Constant Girth Approximation for Directed Graphs in Subquadratic Time[*]

Shiri Chechik
Tel Aviv University
Tel Aviv, Israel
shiri.chechik@gmail.com

Yang P. Liu
Stanford University
Stanford, California, USA
yangpliu@stanford.edu

Omer Rotem
Tel Aviv University
Tel Aviv, Israel
omer.rotem1@gmail.com

Aaron Sidford
Stanford University
Stanford, California, USA
sidford@stanford.edu

ABSTRACT

In this paper we provide a $\tilde{O}(m\sqrt{n})$ time algorithm that computes a 3-multiplicative approximation of the girth of a n-node m-edge directed graph with non-negative edge lengths. This is the first algorithm which approximates the girth of a directed graph up to a constant multiplicative factor faster than All-Pairs Shortest Paths (APSP) time, i.e. $O(mn)$. Additionally, for any integer $k \geq 1$, we provide a deterministic algorithm for a $O(k \log \log n)$-multiplicative approximation to the girth in directed graphs in $\tilde{O}(m^{1+1/k})$ time. Combining the techniques from these two results gives us an algorithm for a $O(k \log k)$-multiplicative approximation to the girth in directed graphs in $\tilde{O}(m^{1+1/k})$ time. Our results naturally also provide algorithms for improved constructions of roundtrip spanners, the analog of spanners in directed graphs.

The previous fastest algorithms for these problems either ran in All-Pairs Shortest Paths (APSP) time, i.e. $O(mn)$, or were due Pachocki *et al.* [16] which provided a randomized algorithm that for any integer $k \geq 1$ in time $\tilde{O}(m^{1+1/k})$ computed with high probability a $O(k \log n)$ multiplicative approximation of the girth. Our first algorithm constitutes the first sub-APSP-time algorithm for approximating the girth to constant accuracy, our second removes the need for randomness and improves the approximation factor in Pachocki *et al.* [16], and our third is the first time versus quality trade-off for obtaining constant approximations.

CCS CONCEPTS

• **Theory of computation → Sparsification and spanners**.

KEYWORDS

Graphs, girth, spanners

[*]Several results and proofs will be deferred to the full version of the paper.

1 INTRODUCTION

The *girth* of a graph G is the length of the shortest cycle in G. It is an important graph quantity that has been studied extensively in both combinatorial settings (see Bollobás's book [3] for a discussion) and computational settings. In particular, exact algorithms for the girth running in time $O(mn)$ in weighted directed graphs [15] are known. On the other hand, a result of Vassilevska W. and Williams show that a truly subcubic algorithm for girth (i.e. running in time $n^{3-\varepsilon}$ for some $\varepsilon > 0$) implies a truly subcubic algorithm for the All Pairs Shortest Path (APSP) problem [22]. As it is a longstanding open problem whether APSP admits a truly subcubic time algorithm, exact computation of the girth in truly subcubic time would be a major breakthrough.

This has motivated the study of efficient *approximation* algorithms for the girth. There has been extensive work on approximating the girth in *undirected* graphs [9, 11, 13, 20]. Many such algorithms use the concept of a α-*spanner* of a graph G, a fundamental combinatorial object which was introduced by Chew [4]. An α-spanner of a graph G is a subgraph of G which multiplicatively preserves distances up to a factor of α. It is well-known that $(2k-1)$-spanners with $O(n^{1+1/k})$ edges exist for any undirected weighted graph [1], and work on the efficient construction of such spanners [2, 19, 21] implies a $O(mn^{\frac{1}{k}})$ time algorithm for $(2k-1)$-multiplicative girth approximation in undirected graphs. There has also been work on improved spanner constructions in the case of undirected unweighted graphs [13, 20], and these algorithms also immediately imply algorithms for girth approximation in undirected unweighted graphs.

Therefore, in order to obtain efficient constant factor girth approximations in directed graphs, it is natural to study an analog of spanners in directed graphs. Unfortunately, approximately computing all pairs distances in directed graphs is a notoriously difficult problem and while sparse spanners do exist in all undirected graphs, they do not exist in all directed graphs. For example, any directed spanner for the "directed" complete bipartite graph with n vertices on the left directed towards n vertices on the right clearly requires all n^2 edges. This problem seems to arise from the fact that the distance metric $d(u, v)$ in directed graphs is asymmetric. Therefore, if we want to construct sparse spanners, it is natural to work instead with the symmetric *roundtrip distance* metric, defined as $d(u \leftrightarrows v) := d(u, v) + d(v, u)$ [8] and similarly define an α-*roundtrip spanner* of a directed graph G to be a subgraph that multiplicatively preserves roundtrip distances up to a factor of α.

Interestingly, there do exist roundtrip spanners for directed graphs with comparable sparsity as spanners for undirected graphs. A result of Roditty, Thorup, and Zwick [18] shows that for any $k \geq 1$ and $\varepsilon > 0$, every graph has a $(2k + \varepsilon)$-roundtrip spanner

with $O(k^2 n^{1+1/k} \log(nW)\varepsilon^{-1})$ edges, where W is the maximum edge weight. Unforunately, this algorithm ran in time $\Omega(mn)$, as it requires the computation of all pairs distances in the graph. Recent work Pachocki *et al.* [16] gave a randomized algorithm running in time $\tilde{O}(m^{1+1/k})$ which on weighted directed graphs G returns a $O(k \log n)$-roundtrip spanner with $\tilde{O}(n^{1+1/k})$ edges and an $O(k \log n)$ approximation to the girth. Up to a logarithmic approximation factor, this matches the sparsity and runtime known for spanners on undirected weighted graphs and girth on sparse graphs.

The result of Pachocki *et al.* [16] constitutes one of small, but rapidly growing [6, 7], set of instances where it is possible to obtain robust nearly linear time approximations to fundamental quantities of directed graphs in nearly linear time, overcoming typical running time gaps between solving problems on directed and undirected graphs.

However, a fundamental open problem left open by this work is whether it is possible to achieve subquadratic algorithms for constant factor approximation of the girth in directed graphs, and more ambitiously to fully close this gap and provide algorithms for $O(k)$ girth approximation and $O(k)$ roundtrip spanners in directed graphs that fully match the runtime and sparsity of those in undirected graphs. This is the primary problem this paper seeks to address and this paper provides multiple new girth approximation algorithms with improved runtime, approximation quality, and dependency on randomness.

1.1 Our Results

In this paper we provide a subquadratic algorithm for constant factor girth approximation in directed graphs and in turn show several improvements on the girth approximation algorithms and roundtrip spanner constructions in the work of Pachocki *et al.* [16]. Here and throughout the remainder of the paper we use $\tilde{O}(\cdot)$ notation to hide factors polylogarithmic in n, where n is the number of vertices in the graph.

In Section 3 we consider obtaining constant approximations to the girth. In particular we provide a randomized algorithm that obtains a 3-approximation to the girth on graphs with non-negative integer edge weights in $\tilde{O}(m\sqrt{n})$ time. Up to logarithmic factors this matches the runtime that would be predicted from the fact that $(2k - 1)$-undirected spanners with $\tilde{O}(n^{1+1/k})$ edges can be constructed in $\tilde{O}(mn^{1/k})$ time for $k = 2$. Further, we show that this procedure can be used to with high probability obtain constant multiplicative roundtrip spanners in directed graphs with arbitrary edge weights in $\tilde{O}(m\sqrt{n})$ time.

