Improved girth approximation in weighted undirected graphs

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

Abstract

Let $G = (V, E, \ell)$ be a n-nodes m-edges weighted undirected graph, where $\ell : E \to (0, \infty)$ is a real *length* function defined on its edges. Let g be the length of the shortest cycle in G.

We present an algorithm that in $O(kn^{1+1/k} \log n + m(k + \log n))$ expected running time finds a cycle of length at most $\frac{4k}{3}g$, for every integer $k \geq 1$. This improves upon the previous best algorithm that in $O((n^{1+1/k} \log n + m) \log(nM))$ time, where $\ell : E \to [1, M]$ is an integral length function, finds a cycle of length at most $2kg$ [KRS+22]. For $k = 1$ our algorithm also improves the result of Roditty and Tov [RT13].

1 Introduction

A fundamental problem in algorithmic graph theory, the problem of computing a shortest cycle in an undirected graph, has been studied extensively for decades (e.g., [IR78, AYZ97, YZ97, LL09, Duc19] and more). Prominent special cases, e.g. detecting triangles in graphs, are foundational to algorithm design and complexity theory, and are useful in practice as well (e.g., [EK10, Chapter 3]). The length of a shortest cycle, known as the *girth* of the graph, is a key parameter often used to shed light on the structure of graph theory problems (e.g., [LW97, OPT01, HW16]).

Given the prominence of a shortest cycle and girth computation problems, extensive effort has gone into developing fast algorithms for them. The best known runtime for computing the girth of an n-vertex, m-edge unweighted graph is $O(\min\{n^\omega, mn\})$ [IR78], where $\omega < 2.373$ is the matrix multiplication exponent. For graphs with nonnegative integer edge lengths bounded by M, the running time is $\tilde{O}(\min\{Mn^\omega, mn\})$ [1] [RV11]. For arbitrary edge lengths and no negative cycles, the fastest algorithms solve All-Pairs Shortest Paths (APSP), whose fastest running time to date is $O(\min\{mn + n^2 \log\log n, n^3/\exp(\sqrt{\log n})\})$ [Pet04, Wil18]. Improving upon these bounds significantly would constitute a major breakthrough in algorithm design and [VW18, LVW18] clarified the difficulty in doing so; faster algorithms would imply breakthroughs in Fine-Grained Complexity.

Given the above hardness for exact girth computation, it is natural to ask which trade-offs between running times and approximation ratios are possible. For unweighted graphs, where all edge lengths are 1, there has been extensive work on this question and there are a variety of results depending on the graph's girth and the desired approximation [LL09, RT13, RV12, DKS17, KRS+22]. Perhaps most relevant to the results of this paper, Kadria et al. [KRS+22] presented a collection of new algorithms for girth approximation, including an algorithm that for any input unweighted n-vertex undirected graph of girth g and an integer parameter $k \geq 1$ finds a cycle of length at most $2k \cdot \lceil g/2 \rceil$ in $O(n^{1+1/k} \log n)$ time. This result improved upon a result of Dahlgaard, Knudsen and

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

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

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

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

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

[1] $\tilde{O}(f(n))$ denotes $O(f(n)\text{poly}\log(n))$.

Stöckel [DKS17] that provided an algorithm which in the same running time of $O(n^{1+1/k}\log n)$ finds a cycle of length at most $2^k g$ with probability $1-1/n$, for $k \geq 2$.

However, for the problem of approximating the girth of n-vertex m-edge undirected graphs with edge lengths in the range $[1, M]$ and girth g, considerably less is known. Until recently, the state-of-the-art included a result of Roditty and Tov [RT13] which obtained an $\widetilde{O}(n^2 \log M)$-time 4/3-approximation algorithm which improved upon a 2-approximation algorithm of Lingas and Lundell [LL09], as well as a result of Ducoffe [Duc19] which obtained an $O(m + n^{5/3}\text{polylog}M)$-time 2-approximation algorithm. Unlike in the case of unweighted graphs, no general trade-off between running time and approximation quality was known.

In the same recent paper, Kadria et al. [KRS+22] obtained the first running time versus approximation trade-off for weighted girth computation algorithms. They presented an algorithm that for any integer $k \geq 1$ finds a cycle of length at most $2k \cdot g$ in $O((n^{1+1/k}\log n + m)\log(nM))$ time. Note that for $k = 1$, this result offers a worse trade-off than both Roditty and Tov [RT13] and Ducoffe [Duc19]. Furthermore, the approximation quality achieved by this result is almost twice as large as that achieved by them for unweighted graphs (when the running times are made comparable).

In light of these results, the central question we ask in this paper is

Is it possible to design a single algorithm that yields improved runtime versus approximation quality trade-offs for undirected weighted graphs?

Our main result is the following theorem:

THEOREM 1.1. (IMPROVED GIRTH APPROXIMATION) *Let $G = (V, E, \ell)$ be a weighted undirected graph, where $\ell : E \to (0, \infty)$. Let g be the unknown girth of G. For every integer $k \geq 1$, there is an algorithm whose expected running time is $O(kn^{1+1/k}\log n + m \log n)$ that finds a cycle C such that $\ell(C) \leq \frac{4k}{3} g$.*

Among the tools used to prove the theorem are: Approximate distance oracles [TZ05], Spira's single-source shortest paths algorithm [Spi73] and a problem related to 2-dimensional orthogonal range reporting from computational geometry, as well as ideas borrowed from Kadria et al. [KRS+22].

Our result (up to logarithmic factors in running time) strictly improves upon Kadria et al. [KRS+22], computing a $(\frac{4k}{3}g)$-approximation (rather than $2kg$) in a comparable running time. Furthermore, the approximation quality versus runtime trade-off matches that of Roditty and Tov [RT13] for $k = 1$.

We note that beating the approximation factor of 4/3 approximation of Roditty and Tov [RT13] in the same running time faces a barrier: any $(4/3-\varepsilon)$-approximation algorithm for $\varepsilon > 0$ would be able to detect whether a graph contains a triangle, and via [VW18], we know that triangle detection is tightly related to Boolean Matrix Multiplication. Thus obtaining a quadratic time combinatorial $(4/3-\varepsilon)$-approximation algorithm would achieve a breakthrough in fast matrix multiplication. This suggests that our scheme might be of the right form.

Beyond improving upon [KRS+22] in approximation quality and matching that of Roditty and Tov [RT13], note that in contrast to these prior result the runtime in Theorem 1.1 is strongly polynomial (it does not depend on M)[2]. Consequently, Theorem 1.1 can be applied to graphs with arbitrary real positive edge lengths (assuming that addition and comparisons still take constant time), not only bounded integers. We thus obtain a strict improvement over the result of Roditty and Tov [RT13]: a trade-off curve that works for graphs with real length edges.

Interestingly, to obtain this result we depart from the approach of Kadria et al. [KRS+22]. Although the running time of the $2k$-approximation algorithm of [KRS+22] was $\tilde{O}(m + n^{1+1/k})$, if the edges incident to each vertex were given in non-decreasing order of their lengths, the algorithm could run in possibly *sublinear* time $\tilde{O}(n^{1+1/k})$. That is, it would not need to examine all the edges for dense enough graphs.

The algorithm, however, is limited in how it accesses the graph. It can only access the edges incident on a given vertex u in sequential order. That is, it can only access the i-th edge of u, in non-decreasing order of length, after accessing the previous $i-1$ edges of u.

We show that this property of the algorithm is what made it weak: if the widely believed Erdös' Girth Conjecture (see e.g., [Erd64]) holds, then any algorithm that accesses the edges of a weighted graph in the above sequential fashion and makes only $o(n^{1+1/k})$ queries can at best return a $(2k+2)$-approximation. That is, the $2k$-approximation algorithm of Kadria et al. [KRS+22] is essentially best possible given how it accesses the graph.

[2]We remark that Roditty and Tov [RT13] presented also an algorithm with $4/3+\epsilon$ approximation and $\tilde{O}((1/\epsilon)n^2)$ running time.

