, 109(10):493–498, 2009. doi:10.1016/J.IPL.2009.01.008.

9 Liam Roditty and Roei Tov. Approximating the girth. *ACM Transactions on Algorithms (TALG)*, 9(2):1–13, 2013. doi:10.1145/2438645.2438647.

10 Liam Roditty and Plia Trabelsi. New algorithms for girth and cycle detection. *arXiv preprint arXiv:2507.02061*, 2025. doi:10.48550/arXiv.2507.02061.

11 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524, 2013. doi:10.1145/2488608.2488673.

12 Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 833–845. SIAM, 2012. doi:10.1137/1.9781611973099.67.

13 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995. doi:10.1006/JCSS.1995.1078.

14 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 605–614. IEEE, 1999. doi:10.1109/SFFCS.1999.814635.

15 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

16 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.