THEOREM 1 (3-MULTIPLICATIVE GIRTH APPROXIMATION). *For any directed graph G with n vertices, m edges, integer non-negative edge weights, and unknown girth g we can compute in $\tilde{O}(m\sqrt{n})$ time an estimate g' such that $g \leq g' \leq 3g$ with high probability in n.*

THEOREM 2 (8-MULTIPLICATIVE ROUNDTRIP SPANNERS). *For any directed graph G with n vertices, m edges, integer non-negative edge weights, we can compute in $\tilde{O}(m\sqrt{n})$ time an 8-multiplicative roundtrip spanner with $\tilde{O}(n^{3/2})$ edges with high probability in n.*

Then, in Section 4 we give algorithms for a $O(k \log \log n)$-multiplicative approximation of the the girth and

construct $O(k \log \log n)$ multiplicative roundtrip spanners with $\tilde{O}(n^{1+1/k})$ edges for a weighted directed graph G with n vertices and m edges in $\tilde{O}(m^{1+1/k})$ time. These algorithms are deterministic and constitute the first deterministic nearly linear time algorithms for $\tilde{O}(1)$ multiplicative approximation of the girth and $\tilde{O}(1)$ multiplicative roundtrip spanners with $\tilde{O}(n)$ edges.

THEOREM 3 (DETERMINISTIC MULTIPLICATIVE GIRTH APPROXIMATION). *For any integer $k \geq 1$ and weighted directed graph G with n vertices, m edges, and unknown girth g we can compute in $\tilde{O}(m^{1+1/k})$ time an estimate g' such that $g \leq g' \leq O(k \log \log n) \cdot g$.*

THEOREM 4 (DETERMINISTIC MULTIPLICATIVE ROUNDTRIP SPANNERS). *For any integer $k \geq 1$ and any weighted directed graph G with n vertices and m edges, we can compute in $\tilde{O}(m^{1+1/k})$ time an $O(k \log \log n)$ multiplicative roundtrip spanner with $\tilde{O}(n^{1+1/k})$ edges.*

Setting $k = \frac{\log n}{\log \log n}$ yields the following corollaries. For $k = \Omega(\log n)$ these results nearly match the optimal algorithms in undirected graphs for $O(k)$ girth approximation and the construction of $O(k)$ spanners.

COROLLARY 1.1. *For any weighted directed graph G with n vertices, m edges, and unknown girth g we can compute in $\tilde{O}(m)$ time an estimate g' such that $g \leq g' \leq O(\log n) \cdot g$.*

COROLLARY 1.2. *For any weighted directed graph G with n vertices and m edges, we can compute in $\tilde{O}(m)$ time an $O(\log n)$ multiplicative roundtrip spanner with $\tilde{O}(n)$ edges.*

Interestingly, our results for constant factor randomized approximations and our results for deterministic approximations are achieved in different ways. Highlighting this, in Section 5 we show how to combine the techniques of these algorithms to obtain both $O(k \log k)$ multiplicative approximations to the girth and $O(k \log k)$ multiplicative roundtrip spanners of size $\tilde{O}(n^{1+1/k})$ in $\tilde{O}(mn^{1/k})$ time with high probability in n.

THEOREM 5 (CONSTANT MULTIPLICATIVE GIRTH APPROXIMATION). *For any integer $k \geq 1$ and any weighted directed graph G with n vertices, m edges, and unknown girth g we can compute in $\tilde{O}(m^{1+1/k})$ time an estimate g' such that $g \leq g' \leq O(k \log k) \cdot g$ with high probability in n.*

THEOREM 6 (CONSTANT MULTIPLICATIVE ROUNDTRIP SPANNERS). *For any integer $k \geq 1$ and any weighted directed graph G with n vertices and m edges, we can compute in $\tilde{O}(m^{1+1/k})$ time an $O(k \log k)$ multiplicative roundtrip spanner with $\tilde{O}(n^{1+1/k})$ edges with high probability in n.*

This shows that for any fixed $\varepsilon > 0$, that there is an algorithm running in time $m^{1+\varepsilon}$ that approximates the girth of a directed graph to within a constant depending on ε, but not on m or n. Additionally, this almost matches the $O(k)$-multiplicative girth approximation algorithms running in $m^{1+1/k}$ time in undirected graphs.

1.2 Comparison to previous work

While the existence of roundtrip spanners matching the quality in undirected graphs was shown in [18], the runtime was $O(mn)$ and

required an APSP computation. Our results, Theorem 1, Theorem 2 are the first to show that constant factor girth approximation and construction of constant factor roundtrip spanners with $\tilde{O}(n\sqrt{n})$ edges can be built in subquadratic $\tilde{O}(m\sqrt{n})$ time. This algorithm leverages new randomized techniques for testing a notion we call *similarity* between vertices not present in previous girth approximation and roundtrip spanner algorithms and we believe is of independent interest.

Our Theorem 3 and Theorem 4 offer direct improvements over the analogous results in [16]. Specifically, our algorithms provide a tighter multiplicative girth approximation and multiplicative spanner stretch in the same runtime as the algorithms in [16], which produce a $O(k \log n)$ girth approximation and $O(k \log n)$ roundtrip spanner with $\tilde{O}(n^{1+1/k})$ edges in time $\tilde{O}(m^{1+1/k})$.

Additionally, our algorithm is deterministic and in our opinion, simpler. The algorithm of Pachocki *et al.* [16] involved the following pieces. First, they use a method of Cohen to estimate ball sizes [5] and resolve the case where there is a vertex whose inball and outball (of some small radius) intersect in a significant fraction of the vertices. In the other case, they use exponential clustering (see [14]) to partition the graph and recurse. Finally, they rerun the algorithm $n^{1/k}$ times. On the other hand, our algorithm simply grows inballs and outballs from various vertices, and uses a delicate cutting condition to decide when to cut and recurse.

Additionally, Theorem 5 and Theorem 6 further improve upon Pachocki *et al.* [16] by combining the ideas from the constant factor girth approximation algorithm and the deterministic ball-growing algorithm, completely removing the dependence on n in the approximation factor while still running in time $\tilde{O}(m^{1+1/k})$. We remark that the ideas for our deterministic algorithm are essential in obtaining this last result, and that more directly combining the ideas of [16] with our constant factor approximation algorithm does not seem to give an $O(k \log k)$ multiplicative girth approximation in $\tilde{O}(m^{1+1/k})$ time.

1.3 Overview of Approach

Summary of randomized $O(1)$ approach. Our approach to obtaining a 3-approximation in Section 3 to the girth is rooted in the simple insight that if a vertex v is in a cycle of length R then every vertex in the ball of radius α from v is at distance at most $\alpha + R$ from every vertex in the cycle. Consequently, for each vertex if we repeatedly prune vertices from its outball of radius R if they do not have the property that they can reach every vertex in this ball by traversing a distance at most $2R$, then we will never prune away vertices in a cycle of length R from that vertex.

Leveraging these insights, we can show that if we randomly compute distances to and from a random $\tilde{O}(\sqrt{n})$ vertices and if a cycle of length $O(R)$ is not discovered immediately then we can efficiently implement a pruning procedure so that each vertex only has in expectation $\tilde{O}(\sqrt{n})$ vertices that could possibly be in a cycle of length $O(R)$ through that vertex. By then checking each of these sets for a cycle and being careful about the degrees of the vertices (and therefore the cost of the algorithm) this approach yields essentially a 4-approximation to the girth in $\tilde{O}(m\sqrt{n})$ time with high probability in n.

Our 3-approximation is then obtained by carefully applying this argument to both outballs and inballs and leveraging the simple fact that if a vertex v is on a cycle C of length R then for every $c \in C$ either $d(v, c) \leq R/2$ or $d(c, v) \leq R/2$.

Overview of deterministic $O(k \log \log n)$ results: Our deterministic algorithm in Section 4 is based on a different approach than our randomized constant approximation algorithms in Section 3. We think this approach is of independent interest and further demonstrate its utility in Section 5 by showing how to combine the insights that underly it with the algorithm from Section 4 to achieve arbitrary constant approximations.

