Tighter Bounds for Weighted and Unweighted Shortest Cycle Approximation

Avi Kadria
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

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

Virginia Vassilevska Williams
Department of Electrical Engineering and Computer Science and CSAIL, MIT, Cambridge, MA, USA

Abstract

We study the problem of approximating the length of a shortest cycle in a given graph, known as the girth of the graph. The state-of-the-art approximation algorithms for unweighted graphs by Kadria et al. [SODA'22] and Roditty and Trabelsi [arXiv'25] achieve the following trade-off: for every integer $k \geq 2$, there is an $\tilde{O}(n^{1+2/k})$ time algorithm that achieves a $(2k/3)$-approximation for the girth in unweighted n-node graphs. The first result of this paper is to achieve the same trade-off for m-edge, n-node graphs with non-negative real edge weights: a $2k/3$-approximation algorithm running in $\tilde{O}(m + n^{1+2/k})$ time. The dependence on m is unavoidable in weighted graphs. Our result improves on the work of Kadria et al. [SODA'23] and Ducoffe [ICALP'19 and SIDMA'21], who were only able to achieve such a trade-off for some values of k. We also prove new fine-grained lower bounds for girth approximation and related problems in unweighted graphs.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Shortest paths; Theory of computation → Complexity classes; Mathematics of computing → Graph algorithms

Keywords and phrases Fine-grained complexity, Graph algorithms, shortest cycle, girth approximations

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.82

Related Version *Full Version*: `https://arxiv.org/abs/2607.00938` [17]

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

1 Introduction

The length of a shortest cycle in a graph, known as the *girth* and denoted by g, is a key parameter often used to shed light on the structure of graph problems (e.g., [19, 21, 11]). The problem of computing a shortest cycle and girth in an undirected graph is a fundamental problem studied extensively for decades, both in unweighted graphs (e.g., [12, 3, 32, 20, 26, 16]), and weighted graphs (e.g., [20, 25, 23, 9, 16, 15]).

Computing the exact value of the girth is computationally expensive: in weighted graphs, it is known to be equivalent to the All-Pairs Shortest Paths (APSP) problem, and in unweighted graphs, it is known that any fast algorithm requires Boolean Matrix Multiplication (BMM) [30]. Because exact computation is deemed prohibitive, fast approximation algorithms have been extensively studied. A (multiplicative) c-approximation algorithm outputs a cycle of length $\hat{g} \leq c \cdot g$ whenever g is the girth of the underlying graph. The factor c, which is the largest ratio $\frac{\hat{g}}{g}$ that the algorithm achieves, is called the *stretch* of the algorithm.

In unweighted graphs, Itai and Rodeh [12] were the first to consider girth approximation, and presented an $O(n^2)$ time algorithm that returns $g \leq \hat{g} \leq 2\lceil g/2 \rceil$. Lingas and Lundell [20] presented an $\tilde{O}(n^{1.5})$ time algorithm[1] that returns $g \leq \hat{g} \leq 4\lceil g/2 \rceil$. Later, Kadria, Roditty, Sidford, Vassilevska Williams, and Zwick [16] generalized these results and obtained the following trade-off. For any **even** integer $k \geq 2$, there is an $O(n^{1+2/k})$ time algorithm that returns $g \leq \hat{g} \leq k \cdot \lceil g/2 \rceil$. Recently, Roditty and Trabelsi [24] obtained the same trade-off for odd values of k. The corresponding stretch is at most $\frac{k}{2}$ when g is even, and at most $\frac{k}{2} \cdot (1 + \frac{1}{g})$ when g is odd. If $g = 3$, that is, there is a triangle in the graph, then $\frac{k}{2} \cdot (1 + \frac{1}{g}) = 2k/3$, which is the stretch in the worst case. We can summarize the state of the art for *unweighted* graphs as:

For every integer $k \geq 2$, there is an $\tilde{O}(n^{1+\frac{2}{k}})$ time $2k/3$-approximation algorithm for the girth of unweighted graphs.

A central and extensively studied question in graph algorithms is whether the running time - approximation trade-off established for unweighted graphs can be matched in the case of weighted undirected graphs with non-negative real edge weights. Therefore, the following problem is natural.

▷ Problem 1. Is it possible, for every integer $k \geq 2$, to obtain an $\tilde{O}(m + n^{1+\frac{2}{k}})$ time algorithm with $\frac{2k}{3}$-stretch in weighted undirected graphs with non-negative real edge weights?

(We allow an additive m in the running time above since in weighted graphs one needs to read the input even for an approximation[2].)

Roditty and Tov [23] almost solved Problem 1 for $k = 2$ and presented an $\tilde{O}(\frac{1}{\varepsilon}n^2)$ time algorithm with $(4/3 + \varepsilon)$-stretch. Later, Ducoffe [9] almost solved Problem 1 for $k = 3$ and presented an $\tilde{O}(\frac{1}{\varepsilon}(m + n^{5/3}))$ time algorithm with $(2 + \varepsilon)$-stretch. [15] solved Problem 1 for every **even** integer $k \geq 2$, and presented an $\tilde{O}(m + n^{1+\frac{2}{k}})$ time algorithm with $\frac{2k}{3}$-stretch.

Therefore, to solve Problem 1 for every $k \geq 2$ and establish that the time/stretch trade-off in weighted graphs matches that of unweighted graphs, it remains to address Problem 1 for **odd** values of $k \geq 3$. In this paper, we completely solve Problem 1 and prove the following.

▶ **Theorem 1.** *Let $G = (V, E, \ell)$ be a weighted graph, where $\ell : E \to \mathbb{R}_{\geq 0}$, and let $k \geq 3$ be an **odd** integer. There exists an algorithm that runs in $\tilde{O}(m + n^{1+\frac{2}{k}})$ time and returns an estimation $\hat{g}$ such that $g \leq \hat{g} \leq \frac{2k}{3}g$.*

We note that our result improves the known trade-off for real weighted graphs for all odd values of $k \geq 3$. In the special case of $k = 3$, Ducoffe's algorithm [9] achieves a $2 = 2 \cdot (3/3)$-approximation only for graphs with polynomially bounded integer weights. For real weights, the stretch of Ducoffe's algorithm is $(2 + \varepsilon)$ for an arbitrarily small constant $\varepsilon > 0$, whereas ours is truly 2.

Among the tools we use to prove the theorem are: approximate distance oracles [28], Spira's single-source shortest paths algorithm [27], ideas from Kadria et al. [15], and a generalization of the ideas of [9].

After matching the trade-off of weighted and unweighted graphs, we consider **fine-grained lower bounds** for girth approximation and related problems in unweighted graphs.

---

[1] $\tilde{O}$ omits poly-logarithmic factors.

[2] Take a complete graph with infinite edge weights and reduce the weight of the edges in some triangle to 1; if an algorithm does not find any of the 1 weight edges, it can return only girth of $3 \cdot \infty$, and finding 3 needles in a stack of $O(n^2)$ size requires $\Omega(n^2)$ time.

Fine-grained Lower Bound for Girth Approximation. Triangle detection is one of the most important special cases of the shortest cycle problem. It plays a fundamental role in algorithm design and complexity theory and also has practical applications (e.g. [10, Chapter 3]). Any $(4/3-\varepsilon)$-approximation algorithm for the girth (for $\varepsilon > 0$) can distinguish between girth ≥ 4 and the existence of a triangle, and thus triangle detection can be viewed as achieving stretch $(4/3-\varepsilon)$ for girth approximation.

The first conditional lower bounds for triangle detection come from the aforementioned equivalence results of Vassilevska W. and Williams [30]. These imply that any $O(n^{3-\varepsilon})$ time algorithm (for $\varepsilon > 0$) for triangle detection would also imply an $O(n^{3-\varepsilon'})$ time algorithm (for $\varepsilon' > 0$) for BMM, and hence under the popular BMM Hypothesis, "combinatorial" techniques cannot achieve $O(n^{3-\varepsilon})$ time for triangle detection and hence for $(4/3-\varepsilon)$-girth approximation. Thus, practical fast algorithms for $(4/3-\varepsilon)$-girth approximation are considered out of reach.

For sparse graphs, [2] showed that triangle detection in an m edge graph with no 4 cycles requires $m^{1.1194-o(1)}$ time, assuming that triangle detection in $\sqrt{n}$-degree graphs requires $n^{2-o(1)}$ time. This also implies a conditional $m^{1.1194-o(1)}$ time lower bound for any $5/3-\varepsilon$ girth approximation algorithm for $\varepsilon > 0$. While the hypothesis used by [2] is somewhat non-standard, a weaker version of it was shown by [14] to be implied by the slightly more popular Strong 3Sum Hypothesis, giving evidence that the non-standard hypothesis could be true. ([14] also based the hardness of the classical problem of triangle detection on the Strong 3Sum hypothesis.[3])

The Strong 3Sum hypothesis studied by [14] states that 3Sum on n numbers from the universe $[\pm O(n^2)]$ requires $n^{2-o(1)}$ time (on a word-RAM).[4] We consider the following Stronger3Sum hypothesis: On a word-RAM with $O(\log n)$ bit words, 3Sum on n numbers from a (Sidon) set[5] $A \subseteq [\pm n^{2+o(1)}]$ with no non-trivial solution to $a+b=c+d$ requires $n^{2-o(1)}$ time.