THEOREM 1.2. (QUERY LOWER BOUND) *Assume that the Girth Conjecture holds for an integer $k \geq 1$. Then for that k and any real value $\tau > 0$, every deterministic algorithm that when run on n-vertex weighted undirected graph, accessed using the edge oracle model outlined above, computes a cycle C with $\ell(C) \leq (2k + 2 - \tau)g$ must make at least $\Omega(n^{1+1/k})$ queries on some graphs.*

2 Preliminaries

2.1 Basic concepts 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 non-decreasing order of length. (If not, this can be easily done in $O(m \log n)$ time.) All graphs considered are assumed to be connected.

For all $u, v \in V$, we let $\delta_G(u, v)$ be the *distance* from u to v in G, i.e. the smallest length of path from u to v in the graph. The length $\ell(P)$ of a path P is the sum of the lengths of its edges, i.e., $\ell(P) = \sum_{e \in P} \ell(e)$. (We usually consider a path P to be a set of edges, but occasionally we also think of it as a set of vertices.) A path from u to v is a *shortest path* if and only if $\ell(P) = \delta_G(u, v)$. As G is undirected, $\delta_G(u, v) = \delta_G(v, u)$ for every $u, v \in V$. When the graph G is clear from the context, which will almost always be the case, we write $\delta(u, v)$ instead of $\delta_G(u, v)$.

A tree T rooted at u and containing the vertices of a set U is said to be a *shortest paths tree* from u to the vertices of U if for every $v \in U$, the path from u to v in T is a shortest path from u to v in G.

If $u \in V$ and $A \subseteq V$, we let $\delta(u, A) = \min_{v \in A} \delta(u, v)$. This is referred to as the distance from u to the set A. (If $A = \emptyset$, then $\delta(u, A) = +\infty$.)

We define the distance $\delta(u, (v, w))$ from a vertex $u \in V$ to an edge $(v, w) \in E$ as follows: $\delta(u, (v, w)) = \min\{\delta(u, v), \delta(u, w)\} + \ell(v, w)$. (Note that $\delta(u, (v, w)) = \delta(u, \{v, w\}) + \ell(v, w)$. Here $\{v, w\}$ is a set of two vertices.)

The *girth* g of a graph $G = (V, E, \ell)$ is the length of a shortest cycle in G. (If (u, v) is an edge, then (u, v, u) is of course not considered to be a cycle.) The length of a cycle C is the sum of the lengths of the edges on C, i.e., $\ell(C) = \sum_{e \in C} \ell(e)$. We also let $M(C) = \max_{e \in C} \ell(e)$ be the maximum edge length on C.

2.2 Balls Given a graph $G = (V, E, \ell)$, a vertex $u \in V$ and $r > 0$, we define the *ball graph* $G_r(u) = (V_r(u), E_r(u))$ of *radius* r around u as follows:

$$\begin{aligned} V_r(u) &= \{v \in V \mid \delta(u, v) \leq r\}, \\ E_r(u) &= \{e \in E \mid \delta(u, e) \leq r\}. \end{aligned}$$

Note that $G_r(u)$ is usually not equal to $G[V_r(u)]$, the subgraph of G induced by the vertex set $V_r(u)$ of $G_r(u)$.

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 simple lemma is useful in proving the correctness of our algorithms.

LEMMA 2.1. *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

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

Thus, $(x, y) \in E_{r + \frac{\ell(C) + M(C)}{2}}(u)$, for every $(x, y) \in C$ and thus $C \subseteq G_{r + \frac{\ell(C) + M(C)}{2}}(u)$, as required. ∎

2.3 Clusters Let $G = (V, E, \ell)$ be a weighted undirected graph and let $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k = \emptyset$ be a hierarchy of vertex sets, where $k \geq 1$. If $u \in A_i \setminus A_{i+1}$, then following [TZ05] we define the *cluster* of u in G to be the graph $CL(u) = (CL_V(u), CL_E(u))$, where

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

Note that unlike [TZ05], we define the cluster $CL(u)$ to be a graph and not just a vertex set.

For a vertex $u \in V$, we let $a(u) = i$ where $u \in A_i \setminus A_{i+1}$. 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. (The choice of $p_i(u)$ is not necessarily unique.) We also let $r_i(u) = \delta(u, A_i)$.

LEMMA 2.2. *Let $u \in A_i \setminus A_{i+1}$. If $v \in CL_V(u)$ and P is a shortest path from u to v, then all the vertices and edges on P are also in $CL(u)$.*

Proof. Let x be a vertex on the shortest path P from u to v. Assume, for contradiction, that $x \notin CL_V(u)$. Let $w = p_{i+1}(x) \in A_{i+1}$. Then, $\delta(w,x) = \delta(x, A_{i+1}) \leq \delta(u,x)$. It follows that $\delta(w,v) \leq \delta(w,x) + \delta(x,v) \leq \delta(u,x) + \delta(x,v) = \delta(u,v)$, contradicting the claim that $v \in CL_V(u)$. The proof for the edges on P is similar. ∎

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 2.3. *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 $\mathbb{E}[\sum_{u \in V} |CL_V(u)|] = O(kn^{1+1/k})$.*

Lemma 2.3 is proved in [TZ05] using a simple probabilistic argument.

Thorup and Zwick [TZ05] describe a simple modification of Dijkstra's algorithm using which $CL(u)$ can be constructed in $\tilde{O}(|E(CL_V(u))|)$, where $E(CL_V(u))$ is the set of all edges in G incident on a vertex of $CL_V(u)$.[3] All clusters can therefore be constructed in $\tilde{O}(kmn^{1/k})$ time. This is too slow for us as we are aiming for a running time of $\tilde{O}(kn^{1+1/k})$. Our girth approximation algorithm constructs most clusters only partially, until a cycle in them is detected. This is described in the next section.

2.4 Initialization We next describe of an initialization algorithm used by our girth approximation algorithm described in Section 4. The initialization algorithm `Initialize`(G, k), see Algorithm 1, receives the input graph $G = (V, E, \ell)$ and the parameter $k \geq 1$. The algorithm starts by sampling the vertex hierarchy $V = A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k = \emptyset$.

Next, it initializes two empty *hash tables* d and π used to store the distances and shortest paths already computed by the algorithm. When the algorithm discovers a distance $\delta(u,v)$ between two vertices $u, v \in V$, it inserts the pair (u,v) to the hash table d with key $\delta(u,v)$. For brevity, we write this as $d(u,v) \leftarrow \delta(u,v)$. When we want to check whether $\delta(u,v)$ was already computed, we search (u,v) in the hash table d. If (u,v) is found we retrieve $\delta(u,v)$. For brevity, we interpret $d(u,v)$ as a search for (u,v) in the hash table d. The search returns $\delta(u,v)$ if (u,v) is in the table, or $+\infty$, if (u,v) is not in the table, i.e., $\delta(u,v)$ is not yet known to the algorithm. (We assume that $d(u,v)$ searches both (u,v) and (v,u), or more efficiently, that all pairs (u,v) stored in the table satisfy $u < v$.)

The hash table π is similarly used to represent the shortest paths already found by the algorithm. Thus, if $d(u,v) < \infty$, then $\pi(u,v)$ is the last edge on a shortest path from u to v. Thus, if $\pi(u,v) = (w,v)$ then $d(u,v) = d(u,w) + \ell(w,v)$.

We assume that each operation on the hash tables d and π takes constant expected time, as can be achieved using standard hashing techniques.

The algorithm then computes the distances $\delta(u, A_i)$, for every $u \in V$ and $0 \leq i < k$. This is easily done by adding an auxiliary vertex s_i, connecting it with 0-length edges to all vertices of A_i and then running Dijkstra

[3] Using the ideas of the next section we can actually improve the running time needed to compute $CL(u)$ to $\tilde{O}(|CL_E(u))|$, but this may still be too slow.