For the sake of simplicity, we focus on unweighted directed graphs G and for a parameter R, construct a subgraph (roundtrip spanner) H so that if the roundtrip distance between u and v is at most R in G, then their roundtrip distance is at most $O(Rk \log \log n)$ in H.

The key insight of guiding our algorithm is the following: instead of partitioning the graph into disjoint pieces and recursing (as is done in [16]), we instead allow the pieces to overlap on the boundaries. This is justified by the following observation. Consider a subgraph W of G, and let W' denote the subgraph consisting of all vertices within distance R of W. Then if we recursively build a roundtrip spanner on W', then we are guaranteed that we can delete W from our graph. Indeed, if $u \in W$ and the roundtrip distance between u and v is at most R, then $u, v \in W'$. This simple observation allows us to overcome the critical challenge in [16], arguing that that graph can be broken apart, while nevertheless preserving roundtrip distance.

This observation also forms the basis of an optimal spanner construction on unweighted undirected graphs, which appears in a book of Peleg (exercise 3 on page 188 in [17]). Specifically, for any integer $k \geq 1$, we can construct a $(2k - 1)$-spanner with $O(n^{1+1/k})$ edges in time $O(m)$. The construction works as follows. Start at any vertex v, let B_i denote the ball of radius i centered at v, and let $|B_i|$ denote the number of vertices in B_i. Grow such balls around v until we find an index i with $|B_{i+1}| \leq n^{1/k}|B_i|$. We can clearly guarantee that $i \leq k$. At this point, add a spanning tree on B_{i+1} to your spanner and delete all vertices in B_i. Now, recurse on the remaining graph. It is easy to check that the resulting spanner is as desired. Our algorithm for directed graphs is similar, and we give a more specific overview in Section 4.1.

We gain an $O(\log \log n)$ dependence over the undirected spanner algorithm presented because we must recurse on the balls we grew instead of simply building a spanning tree on them. The precise condition for recursion and corresponding calculation are performed in the algorithms GOODCUT (Algorithm 4) and Lemma 4.2.

Further, our $O(k \log k)$ approximations of Section 5 are then achieved by using the techniques of the algorithms in Section 3 to better control the size of the outballs and inballs in an invocation of the deterministic algorithm of Section 4.

2 PRELIMINARIES

For weighted directed graph G, we let $V(G)$ and $E(G)$ denote the vertex and edge sets of G. We assume all edge lengths are nonnegative. For a subgraph $S \subseteq G$ (not necessarily vertex induced), let $V(S)$ denote the set of vertices of G, and let $E(S)$ denote the set

of edges. For a subset $W \subseteq V(G)$, we define $G[W]$ to be subgraph induced by W. When the graph G is clear from context, we let n and m denote $|V(G)|$ and $|E(G)|$ respectively.

For a weighted directed graph G with non-negative edge lengths, we let $d_G(u, v)$ denote the (shortest path) distance from u to v in G. When the graph G is clear from context, we simply denote this as $d(u, v)$. If there is no path from u to v, we let $d(u, v) = \infty$. When S is a subgraph of G, we let $d_S(u, v)$ denote the (shortest path) distance from u to v only using the edges in $E(S)$. We denote the *roundtrip distance* between u and v as $d_G(u \leftrightarrows v) := d_G(u, v) + d_G(v, u)$ and define a roundtrip spanner.

DEFINITION 2.1 (ROUNDTRIP SPANNER). *We say that a subgraph $S \subseteq G$ is an α-roundtrip spanner if $d_S(u \leftrightarrows v) \leq \alpha \cdot d_G(u \leftrightarrows v)$ for all $u, v \in V(G)$.*

For weighted directed graph G we define the *inball* and *outball* of radius r around a vertex v as

$$B_v^{\text{in}}(r) := G[\{u : d(u, v) \leq r\}] \text{ and } B_v^{\text{out}}(r) := G[\{u : d(v, u) \leq r\}]$$

respectively. In other words, the inball of radius r around v is the subgraph induced by vertices u with $d(u, v) \leq r$. The outball is defined similarly. We define the *ball* of radius r around vertex v as

$$B_v(r) := G[\{u : d(u \leftrightarrows v) \leq r\}].$$

In other words, the ball of radius r around v is the subgraph induced by vertices u within roundtrip distance r of v.

3 RANDOMIZED CONSTANT APPROXIMATIONS

Here we provide algorithms for efficiently computing a 3-approximation to the girth Section 3.1. To simplify our algorithm and analysis we assume that the maximum degree of G is bounded by $O(m/n)$, i.e. we assume it is only a constant larger than the average degree, which is $2m/n$. We justify this assumption by showing that we can always reduce to this case as is formalized in the following lemma.

LEMMA 3.1. *Given a directed weighted graph $G = (V, E)$ of n vertices and m edges with non negative edge weights, one can construct a graph H in $O(m)$ time of $O(n)$ vertices and $O(m)$ edges with non negative edge weights and of maximum degree $O(m/n)$ such that*

(1) *All roundtrip distances (between pairs of vertices in G) in H and in G are the same.*

(2) *Given a cycle in H, one can find in $O(m)$ time a cycle in G of the same length.*

(3) *Given a subgraph H' of H, one can find in $O(m)$ time a subgraph G' of G such that the number of edges in G' is at most the number of edges in H' and the roundtrip distances in H' and G' are the same.*

3.1 An $\tilde{O}(m\sqrt{n})$ Time 3-approximation to Girth

In this section we show a procedure that given a directed weighted graph G and a girth estimate R, returns a cycle of length at most $3R$ if the girth in G is at most R. The algorithm is given by GIRTHAPPROX (See Algorithm 1) which in turn invokes the subroutine SIMILARSET (See Algorithm 2).

In order to approximate the girth of G we invoke this procedure for every $r = (1 + \epsilon)^i$ for $1 \leq i \leq \log_{1+\epsilon} nW$ and stop once the procedure returns a cycle. If g is the girth of G this incurs an additional $\log_{1+\epsilon} g$ factor to the running time (as for the first index i such that $(1 + \epsilon)^i > g$ the algorithm will return a cycle w.h.p.) and an additional $(1 + \epsilon)$ factor in the approximation ratio. The additional $(1 + \epsilon)$ factor in the approximation ratio can be avoided if the weights are integers by simply using binary search on the range between 1 and nW (where W is the maximum edge weight in G) and finding two consecutive integers i and $i + 1$ such that the procedure returned a cycle of length at most $3(i + 1)$ when invoked on $i + 1$ but not a cycle when invoked on i. This incurs a $\log nW$ factor in the running time that can be improved to $O(\log n)$ by the same method as done in [16] of contracting small weight strongly connected components and deleting large weight edges (see Section 5.1 in [16] for more details).

Let $G = (V, E)$ be a directed graph with n vertices and m edges. We assume the graph G is of average degree $\delta = 2m/n$ and that also the maximum degree in the graph is also $O(\delta)$.

The subroutine SIMILARSET gets as an input the graph G and the target distance R and either returns a cycle of length at most $3R$ or returns a subset A_v of vertices for every $v \in V$. The subset A_v for a vertex $v \in V$ consists of vertices at distance at most $R/2$ from v with the guarantee that A_v contains all vertices that are

(1) At distance at most $R/2$ from v and

(2) On a cycle of length R with v.

Procedure GIRTHAPPROX invokes the Procedure SIMILARSET twice, once on G and once on the reversed graph of G (the graph obtained by reversing every edge of G). If a cycle of length $3R$ is returned in one of these calls then procedure GIRTHAPPROX returns such a cycle. Otherwise, let $\{A_v^{\text{in}}\}_{v \in V}$ be the sets returned from invoking SIMILARSET on the graph G and $\{A_v^{\text{out}}\}_{v \in V}$ on the reversed graph. Next, the procedure for every $v \in V$ checks if there is a cycle containing v of length at most R in the induced graph of $A_v^{\text{in}} \cup A_v^{\text{out}}$. If such a cycle exists then the procedure returns such a cycle.