This hypothesis seems much stronger than the Strong 3Sum hypothesis. However, it is still plausible. In fact, [14] explicitly ask (Open problem 5) whether their techniques for fine-grained reductions for 3Sum in Sidon Sets and Sidon Set verification can be improved so that the range of the integers does not increase by much and the problems are still hard for n integers in the range $[\pm n^{2+\delta}]$ for small δ. So far, all known techniques for removing additive structure in fine-grained reductions really blow up the size of the integers so that it is unclear whether one can prove the Stronger3Sum hypothesis with current techniques. The hypothesis could also be false due to the highly structured nature of the input, but proving that it is false would also seem to require interesting new techniques. We show:

▶ **Theorem 2.** *Under the Stronger3Sum Hypothesis, there is no $O(n^{1.5-\varepsilon})$ time algorithm for $C_{\leq 4}$ detection and hence for $(5/3-\varepsilon)$-girth approximation in graphs with maximum degree $n^{1/4}$.*

A brute-force algorithm can detect if a given n-node graph with maximum degree $n^{1/4}$ has a triangle or a C_4 in time $O(n^{1.5})$: for every vertex u, try all pairs of its neighbors v, v' and check whether (v, v') is an edge, thus checking if a triangle exists; if no triangle is found, the previous step actually lists all two-edge paths in the graph $v-u-v'$ and one can detect

[3] Later, Chan and Xu [5] improved the triangle detection lower bound to $m^{9/7-o(1)}$, under the so-called Strong Exact Triangle Hypothesis.
[4] Where $[\pm n] = \{i \in \mathbb{Z} \mid |i| \leq n\}$
[5] A set A of integers is a *Sidon set* if there are no distinct $a, b, c, d \in A$ such that $a+b=c+d$.

a 4-cycle just by sorting[6] these by their end-points and checking for a collision. Thus our theorem shows that under the Stronger3Sum Hypothesis, the brute-force algorithm is essentially optimal.

Previously, the best known lower bound for $(5/3-\varepsilon)$-girth approximation was by [2]. It is instructive to compare the time lower bounds in terms of the number of edges. That of [2] is $m^{1.1194-o(1)}$ (as mentioned earlier) and ours is $m^{6/5-o(1)} = m^{1.2-o(1)}$. Of course, they are under different hypotheses that may or may not be true.

Listing Cycles or All Edge Triangles. A relatively recent line of work concerns algorithms and conditional lower bounds for listing and enumerating cycles in graphs. Listing triangles has been studied extensively both from an algorithmic point of view (e.g. [4]) and from a fine-grained perspective [31, 18, 22]. Listing and enumerating cycles of larger constant length is an even more recent topic of study [29, 13, 2, 14, 1].

Most known conditional lower bounds for cycle listing and enumeration go through the All-Edge Sparse Triangle problem: given an m-edge graph, determine for every edge e whether e appears in a triangle. In 2010, Patrascu [22] showed that under the 3Sum hypothesis, All-Edge Sparse Triangle requires $m^{4/3-o(1)}$ time. Since then various refinements of this problem are studied from a lower bounds perspective, most notably for solving All-Edge Sparse Triangle in graphs with a small number of cycles of length at most k [2, 14, 1].

We study the recent techniques for conditional lower bounds for listing cycles and discover that they can be used to prove hardness for an easier problem as well.

Consider the following AllEdgeTriangleOrListing problem: for an integer $k \geq 4$, given a graph G and an integer t, either solve the All-Edge Sparse Triangle in G, or list t cycles of length at most k.

This problem is easier than both All-Edge Sparse Triangle and the problem of listing at most t cycles of length at most k, as an algorithm for it can choose which problem to solve depending on the input graph.

Listing t cycles of length at most k is an easier problem than listing t cycles of length exactly k, the problem of recent interest as described earlier. For instance, there is a very simple $O(n^2)$ time algorithm for finding a single cycle of length at most $2k$ for any integer k [7], whereas an $O(n^2)$ time algorithm for finding a cycle of length exactly $2k$ is much more involved [32]. However, the problem of listing all cycles of length at most k is at least as hard as that of listing all cycles of length k. So the two listing problems are tightly linked.

By simple modifications of the known techniques, we show:

▶ **Theorem 3.** *Under the 3SUM Hypothesis, there is no $O(m^{1+1/(k-1)-\varepsilon}+t)$ time algorithm for AllEdgeTriangleOrListing.*

As this is an intermediary problem that has not been studied before, it may not be clear how close this lower bound is to the true complexity of the problem. We show that, up to constants in front of k, the lower bound is in fact tight.

▶ **Theorem 4.** *There is an $O(m^{1+1/(k+1)}+t)$ time algorithm that given an m-edge graph G either returns t cycles of length at most $2k$ in G or solves the All-Edge Sparse Triangle problem in G.*

---

[6] Sorting here can be done in linear time because the n nodes can be uniquely labeled by the integers in $[n]$.

[7] Run BFS from every node up to k levels and stop when the first cycle is closed. If no cycle is closed, each BFS only sees a tree and runs in $O(n)$ time.

This should be compared with the fastest known algorithm for listing $2k$ cycles in terms of m. It is believed that the "right" running time for listing t $2k$-cycles should be $\tilde{O}(m^{2k/(k+1)}+t)$ [29] as the best known running time for $2k$-cycle detection is $\tilde{O}(m^{2k/(k+1)})$ [7]. This running time has been achieved for $k = 2$ [14, 1] and is a big open problem for $k \geq 3$ (though it has been almost achieved for $k = 3$ [29]). Our theorem above can be viewed as a step in the direction of understanding the complexity of listing cycles of length *at most* k (as opposed to exactly k).

2 Preliminaries

Let $G = (V, E, \ell)$ be a weighted undirected graph, where $\ell : E \to (0, \infty)$ is a real *length* function defined on its edges. Let $n = |V|$ and $m = |E|$. The graph is represented using an adjacency-list representation. We assume that the edges incident on a vertex u are sorted in a non-decreasing order of length. [8]

For all $u, v \in V$, let $P(u, v) = \{u = x_1, \ldots, x_t = v\}$ be a shortest path between u and v in G, let $\pi(u, v)$ be the last edge on a shortest path from u to v, and let $P_2(u, v)$ be a second-shortest path from u to v (i.e. $P_2(u, v)$ is a shortest path from u to v that differs from $P(u, v)$ by at least one edge). Let $\ell(P(u, v)) = \sum_{i=1}^{t-1} \ell(x_i, x_{i+1})$. Let the *distance* $\delta(u, v)$ between u and v be $\ell(P(u, v))$. A sequence of vertices $C = (x_1, x_2, \ldots, x_t)$ is a cycle if $x_1 = x_t$, and $(x_i, x_{i+1}) \in E$, for every $1 \leq i \leq t-1$. If x_1 is the only vertex that appears twice in C, then C is a *simple* cycle. Let C be a simple cycle. We denote $\ell(C) = \sum_{i=1}^{t-1} \ell(x_i, x_{i+1})$. The *girth* g of a graph $G = (V, E, \ell)$ is the length of a shortest simple cycle in G. That is, C is a shortest cycle if $\ell(C) = g$. Let $M(C)$ be an edge with the largest length in C. We remark that throughout the paper, we treat shortest paths $P(\cdot, \cdot)$ and cycles C interchangeably as sets of edges and sets of vertices. For the rest of the paper, we also assume that the graph is connected and contains a simple cycle, as otherwise the problem is trivial.

`Sample`(S, p) is a procedure that returns a subset S^* of S where each element of S is included in S^* independently with probability p. Given a set $S \subseteq V$, the induced subgraph $G[S]$ is defined as $(S, \{(u, v) \in E \mid u, v \in S\})$. If $u \in V$ and $A \subseteq V$, we let $\delta(u, A) = \min_{v \in A} \delta(u, v)$ denote the *distance from u to the set A*. (If $A = \emptyset$, then $\delta(u, A) = +\infty$.) Given a set S we define $B_S(u) = \{v \in V \mid \delta(u, v) < \delta(u, S)\}$.

▶ **Lemma 5** (Nearest set computation [6]). *Let $S =$ `Sample`(V, n^{-x}). Computing $G[B_S(u)]$ for every $u \in V$ takes $O((m + n^{1+2x}) \log n)$ time.*

We denote by `MinCycle`(u, G) an $O(m + n \log n)$ time procedure that runs a modification of Dijkstra's algorithm from u in G and computes both $P(u, v)$ and $P_2(u, v)$, for every $v \in V$.[9] `MinCycle` then computes $v = \arg\min_{v \in V}(\ell(P(u, v)) + \ell(P_2(u, v)))$, removes the common prefix and suffix of $P(u, v)$ and $P_2(u, v)$, and returns the resulting simple cycle. Notice that if u is on a shortest cycle, then $\ell($`MinCycle`$(u, G)) = g$.

Following [15], we define the distance from a vertex $u \in V$ to an edge $(v, w) \in E$ as follows: $\delta(u, (v, w)) = \min\{\delta(u, v), \delta(u, w)\} + \ell(v, w)$. [10] Let $u \in V$ and $r > 0$. We define the *ball graph* $G_r(u) = (V_r(u), E_r(u))$ of *radius* r around u as follows:

$$V_r(u) = \{v \in V \mid \delta(u, v) \leq r\} \text{ and } E_r(u) = \{e \in E \mid \delta(u, e) \leq r\} .$$

[8] A priori, the algorithms of [16] work in $o(m)$ if we assume the edges are sorted. However, [15] proved that even in this model, $\Omega(m)$ time is required to achieve approximation that is better than $4k$ stretch in $O(n^{1+2/k})$ time. Therefore to get the improved $8k/3$-stretch, even in this model our algorithm needs to have $\Omega(m)$ time.