Algorithm 1: `Initialize`$(G = (V, E, \ell), k)$

```
1  A_0 ← V ; A_k ← ∅
2  for i ← 1 to k − 1 do
3      A_i ← Sample(A_{i−1}, n^{−1/k})
4  d ← HashTable()      // Used to store computed distances.
5  π ← HashTable()      // Used to store computed shortest paths.
6  for i ← 1 to k − 1 do
7      Dijkstra(G, A_i)    // Finds δ(u, A_i) and p_i(u) for every u ∈ V.
8      for u ∈ V do
9          d(p_i(u), u) ← δ(u, A_i)
10 Preprocess(G)
```

from s_i, as done in [TZ05]. This also computes $p_i(u) = \arg\min_{v \in A_i} \delta(u, v)$ for every $u \in V$ and $0 \leq i < k$ and a corresponding shortest path from u to $p_i(u)$.

Finally, `Initialize` calls `Preprocess` that performs preprocessing operations on the adjacency lists of all vertices. This processing includes sorting each adjacency list in non-decreasing order of edge length, and for every $0 \leq i < k$ building a binary tree on the edges of vertex, as explained in Section 3.1. The total cost of all these preprocessing operations is $O(m \log n)$.

LEMMA 2.4. *The running time of* `Initialize` *is* $O((m + kn) \log n)$.

Proof. The k calls to Dijkstra's algorithm take $O(k(m + n \log n))$ time. Preprocessing the adjacency lists takes $O(m \log n)$. As we may assume that $k \leq \log n$, the total time is $O((m + kn) \log n)$. ∎

2.5 Cycle detection and compact cycle representation When the girth approximation algorithm discovers an edge (v, w) such that both distances $\delta(u, v)$ and $\delta(u, w)$ are known, for some $u \in V$, it checks whether $(v, w) \notin \pi(u, v), \pi(u, w)$. If so, a cycle is detected. (Recall that $\pi(u, v)$ and $\pi(u, w)$ are last edges on shortest paths from u to v and w, respectively.) The actual cycle is composed of the shortest paths from u' to v and w, where u' is the LCA (Lowest Common Ancestor) of v and w in the shortest paths tree rooted at u, and the edge (v, w).

This cycle and its length can be easily found in time proportional to the number of edges on the cycle. In some cases this is not fast enough. We thus succinctly represent the discovered cycle by the triplet (u, v, w) and use $\delta(u, v) + \delta(u, w) + \ell(v, w) = d(u, v) + d(u, w) + \ell(v, w)$ as an upper bound on its length. [4]

3 Algorithm `ClusterOrCycle`

In this section we describe an algorithm `ClusterOrCycle` that satisfies the following lemma:

LEMMA 3.1. *Let $G = (V, E, \ell)$ be a weighted undirected graph and let $u \in V$. If $CL(u)$ is a tree, then* `ClusterOrCycle`(u) *finds $CL(u)$, the distance $\delta(u, v)$ for every $v \in CL(u)$, and a tree of shortest paths from u to all vertices of $CL(u)$. Otherwise, if $r > 0$ is the smallest number such that $CL(u) \cap G_r(u)$ contains a cycle, then* `ClusterOrCycle`(u) *returns a description of a cycle in $CL(u) \cap G_r(u)$ whose length is at most $2r$. Furthermore, it returns $c\ell(u) = CL(u) \cap G_{<r}(u)$ and a tree containing shortest paths from u to all vertices of $c\ell(u)$. The running time of* `ClusterOrCycle`(u) *is $O(|CL_V(u)| \log n)$.*

Algorithm `ClusterOrCycle`(u) starts constructing the cluster $CL(u)$. It stops, however, when the first cycle in $CL(u)$ is encountered. This ensures that the running time `ClusterOrCycle`(u) is proportional to the number

[4] In principle we can use an LCA data structure to find u' and the actual length of the cycle in $O(1)$ time. This complicates the algorithm and does not lead to improved results.

of vertices in $CL(u)$, and not to the number of edges in $CL(u)$, which would have been too expensive. It uses a modification of Spira's [Spi73] single-source shortest paths algorithm.

Spira's algorithm assumes that the edges incident on each vertex are sorted in non-decreasing order of length. It may be viewed as a lazy version of Dijkstra's [Dij59] algorithm. In certain cases it may find distances to all vertices without examining all edges. (This is possible as the adjacency lists of all vertices are assumed to be sorted by length. For a recent application of Spira's algorithm see [WZ15].)

When Dijkstra's algorithm discovers the distance from the source u to a new vertex v, it immediately relaxes all the outgoing edges (v, w) of v. Spira's algorithm relaxes only the first outgoing edge of v. The heap Q used by Spira's algorithm contains edges rather than vertices. Relaxing an edge (v, w) amounts to inserting it to Q with key $d(u, v) + \ell(v, w)$. The algorithm also maintains a set U of vertices whose distance from the source u was already found. Initially $U = \{u\}$. In each iteration, Spira's algorithm extracts an edge (v, w) of minimum key from the heap Q. If $w \notin U$, it adds w to U and sets $d(u, w) \leftarrow d(u, v) + \ell(v, w)$ which is guaranteed to be the distance from u to w. It now relaxes the first edge of w and the next edge of v, i.e., the edge following (v, w) in the sorted adjacency list of v, if there is such an edge. If $w \in U$, the algorithm simply relaxed the next edge of v. When $U = V$ the algorithm stops, even if there are still edges left in the heap Q and even if some edges were not examined yet.

The correctness of Spira's algorithm follows easily from the correctness of Dijkstra's algorithm, or can be proved directly using the same ideas used to prove the correctness of Dijkstra's algorithm.

Algorithm `ClusterOrCycle`(u), shown as Algorithm 2, uses the following modification of Spira's algorithm. It starts constructing $CL(u)$. The set $c\ell(u)$ denotes the set of vertices of the cluster discovered so far. Initially $c\ell(u) = \{u\}$. When the first edge (v, w) for which $w \in c\ell(u)$ is extracted from the heap Q, the algorithm stops as a cycle in $c\ell(u)$ is discovered.

A non-trivial complication arises from the fact that we want `ClusterOrCycle`(u) to only examine edges that belong to $CL(u)$. Furthermore, for a correct implementation of Spira's algorithm we need to examine these edges in non-decreasing order of length.

For the high-level description of Algorithm `ClusterOrCycle`(u), we assume that we have a function `Next`(u, v) that given a vertex v already known to be in $CL(u)$ gives us the next incident edge (v, w) of v that leads to a vertex w also in $CL(u)$, in non-decreasing order of length. If there is no such next edge then `Next`(u, v) returns *null*. The implantation of `Next`(u, v) is described in Section 3.1. It is shown there that it can be implemented in $O(\log n)$ time.

`ClusterOrCycle`(u) uses `Next`(u, v) via a function `RelaxNext`(u, v), see Algorithm 3, that uses `Next`(u, v) to extract the next eligible edge e, if there is any, and relax it, i.e., add e to the heap Q with key $d(v) + \ell(e)$.

We end this section with a proof of Lemma 3.1.

Proof. [Proof of Lemma 3.1] `ClusterOrCycle`(u) starts running Spira's algorithm on the implicitly represented cluster graph $CL(u)$. Spira's algorithm extracts the edges (v, w) of $CL(u)$ from the heap Q in non-decreasing order of their key $\delta(u, v) + \ell(v, w)$. When the first edge (v, w) reaching a vertex w of $CL(u)$ is extracted from Q, then $\delta(u, w) = \delta(u, v) + \ell(v, w)$. The distance $d(u, w)$ is set accordingly and w is added to $c\ell(u)$, the set of vertices of the cluster discovered so far. If a second edge (v', w) reaching a vertex w is extracted from Q, then a cycle is detected and Spira's algorithm is aborted.