Procedure SIMILARSET works as follows. The algorithms starts by sampling $O(\log n)$ independent subsets S_i of expected size $O(\sqrt{n})$ each for $1 \leq i \leq M$ where $M = 50 \log n$. From every vertex $w \in \bigcup_{1 \leq i \leq M} S_i$ the algorithm runs Dijkstra from and to w in G. If a cycle of length $3R$ is detected then the algorithm returns it.

Next for every vertex $v \in V$ and index $1 \leq i \leq M$ the algorithm defines a set $T_i(v) \subseteq S_i$. The sets $T_i(v)$ will be used to reduce the number of potential vertices that can be on a cycle of length at most R with v. First, the set $T_0(v)$ consists of all vertices in S_0 that are at distance at most $R/2$ from v. Let $R_0(v)$ be a sampled set of $O(\log n)$ vertices from $T_0(v)$. Now, the sets $T_i(v)$ and $R_i(v)$ are defined as follows. The set $T_i(v)$ is the set of all vertices $s \in S_i$ such that $d(v, s) \leq R/2$ and $d(s, t) \leq 3R/2$ for all $t \in \bigcup_{0 \leq j \leq i-1} R_j(v)$. Again, define $R_i(v)$ as a sampled set of $O(\log n)$ vertices from $T_i(v)$.

To gain intuition for the definition of $T_i(v)$ and $R_i(v)$, consider the set $G_i(v)$ of all vertices $s \in V(G)$ such that $d(v, s) \leq R/2$ and $d(s, t) \leq 3R/2$ for all $t \in \bigcup_{0 \leq j \leq i-1} R_j(v)$. We remark that our algorithm does not compute $G_i(v)$, but its definition is essential for the analysis. Intuitively, the set $G_i(v)$ consists of the vertices after i rounds that the algorithm still believes could be in a cycle of length R with v. If $|G_i(v)| \geq 100\sqrt{n} \log n$, then by the choice of S_i as an

independent random set of expected size $O(\sqrt{n})$, we have that $T_i(v)$ is a random sample of $G_i(v)$ of expected size at least $100 \log n$. In this way, $R_i(v)$ is just a random sample of $G_i(v)$ of size $O(\log n)$.

As we show in Lemma 3.4, if $|G_M(v)| \geq 100\sqrt{n} \log n$, our algorithm discovers w.h.p. a cycle of length at most $3R$ sometime during the shortest path computations done at the beginning. On the other hand, if $|G_M(v)| \leq 100\sqrt{n} \log n$, then we can grow a shortest path tree from v but only include vertices in $G_M(v)$ to search for a cycle of length R, only paying runtime $|G_M(v)| = \tilde{O}(\sqrt{n})$ for that vertex v.

Formalizing this final step, the algorithm computes a shortest path tree $T(v)$ from v up to depth $R/2$, keeping only vertices $s \in V$ such that $d(v, s) \leq R/2$ and $d(s, t) \leq 3R/2$ for all $t \in \bigcup_{0 \leq j \leq M} R_j(v)$. The set A_v is the set of vertices in $T(v)$.

Algorithm 1 GirthApprox(G, R), takes a graph G and a parameter R. If the girth of G is at most R this algorithm outputs w.h.p. a cycle of length at most $3R$

1: Invoke SimilarSet(G, R) to either find a cycle of length at most $3R$ or set $A_v^{\text{out}} \subseteq V$ for each $v \in V(G)$.
2: Invoke SimilarSet(G^{rev}, R) where G^{rev} is the graph where the direction of every edge is reversed to either find a cycle of length at most $3R$ or sets $A_v^{\text{in}} \subseteq V$ for each $v \in V(G)$.
3: If a cycle of length at most $3R$ has yet to be found for each $v \in V(G)$ perform Dijkstra from v in the graph induced by $A_v^{\text{out}} \cup A_v^{\text{in}}$ to find a cycle of length at most R through one of the v.
4: Return any cycle of length at most $3R$ found.

Next we prove the correctness of our girth computation algorithm GirthApprox (Algorithm 1) and bound its running time. First we prove the following lemma which provides a fairly straightforward argument that the algorithm always outputs the correct result. The more challenging part of the analysis will be to bound its running time.

Lemma 3.2. *If G contains a cycle of length at most R then GirthApprox(G, R) (Algorithm 1) returns a cycle of length at most $3R$.*

Proof. Assume G contains a cycle C of length at most R. Let v be a vertex in C. If the algorithm returns a cycle in line 3 of SimilarSet (Algorithm 2) then since this cycle has length at most $3R$, the algorithm works as desired.

Consequently, we assume that this is not the case. Our goal is now to show that A_v^{out} contains all vertices $c \in C$ such that $d(v, c) \leq R/2$ and that A_v^{in} contains all vertices $c \in C$ such that $d(c, v) \leq R/2$. Since for all $c \in C$ either $d(v, c) \leq R/2$ or $d(c, v) \leq R/2$ this will imply that $C \subseteq A_v^{\text{out}} \cup A_v^{\text{in}}$ and therefore a cycle of length at most R will be found in Line 3 of GirthApprox(G, R) (Algorithm 1) and the algorithm works as desired. Further, note that it suffices to show that A_v^{out} contains all vertices $c \in C$ such that $d(v, c) \leq R/2$ as this will imply the desired claimed regarding A_v^{in} by symmetry.

Consider the execution of SimilarSet (Algorithm 2) from Line 1 of GirthApprox(G, R) (Algorithm 1). Further, consider a vertex $t \in T_i(v)$ for some $0 \leq i \leq M - 1$. Recall that $d(v, t) \leq R/2$ (by definition and construction of $T_i(v)$). Consider a vertex c in C. As v

---

Algorithm 2 SimilarSet(G, R), takes a graph G and a parameter R. This algorithm either computes a cycle of length $3R$ or a set $A_v \subseteq V(G)$ of "similar" vertices to v (with respect to balls of radius $R/2$) for each $v \in V(G)$.

1: For $M = 50 \log n$, sample sets $S_0, S_1, \cdots, S_M \subseteq V(G)$, each of expected size $O(n^{1/2})$ by sampling every vertex $v \in V$ independently with probability $p = n^{-1/2}$.
2: Run Dijkstra to/from each vertex $v \in S_i$ for every $1 \leq i \leq M$.
3: If there exists a vertex $v \in \bigcup_{1 \leq i \leq M} S_i$ such that v is on a cycle of length $3R$ then return the shortest such cycle.
4: **for** every vertex $v \in V$ **do**
5:    Set $T_0(v) \leftarrow \{s \in S_0 \mid d(v, s) \leq R/2\}$.
6:    **for** $i = 1, \ldots, M$ **do**
7:       **if** $|T_{i-1}(v)| > 100 \log n$ **then**
8:          Let $R_{i-1}(v)$ be $100 \log n$ vertices chosen independently at random from $T_{i-1}(v)$
9:       **else**
10:          Let $R_{i-1}(v) = T_{i-1}(v)$.
11:       $T_i(v) \leftarrow \{s \in S_i \mid d(v, s) \leq R/2 \text{ and } d(s, t) \leq 3R/2 \text{ for all } t \in \bigcup_{0 \leq j \leq i-1} R_j(v)\}$
12:    Compute a shortest path tree $T(v)$ up to depth $R/2$ keeping only vertices s such that $d(s, t) \leq 3R/2$ for all $t \in \bigcup_{0 \leq j \leq M} R_j(v)\}$.
13:    Set A_v to be the set of vertices in $T(v)$.
    return A_v for all $v \in V$

---

and c are on a cycle of length R we have $d(c, v) \leq R$ and therefore $d(c, t) \leq d(c, v) + d(v, t) \leq 3R/2$ by triangle inequality. It follows by construction that each vertex $c \in C$ with $d(v, c) \leq R/2$ will be added to A_v as desired. □

With the correctness of GirthApprox (Algorithm 1) established, in the remainder of this section we focus on analyzing its running time. To do this we will consider an invocation of SimilarSet (Algorithm 2) and both bound its running time and the size of the sets A_v it computes.