[9] This can be implemented by maintaining two priority queues during Dijkstra's algorithm-one for the shortest distances and another for the second-shortest distances.

[10] Note that $\delta(u, (v, w)) = \delta(u, \{v, w\}) + \ell(v, w)$. Here $\{v, w\}$ is a set of two vertices.

We let $G_{<r}(u) = (V_{<r}(u), E_{<r}(u))$ denote the *open* ball graph of radius r around u. The definitions of $V_{<r}(u)$ and $E_{<r}(u)$ are identical to those of $V_r(u)$ and $E_r(u)$ with the weak inequalities $\delta(u,v) \leq r$ and $\delta(u,e) \leq r$ replaced by strict inequalities.

The following general and simple lemma from [15] is used in the correctness proof of our algorithm, and therefore, we provide its proof here for completeness.

▶ **Lemma 6** ([15]). *Let $G = (V, E, \ell)$ be a weighted undirected graph, C a cycle in G, $u \in V$, and $r > 0$. If $V_r(u) \cap C \neq \emptyset$, then $C \subseteq G_{r+\frac{1}{2}(\ell(C)+M(C))}(u)$.*

Proof. Let $v \in V_r(u) \cap C$. By definition $\delta(u,v) \leq r$. Let $(x,y) \in C$. Assume, without loss of generality, that $\delta(v,x) \leq \delta(v,y)$. As $\delta(v,x) + \ell(x,y) + \delta(v,y) \leq \ell(C)$, we get that $\delta(v,x) \leq \frac{1}{2}(\ell(C) - \ell(x,y))$. Thus

$$\begin{aligned}\delta(u,(x,y)) &\leq \delta(u,v) + \delta(v,x) + \ell(x,y)\\ &\leq r + \frac{1}{2}(\ell(C) - \ell(x,y)) + \ell(x,y)\\ &= r + \frac{1}{2}(\ell(C) + \ell(x,y)) \leq r + \frac{1}{2}(\ell(C) + M(C)),\end{aligned}$$

where the last inequality follows from the fact that $(x,y) \in C$, and therefore $\ell(x,y) \leq M(C)$. Thus, $(x,y) \in E_{r+\frac{\ell(C)+M(C)}{2}}(u)$, for every $(x,y) \in C$ and therefore $C \subseteq G_{r+\frac{\ell(C)+M(C)}{2}}(u)$, as required. ◀

Let $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k = \emptyset$ be a hierarchy of vertex sets, where $k \geq 1$. Following [28] we let $B_i(u) = \{w \in A_i \mid \delta(u,w) < \delta(A_{i+1}, u)\}$.

Following [15] we define the *cluster graph* of $u \in A_i \setminus A_{i+1}$ in G to be the graph $CL(u) = (CL_V(u), CL_E(u))$, where

$$\begin{aligned}CL_V(u) &= \{v \in V \mid \delta(u,v) < \delta(A_{i+1}, v)\}\ ,\\ CL_E(u) &= \{(v,w) \in E \mid \delta(u,v) + \ell(v,w) < \delta(A_{i+1}, w)\}\ .\end{aligned}$$

The cluster graph can be viewed as an extension of the clusters of Thorup and Zwick [28] to a subgraph rather than a set of vertices. Notice that in the definition of $CL_E(u)$, we have that v, w have asymmetrical roles, and it might be that v, w would satisfy the condition, whereas w, v would not.

For any $u \in V$ and $0 \leq i < k$, we let $p_i(u) = \arg\min_{v \in A_i} \delta(u,v)$, i.e., $p_i(u)$ is a vertex of A_i closest to u (ties are broken lexicographically). [15] proved the following property on cluster graphs.

▶ **Lemma 7** ([15]). *Let $u \in A_i \setminus A_{i+1}$. If $v \in CL_V(u)$ then $P(u,v) \subseteq CL(u)$.*

Clusters have especially nice properties when the hierarchy $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k = \emptyset$ is obtained using random sampling. Lemma 8 gives one such property and is proven in [28] using a simple probabilistic argument.

▶ **Lemma 8** ([28]). *If A_{i+1}, for $i = 0, 1, \ldots, k-2$, is obtained by including each vertex of A_i independently with probability $n^{-1/k}$, then $\mathbb{E}[\sum_{u \in V} |CL_V(u)|] = O(kn^{1+1/k})$.*

The main tool from [15] that we need is algorithm `ClusterOrCycle` (see Appendix F for the pseudocode), which either constructs a cluster or reports a short cycle. In [15] the vertex hierarchy $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k = \emptyset$, where $k \geq 1$, is initialized using the procedure `Initialize` (see Appendix F for the pseudocode). The value of $\delta(u, A_i)$ and $p_i(u)$ are computed as well, for every $u \in V$ and $i \in [k]$. In the `Initialize` algorithm, `Preprocess` is also called to allow efficient access to edges in `ClusterOrCycle`. The properties of `ClusterOrCycle` are summarized in the following lemma:

▶ **Lemma 9** (Lemma 5 in [15]). *Let $G = (V, E, \ell)$ be a weighted undirected graph on which we run procedure* `Initialize` *and let $u \in V$. If $CL(u)$ contains a cycle, let $r > 0$ be the smallest number such that $CL(u) \cap G_r(u)$ contains a cycle, then* `ClusterOrCycle`(u) *returns a description of a cycle of length at most $2r$. Furthermore, it returns the shortest paths from u to all vertices of $CL(u) \cap G_{<r}(u)$. Otherwise, if $CL(u)$ is a tree, then* `ClusterOrCycle`(u) *returns all the shortest paths from u to $CL(u)$.* `ClusterOrCycle`(u) *can be implemented in $O(|CL_V(u)| \log n)$ time.*

3 Technical overview

Algorithm Overview. Our techniques build upon the distance oracle of Thorup and Zwick [28] and the algorithm of [15], with an important and interesting difference. While both the prior works [28] and [15] employ uniform sampling to construct a vertex hierarchy of k levels, our algorithm instead uses *non-uniform sampling*. This non-uniform sampling enables the creation of what can be viewed as a "half-level" in the hierarchy, an idea we elaborate on below, which is crucial for achieving improved approximation guarantees for odd values of $k = 2\alpha + 1$.

Let $A_0 = V$, and let $A_1 \subseteq A_0$ be a random subset where each vertex of A_0 remains in A_1 with probability $n^{-1/k}$. Then, for every $2 \leq i \leq \alpha$, $A_i \subseteq A_{i-1}$ is a random subset where each vertex of A_{i-1} remains in A_i with probability $n^{-2/k}$, and let $A_{\alpha+1} = \emptyset$.

Let C be a shortest cycle of length g. Since A_1 is sampled with probability $n^{-1/k}$, the expected size of $B_0(u)$ is $O(n^{1/k})$ (whereas $|B_i(u)| = O(n^{2/k})$, for every $i > 0$). Our algorithm combines ideas from Ducoffe [9] and Kadria, Roditty, Sidford, Vassilevska Williams, and Zwick [15] with the fact that $B_0(u)$ is relatively small. This allows us to efficiently incorporate the "half-level" A_1 together with A_2 and to either find a short cycle satisfying $\hat{g} \leq 2g$, or to guarantee that the cycle C has a vertex sufficiently close to A_2, specifically $\delta(C, A_2) \leq 1.5 \cdot \frac{2}{3}g = g$, where $\delta(C, A_2) = \min\{\delta(c, a) \mid \langle c, a\rangle \in C \times A_2\}$.

Then, by utilizing ideas from [15] for the rest of the levels we show that either a cycle of length $\hat{g} \leq 2\delta(A_{i-1}, C) + 4g/3$ is found or $\delta(A_i, C) \leq \delta(A_{i-1}, C) + \frac{2g}{3}$. Combined with the fact that $\delta(A_2, C) \leq g$ we show that $\hat{g} \leq \frac{2k}{3}g$.

Roughly speaking, our algorithm works as follows. Let $\hat{g}$ be the girth approximation, and let $\hat{C}$ be the cycle of length $\hat{g}$ found. In Phase 1, we call `MinCycle`$(G[B_0(u)])$, and update $\hat{g}$ and $\hat{C}$ accordingly. Then, in Phase 2, for every $u \in A_1$ we call `ClusterOrCycle`(u) and update $\hat{g}$ and $\hat{C}$ accordingly, and finally, in Phase 3, we iterate over every edge $(u, v) \in E$, and check whether a short cycle can be closed using the edge (u, v) and a precomputed distance.

The correctness proof is divided into two cases: the case that $M(C) \leq g/3$, and the case that $M(C) \geq g/3$, where $M(C)$ is the maximum edge weight in C. (See Lemmas 15 and 16.)

Let $(u, u') \in C$ be an edge such that $\ell(u, u') = M(C)$, and let $h'_i(C) = \min(\delta(u, A_i), \delta(u', A_i))$, and let $h_i(C) = \min_{w \in A_i}(\delta(w, C))$ (see Figure 2 for an illustration).