Let $r > 0$ be the smallest number, as in the statement of the lemma, such that $CL(u) \cap G_r(u)$ contains a cycle. As $CL(u) \cap G_{<r}(u)$ does not contain a cycle, `ClusterOrCycle`(u) finds distances and shortest paths to all vertices of $CL(u) \cap G_{<r}(u)$ before a second edge reaching a vertex is found. The algorithm then starts finding vertices of distance exactly r from u. As $CL(u) \cap G_r(u)$ contains a cycle, at some stage a second edge reaching a vertex in $CL(u) \cap G_r(u)$ must be found and Spira's algorithm is aborted. This edge clearly closes a cycle of length at most $2r$ which is returned by the algorithm, as required.

Spira's algorithm spends $O(\log n)$ time on each edge (v, w) it considers. This includes the $O(\log n)$ time taken by `Next`(u, v) to return the edge, the $O(\log n)$ (or $O(1)$) time needed to insert the edge to the heap Q, and the $O(\log n)$ time needed for extracting it from the heap. The size of the heap is always at most the number of vertices in $c\ell(u)$, i.e., the vertices of the cluster discovered so far. As long as no cycles are found, the number of edges examined by Spira's algorithm is at most $2|c\ell(u)| - 1$: the number of edges extracted from Q is $|c\ell(u)| - 1$ and the number of edges in Q is at most $|c\ell(u)|$. When a cycle is found, the total number of edges examined is at most $2|c\ell(u)|$. The total running time is therefore $O(|c\ell(u)| \log n) = O(|CL(u)| \log n)$, as claimed. ∎

Algorithm 2: `ClusterOrCycle`(u)

```
1  d(u,u) ← 0
2  π(u,u) ← null
3  cℓ(u) ← {u}
4  Q ← Heap()
5  RelaxNext(u,u)
6  while Q ≠ ∅ do
7      (v,w) ← Q.ExtractMin()
8      if w ∈ cℓ(u) then
9          return ⟨ (u,v,w) , d(u,v) + ℓ(v,w) + d(u,w) ⟩
10     d(u,w) ← d(u,v) + ℓ(v,w)
11     π(u,w) ← (v,w)
12     cℓ(u) ← cℓ(u) ∪ {w}
13     RelaxNext(u,v)
14     RelaxNext(u,w)
15 return ⟨ null , ∞ ⟩
```

Algorithm 3: `RelaxNext`(u,v)

```
1  e ← Next(u,v)
2  if e ≠ null then
3      Q.Insert(e, d(u,v) + ℓ(e))
```

3.1 Examining cluster edges in non-decreasing order of length Recall that if $u \in A_i \setminus A_{i+1}$ then $CL(u) = (CL_V(u), CL_E(u))$, where

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

Algorithm `Preprocess`, called by `Initialize`, defines k *shifted lengths* $\ell_i(v,w) = \ell(v,w) - \delta(w, A_{i+1})$ for each edge $(v,w) \in E$. Now, if $u \in A_i \setminus A_{i+1}$ and $v \in CL_V(u)$ then $(v,w) \in CL_E(u)$ if and only if $\ell_i(v,w) < -\delta(u,v)$. We want to iterate over the edges of v that satisfy this condition in increasing order of their original length.

Abstractly, we are faced with the following situation. We have a sequence $e_1, e_2, \ldots, e_n$ of items. (In our concrete situation these are the edges incident on some vertex v and n is the degree of v, where we already know that $v \in CL(u)$ and also have $d(u,v) = \delta(u,v)$.) Each item e has two lengths, $x(e)$ and $y(e)$. (In the concrete case, these are $\ell(e)$ and $\ell_i(e)$, where $u \in A_i \setminus A_{i+1}$.) We are given a bound y_0 and are required to iterate over the items that satisfy $y(e) < y_0$ in non-decreasing order of $x(e)$, until we decide that we do not want to see additional items. (In our case $y(e) = \ell_i(e)$ and $y_0 = -\delta(u,v)$.) We want to produce each item in, say, at most $O(\log n)$ time.

This is closely related to the *2-dimensional orthogonal range reporting* problem. In this problem we are given a collection of n points (x_j, y_j) in the plane. Given four thresholds $a < b$ and $c < d$, we want to return all the points in the box $[a,b] \times [c,d]$, i.e., all the points satisfying $a \le x_j \le b$ and $c \le y_j \le d$. A classical result of Chazelle [Cha86], which improves on a result of Willard [Wil85], says that this can be done in $O(k + \log n)$ time using $O(n \log n / \log\log n)$ space, where k is the number of points returned.

Our problem is slightly easier, on the one hand, as we have only one threshold d. On the other hand, we want to produce the items satisfying $y_j < d$, one by one, in non-decreasing order of their x-coordinate. We are not allowed to first collect all items satisfying $y_j < d$ and then sort them according to their x-coordinates, as we may only want to look at the first few points satisfying the condition, or even just the first.

As we have only one threshold, we can solve our problem using ideas borrowed from the *priority search tree*

of McCreight [McC85]. (These ideas work, in fact, for up to three thresholds.)

We sort the n points according to their x-coordinate and put them at the leaves of a binary tree. (For simplicity we may assume that n is a power of 2.) Each node of the tree contains the minimum y-coordinate among all the items in its subtree. All these values can be easily computed in $O(n)$ time by letting the value of each vertex be the minimum of the values of its two children.

Given an upper bound y_0 we can now easily find the item (x_j, y_j) with the minimum x-coordinate that satisfies $y_j < y_0$. First we check if the minimum y-value of the root is less than y_0. If not, then there is no point satisfying the condition. Then, staring at the root we repeatedly go to the left child, if its minimum y value is less than y_0, and to the right child otherwise. The first item can thus be found in $O(\log n)$ time. Similarly, given an item we can easily find the next item in $O(\log n)$ time. Thus, the first k items, in non-decreasing order of their x-coordinates, can be found in $O(k \log n)$ time.

Agarwal [Aga22] pointed out a more efficient, but slightly more complicated, solution. Insert the points in non-decreasing order of their y-coordinates into a *persistent* red-black tree. (See Sarnak and Tarjan [ST86].) The keys of the points are their x-coordinates. Given a threshold y_0, do a binary search on the y-coordinates to find the appropriate version of the red-black tree and start listing the items in this tree in non-decreasing order of their x-coordinate. Producing the first k points then takes only $O(\log n + k)$ instead of $O(k \log n)$.

Producing each edge in $O(\log n)$ time is enough for our purposes as we spend $\Omega(\log n)$ time on each edge in any case. Thus, Agarwal's elegant idea does not lead to an improved running time of the whole algorithm.

4 Girth approximation algorithm

In this section we prove Theorem 1.1 which we restate for convenience:

Theorem 1.1. *Let $G = (V, E, \ell)$ be a weighted undirected graph, where $\ell : E \to (0, \infty)$. Let g be the girth of G. For every integer $k \geq 1$, there is an algorithm whose expected running time is $O(kn^{1+1/k} \log n + m(k + \log n))$ that finds a cycle C such that $\ell(C) \leq \frac{4k}{3} g$.*

To prove Theorem 1.1 we present an Algorithm `Cycle` that receives as an input a weighted undirected graph $G = (V, E, \ell)$ with girth g and an integer parameter $k \geq 1$ and finds a cycle of length at most $\frac{4k}{3} g$.

Algorithm `Cycle`, see Algorithm 4, works as follows. It starts by calling `Initialize`(G). It then sets α and W to ∞ and $\emptyset$, respectively. Here, α is an upper bound on the length of the smallest cycle found so far and W is a triplet describing this shortest cycle, as explained in Section 2.5.