Before setting up the proofs, for each vertex $v \in V$ we define

$$G_0(v) = \{s \in V \mid d(v, s) \leq R/2\} \quad \text{and}$$
$$G_i(v) = \{s \in V \mid d(v, s) \leq R/2 \text{ and}$$
$$d(s, t) \leq 3R/2 \text{ for all } t \in \bigcup_{0 \leq j \leq i-1} R_j(v)\}.$$

Notice that the distribution of $T_i(v)$ is the distribution on vertices that results from taking each $s \in G_i(v)$ and including it in $T_i(v)$ with probability $p = 1/\sqrt{n}$.

Loosely speaking, the analysis of the running time is roughly as follows. The main non trivial part is to show that the expected size of the sets A_v^{in} and A_v^{out} is $\tilde{O}(\sqrt{n})$. This, together with the assumption that the maximum degree is $O(m/n)$, will imply that the running time of our algorithm is $\tilde{O}(m\sqrt{n})$. We roughly speaking show the following for the set A_v^{out} (similarly for the set A_v^{in}). We want to claim that w.h.p. the sets $G_i(v)$ are decreasing by at least a constant factor until there is a set $G_i(v)$ of $\tilde{O}(\sqrt{n})$ size. As A_v^{out} is a subset of $G_M(v) \subseteq G_i(v)$, the claim follows. Assume this is not case, i.e., there exists an index i such that $|G_{i+1}(v)| > 0.8|G_i(v)|$. Note that by construction for every vertex s in $G_{i+1}(v)$ all vertices in $R_i(v)$ are at

distance at most $3R/2$ from it. As $R_i(v)$ is a sampled set of $G_i(v)$, we can show that w.h.p. most vertices in $G_i(v)$ (say 0.9 fraction of them) are at distance at most $3R/2$ from s. As $|G_{i+1}(v)| > 0.8|G_i(v)|$, this means that this is also true for most vertices in $G_i(v)$. That is, most vertices in $G_i(v)$ are at distance at most $3R/2$ to most of the other vertices in $G_i(v)$. We show by counting argument that in this case there must be many pairs of vertices u and v such that $u, v \in G_i(v)$ and $d_G(u, v) \leq 3R/2$ and $d_G(v, u) \leq 3R/2$ (hence u and v are both on a cycle of length at most $3R$). That is, w.h.p. $G_i(v)$ contains many vertices that are on cycles of length at most $3R$. W.h.p. we can show that such a vertex will belong to $S_i(v)$ and therefore the algorithm will detect a cycle of length $3R$ and will not continue to computing the sets A_v^{out}.

LEMMA 3.3. *Consider a vertex v, index $i \in [M]$ such that $|G_i(v)| \geq 200\sqrt{n}\log n$ and a vertex $u \in V$. If there are less than $0.9|G_i(v)|$ vertices $s \in G_i(v)$ such that $d(u, s) \leq 3R/2$ then with probability at least $1 - 2/n^{10}$, $u \notin G_{i+1}(v)$.*

PROOF. Note that the distribution of obtaining $T_j(v)$ is equivalent to the distribution of picking every vertex in $G_j(v)$ with probability p for every $1 \leq j \leq M$.

We first show that with high probability $T_i(v)$ contains at least $100\log n$ vertices (and therefore also $R_i(v)$). As $|G_i(v)| \geq 200\sqrt{n}\log n$ then the expected size of $|T_i(v)|$ is at least $200\log n$. Therefore, by Chernoff Bound the probability that $|T_i(v)| \leq 100\log n$ is at most $\left(\frac{e^{-1/2}}{1/2^{1/2}}\right)^{100\log n} < 1/n^{10}$.

Assume this is indeed the case, that is, $T_i(v)$ contains at least $100\log n$ vertices. The set $R_i(v)$ is a sampled set of $100\log n$ vertices from $T_i(v)$. As the distribution of obtaining the set $T_i(v)$ is equivalent to distribution of picking every vertex in $G_i(v)$ with probability p then the distribution of $R_i(v)$ is equivalent to picking $100\log n$ vertices from $G_i(v)$ (every vertex in $G_i(v)$ has the same probability appearing in $R_i(v)$). Consider a uniformly random vertex s from $G_i(v)$. With probability at least $1/10$ we have $d(u, s) > 3R/2$. In other words with probability at most $9/10$ we have $d(u, s) \leq 3R/2$. Therefore, the probability that for every vertex s in $R_j(v)$ we have $d(u, s) \leq 3R/2$ is at most $(9/10)^{100\log n} \leq 1/n^{10}$.

The lemma follows by union bound over the events that either $|T_i(v)|$ is smaller than $100\log n$ or for all $s \in R_i(v)$ we have $d(u, s) \leq 3R/2$.   □

LEMMA 3.4. *If there exists a vertex v and an index i such that $|G_i(v)| \geq 200\sqrt{n}\log n$ and $|G_{i+1}(v)| \geq 0.8|G_i(v)|$ then with probability at least $1 - 1/n^8$ there exists a vertex in $T_i(v)$ that is contained in a cycle of length at most $3R$.*

PROOF. Assume such a vertex v and index i exist.

We say that a vertex u is (v, i)-dense if there are at least $0.9|G_i(v)|$ vertices $s \in G_i(v)$ such that $d(u, s) \leq 3R/2$.

By union bound on all vertices $v \in V$ on Lemma 3.3, with probability at least $1 - 2/n^9$, all vertices in $G_{i+1}(v)$ are (v, i)-dense.

As $G_{i+1}(v) \subseteq G_i(v)$ and $|G_{i+1}(v)| \geq 0.8|G_i(v)|$, we also have that with probability at least $1 - 2/n^9$, $0.8|G_i(v)|$ vertices in $G_i(v)$ are (v, i)-dense. Assume this is indeed the case.

Imagine constructing the following directed graph H whose set of vertices is $G_i(v)$ and set of edges is the following. For every

vertex u in $G_i(v)$ add an outgoing edge for every vertex s such that $d(u, s) \leq 3R/2$. Note that if there exists two edges in the graph (u, s) and (s, u) then both u and s are on a cycle of length at most $3R$. We next show that by counting argument there are many vertices in $G_i(v)$ that are on a cycle of length at most $3R$. Every (v, i)-dense vertex u has $0.9|G_i(v)|$ outgoing edges in H. There are at least $0.9|G_i(v)|$ (v, i)-dense vertices in H. We get that the number of edges $E(H)$ is at least $0.71|G_i(v)|^2$, that is, $|E(H)| \geq 0.71|G_i(v)|^2$.