First, we show that in Phase 1 of our algorithm, either a short cycle is found or $h_2(C), h'_2(C) \leq g$ (See Lemma 14).

In the case that $M(C) \leq g/3$, we then show that in Phase 2 of the algorithm, in every level we have that either $\hat{g} \leq 2h_i(C) + \frac{4}{3}g$ or $h_i(C) \leq h_{i-1}(C) + 2g/3$ (See Claim 11.2).

Since $A_{\alpha+1} = \emptyset$ and therefore $h_{\alpha+1}(C) = \infty$, by applying the above inequality $(\alpha - 2)$ times, we get that:

$$\hat{g} \leq 2\left(h_2(C) + \frac{2}{3}(\alpha-2)g\right) + \frac{4}{3}g \overset{h_2(C)\leq g}{\leq} 2\left(g + \frac{2}{3}(\alpha-2)g\right) + \frac{4}{3}g = \frac{2}{3}(2\alpha+1)g,$$

as required.

Similarly, in the case that $M(C) > g/3$, we show that either a short cycle is found in Phase 2 or 3, or that $h'_i(C) \leq h'_{i-1}(C) + 2g/3$, and using the above inequalities for $h'_i(C)$ we have that $\hat{g} \leq \frac{2}{3}(2\alpha+1)g$.

Conditional Lower Bounds for $C_{\leq 4}$ detection. We consider the $C_{\leq 4}$ detection problem, which asks whether a graph contains a triangle or a cycle of length 4. Here we modify a reduction by [14] from Strong3Sum to triangle detection. We show that if one instead starts from the Stronger3Sum Hypothesis, where the input 3Sum instance is assumed to both be a Sidon Set and have small numbers, then one can create polylogarithmically more instances of triangle detection that now contain no 4-cycles and whp the 3Sum instance has a solution if and only if one of these instances contains a triangle. Hence, any efficient $C_{\leq 4}$ detection algorithm can be used to solve the original 3Sum instance efficiently.

Upper and lower bound for AllEdgeTriangleOrListing. We also consider the (k,t)-AllEdgeTriangleOrListing problem, which asks to either solve the AllEdgeTriangle or list *at least* t cycles of length at most k. To show a reduction from 3Sum to (k,t)-AllEdgeTriangleOrListing, we consider the problem of AllEdgeTriangle in sparse graphs with a bounded number of short cycles that was proven by [14] to require $n^{2-o(1)}$ time. We then leverage a sampling technique used by [2, 1, 14] to further reduce the number of cycles of length at most k to a point where the number of cycles found is less than t, and therefore the AllEdgeTriangleOrListing in fact solves the AllEdgeTriangle in these subgraphs. Using this technique we obtain the $O(m^{1+\frac{1}{k-1}})$ lower bound for the AllEdgeTriangleOrListing problem.

In addition, we provide an upper bound algorithm for the AllEdgeTriangleOrListing problem. To achieve this, we extend the `DegenerateOrCycle` and `BallOrCycle` procedures of [16] to handle the more challenging task of AllEdgeTriangleOrListing, rather than girth approximation. Using these extended tools, we develop an algorithm that solves the $(2k,t)$-AllEdgeTriangleOrListing problem in $O(m + \min(m^{1+\frac{1}{k+1}}, n^{1+2/k}) + tk)$ time.

4 Weighted girth approximation algorithm

In this section[11], we prove the following theorem:

Reminder of Theorem 1. *Let $G = (V, E, \ell)$ be a weighted graph, where $\ell : E \to \mathbb{R}_{\geq 0}$, and let $k \geq 3$ be an **odd** integer. There exists an algorithm that runs in $\tilde{O}(m + n^{1+\frac{2}{k}})$ time and returns an estimation $\hat{g}$ such that $g \leq \hat{g} \leq \frac{2k}{3}g$.*

Let $k = 2\alpha + 1$. Our algorithm is composed of an initialization phase and three different phases of girth approximations. Roughly speaking, Phase 1 handles the first "half level" using ideas from [8], and Phases 2,3 use ideas from [15] for the rest of the levels.

[11] This section has missing proofs due to the line limit, for the full version see Appendix A.

In the initialization phase, we initialize all the data structures needed for the algorithm; this phase is similar to `Initialize` from [15]. For completeness, we describe `Initialize` in Appendix F.1, with one major difference: we create a hierarchy of vertex sets using non-uniform sampling. We have $A_0 = V$, $A_{\alpha+1} = \emptyset$, $A_1 = \texttt{Sample}(A_0, n^{-1/(2\alpha+1)})$, and $A_i = \texttt{Sample}(A_{i-1}, n^{-2/(2\alpha+1)})$, for every $2 \leq i \leq \alpha$. The reason for these different probabilities is to create a first "half level". Using this "half level", the algorithm can address odd values of k. In this phase, the current girth estimation $\hat{g}$ is initialized to ∞, the current shortest cycle $\hat{C}$ is set to $\emptyset$, and both the distance hash table d and the predecessor hash table π (used to reconstruct shortest paths) are initialized as empty hash tables. In addition, the algorithm also computes $\delta(p_i(u), u)$ for every $0 \leq i \leq \alpha$, and calls $\texttt{Preprocess}(G)$, as required by [15], to efficiently implement `ClusterOrCycle`.

In Phase 1, for every vertex $u \in V$, the algorithm computes the subgraph $G[B_0(u)]$ using Lemma 5 and calls $\texttt{MinCycle}(u, G[B_0(u)])$. If the length of a shortest cycle that goes through u in $G[B_0(u)]$ is smaller than $\hat{g}$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

In Phase 2, for every vertex $u \in A_1$, the algorithm computes $\texttt{ClusterOrCycle}(u)$. If a cycle shorter than the current estimate $\hat{g}$ is detected by $\texttt{ClusterOrCycle}(u)$ in $CL(u)$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

Finally, in phase 3, for every edge $(v, w) \in E$ and every $0 \leq i \leq \alpha$, the algorithm checks whether the edge (v, w) closes a cycle in the shortest paths tree rooted at $u = p_i(v)$. If (v, w) indeed closes a cycle and the length of this cycle is smaller than $\hat{g}$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

The algorithm checks whether (v, w) actually closes a cycle rather than merely retracing a path in the shortest paths tree rooted at u by verifying that $\pi(u, v) \neq (w, v)$ and $\pi(u, w) \neq (v, w)$. If $\pi(u, v) \neq (w, v)$ and $\pi(u, w) \neq (v, w)$, a cycle is indeed formed, and the length of the cycle is bounded from above by $\hat{g}' = d(u, v) + \ell(v, w) + \delta(u, w)$. If $\hat{g}' < \hat{g}$, the algorithm updates $\hat{g}$ accordingly. In this case, we succinctly represent the discovered cycle $\hat{C}$ with the triplet (u, v, w).

At the end of the algorithm, to obtain a cycle from the triplet (u, v, w), the algorithm computes u', the Lowest Common Ancestor (LCA) of v and w in the shortest paths tree rooted at u, and returns the simple cycle $P(u', v) \cup P(u', w) \cup (v, w)$. The algorithm returns $\hat{g}$ as the estimated girth, along with $\hat{C}$ as the corresponding cycle.

Next, we bound the running time of the algorithm and show that $\texttt{Cycle}(G, k)$ takes $\tilde{O}(m + n^{1+2/k}) = \tilde{O}(m + n^{1+2/(2\alpha+1)})$ time, since $k = 2\alpha + 1$.

▶ **Lemma 10** (Lemma 13 in Appendix A). *`Cycle`(G, k) takes $O((m + kn^{1+2/(2\alpha+1)}) \log n + \alpha m)$ time.*

Proof. See Lemma 13 for the complete proof. ◀

Next, we prove the following lemma, which is a main tool used in our correctness proof.

▶ **Lemma 11** (Lemma 14 in Appendix A). *Let C be a shortest cycle; that is, $\ell(C) = g$. Let $u \in C$, and let (x, y) be the farthest edge from u in C, i.e., $(x, y) = \arg\max_{(x,y)\in C}(\delta(u, (x, y)))$.*

Then either $\hat{g} \leq 2g$ or $\min(\delta(A_2, x), \delta(A_2, y)) \leq g$.

Proof. Let $u \in C$, and let (x, y) be the farthest edge from u in C. Without loss of generality, assume that $\delta(u, x) \geq \delta(u, y)$. If $C \subseteq G[B_0(u)]$, then we have $\hat{g} \leq \ell(\texttt{MinCycle}(u, G[B_0(u)])) = g \leq 2g$, and the claim holds. Therefore, we assume that $C \not\subseteq G[B_0(u)]$.