Next, `Cycle` calls `ClusterOrCycle`(u) for every $u \in V$. The result of `ClusterOrCycle`(u) is the pair $\langle W', \alpha' \rangle$. If $\alpha' < \alpha$ then α and W are updated to be α' and W', respectively. Finally, for every $(v, w) \in E$ and every $0 \leq i \leq k-1$, the algorithm checks whether the edge (v, w) closes a cycle in shortest paths tree of $u = p_i(v)$, and if this cycle is shorter than the shortest cycle found so far. More precisely, the algorithm checks whether $d(u, v)$ and $d(u, w)$ are defined, by two accesses to the hash table d. If they are defined, they correspond to the actual distances $\delta(u, v)$ and $\delta(u, w)$. Otherwise, they are $+\infty$. Next, the algorithm checks that $\pi(u, v) \neq (w, v)$ and $\pi(u, w) \neq (v, w)$. If this condition holds then a cycle is indeed formed and $\alpha' = d(u, v) + \ell(v, w) + d(u, w)$ is an upper bound on its length. If $\alpha' < \alpha$ we update α and W accordingly.

Let C be a shortest cycle in G. We break the correctness proof of `Cycle` into two cases: either $M(C) \leq g/3$ or $M(C) > g/3$. (Recall that $M(C)$ is the length of the longest edge on C.)

We assume at first that $M(C) \leq g/3$. We show that if there is $w \in A_i$ that is relatively close to C then either `Cycle` finds a cycle within the desired bound or there is $w' \in A_{i+1}$ that is relatively close to C.

LEMMA 4.1. *Let C be a cycle in G such that $\ell(C) = g$ and $M(C) \leq g/3$. Let $0 \leq i \leq k-1$. If there exists $w \in A_i$ such that $\delta(w, C) \leq \frac{2i}{3} g$ then either* `Cycle` *finds a cycle of length at most $\frac{4(i+1)}{3} g$, or there exists $w' \in A_{i+1}$ such that $\delta(w', C) \leq \frac{2(i+1)}{3} g$.*

Proof. Let $0 \leq i \leq k-1$ and let $w \in A_i$ such that $\delta(w, C) \leq \frac{2i}{3} g$. If there exists $x \in C$ such that $r_{i+1}(x) \leq \frac{2(i+1)}{3} g$ then there exists $p_{i+1}(x) = w' \in A_{i+1}$ such that $\delta(w', C) \leq \frac{2(i+1)}{3} g$, and the claim holds. Thus, we assume for the rest of the proof that $r_{i+1}(x) > 2(i+1)g/3$, for every vertex $x \in C$.

We show that $C \subseteq CL(w) \cap G_{2(i+1)g/3}(w)$. We first show that $C \subseteq G_{2(i+1)g/3}(w)$. Let $y = \arg\min_{z \in C} \delta(w, z)$. Since $\delta(w, y) \leq 2ig/3$ we have $y \in V_{2ig/3}(w) \cap C \neq \emptyset$ and we can apply Lemma 2.1 with $r = 2ig/3$ and $M(C) \leq g/3$ to get that $C \subseteq G_{2ig/3+g/2+g/6}(w) = G_{2(i+1)g/3}(w)$. By the definition of ball graphs, this implies that $\delta(w, (s, t)) \leq 2(i+1)g/3$, for every edge $(s, t) \in C$.

Algorithm 4: `Cycle`$(G = (V, E, \ell), k)$

```
Initialize(G)
α ← ∞ ; W ← ∅
for u ∈ V do
    ⟨W′, α′⟩ ← ClusterOrCycle(u)
    if α′ < α then
        α ← α′ ; W ← W′
for (v, w) ∈ E do
    for i ← 0 to k − 1 do
        u ← p_i(v) ; α′ ← d(u, v) + ℓ(v, w) + d(u, w)
        if α′ < α and π(u, v) ≠ (w, v) and π(u, w) ≠ (v, w) then
            α ← α′ ; W ← (u, v, w)
return ⟨W, α⟩
```

We now show that $C \subseteq CL(w)$. Recall that we are in the case that for every vertex $x \in C$ we have $r_{i+1}(x) > 2(i+1)g/3$, thus, $\delta(w,(s,t)) \leq 2(i+1)g/3 < r_{i+1}(t)$ and $\delta(w,(s,t)) \leq 2(i+1)g/3 < r_{i+1}(s)$, for every edge $(s,t) \in C$.

By definition we have $\delta(w,(s,t)) = \min\{\delta(w,s), \delta(w,t)\} + \ell(s,t)$. Assume, without loss of generality, that $\delta(w,s) \leq \delta(w,t)$. Thus, $\delta(w,s) + \ell(s,t) = \delta(w,(s,t)) < r_{i+1}(t)$ which implies that $(s,t) \in CL_E(w)$ and $C \subseteq CL(w)$.

It follows from Lemma 3.1 that if $C \subseteq CL(w) \cap G_{2(i+1)g/3}(w)$ then `ClusterOrCycle`(w) finds a cycle of length at most $2 \cdot 2(i+1)g/3 = 4(i+1)g/3$, and the claim follows. ∎

Next, we use Lemma 4.1 and prove:

LEMMA 4.2. *Let C be a cycle in G such that $\ell(C) = g$ and $M(C) \leq g/3$. Let $0 \leq i \leq k-1$. Either* `Cycle` *finds a cycle of length at most $\frac{4(i+1)}{3}g$, or there exists $w' \in A_{i+1}$ such that $\delta(w', C) \leq \frac{2(i+1)}{3}g$.*

Proof. We prove the claim by induction on i. For the base case $i = 0$, thus $A_0 = V$ and $C \cap A_0 \neq \emptyset$. Let $z \in C \cap A_0$. Since $\delta(z,z) = 0$ from Lemma 4.1 we get that `Cycle` either finds a cycle of length at most $4g/3$ or there exists $w \in A_1$ such that $\delta(w, C) \leq 2g/3$, as required.

Next, we assume the claim holds for $i-1$ and prove the claim for i. Since the claim holds for $i-1$ then either `Cycle` finds a cycle of length at most $4ig/3$ and since $4ig/3 \leq 4(i+1)g/3$ the claim holds, or there exists a vertex $w \in A_i$ such that $\delta(w, C) \leq 2ig/3$. In this case it follows from Lemma 4.1 that either `Cycle` finds a cycle of length at most $4(i+1)g/3$ or there exists $w' \in A_{i+1}$ such that $\delta(w', C) \leq 2(i+1)g/3$, as required. ∎

Using Lemma 4.2 it is straightforward to establish the correctness of `Cycle` for the case that $M(C) \leq g/3$.

COROLLARY 4.1. *Let C be a cycle in G such that $\ell(C) = g$ and let $M(C) \leq g/3$.* `Cycle` *finds a cycle of length at most $\frac{4k}{3}g$.*

Proof. When $i = k-1$ we have $A_k = \emptyset$ so there is no w in A_k and from Lemma 4.2 it follows that `Cycle` finds a cycle of length at most $\frac{4k}{3}g$. ∎

Next, we consider the case that $M(C) > g/3$. Let $(u,u') \in C$ such that $\ell(u,u') = M(C)$. We will show that if $\min\{r_i(u), r_i(u')\}$ is relatively small then either `Cycle` finds a cycle of length at most $\frac{4(i+1)}{3}g$ or $\min\{r_{i+1}(u), r_{i+1}(u')\}$ is relatively small.

LEMMA 4.3. *Let C be a cycle in G such that $\ell(C) = g$. Let $M(C) > g/3$, $(u,u') \in C$ and $\ell(u,u') = M(C)$. Let $0 \leq i \leq k-1$. If $\min\{r_i(u), r_i(u')\} \leq i \cdot (g - M(C))$ then either* `Cycle` *finds a cycle of length at most $\frac{4(i+1)}{3}g$ or $\min\{r_{i+1}(u), r_{i+1}(u')\} \leq (i+1) \cdot (g - M(C))$.*

Proof. Let $0 \leq i \leq k-1$ and let $\min\{r_i(u), r_i(u')\} \leq i \cdot (g - M(C))$. If $\min\{r_{i+1}(u), r_{i+1}(u')\} \leq (i+1) \cdot (g - M(C))$ then the claim holds. We can assume, therefore, that let $r_{i+1}(u') > (i+1) \cdot (g - M(C))$ and $r_{i+1}(u) > (i+1) \cdot (g - M(C))$.