On the other hand let α be the fraction of vertices in $G_i(v)$ that do appear on a cycle of length at most $3R$. For every edge in H give a credit of $1/2$ for each of its endpoints vertices. Note that every vertex x that do not belong to a cycle of length at most $3R$ can get a credit of less than $|G_i(v)|/2$. To see this, note that there is no other vertex with both incoming and outgoing edge to x (as otherwise x is on a cycle of length at most $3R$) so the total number of incoming and outgoing edges of x is at most $|G_i(v)| - 1 < |G_i(v)|$. Hence, the total credit of x is less than $|G_i(v)|/2$. The total credit of a vertex x that do participate in a cycle of length at most $3R$ is less than $|G_i(v)|$. We get that the total credit of all vertices, which is also equal to the total number of edges in H, is less than $\alpha|G_i(v)||G_i(v)|/2 + (1 - \alpha)|G_i(v)|^2$. It follows that $0.71|G_i(v)|^2 \leq \alpha|G_i(v)||G_i(v)|/2 + (1 - \alpha)|G_i(v)|^2$. Straight forward calculation show that $\alpha < 0.58$ and thus $1 - \alpha > 0.42$. In other words, at least $0.42|G_i(v)|$ vertices in $G_i(v)$ belong to a cycle of length at most $3R$.

Next, we claim that w.h.p. there is such a vertex in $T_i(v)$. Recall that the distribution of $T_i(v)$ is equivalent to picking every vertex in $G_i(v)$ with probability p. Consider one vertex that participates in a cycle of length at most $3R$ the probability it does not belong to $T_i(v)$ is $1 - p$. The probability that none of the $0.42|G_i(v)|$ vertices belong to $T_i(v)$ is at most $(1 - p)^{0.42|G_i(v)|} \leq (1 - p)^{84\log n/p} \leq 1/n^{10}$.

The lemma follows (as $1/n^{10} + 2/n^9 < 1/n^8$ for large enough n).   □

Finally, the following concludes the running time of our algorithm.

LEMMA 3.5. *The expected running time of Algorithm 1 is $O(m\sqrt{n}\log n + n\sqrt{n}\log^3 n) = \tilde{O}(m\sqrt{n})$.*

PROOF. Consider one of the executions of SIMILARSET (Algorithm 2) by GIRTHAPPROX (Algorithm 1). This algorithm computes Dijkstra to/from each vertex $w \in S_i$ for every $1 \leq i \leq M$ in $O(m + n\log n)$ time. The expected size of each S_i is $O(n^{1/2})$. Thus, the expected time of this computation for S_i is $O(m\sqrt{n} + n\sqrt{n}\log n)$. There are $O(\log n)$ sets S_i and therefore there is at most $O(m\sqrt{n}\log n + n\sqrt{n}\log^2 n)$ expected time for the computation of all Dijkstra's. Next, for every vertex v the algorithm computes the sets $T_i(v)$ for every $i \in [M]$. The set $T_0(v)$ can be computed easily in $O(|S_0|)$ time which is $O(n^{1/2})$ in expectation. In order to compute $T_i(v)$ for $i > 0$, the algorithm considers every vertex $s \in S_i$ and it check if s is at distance at most $3R/2$ from every vertex in $t \in \cup_{j\in[0,\ldots,i-1]}R_j(v)$. There are $O(\log^2 n)$ vertices t in $\cup_{j\in[0,\ldots,i-1]}R_j(v)$. The distance $d(s, v)$ is already computed and thus can be retrieved in $O(1)$ time. Overall, computing the set $T_i(v)$ takes $O(n^{1/2}\log^2 n)$ in expectation. Therefore, $O(n^{1/2}\log^3 n)$ for all indices $i \in [M]$. Hence, for all vertices v $O(n^{3/2}\log^3 n)$ expected time for this part.

Next, we bound the cost of computing the balls, A_v, and we bound their size. By a slight abuse of notation we call a vertex s

(v, M)-dense if it satisfies

$$d(v, s) \leq R/2 \text{ and } d(s, t) \leq 3R/2 \text{ for all } t \in \cup_{j \in [0, \dots, M]} R_j(v).$$

The algorithm grows a ball from every vertex $v \in V$ by only keeping vertices s that are (v, M)-dense to compute A_v.

We first show that if there is no index i such that $|G_i(v)| \geq 200\sqrt{n} \log n$ and $|G_{i+1}(v)| \geq 0.8|G_k(v)|$ then the expected time to compute the ball of v is $O(n^{1/2} \log^3 n + n^{1/2} \log n \cdot \delta)$. We do that by showing that the expected number of vertices in $G_M(v)$ is $O(n^{1/2} \log n)$. As the maximum degree in G is $O(\delta)$ and checking if a vertex s is (v, M)-dense takes $O(\log^2 n)$ time, then the claim follows.

As for every i such that
$|G_i(v)| \geq O(\sqrt{n} \log n)$ we have $|G_{i+1}(v)| \leq 0.8|G_i(v)|$ then straight forward calculation shows that there exists an index $M' \in [1..M]$ such that $|G_{M'}(v)| < O(\sqrt{n} \log n)$. Note that the ball of v contains only vertices from $G_{M'}(v)$ and thus the claim follows.

We now assume that there exists a vertex v and index i such that $|G_i(v)| \geq O(\sqrt{n} \log n)$ and $|G_{i+1}(v)| \geq 0.8|G_i(v)|$. By claim 3.4 in this case with probability at least $1 - 1/n^8$ the algorithm finds a cycle of length $3R$ and returns it in Line 3. Therefore, in this case the algorithm does not compute the balls in Line 13. With probability at most $1/n^8$ the algorithm does not find a cycle in Line 3 and therefore continues to computing the balls in Line 13. The computation of all balls in Line 13 is bounded by $O(mn)$ in this case. As this happens with very small probability this does not effect the asymptotic bound of the expected running time. The lemma follows. □

We conclude this section with the proof of Theorem 1.

Proof of Theorem 1. The algorithm calls Algorithm GirthApprox using a binary search on the range $[1, nW]$ to find a parameter R such that Algorithm GirthApprox returns a cycle (of length at most $3(R + 1)$) when invoked on $R + 1$ but not on R. As mentioned above the dependency on $\log nW$ can be improved to $\log n$ using the method used in [16] (Section 5.1). Roughly speaking this method constructs in $O(m \log n)$ time a set of graphs such that the number of vertices in all these graphs together is $O(n \log n)$, the number of edges is $O(m \log n)$, the ratio between the maximum edge weight and the minimum edge weight in all these graphs is $O(n)$ and the shortest cycle is contained in one of these graphs. Instead of running binary search on G, we run it in each of these graphs.

Now using Lemma 3.2 and Lemma 3.5 the theorem follows. □

We give our result on constant approximation roundtrip spanners in $\tilde{O}(m\sqrt{n})$ time and show Theorem 2 in the full version.

4 DETERMINISTIC $O(k \log \log n)$ APPROXIMATION ALGORITHMS

In this section we present our deterministic algorithms for computing a $O(k \log \log n)$ approximation to the girth and computing $O(k \log \log n)$ multiplicative roundtrip spanners. Our main result will be showing how to compute improved roundtrip covers as defined originally in [18]. Leveraging this result we will prove Theorem 3 and Theorem 4.

First, leveraging the definitions of balls in Section 2 we define roundtrip covers. Intuitively, roundtrip covers are a union of balls of radius kR such that if vertices $u, v \in V(G)$ satisfy $d(u \leftrightarrows v) \leq R$ then u, v are both in some ball in the cover.

DEFINITION 4.1 (ROUNDTRIP COVERS). *A collection C of balls is a (k, R) roundtrip cover of a weighted directed graph G if and only if every ball in C has radius at most kR, and for any $u, v \in V(G)$ with $d(u \leftrightarrows v) \leq R$ there is a ball $B \in C$ such that $u, v \in B$.*

Specifically, we show the following theorem.