Since $C \not\subseteq G[B_0(u)]$, there exists a vertex $w \in C$ such that $w \notin B_0(u)$. From the definition of $B_0(u)$, we have $\delta(u, A_1) \leq \delta(u, w)$. Since x is the farthest vertex from u in C, we get:

$$\delta(u, A_1) \leq \delta(u, w) \leq \delta(u, x) \tag{1}$$

Since C is a shortest cycle and $x \in C$, we have that $C = P(u, x) \cup P_2(u, x)$, and therefore $\ell(C) = \ell(P(u, x)) + \ell(P_2(u, x))$. Thus,

$$g = \ell(C) = \ell(P(u, x)) + \ell(P_2(u, x)) = \delta(u, x) + \ell(P_2(u, x)). \tag{2}$$

Let $r = \delta(u, A_1) + \ell(P_2(u, x))$. We show that:

▷ Claim 11.1 (Claim 14.1 in Appendix A). $C \subseteq G_r(p_1(u))$

Proof. See Claim 14.1 in Appendix A for the complete proof. ◁

We divide the rest of the proof into the case that $C \subseteq CL(p_1(u))$, and the case that $C \not\subseteq CL(p_1(u))$. If $C \subseteq CL(p_1(u))$, then together with Claim 14.1 we have that $C \subseteq G_r(p_1(u)) \cap CL(p_1(u))$, and therefore from Lemma 9 we have that:

$$\begin{aligned}\hat{g} &\leq \ell(\texttt{ClusterOrCycle}(p_1(u))) \leq 2r = 2(\delta(p_1(u), u) + \ell(P_2(u, x))) \\ &\overset{1}{\leq} 2(\delta(u, x) + \ell(P_2(u, x))) \overset{2}{\leq} 2g,\end{aligned}$$

as required.

If $C \not\subseteq CL(p_1(u))$, then we must have a cycle edge $(s, t) \in C$ such that $(s, t) \notin CL(p_1(u))$. If $(s, t) \neq (x, y)$ (recall that (x, y) is the farthest edge from u in C), then since C is a shortest cycle we have that $(s, t) \subseteq P(u, t)$ or $(t, s) \subseteq P(u, s)$, and therefore $\delta(u, (s, t)) = \delta(u, s)$ or $\delta(u, (s, t)) = \delta(u, t)$. Wlog, we assume that $\delta(u, (s, t)) = \delta(u, s) + \ell(s, t) = \delta(u, t)$. Since $(s, t) \notin CL(p_1(u))$, it follows from the definition of $CL(p_1(u))$ that

$$\delta(A_2, t) \leq \delta(p_1(u), s) + \ell(s, t) \overset{\triangle}{\leq} \delta(A_1, u) + \delta(u, s) + \ell(s, t) = \delta(A_1, u) + \delta(u, t) \tag{3}$$

Since $(s, t) \neq (x, y)$ it follows that $(s, t) \in P(u, x) \cup P(u, y)$. (See Figure 1 for an illustration of this case.) If $(s, t) \in P(u, y)$, then $\delta(u, y) = \delta(u, t) + \delta(t, y)$. Therefore:

$$\begin{aligned}\delta(A_2, y) &\overset{\triangle}{\leq} \delta(y, t) + \delta(A_2, t) \overset{3}{\leq} \delta(y, t) + \delta(p_1(u), u) + \delta(u, t) \\ &\overset{x \notin B_0(u)}{\leq} \delta(u, t) + \delta(t, y) + \delta(u, x) \overset{t \in P(u,y)}{\leq} \delta(u, y) + \delta(u, x) \leq g/2 + g/2 = g,\end{aligned}$$

where the last inequality follows from the fact that $\delta(u, z) \leq \frac{g}{2}$ for every $z \in C$, as required.

If $(s, t) = (y, x)$, then $(y, x) \notin CL(p_1(u))$, and by the definition of $CL(p_1(u))$ we have $\delta(A_2, x) \leq \delta(p_1(u), y) + \ell(y, x)$. Therefore:

$$\begin{aligned}\delta(A_2, x) &\leq \delta(p_1(u), y) + \ell(y, x) \overset{\triangle}{\leq} \delta(p_1(u), u) + \delta(u, y) + \ell(y, x) \\ &\overset{1}{\leq} \delta(u, x) + \ell(P_2(u, x)) \overset{2}{=} g,\end{aligned}$$

as required. ◀

Next, using Lemma 14 we bound the value of $\hat{g}$ from above by $\frac{2}{3}(2\alpha + 1)g$. Let C be a shortest cycle in G, i.e. $\ell(C) = g$. Recall that $M(C)$ is the length of a longest edge in C. To prove that $\hat{g} \leq \frac{2}{3}(2\alpha + 1)g$, we consider the following two cases: the case that $M(C) \leq g/3$

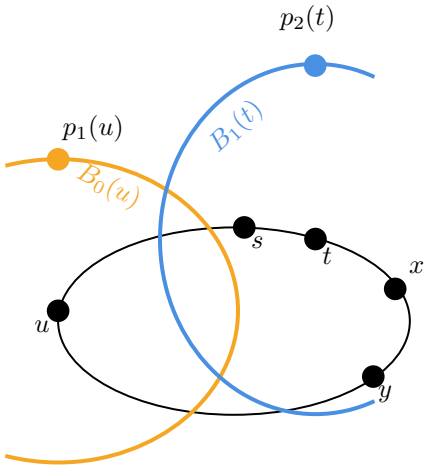
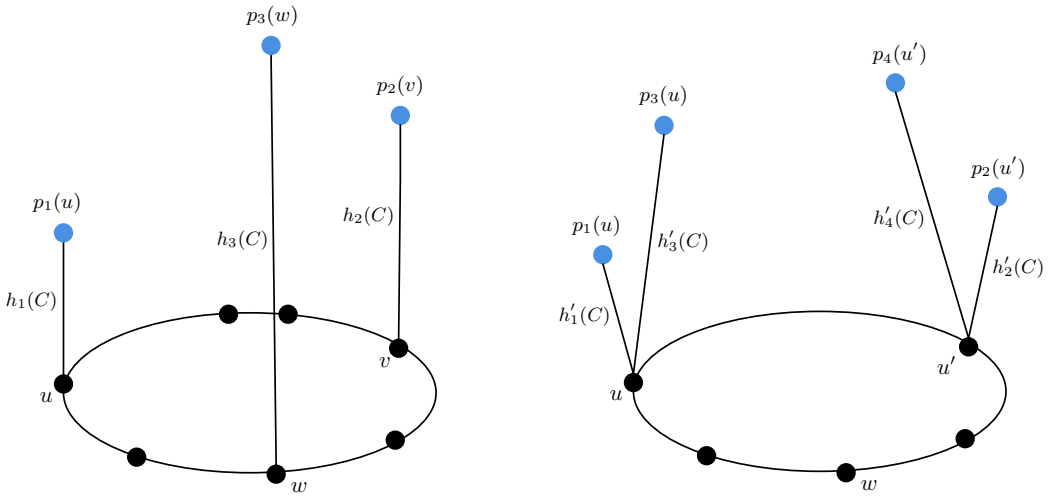


Figure 1 $\ell(C) = g$, $u \in C$, $(x, y) \in C$ farthest edge from u, $(s, t) \in C$ such that $(s, t) \notin CL(p_1(u))$ and $(s, t) \neq (x, y)$.

(Lemma 15) and the case that $M(C) > g/3$ (Lemma 16).[12] If $M(C) \leq g/3$, then we show that in the first two loops of the algorithm, a cycle of length at most $\frac{2}{3}(2\alpha + 1)g$ is found. Otherwise, if $M(C) > g/3$, we consider a longest edge $(u, u') \in C$ and show that either a short cycle is found during the first two loops, or that while considering the edge (u, u') in the third loop of the algorithm a cycle of length at most $\frac{2}{3}(2\alpha + 1)g$ is found. For the complete proofs, see Appendix A. Theorem 1 follows from Lemmas 13, 15, and 16.



$(a) : M(C) \leq g/3$
$h_1(C) = \delta(A_1, u),\ h_2(C) = \delta(A_2, v)$
$h_3(C) = \delta(A_3, w)$

$(b) : M(C) \geq g/3\ , \ell(u, u') = M(C)$
$h'_1(C) = \delta(A_1, u),\ h'_2(C) = \delta(A_1, u')$
$h'_3(C) = \delta(A_3, u),\ h'_4(C) = \delta(A_4, u')$

Figure 2 Illustration of $h_i(C)$ and $h'_i(C)$, where C is a shortest cycle in G.

▷ Claim 11.2. Either $\hat{g} \leq 2h_i(C) + 4g/3$ or $h_{i+1}(C) \leq h_i(C) + \frac{2}{3}g$

Proof. Appears in the full version [17]. ◁

[12] We remark that the distinction between $M(C) \leq g/3$ and $M(C) > g/3$ was also considered in the correction proof of [15].

5 Conditional lower bound for $(5/3-\epsilon)$-girth approximation under the Stronger3Sum hypothesis

In this section, we show a girth approximation lower bound under the Stronger3Sum hypothesis, which is defined as follows.

▷ Hypothesis 12 (Stronger3Sum). There is no $O(n^{2-\varepsilon})$ time algorithm for 3Sum on a set A where $A \subseteq [\pm O(n^2)]$ and A is a Sidon set.

We remark that although this hypothesis seems much stronger than the Strong 3Sum hypothesis, it is still plausible. In fact, [14] explicitly asks (Open problem 5) whether their techniques for fine-grained reductions for 3Sum in Sidon Sets can be used to prove the Stronger3Sum hypothesis assuming the Strong 3Sum hypothesis.

[14] proved a conditional lower bound for detecting a triangle under the Strong 3Sum hypothesis. Next, we prove a conditional lower bound for detecting $C_{\leq 4}$, i.e. whether a cycle contains a triangle or a C_4.