Assume, without loss of generality, that $r_i(u) \leq r_i(u')$. Since $\min\{r_i(u), r_i(u')\} \leq i \cdot (g - M(C))$ this implies that $r_i(u) = \delta(p_i(u), u) \leq i(g - M(C))$.

Let r be the smallest number such that $CL(p_i(u)) \cap G_r(p_i(u))$ contains a cycle. If $r \leq (i+1) \cdot (g - M(C))$ then by Lemma 2.2 `ClusterOrCycle`$(p_i(u))$, when called, finds a cycle of length at most $2r \leq 2 \cdot (i+1) \cdot (g - M(C))$. Since $g - M(C) \leq \frac{2}{3}g$ we have $2r \leq \frac{4(i+1)}{3}g$, and the claim holds. Thus, we assume that $r > (i+1) \cdot (g - M(C))$.

Next, we show that $u' \in CL_V(p_i(u)) \cap V_{(i+1)\cdot(g-M(C))}(p_i(u))$. Since $\ell(C) = g$, $(u, u') \in C$ and $\ell(u, u') = M(C)$ we get that $\delta(u, u') = \min\{M(C), g - M(C)\} \leq g - M(C)$. By the triangle inequality, $\delta(p_i(u), u') \leq r_i(u) + d(u, u')$. Combining these two inequalities with our assumption that $r_i(u) \leq i \cdot (g - M(C))$ we get

$$\begin{aligned} \delta(p_i(u), u') &\leq r_i(u) + \delta(u, u') \\ &\leq i \cdot (g - M(C)) + \delta(u, u') \\ &\leq (i+1) \cdot (g - M(C)) \end{aligned}$$

Thus, $u' \in V_{(i+1)\cdot(g-M(C))}(p_i(u))$. Since $(i+1) \cdot (g - M(C)) < r_{i+1}(u')$ we get that $\delta(p_i(u), u') < r_{i+1}(u')$ and thus $u' \in CL_V(p_i(u))$. We conclude that u' is in the graph $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$.

Now since $r > (i+1) \cdot (g - M(C))$ it follows from Lemma 3.1 that `ClusterOrCycle`$(p_i(u))$ computes $d(p_i(u), u') = \delta(p_i(u), u')$ and a shortest paths tree rooted at $p_i(u)$ that contains u'.

Next, we show that when `Cycle` considers the edge (u, u') it holds that $\pi(p_i(u), u) \neq (u', u)$ and $\pi(p_i(u), u') \neq (u, u')$. We first show that $\pi(p_i(u), u) \neq (u', u)$. Assume for the sake of contradiction that $\pi(p_i(u), u) = (u', u)$. This implies that $\delta(p_i(u), u') < \delta(p_i(u), u)$. Since it always holds that $r_i(u') \leq \delta(p_i(u), u')$, we get that $r_i(u') < r_i(u)$, a contradiction to our assumption that $r_i(u) \leq r_i(u')$.

We now show that $\pi(p_i(u), u') \neq (u, u')$. Assume, for the sake of contradiction, that $\pi(p_i(u), u') = (u, u')$. This implies that $\delta(p_i(u), (u, u')) \leq \delta(p_i(u), u') \leq (i+1) \cdot (g - M(C))$, and hence (u, u') is in $G_{(i+1)\cdot(g-M(C))}(p_i(u))$.

Since u' is in $CL(p_i(u))$ it follows from Lemma 2.2 that the shortest path between $p_i(u)$ and u' is in $CL(p_i(u))$, thus its last edge (u, u') is in $CL(p_i(u))$. We conclude that (u, u') is in $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$. Consider a path $C'(u, u')$ between u and u' that uses the edges of $C \setminus \{(u, u')\}$. The length of this path is $g - M(C)$. Let $P(p_i(u), u)$ be a shortest path between $p_i(u)$ and u. The length of this path is $r_i(u) \leq i(g - M(C))$. The concatenation of $P(p_i(u), u)$ with $C'(u, u')$ is path between $p_i(u)$ and u' of length at most $(i+1)(g - M(C))$ and thus the distance between $p_i(u)$ and each of the edges $C \setminus \{(u, u')\}$ is at most $(i+1)(g - M(C))$ which implies the edges of $C \setminus \{(u, u')\}$ are in $G_{(i+1)\cdot(g-M(C))}(p_i(u))$.

Let $(s, t) \in C'(u, u')$ and assume that when going from u to u' on $C'(u, u')$ we first encounter s. Let $C'(t, u')$ be the path from t to u' in C avoiding the edge (u, u'). Next we show that (s, t) satisfies $\delta(p_i(u), s) + \ell(s, t) < r_{i+1}(t)$, and thus in $CL(p_i(u))$.

From the triangle inequality we get that $\delta(p_i(u), s) \leq \delta(p_i(u), u) + g - M(C) - \ell(s, t) - \ell(C'(t, u'))$. Thus, $\delta(p_i(u), s) + \ell(s, t) \leq \delta(p_i(u), u) + g - M(C) - \ell(C'(t, u'))$. Since $r_{i+1}(u') \leq \ell(C'(t, u')) + r_{i+1}(t)$ we get that $\delta(p_i(u), s) + \ell(s, t) \leq i(g - M(C)) + g - M(C) - \ell(C'(t, u')) < r_{i+1}(u') - \ell(C'(t, u')) \leq r_{i+1}(t)$.

We conclude that (s, t) is in $CL(p_i(u))$. Thus, there is a path between $p_i(u)$ and u' in $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$ that does not use the edge (u, u').

We reach a contradiction since there is a path between $p_i(u)$ and u' in $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$ that does not use the edge (u, u') and the edge (u, u') is in $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$, as well. However, $CL(p_i(u)) \cap G_{(i+1)\cdot(g-M(C))}(p_i(u))$ does not contain a cycle. We conclude that the condition in line 10 is true.

Since $\delta(p_i(u), u') \leq (i+1) \cdot (g - M(C))$, $\delta(p_i(u), u) = r_i(u) \leq i \cdot (g - M(C))$, and $\ell(u, u') = M(C)$ we get that:

$$\delta(p_i(u), u) + d(p_i(u), u') + M(C) \leq i \cdot (g - M(C)) + (i+1) \cdot (g - M(C)) + M(C) = 2i(g - M(C)) + g,$$

and algorithm `Cycle` finds a cycle of length at most $2i(g - M(C)) + g \leq \frac{4(i+1)}{3}g$, as required. ∎

Using Lemma 4.3 we show:

LEMMA 4.4. *Let C be a cycle in G such that $\ell(C) = g$. Let $M(C) > g/3$, $(u, u') \in C$ and $\ell(u, u') = M(C)$. Let $0 \leq i \leq k-1$. Either* `Cycle` *finds a cycle of length at most $\frac{4(i+1)}{3}g$ or $\min\{r_{i+1}(u), r_{i+1}(u')\} \leq (i+1) \cdot (g - M(C))$.*

Proof. We prove the claim by induction on i. For the base case we have $i = 0$, thus $A_0 = V$ and $C \cap A_0 \neq \emptyset$. Let $z \in \{u, u'\}$. Since $\delta(z, z) = 0$ we use Lemma 4.3 and get that either `Cycle` finds a cycle of length at most $\frac{4}{3}g$ or $\min\{r_1(u), r_1(u')\} \leq (g - M(C))$.

Next, we assume the claim holds for $i - 1$ and prove the claim for i. Since the claim holds for $i - 1$ then `Cycle` either finds a cycle of length at most $\frac{4i}{3}$ and the claim holds, or $\min\{r_i(u), r_i(u')\} \leq i \cdot (g - M(C))$ and we use Lemma 4.3 and get that `Cycle` either finds a cycle of length at most $\frac{4(i+1)}{3}g$ or $\min\{r_{i+1}(u), r_{i+1}(u')\} \leq (i + 1) \cdot (g - M(C))$, as required. ∎

Using Lemma 4.4 it is straightforward to establish the correctness of `Cycle` for the case that $M(C) > g/3$.