THEOREM 7 (IMPROVED ROUNDTRIP COVERS). *For an n-vertex m-edge graph G, an execution of ROUNDTRIPCOVER(G, k, R) returns a collection C of balls that forms a $(O(k \log \log n), R)$ roundtrip cover of a weighted directed graph G in time $m^{1+O(1/k)}$ where $\sum_{B \in C} |V(B)| = n^{1+O(1/k)}$.*

To show Theorem 3 from Theorem 7, we can compute $(k, 2^i)$ roundtrip covers for all $0 \leq i \leq O(\log n)$, and set our girth estimate as the minimum radius of any ball in the cover that has a cycle. To compute a roundtrip spanner, simply take the union of all the balls in the $(k, 2^i)$ roundtrip covers for all $i = O(\log n)$.

The rest of the section is organized as follows. In Section 4.2 we state our main algorithm. In Section 4.3 we analyze the algorithm and prove Theorem 7. In Section 4.4 we use Theorem 7 to formally prove Theorem 3 and Theorem 4.

4.1 Technical Overview

We focus on unweighted directed graphs G and for a parameter R, construct a roundtrip spanner H so that if the roundtrip distance between u and v is at most R in G, then their roundtrip distance is at most $O(Rk \log \log n)$ in H.

Our approach is based on growing inballs and outballs in the graph G. Fix a vertex v, and let $B_i^{\text{in}}, B_i^{\text{out}}$ denote the inball and outball of radius iR around v, and let $|B_i^{\text{in}}|, |B_i^{\text{out}}|$ denote the number of vertices in the balls and fix $d = O(k \log \log n)$. We start by growing and inball and outball around v. First, if $|B_d^{\text{in}} \cap B_d^{\text{out}}| \geq \frac{n}{2}$, then we can build a roundtrip ball of radius $2dR + R$ and delete $B_d^{\text{in}} \cap B_d^{\text{out}}$ from our graph. This is safe essentially by our observation above. Otherwise, we find an index i such that $|B_{i+1}^{\text{in}}|$ isn't much larger than $|B_i^{\text{in}}|$, we recursively build a roundtrip cover on B_{i+1}^{in} and then delete B_i^{in}. This is safe to do by our observation above. Similarly, if there is an index i such that $|B_{i+1}^{\text{out}}|$ isn't much larger than $|B_i^{\text{out}}|$, we recursively build a roundtrip cover on B_{i+1}^{out} and then delete B_i^{out}. Through standard ball cutting inequalities we can show that such an index i exists (Lemma 4.2). We would like to elaborate on a few points. First, when we compare the sizes of $|B_{i+1}^{\text{in}}|$ and $|B_i^{\text{in}}|$, we compare both the number of vertices *and* edges, the former to control the size of the roundtrip spanner constructed, and the latter to control runtime. Second, we grow the inball and outball at the same rate, i.e. we alternately add an edge at a time to the inball and outball to maintain that the work spent on each is the same.

4.2 Main Algorithm

We first give a high-level description of our algorithm for computing Roundtrip Covers, ROUNDTRIPCOVER, which is presented formally as Algorithm 3.

High-level Description of Algorithm. As discussed in Sections 1.3 and 4.1, our algorithm is based on ball growing along with the following observation: if for a radius r' we compute a roundtrip cover of $B_v^{\text{in}}(r' + R)$ and add all the balls in the computed roundtrip cover on $B_v^{\text{in}}(r' + R)$ to our final cover, then we can safely delete all vertices $u \in B_v^{\text{in}}(r')$ from our graph and recurse on the rest of graph; the deleted vertices are already satisfied in the sense that for every $u' \in V(G)$ with $d(u \leftrightarrows u') \leq R$ there is a ball B in the cover such that $u, u' \in B$. Indeed, if $u \in B_v^{\text{in}}(r')$ and $d(u \leftrightarrows u') \leq R$ then $u, u' \in B_v^{\text{in}}(r' + R)$ and therefore we are guaranteed that the roundtrip cover on $B_v^{\text{in}}(r' + R)$ contains a ball B such that $u, u' \in B$. Using this observation, we grow inballs and outballs around vertices in our graph G to "partition" our graph into pieces that possibly overlap, where the overlap corresponds to the boundary $B_v^{\text{in}}(r' + R) \backslash B_v^{\text{in}}(r')$ in our example.

We describe our algorithm in more detail now. Consider any vertex v. We grow an inball and outball around v at the same rate, spending the same time on the inball and outball. First, we consider the case that $|V(B_v^{\text{in}}(r))|, |V(B_v^{\text{out}}(r))| \geq \frac{3}{4}n$ for some $r = O(Rk \log \log n)$, as was done in Pachocki *et al.* [16]. Then we know that $|V(B_v^{\text{in}}(r)) \cap V(B_v^{\text{out}}(r))| \geq \frac{n}{2}$. By our observation above, we can add the ball $B_v(2r+R)$ to our roundtrip cover, delete $B_v^{\text{in}}(r) \cap B_v^{\text{out}}(r)$ from G, and recurse on the remainder. Otherwise, if we find a radius r' such that say $B_v^{\text{in}}(r')$ and $B_v^{\text{in}}(r' + R)$ satisfy the conditions of GoodCut (Algorithm 4), then we recurse on $B_v^{\text{in}}(r' + R)$ and delete $B_v^{\text{in}}(r')$ from our graph and recurse on the remaining graph. This is safe to do by our observation above. We can also do an analogous process on $B_v^{\text{out}}(r')$ and $B_v^{\text{out}}(r' + R)$. By a variant of the standard ball-growing inequality (Lemma 4.2) we can show that a good cut always exists.

We now will give some intuition about the condition in Good-Cut and the (somewhat strange) appearance of the $O(\log \log n)$ in our algorithm. First, we remark that the condition in GoodCut must track both the number of vertices and edges in the ball: the former to control recursion depth and roundtrip cover size, and the latter to control runtime. Now we give intuition for why we require an $O(k \log \log n)$ approximation factor in our algorithm. Consider growing inballs $B_v^{\text{in}}(r)$ from v for various radii r, and recall that we make a cut depending on the relative sizes of $|V(B_v^{\text{in}}(r))|$ and $|V(B_v^{\text{in}}(r + R))|$. Now, note that if for example $|V(B_v^{\text{in}}(r))| = O(1)$, we can afford to have $|V(B_v^{\text{in}}(r + R))| = O(n^{1/k})$, as we can simply run a naive algorithm on $B_v^{\text{in}}(r + R)$ now. On the other hand, if for example $|V(B_v^{\text{in}}(r))| = \Omega(n)$, we can essentially only afford to have $|V(B_v^{\text{in}}(r + R))| \leq \left(1 + \frac{1}{k}\right)|V(B_v^{\text{in}}(r))|$. To see the latter, note that the recurrence $T(m) = \left(1 + \frac{1}{k}\right)(T(m/2) + T(m/2))$ has solution $T(m) = m^{1+O(1/k)}$. Now, interpolating between these two extremes allows us to compute the optimal way to do ball cutting (which is done in GoodCut). This leads to a ball cutting procedure with $O(k \log \log n)$ levels, and thus results in an $O(k \log \log n)$ approximation ratio.