The reduction first follows the same steps as the reduction of Theorem 1.14 of [14]. Then, we show how to take a small number of subgraphs of the graphs created in the reduction of [14] so that if the input set A does not have a non-trivial solution to $a+b=c+d$ then the subgraphs created have no 4-cycles. Moreover, as before, any triangle corresponds to a 3SUM solution, and if A has a 3SUM solution, one of the subgraphs is guaranteed to have a triangle. Therefore, detecting a triangle in a sparse 4-cycle free graph is hard. We prove:

Reminder of Theorem 2. *Under the Stronger3Sum Hypothesis, there is no $O(n^{1.5-\varepsilon})$ time algorithm for $C_{\leq 4}$ detection and hence for $(5/3-\varepsilon)$-girth approximation in graphs with maximum degree $n^{1/4}$.*

Proof. Assume for the sake of contradiction that for some $\varepsilon > 0$ there exists an $O(n^{1.5-\varepsilon})$ time algorithm Alg for $C_{\leq 4}$ detection in graphs where $\deg(u) = O(n^{1/4})$, for every $u \in V$. We will show that there exists an $O(n^{2-\varepsilon})$ time algorithm for 3Sum on a (Sidon) set A where $|A| = n$, $A \subseteq [\pm C_1 n^2]$, for some constant C_1 and there are no non-trivial solutions to $a+b=c+d$ with $a,b,c,d \in A$.

First, we describe the construction of [14]. Let $A \subseteq [\pm n^2]$ be the input for 3Sum. Let $p,q,r \in [2C_1 n^{2/3}\log^2 n, 4C_1 n^{2/3}\log^2 n]$ be three distinct primes, sampled uniformly at random. Let $A' = \{a \pmod{pqr} \mid a \in A\}$. Let U (resp. V,W) be a vertex set identified by all numbers $[0,pqr)$ that are congruent to 0 mod p (resp. q,r). We add an edge between $u \in U$ and $v \in V$ if $v-u \pmod{pqr} \in A'$, between $v \in V$ and $w \in W$ if $w-v \pmod{pqr} \in A'$ and between $w \in W$ and $u \in U$ if $u-w \pmod{pqr} \in A'$. See Figure 3 for an illustration of this graph.

[14] shows that these edges can be added efficiently: for each $a \in A'$, if $v-u=a \pmod{pqr}$, then $v-u=a \pmod{q}$. Also, by construction, $u=0 \pmod{p}$ and $v=0 \pmod{q}$, which imply $u=-a \pmod{q}$. For every fixed $a \in A'$ and $y \in [0,\ldots,pqr-1)$, if $u'=0 \pmod{p}, u'=-a \pmod{q}, u'=y \pmod{r}$, then $u=(u' \pmod{pqr})$ is uniquely determined by the Chinese Remainder Theorem. Hence, we can go over all $a \in A', y \in [0,\ldots,pqr-1)$ which determine $u \in U$ and then add an edge to $(u+a \pmod{pqr}) \in V$. It thus takes $\tilde{O}(n \cdot r) = \tilde{O}(n^{5/3})$ time to add all the edges. (The addition of the edges in $V \times W$ and $W \times U$ is done similarly.) Finally, we remove all the edges in the graph whose degree is larger than $Cn^{1/3}$.

[14] shows that the graph has a triangle if and only if there is a solution for 3Sum in A; for completeness, we prove this for our construction in the appendix.

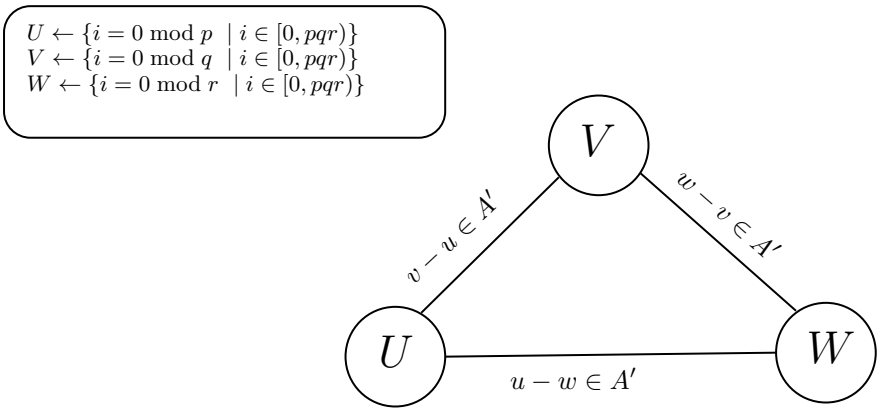


Figure 3 Illustration of Theorem 2.

▷ Claim 12.1. The graph has a triangle if and only if there is a solution for 3SUM in A.

Proof. Appears in the full version [17]. ◁

If A had a solution $x + y + z = 0$ where $x = z$, then $(A, 2A)$ would have a 2SUM solution. Since 2SUM can be solved in $O(n \log n)$ time by sorting, we can solve 3SUM in A in that much time. Hence, we assume that for any 3SUM solution in A, we have $x \neq z$. Since this means that $(x - z) \neq 0$, and since $(x - z)$ has $O(\log n)$ prime factors, we can assume that whp our random prime p does not divide $x - z$ and hence $x \neq z \pmod p$.

Now, notice that if A has a 3SUM solution $x + y + z = 0$, then whp in one of our trials, there is a triangle (u, v, w) in the graph where

$$u = 0 \pmod p, v = x \pmod p, w = -z \pmod p.$$

Since we can assume that $x \neq z \pmod p$ in any 3SUM solution, we can reduce 3SUM to $O(\log n)$ instances of subgraphs of our graphs above, where for every $v \in V, w \in W$ we have that $v \neq -w \pmod p$ and such that we are guaranteed that whp one of the subgraphs will contain a triangle if A had a 3SUM.

We do this as follows. Fix one of our graphs G from the trials above. For each $i \in [0, \ldots, \log p)$ and each choice of $(b, b') \in \{(0, 1), (1, 0)\}$, we create a subgraph $G_{bb'}$ of G by only keeping those $v \in V$ such that the ith bit of $v \pmod p$ is b, and those $w \in W$ such that the ith bit of $-w \pmod p$ is b'. Since a triangle (u, v, w) corresponding to a 3SUM solution has $v \neq -w \pmod p$, there is some bit i for which the ith bits of $v \pmod p$ and $-w \pmod p$ (b and b') differ. For the corresponding subgraph $G_{bb'}$, both v and w appear and hence (u, v, w) is a triangle. Meanwhile, for every graph $G_{bb'}$, for every v, w we have that $v \neq -w \pmod p$.

Now let's focus on any one of the graphs $G_{bb'}$ mentioned above. Suppose, for contradiction, that $G_{bb'}$ contains a 4-cycle. There are two types of 4-cycles. The first is w.l.o.g of the form $u, u' \in U, v, v' \in V$ that goes $u \to v \to u' \to v' \to u$ (or (U, V) can be (V, W) or (W, U)). The second is wlog of the form $u \to v \to u' \to w \to u$, for $u, u' \in U$, $v \in V$, $w \in W$ (or some permutation of U, V, W).

Let's consider the first type of 4-cycle: $u \to v \to u' \to v' \to u$. By construction, we have that $v - u = a \pmod{pqr}, v' - u = a' \pmod{pqr}, v - u' = a'' \pmod{pqr}, v' - u' = a''' \pmod{pqr}$. As $v \neq v', u \neq u'$ we get that $a \neq a'$, $a'' \neq a'''$, $a \neq a''$ and $a' \neq a'''$.

However, we have that $a - a'' = v - v' = a' - a''' \mod (pqr)$, so that $a + a''' = a' + a'' \mod (pqr)$. Since pqr is chosen to be large enough, we also have that $a + a''' = a' + a''$ for the corresponding integers in A. We showed that $a \neq a', a''$ and $a''' \neq a', a''$ so we have a non-trivial solution to $x + y = z + w$, a contradiction to A being a Sidon set.

For the second type of 4-cycle $u - v - u' - w - u$ we obtain a similar contradiction but with a bit more work. We get $u - w = a \mod (pqr), v - u = a' \mod (pqr), u' - w = a'' \mod (pqr), v - u' = a''' \mod (pqr)$. We have that $a + a' = v - w = a'' + a''' \mod (pqr)$ so there is a solution in A

$$a + a' = a'' + a'''.$$

Again, since $u \neq u'$ we get that $a \neq a''$, $a' \neq a'''$.

Suppose for the sake of contradiction that $a = a'''$ (a similar argument can be made for $a' = a''$). Then from the definition of a and a''' we get that $(u - w) = (v - u') \mod (pqr)$ and hence

$$u + u' = v + w \mod (pqr).$$

However, consider this equation modulo p. Since for all $u \in U$ we have $u = 0 \pmod p$, we obtain $u + u' = 0 \pmod p$. Meanwhile, $v + w \neq 0 \pmod p$ because in $G_{bb'}$ we ensured that for all v, w, $v \neq -w \pmod p$, a contradiction.

Thus, we must have that $a \neq a'''$ and $a' \neq a''$, in addition to the fact that $a \neq a''$ and $a' \neq a'''$, and therefore $a + a' = a'' + a'''$ is a non-trivial solution to $x + y = z + w$, a contradiction to A being a Sidon set. Thus, such cycles cannot exist in $G_{bb'}$.