COROLLARY 4.2. *Let C be a cycle in G such that $\ell(C) = g$ and let $M(C) > g/3$. Let $(u, u') \in C$ and $\ell(u, u') = M(C)$.* `Cycle` *finds a cycle of length at most $\frac{4k}{3}g$.*

Proof. Since $A_k = \emptyset$, we get that $r_k(u) = r_k(u') = \infty$, thus, it follows from Lemma 4.4 that `Cycle` finds a cycle of length at most $\frac{4k}{3}g$. ∎

LEMMA 4.5. *The expected running time of* `Cycle` *is $O((m + kn^{1+1/k}) \log n + km)$.*

Proof. From Lemma 2.4 it follows that the call to `Initialize` takes $O((m + kn) \log n)$ time. For every $u \in V$ we call to `ClusterOrCycle`(u). From Lemma 3.1 it follows that the running time of `ClusterOrCycle`(u) is $O(|CL_V(u)| \log n)$. From Lemma 2.3 it follows that $\mathbb{E}[\sum_{u \in V} |CL_V(u)|] = O(kn^{1+1/k})$. Thus, calling to `ClusterOrCycle`$(u)$ for every $v \in V$ takes $O((kn^{1+1/k}) \log n)$ expected running time.

For every $(v, w) \in E$ we iterate over k vertices. The cost of this is $O(km)$. ∎

The proof of Theorem 1.1 follows from Corollary 4.1, Corollary 4.2 and Lemma 4.5.

5 Lower bound for weighted approximation

Here we prove a lower bound for girth computation (Theorem 1.2) under the following oracle model for accessing the edges of an $n = |V|$ vertex graph $G = (V, E, \ell)$. Every vertex $v \in V$ has a counter, initialized at 1. The following queries are allowed:

- For any $j \in \{1, \ldots, n\}$, access the jth vertex v of G and return its degree $\deg(v)$,
- for any $j \in \{1, \ldots, n\}$, access the jth vertex v and return the $c(v)$th edge of v in a predetermined sorted order in terms of non-decreasing edge weights; then increment $c(v)$.

In other words, at any point the algorithm can access the next weighted edge out of any vertex, so that to see the ith edge out of a vertex, the algorithm must have accessed all $i - 1$ edges before it in the sorted order.

We prove:

Theorem 1.2. *Assume that the Girth Conjecture holds for an integer $k \geq 1$. Then for that k and any real value $\tau > 0$, every deterministic algorithm that when run on n-vertex weighted undirected graph accessed using the edge oracle model outlined above computes a cycle C with $\ell(C) \leq (2k + 2 - \tau)g$ (with constant probability), must make at least $\Omega(n^{1+1/k})$ queries.*

To prove Theorem 1.2 we provide a transformation from any unweighted graph of a given girth to a weighted graph where the girth has only increased and there are many vertices of large degree where if their largest incident edge length is sufficiently decreased then the girth is decreased. The number of vertices and the degree in this transformation depend only on the average vertex degree in the original graph. Applying this transformation to a high-girth graph of large average degree, randomly decreasing the length of the longest edge incident to one of these high-degree vertices and slightly perturbing the edge weights yields the distribution of graphs for our lower bound.

In the rest of this section we first provide the transformation in Lemma 5.1 below and then we use it to prove Theorem 1.2. Lemma 5.1 proves more properties about the transformation that are actually needed to prove the lower bound, but we include the proof as it is illustrated and of possible independent interest.

LEMMA 5.1. (WEIGHTED SHORT-CYCLE PLANTING) *For all $\epsilon \in [0, 1)$ if there exists an n_0-vertex m_0-edge unweighted graph of girth $g \in [3, \infty)$ then there exists a n-vertex m-edge weighted graph $G = (V, E, \ell)$ and vertex subset $S \subseteq V$ with the following properties*

- ***sizes**: $n \in [3n_0, 4n_0]$, $|S| \geq n_0$, and $\ell_e \in [\epsilon, g]$ for all $e \in E$,*
- ***girth**: the girth of G is at least g,*
- ***cycle planting**: each vertex in S is incident to exactly one edge of length ϵ, between 1 and 2 edges of length g, and between $\lfloor m_0/(2n_0) \rfloor$ and $\lceil m_0/n_0 \rceil$ edges of length 1. Each length g edge has both endpoints in S and if it is changed to have length 1, then the resulting graph has a cycle of length $1 + 2\epsilon$.*

Proof. Let $G_0 = (V_0, E_0, \ell_0)$ be an n_0-vertex, m_0-edge, unweighted graph ($\ell_0(e) = 1$ for all $e \in e$) of girth g. Further, for all $v \in V_0$ let $\deg(v)$ denote the degree of v in G_0 and let $d_{\text{avg}} := \frac{2m_0}{n_0} = \frac{1}{n_0} \sum_{v \in V_0} \deg(v)$ denote the average degree of the vertices of G_0.

Given G_0 we construct $G = (V, E, \ell)$ and S from it as follows. Informally, G is the result of replacing every vertex v in G_0 with a star connecting $\lceil \frac{2\deg(v)}{d_{\text{avg}}} \rceil$ vertices with edges of length ϵ and then dividing the edges of the original graph evenly over these new vertices (see Figure 1 below for a picture).

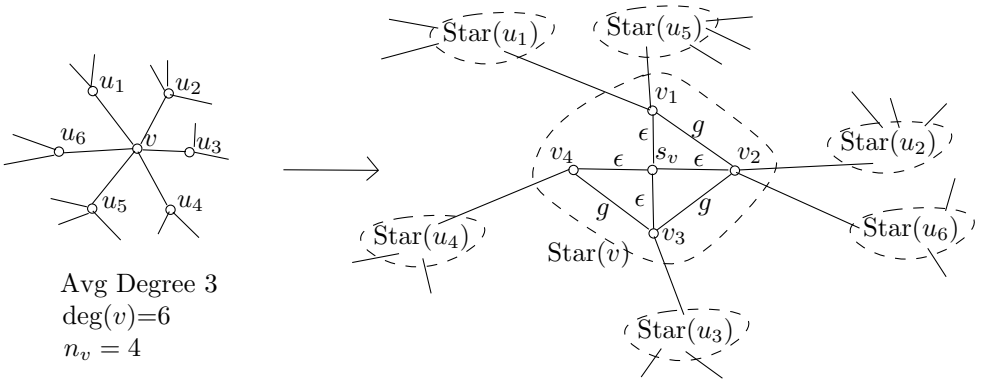

Figure 1: An example of the lower bound construction. Here each vertex v of G_0 gets replaced by a construction $\text{Star}(v)$ and each edge (v, v') of G_0 is now between $\text{Star}(v)$ and $\text{Star}(v')$ as shown.

Formally, to construct G and S we start with $G = G_0$ and $S = \emptyset$ and then apply the following procedure for every $v \in V_0$ one at a time. First, we create new vertices $v_1, \ldots, v_{n_v}$ for $n_v := \lceil \frac{2\deg(v)}{d_{\text{avg}}} \rceil$ as well as a vertex s_v. Then for the edges $(v, u_1), \ldots, (v, u_{\deg(v)})$ incident to v we replace them with the edges $(v_{(1 \bmod n_v)+1}, u_1), \ldots, (v_{(\deg(v) \bmod n_v)+1}, u_{\deg(v)}) \in E_0$ each of length 1. After this, we delete the previous edges and v. Further, we add an edge (s_v, v_i) of length ϵ for all $i \in [n_v]$. Finally, for all $i \in [n_v - 1]$ we add an edge of length g between v_i and v_{i+1} (note that if $n_v = 1$, then we add no such edges of length g) and if $n_v > 1$ then $v_1, \ldots, v_{n_v}$ is added to S.