This means that in $G_{bb'}$ we have removed the 4-cycles with two nodes in U. There could be 4-cycles with two nodes in V (and one in U, W each) or two nodes in W (and one in U, V). To handle these, we create $O(\log^2 n)$ subgraphs of $G_{bb'}$ where we also go through all $O(\log^2 n)$ choices of pairs of bits of q and r to make sure that $u + w \neq 0 \pmod q$ and $u + v \neq 0 \pmod r$. We obtain a poly-logarithmic number of subgraphs where we are guaranteed that if there were a 3Sum in A, in one of the subgraphs, this 3Sum is represented by a triangle and there are no 4-cycles.

The number of vertices N in each of the poly-logarithmic number of graphs is $\Theta(n^{4/3})$, and the maximum degree is $O(n^{1/3}) = O(N^{1/4})$, so the $C_{\leq 4}$ detection instance requires $n^{2-o(1)} = N^{1.5-o(1)}$ time under the Stronger3Sum hypothesis. ◀

References

1. Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023. doi:10.1145/3564246.3585240.
2. Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in p via short cycle removal: cycle detection, distance oracles, and beyond. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1487–1500. ACM, 2022. doi:10.1145/3519935.3520066.
3. Noga Alon, Raphy Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997. doi:10.1007/BF02523189.
4. Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014. doi:10.1007/978-3-662-43948-7_19.

5 Timothy M. Chan and Yinzhan Xu. Simpler reductions from exact triangle. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 28–38. SIAM, 2024. doi:10.1137/1.9781611977936.4.

6 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052. SIAM, 2014. doi:10.1137/1.9781611973402.78.

7 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. doi:10.1145/3055399.3055459.

8 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. doi:10.4230/LIPIcs.ICALP.2019.49.

9 Guillaume Ducoffe. Faster approximation algorithms for computing shortest cycles on weighted graphs. *SIAM J. Discret. Math.*, 35(2):953–969, 2021. doi:10.1137/20M1330415.

10 D. Easley and J. Kleinberg. *Networks, crowds, and markets: reasoning about a highly connected world*. Cambridge Univ Press, Cornell, NY, 2010.

11 Carlos Hoppen and Nicholas Wormald. Properties of regular graphs with large girth via local algorithms. *Journal of Combinatorial Theory, Series B*, 121:367–397, 2016. doi:10.1016/J.JCTB.2016.07.009.

12 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. doi:10.1137/0207033.

13 Ce Jin, Virginia Vassilevska Williams, and Renfei Zhou. Listing 6-cycles. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 19–27. SIAM, 2024. doi:10.1137/1.9781611977936.3.

14 Ce Jin and Yinzhan Xu. Removing additive structure in 3sum-based reductions. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 405–418, 2023. doi:10.1145/3564246.3585157.

15 Avi Kadria, Liam Roditty, Aaron Sidford, Virginia Vassilevska Williams, and Uri Zwick. Improved girth approximation in weighted undirected graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2242–2255. SIAM, 2023. doi:10.1137/1.9781611977554.CH85.

16 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 2022)*, 2022.

17 Avi Kadria, Liam Roditty, and Virginia Vassilevska Williams. Tighter bounds for weighted and unweighted shortest cycle approximation. *arXiv preprint*, 2026. arXiv:2607.00938.

18 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. doi:10.1137/1.9781611974331.CH89.

19 Felix Lazebnik and Ping Wang. On the structure of extremal graphs of high girth. *Journal of Graph Theory*, 26(3):147–153, 1997. doi:10.1002/(SICI)1097-0118(199711)26:3\%3C147::AID-JGT5\%3E3.0.CO;2-R.

20 Andrzej Lingas and Eva-Marta Lundell. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.*, 109(10):493–498, 2009. doi:10.1016/j.ipl.2009.01.008.

21 Deryk Osthus, Hans Jürgen Prömel, and Anusch Taraz. Almost all graphs with high girth and suitable density have high chromatic number. *Journal of Graph Theory*, 37(4):220–226, 2001. doi:10.1002/JGT.1017.

22 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

23 Liam Roditty and Roei Tov. Approximating the girth. *ACM Trans. Algorithms*, 9(2):15:1–15:13, 2013. doi:10.1145/2438645.2438647.

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

25 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.27.

26 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. doi:10.1137/1.9781611973099.67.

27 Philip M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$. *SIAM J. Comput.*, 2(1):28–32, 1973. doi:10.1137/0202004.

28 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

29 Virginia Vassilevska Williams and Alek Westover. Listing 6-cycles in sparse graphs. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*, volume 325 of *LIPIcs*, pages 92:1–92:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ITCS.2025.92.

30 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.

31 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic triangles, triangle listing and apsp. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797. IEEE, 2020. doi:10.1109/FOCS46700.2020.00078.

32 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM J. Discret. Math.*, 10(2):209–222, 1997. doi:10.1137/S0895480194274133.

A Weighted girth approximation algorithm

In this section, we prove the following theorem:

Reminder of Theorem 1. *Let $G = (V, E, \ell)$ be a weighted graph, where $\ell : E \to \mathbb{R}_{\geq 0}$, and let $k \geq 3$ be an **odd** integer. There exists an algorithm that runs in $\tilde{O}(m + n^{1+\frac{2}{k}})$ time and returns an estimation $\hat{g}$ such that $g \leq \hat{g} \leq \frac{2k}{3} g$.*

Let $k = 2\alpha + 1$. Our algorithm is composed of an initialization phase and three different phases of girth approximations. Roughly speaking, Phase 1 handles the first "half level" using ideas from [8], and Phases 2,3 use ideas from [15] for the rest of the levels.

In the initialization phase, we initialize all the data structures needed for the algorithm; this phase is similar to `Initialize` from [15]. For completeness, we describe `Initialize` in Appendix F.1, with one major difference: we create a hierarchy of vertex sets using non-uniform sampling. We have $A_0 = V$, $A_{\alpha+1} = \emptyset$, $A_1 = \texttt{Sample}(A_0, n^{-1/(2\alpha+1)})$, and

$A_i = \texttt{Sample}(A_{i-1}, n^{-2/(2\alpha+1)})$, for every $2 \leq i \leq \alpha$. The reason for these different probabilities is to create a first "half level". Using this "half level", the algorithm can address odd values of k. In this phase, the current girth estimation $\hat{g}$ is initialized to ∞, the current shortest cycle $\hat{C}$ is set to $\emptyset$, and both the distance hash table d and the predecessor hash table π (used to reconstruct shortest paths) are initialized as empty hash tables. In addition, the algorithm also computes $\delta(p_i(u), u)$ for every $0 \leq i \leq \alpha$, and calls $\texttt{Preprocess}(G)$, as required by [15], to efficiently implement $\texttt{ClusterOrCycle}$.

In Phase 1, for every vertex $u \in V$, the algorithm computes the subgraph $G[B_0(u)]$ using Lemma 5 and calls $\texttt{MinCycle}(u, G[B_0(u)])$. If the length of a shortest cycle that goes through u in $G[B_0(u)]$ is smaller than $\hat{g}$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

In Phase 2, for every vertex $u \in A_1$, the algorithm computes $\texttt{ClusterOrCycle}(u)$. If a cycle shorter than the current estimate $\hat{g}$ is detected by $\texttt{ClusterOrCycle}(u)$ in $CL(u)$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

Finally, in phase 3, for every edge $(v, w) \in E$ and every $0 \leq i \leq \alpha$, the algorithm checks whether the edge (v, w) closes a cycle in the shortest paths tree rooted at $u = p_i(v)$. If (v, w) indeed closes a cycle and the length of this cycle is smaller than $\hat{g}$, then the algorithm updates $\hat{C}$ and $\hat{g}$.

The algorithm checks whether (v, w) actually closes a cycle rather than merely retracing a path in the shortest paths tree rooted at u by verifying that $\pi(u, v) \neq (w, v)$ and $\pi(u, w) \neq (v, w)$. If $\pi(u, v) \neq (w, v)$ and $\pi(u, w) \neq (v, w)$, a cycle is indeed formed, and the length of the cycle is bounded from above by $\hat{g}' = d(u, v) + \ell(v, w) + \delta(u, w)$. If $\hat{g}' < \hat{g}$, the algorithm updates $\hat{g}$ accordingly. In this case, we succinctly represent the discovered cycle $\hat{C}$ with the triplet (u, v, w).

At the end of the algorithm, to obtain a cycle from the triplet (u, v, w), the algorithm computes u', the Lowest Common Ancestor (LCA) of v and w in the shortest paths tree rooted at u, and returns the simple cycle $P(u', v) \cup P(u', w) \cup (v, w)$. The algorithm returns $\hat{g}$ as the estimated girth, along with $\hat{C}$ as the corresponding cycle.

Next, we bound the running time of the algorithm and show that $\texttt{Cycle}(G, k)$ takes $\tilde{O}(m + n^{1+2/k}) = \tilde{O}(m + n^{1+2/(2\alpha+1)})$ time, since $k = 2\alpha + 1$.

▶ **Lemma 13.** *$\texttt{Cycle}(G, k)$ takes $O((m + kn^{1+2/(2\alpha+1)}) \log n + \alpha m)$ time.*

Proof. Appears in the full version [17]. ◀

Next, we prove the following lemma, which is a main tool used in our correctness proof.

▶ **Lemma 14.** *Let C be a shortest cycle; that is, $\ell(C) = g$. Let $u \in C$, and let (x, y) be the farthest edge from u in C, i.e., $(x, y) = \arg\max_{(x,y)\in C}(\delta(u, (x, y)))$.*

Then either $\hat{g} \leq 2g$ or $\min(\delta(A_2, x), \delta(A_2, y)) \leq g$.

Proof. Appears in the full version [17].

▷ Claim 14.1. $C \subseteq G_r(p_1(u))$

Proof. Appears in the full version [17]. ◁

◀

Next, using Lemma 14 we bound the value of $\hat{g}$ from above by $\frac{2}{3}(2\alpha + 1)g$. Let C be a shortest cycle in G, i.e. $\ell(C) = g$. Recall that $M(C)$ is the length of a longest edge in C. To prove that $\hat{g} \leq \frac{2}{3}(2\alpha + 1)g$, we consider the following two cases: the case that $M(C) \leq g/3$

Algorithm 1 `Cycle`(G, α).

```
   /* Initialization phase                                                    */
 1 ĝ ← ∞, Ĉ ← ∅, d ← HashTable(), π ← HashTable()
 2 A_0 ← V ; A_{α+1} = ∅ ; A_1 ← Sample(A_0, n^{-1/(2α+1)})
 3 for i ∈ [2, α] do  A_i ← Sample(A_{i-1}, n^{-2/(2α+1)}) ;
 4 Preprocess(G)
 5 Compute p_i(u) and d(u, p_i(u)) for every ⟨u, i⟩ ∈ ⟨V, [α]⟩
   /* Phase 1:                                                                */
 6 for u ∈ V do
 7     Compute G[B_0(u)] using Lemma 5
 8     ⟨ĝ', Ĉ'⟩ ← MinCycle(u, G[B_0(u)])
 9     if ĝ' ≤ ĝ then
10         ĝ ← ĝ' ; Ĉ ← Ĉ'
   /* Phase 2:                                                                */
11 for u ∈ A_1 do
12     ⟨ĝ', Ĉ'⟩ ← ClusterOrCycle(u)
13     if ĝ' ≤ ĝ then
14         ĝ ← ĝ' ; Ĉ ← Ĉ'
   /* Phase 3:                                                                */
15 for (v, w) ∈ E do
16     for i ← 0 to α do
17         u ← p_i(v) ; ĝ' ← d(u, v) + ℓ(v, w) + d(u, w) ;
18         if ĝ' < ĝ and π(u, v) ≠ (w, v) and π(u, w) ≠ (v, w) then
19             ĝ ← ĝ' ; Ĉ ← (u, v, w) ;
20 if Ĉ is a triplet (u, v, w) then u' ← LCA(u, v, w); Ĉ = P(u', v) ∪ P(u', w) ∪ (v, w);
21 return ⟨ĝ, Ĉ⟩
```

(Lemma 15) and the case that $M(C) > g/3$ (Lemma 16).[13] If $M(C) \leq g/3$, then we show that in the first two loops of the algorithm, a cycle of length at most $\frac{2}{3}(2\alpha+1)g$ is found. Otherwise, if $M(C) > g/3$, we consider a longest edge $(u, u') \in C$ and show that either a short cycle is found during the first two loops, or that while considering the edge (u, u') in the third loop of the algorithm a cycle of length at most $\frac{2}{3}(2\alpha+1)g$ is found. We start with the case where $M(C) \leq g/3$.

▶ **Lemma 15.** *If $M(C) \leq g/3$ then $\hat{g} \leq \frac{2}{3}(2\alpha+1)g$*

Proof. Appears in the full version [17]. ◀

Next, we show that $\hat{g} \leq \frac{2}{3}(2\alpha+1)g$ in the case where $M(C) > g/3$.

▶ **Lemma 16.** *If $M(C) > g/3$ then $\hat{g} \leq \frac{2}{3}(2\alpha+1)g$.*

Proof. Appears in the full version [17]. ◀

Theorem 1 follows from Lemmas 13, 15, and 16.

[13] We remark that the distinction between $M(C) \leq g/3$ and $M(C) > g/3$ was also considered in the correction proof of [15].

B Proof of Claim 12.1

▷ Claim 17 (Claim 12.1). The graph has a triangle if and only if there is a solution for 3Sum in A.

Proof. Appears in the full version [17]. ◁

C Upper and lower bounds for the AllEdgeTriangleOrListing problem

Appears in the full version [17].

D Definitions

Appears in the full version [17].

E Lower bounds for all edge shortest cycle approximation

E.1 $m^{1+\frac{1}{k-1}-o(1)}$ lower bound for $k/3$-stretch for all edge shortest cycle

In this section, we show that using the techniques of [14] it is also possible to prove a lower bound for all edge shortest cycle. We use the following lemma:

▶ **Lemma 18** ([14]). *Fix any constant $\sigma \in (0, 0.5)$, and any integer $k \geq 3$. Under the 3Sum hypothesis, it requires $n^{2-o(1)}$ time to solve $n^{3\sigma}$ instances of All-Edges Sparse Triangle on tripartite graphs with $O(n^{1-\sigma})$ vertices and maximum degree $O(n^{0.5-\sigma})$, such that the total number of cycles of length at most k over all instances is $O(n^{k/2-(k-3)\sigma})$*

Next, we prove the lower bound for AESC using the methods of [14].

▶ **Theorem 19.** *Under the 3Sum hypothesis, given a graph $G = (V, E)$ with n vertices and maximum degree $n^{1/(k-2)}$, there is no $O(n^{1+2/(k-2)-\varepsilon})$, for $\varepsilon > 0$, time algorithm, for a $(k/3 - \delta)$, for $\delta > 0$, approximation for AESC. In terms of the number of edges m, the lower bound is $m^{1+1/(k-1)-o(1)}$.*

Proof. Appears in the full version [17]. ◀

E.2 Alternative lower bound proof for the $(5/3 - \epsilon)$ short cycle approximations

Next, we prove the lower bound for $k/3 - \delta$ stretch for all edge cycle in an alternative way. We show that under the 3Sum hypothesis, for $\rho \geq 2/3$ there is no $O(n^{3-2\rho-\varepsilon})$ time algorithm for the $(4, \rho, O(1))$-AllEdgeCycleListing. We prove:

▶ **Theorem 20.** *Under the 3Sum hypothesis, given a graph $G = (V, E)$ with n vertices and maximum degree $n^{1/3}$, there is no $O(n^{5/3-\varepsilon})$-time algorithm, for a $(5/3 - \varepsilon)$-approximation for AESC. In terms of the number of edges m, the lower bound is $m^{5/4-o(1)}$.*

The reduction follows the same steps as the reduction of [31], however we show that if the weights of the input graph are a Sidon set then the number of C_4 is very small (Claim 21.1), and therefore even approximating a shortest cycle for each edge, that is, reporting either a triangle or a C_4 is hard.

▶ **Theorem 21.** *Let $\rho \geq 2/3$. Under the 3Sum hypothesis, there is no $O(n^{3-2\rho-\varepsilon})$, for $\varepsilon > 0$, time algorithm for $(4, \rho, O(1))$-AllEdgeCycleListing.*

Proof. Appears in the full version [17].

▷ Claim 21.1. Let $(a, b) \in E'_{i,j,k}$ such that for some c, (a, b, c) is a zero triangle in $E'_{i,j,k}$. $\mathbb{E}(|\{C_4 \mid (a, b) \in C_4\}|) = O(n^{2-3\rho})$

Proof. Appears in the full version [17]. ◁

▷ Claim 21.2 (Claim 3.6 [31]). Let $(a, b) \in E_{i,j,k}$ such that for some c, (a, b, c) is a zero triangle in $E_{i,j,k}$. $\mathbb{E}[|\{C_3 \mid (a, b) \in C_3 \text{ and } \ell(C_3) \neq 0\}|] = O(n^{1-2\rho})$

Proof. Appears in the full version [17]. ◁

◀

To complete the proof of Theorem 20 we show that $(4, \rho)$-AllEdgeCycle is equivalent to $(4, \rho, O(1))$-AllEdgeCycleListing in the following lemma. (Note that we don't need the lemma in full generality as we only need that for every edge either a triangle is found, or $O(1)$ 4-cycles are listed, but we provide the lemma for completeness.)

▶ **Lemma 22.** *If there is an $\tilde{O}(n^{1+\alpha})$ time algorithm for $(4, \rho)$-AllEdgeCycle then there is also an $\tilde{O}(n^{1+\alpha})$ time algorithm for $(4, \rho, O(1))$-AllEdgeCycleListing that succeeds whp.*

Proof. Appears in the full version [17]. ◀

Combining Theorem 21 and Lemma 22, we obtain the following hardness result for all edge shortest cycle approximation.

▶ **Corollary 23.** *Let $\rho \geq 2/3$. Under the 3Sum hypothesis, there is no $O(n^{3-2\rho-\varepsilon})$, for $\varepsilon > 0$, time algorithm for $(4, \rho)$-AllEdgeCycle.*

Theorem 20 follows from Corollary 23 with $\rho = 2/3$.

F `Initialize` and `ClusterOrCycle` of [15]

Appears in the full version [17].

F.1 `Initialize`

Appears in the full version [17].

F.2 Algorithm `ClusterOrCycle`

Appears in the full version [17].