In the remainder of the proof we show that G and S have the desired properties:

sizes: For every vertex $v \in V_0$ we add $n_v + 1$ vertices to G. Consequently,

$$n = \sum_{v \in V_0} (n_v + 1) = n_0 + \sum_{v \in V_0} \left\lceil \frac{2\deg(v)}{d_{\text{avg}}} \right\rceil .$$

Further, note that $\sum_{v \in V} n_v \in [2n_0, 3n_0]$ as $\sum_{v \in V_0} \frac{2 \cdot \deg(v)}{d_{\text{avg}}} = \frac{4m_0}{d_{\text{avg}}} = 2n_0$. Consequently, $n \in [3n_0, 4n_0]$ and $|S| \geq [\sum_{v \in V_0} n_v - 1] \geq 2n_0 - n_0 = n_0$. Finally $\ell_e \in [\epsilon, g]$ for all $e \in E$ by construction.

girth: Let C be a cycle in G and let $u_1, \ldots, u_c$ denote the vertices of the cycle in order. Further, let $v^{(1)}, \ldots, v^{(c)}$ denote their associated vertices in V_0 (i.e. $u_i = v^{(i)}_{j_i}$ or $s_{v^{(i)}}$ for all $i \in [c]$ for some j_i). By construction of G for all $i \in [c]$ either $(v^{(i)}, v^{((i \bmod c)+1)}) \in E$ or $v^{(i)} = v^{((i \bmod c)+1)}$. Consequently, if we simply remove vertex duplication in $v^{(1)}, \ldots, v^{(c)}$ then the resulting vertex subsequence is either a cycle in G_0, in which case $\ell(C) \geq g$, or the sequence has exactly 1 vertex, i.e. all the $v^{(i)}$ are the same (note that there cannot be

only 2 distinct $v^{(i)}$ since an edge in E_0 would then have to be used twice). However, if all the $v^{(i)}$ are the same, then the cycle must be among the vertices $v_1, \ldots, v_{n_v}$ and s_v for some $v \in V_0$. Since the edges among these vertices of length ϵ is acyclic the cycle must use one of the edges of length g and therefore, in all cases $\ell(C) \geq g$.

cycle planting: Note that by construction every vertex in S is v_i for some $v \in V_0$ and $i \in [n_v - 1]$. Further, by construction v_i is incident to one edge of length ϵ, between 1 and 2 edges of length g, and some number of length 1. Further, if the edge of length g is given length 1 then v_i, v_{i+1}, s_v yields a cycle of length $1 + 2\epsilon$. Consequently, it only remains to bound the number of edges incident to v_i of length 1. However, the number of such edges is either $\lfloor \deg(v)/n_v \rfloor$ or $\lceil \deg(v)/n_v \rceil$. Further, since $n_v > 1$ (as $v_i \in S$) this implies that $1 < \frac{2\deg(v)}{d_{\text{avg}}}$ and the result then follows as

$$\left\lfloor \frac{\deg(v)}{n_v} \right\rfloor \geq \left\lfloor \frac{\deg(v)}{\left(\frac{2\deg(v)}{d_{\text{avg}}}\right) + 1} \right\rfloor > \left\lfloor \frac{\deg(v)}{\left(\frac{4\deg(v)}{d_{\text{avg}}}\right)} \right\rfloor = \left\lfloor \frac{m_0}{2n_0} \right\rfloor \text{ and}$$

$$\left\lceil \frac{\deg(v)}{n_v} \right\rceil \leq \left\lceil \frac{\deg(v)}{\left(\frac{2\deg(v)}{d_{\text{avg}}}\right)} \right\rceil = \left\lceil \frac{d_{\text{avg}}}{2} \right\rceil = \left\lceil \frac{m_0}{n_0} \right\rceil .$$

□

Proof. [Proof of Theorem 1.2] Let $k \geq 1$ be a fixed integer, and let $\tau \in (0, 1)$ be given. Select a Girth Conjecture graph G_k with n_0 vertices and $m_0 = \Theta(n_0^{1+1/k})$ edges and girth $2k + 2$. Further, apply Lemma 5.1 to G_k and any $\epsilon < \tau/(2(2k - 2 - \tau))$, obtaining a graph G. We know that no matter which edge of weight g we pick, if we change its weight to 1, the girth goes from $\geq g$ to $1 + 2\epsilon$. Recall also that G contains a vertex set $S \subseteq V$ with $|S| \geq n_0$ such that the number of edges incident to each vertex of S is between $2 + \lfloor m_0/(2n_0) \rfloor$ and $3 + \lceil m_0/n_0 \rceil$, at most 2 of which are of weight > 1.

Leveraging G_k, G, and the properties of G given Lemma 5.1 we prove our lower bound below:

Consider any deterministic algorithm A for girth approximation running on G. Suppose that A accesses $< \frac{n_0}{4} \lfloor m_0/(2n_0) \rfloor$ edges and let S_q denote the subset of S consisting of vertices which the algorithm queried at least $\lfloor m_0/(2n_0) \rfloor$ times. Note that $|S_q| < \frac{n_0}{4}$. Further, every vertex in S_q is incident to at least $\lfloor m_0/(2n_0) \rfloor$ edges of length 1 and at most 2 edges of length $g > 1$ and all edges of length g have both endpoints in S. Consequently, at most $2|S_q| < \frac{n_0}{2}$ edges of length g are accessed via queries. However, there are at least $|S|/2 \geq n_0/2$ edges of length g in the graph. Therefore, at least one of the edges of length g is not accessed and A would perform the same when run on G and when run on G for which one of the weight g edges incident to s were changed to have any length ≥ 1.

Thus A will fail to distinguish between girth $1 + 2\epsilon$ and girth $\geq 2k + 2$. Further, if the algorithm outputs a cycle containing the edge of length g that was not accessed then its length could be changed to be arbitrarily large and the accesses would be consistent. On the other hand if the algorithm does not output a cycle containing this edge of length g its length could be changed to have length 1 and the cycle will have length $\geq 2k + 2$ although the girth is $1 + 2\epsilon$. Consequently, in the worst case the ratio of the girth to the length of the cycle output is at least $(2k + 2)/(1 + 2\epsilon)$ Since $\epsilon < \tau/(2(2k - 2 - \tau))$, we have that $-\tau(1 + 2\epsilon) + 2\epsilon(2k - 2) < 0$ and $(2k-2-\tau)(1+2\epsilon) < (2k-2)$. Thus any deterministic algorithm that makes fewer than $\frac{n_0}{4} \lfloor m_0/(2n_0) \rfloor = \Theta(n_0^{1+1/k})$ queries, will not be able to compute a cycle C with $\ell(C) \leq (2k + 2 - \tau)g$ on some input. □

Acknowledgement

Thank you to the anonymous reviewers for the helpful comments and suggestions and their thoughtful review of an earlier version of this paper. Also thank you to Pankaj Agarwal, Haim Kaplan and Micha Sharir for information on the 2-dimensional range searching problem.

References

[Aga22] Pankaj Agarwal. Personal communication, 2022.

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

[Cha86] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DKS17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. New subquadratic approximation algorithms for the girth. *CoRR*, abs/1704.02178, 2017.

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

[EK10] D. Easley and J. Kleinberg. *Networks, crowds, and markets: reasoning about a highly connected world.* Cambridge Univ Press, Cornell, NY, 2010.

[Erd64] Paul Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague, 1964.

[HW16] 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.

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

[KRS+22] Avi Kadria, Liam Roditty, Aaron Sidford, Virginia Vassilevska Williams, and Uri Zwick. Algorithmic trade-offs for girth approximation in undirected graphs. In *SODA*, pages 1471–1492. SIAM, 2022.

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

[LVW18] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018.

[LW97] Felix Lazebnik and Ping Wang. On the structure of extremal graphs of high girth. *Journal of Graph Theory*, 26(3):147–153, 1997.

[McC85] Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[OPT01] 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.

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

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

[RV11] 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.

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

[Spi73] 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.

[ST86] Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

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

[Wil85] Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985.

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

[WZ15] David Bruce Wilson and Uri Zwick. A forward-backward single-source shortest paths algorithm. *SIAM J. Comput.*, 44(3):698–739, 2015.

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