Explanation of Algorithm 3: We now explain what each piece of Algorithm 3 is doing. Here, i_{in} and i_{out} track the radius of the inball and outball that we are growing. We grow the balls at the same rate. If we notice that at any point we are in position to make a good cut (see lines 7, 9) then we do so. Otherwise, we know that both balls

---

Algorithm 3 RoundtripCover(G, k, R), takes a graph G with n vertices, m edges, and parameters k and R. Returns a $(O(k \log \log n), R)$ roundtrip cover $C = \{B_1, B_2, \dots, \}$

1: $i_{\text{in}}, i_{\text{out}} \leftarrow 0$.
2: $r \leftarrow 5kR \log \log n$.
3: Take any $v \in V(G)$.
4: **while true do** *some condition below in lines 5, 7, 9 will trigger eventually*
5:     **if** $\min(|V(B_v^{\text{in}}((i_{\text{in}} + 1)R))|, |V(B_v^{\text{out}}((i_{\text{out}} + 1)R))| \geq \frac{3n}{4}$ **then**
6:         **return** $\{B_v(2r + R)\} \cup$ RoundtripCover$(G\backslash(B_v^{\text{in}}((i_{\text{in}} + 1)R) \cap B_v^{\text{out}}((i_{\text{out}} + 1)R)), R, k)$.
7:     **if** GoodCut$(G, B_v^{\text{in}}(i_{\text{in}}R), B_v^{\text{in}}((i_{\text{in}} + 1)R))$ **then**
8:         **return** RoundtripCover$(B_v^{\text{in}}((i_{\text{in}} + 1)R), R, k) \cup$ RoundtripCover$(G\backslash B_v^{\text{in}}(i_{\text{in}}R), R, k)$.
9:     **if** GoodCut$(G, B_v^{\text{out}}(i_{\text{out}}R), B_v^{\text{out}}((i_{\text{out}} + 1)R))$ **then**
10:         **return** RoundtripCover$(B_v^{\text{out}}((i_{\text{out}} + 1)R), R, k) \cup$ RoundtripCover$(G\backslash B_v^{\text{out}}(i_{\text{out}}R), R, k)$.
11:     **if** $|E(B_v^{\text{in}}(i_{\text{in}}R))| \leq |E(B_v^{\text{out}}(i_{\text{out}}R))|$ or $|V(B_v^{\text{out}}(i_{\text{out}}R))| \geq \frac{3n}{4}$ **then**
12:         $i_{\text{in}} \leftarrow i_{\text{in}} + 1$
13:     **else**
14:         $i_{\text{out}} \leftarrow i_{\text{out}} + 1$

Algorithm 4 GoodCut(G, B_1, B_2), takes a graph G with n vertices and m edges, balls $B_1 \subseteq B_2$, and determines whether recursing on B_2 and then deleting B_1 from our graph is good progress

---

1: **if** $|V(B_2)| \leq \frac{3}{4}n$ and $|V(B_2)| \leq |V(B_1)|^{\frac{k-1}{k}} n^{\frac{1}{k}}$ and
2:   $|E(B_2)| \leq \max((1 + \frac{1}{k})|E(B_1)|, |E(B_1)|^{\frac{k-1}{k}} m^{\frac{1}{k}})$ **then**
3:     **return true**
4: **else**
5:     **return false**

---

will eventually contain many vertices (see line 5). In this case, we add $B_v(2r + R)$ to our roundtrip cover, delete $B_v^{\text{in}}(i_{\text{in}}R) \cap B_v^{\text{out}}(i_{\text{out}}R)$ from our graph, and recurse. To grow the inball and outball at the same rate, we run Dijkstra to grow the inball and outball, alternately processing an edge at a time from the inball and outball. We check the condition of GoodCut on a ball when we have certified that we have processed all vertices up to distance $i_{\text{in}}R$ or $i_{\text{out}}R$ respectively.

4.3 Analysis of RoundtripCover and proof of Theorem 7

In this section we prove Theorem 7, bounding the performance of our roundtrip cover algorithm Algorithm 3. We start by showing that $1 + \max(i_{\text{in}}, i_{\text{out}}) \leq 5k \log \log n$ at all points in the algorithm, hence some condition in lines 5, 7, 9 will trigger eventually.

LEMMA 4.2. *At all points during Algorithm 3, we have that* $1 + \max(i_{\text{in}}, i_{\text{out}}) \leq 5k \log \log n$.

PROOF. We show $1 + i_{\text{in}} \leq 5k \log \log n$, and the bound on $1 + i_{i_{\text{out}}}$ is analogous. To prove this we assume that none of the conditions

in the inner loop of the algorithm trigger, and compute the resulting vertex and edge sizes of $B_v^{\text{in}}(i_{\text{in}}R)$ and $B_v^{\text{out}}(i_{\text{out}}R)$. To this end, assume that $|V(B_v^{\text{in}}(i_{\text{in}}R))| \leq \frac{3n}{4}$ and $|E(B_v^{\text{in}}(i_{\text{in}}R))| \leq m$. By the conditions of lines 5, 7, and 11 we know that each time we increment i_{in} either

$$|V(B_v^{\text{in}}((i_{\text{in}}+1)R))| \geq |V(B_v^{\text{in}}(i_{\text{in}}R))|^{\frac{k-1}{k}} n^{\frac{1}{k}} \qquad (1)$$

or

$$|E(B_v^{\text{in}}((i_{\text{in}}+1)R))| \geq |E(B_v^{\text{in}}(i_{\text{in}}R))|^{\frac{k-1}{k}} m^{\frac{1}{k}} \text{ and} \qquad (2)$$

$$|E(B_v^{\text{in}}((i_{\text{in}}+1)R))| \geq \left(1 + \frac{1}{k}\right)|E(B_v^{\text{in}}(i_{\text{in}}R))|. \qquad (3)$$

We first show that Eq. (1) can only hold for $2k \log 4 \log n$ values of i_{in}. To this end, define a sequence $\{x_i\}_{i \geq 0}$ as $x_0 = 1$ and $x_{i+1} = x_i^{\frac{k-1}{k}} n^{\frac{1}{k}}$. By induction it follows that $x_i = n^{1-\left(\frac{k-1}{k}\right)^i}$. In particular,

$$x_{2k \log 4 \log n} = n^{1-\left(\frac{k-1}{k}\right)^{2k \log 4 \log n}} \geq \frac{3}{4}n.$$

This shows that the condition in Eq. (1) can only hold at most $2k \log 4 \log n$ times. Similarly, after Eq. (2) holds for $2k \log 4 \log n$ different i_{in}, we will have that $|E(B_v^{\text{in}}(i_{\text{in}}R))| \geq \frac{3m}{4}$. At this point, Eq. (3) can hold at most k times. This gives us that in total

$$1 + i_{\text{in}} \leq 1 + 2k \log 4 \log n + 2k \log 4 \log n + k \leq 5k \log \log n$$

as desired. □

Now we proceed to proving Theorem 7.

PROOF OF THEOREM 7. We first show that the algorithm indeed returns a $(O(k \log \log n), R)$ roundtrip cover. Then we bound the total size of balls in the roundtrip cover, as well as the runtime.

Returns a $(O(k \log \log n), R)$ *roundtrip cover.* We analyze lines 6, 8, and 10. In line 6, note that by Lemma 4.2, we know that $(i_{\text{in}}+1)R, (i_{\text{out}}+1)R \leq r$. Therefore, we know that $B_v^{\text{in}}(i_{\text{in}}R) \cap B_v^{\text{out}}(i_{\text{out}}R) \subseteq B_v(2r)$. Additionally, it is clear that for any vertex $u \in B_v^{\text{in}}(i_{\text{in}}R) \cap B_v^{\text{out}}(i_{\text{out}}R)$, if another vertex u' satisfies $d(u \leftrightarrows u') \leq R$ then $u' \in B_v(2r + R)$. Therefore, the ball $B_v(2r + R)$ contains both u and u', so we can safely delete $B_v^{\text{in}}(i_{\text{in}}R) \cap B_v^{\text{out}}(i_{\text{out}}R)$ from G and recurse. This is exactly what is happening in line 6. In line 8, note that for any vertex $u \in B_v^{\text{in}}(i_{\text{in}}R)$, if another vertex u' satisfies $d(u \leftrightarrows u') \leq R$ then $u' \in B_v^{\text{in}}((i_{\text{in}}+1)R)$. Therefore, if we construct a roundtrip cover on $B_v^{\text{in}}((i_{\text{in}}+1)R)$, then we can safely delete $B_v^{\text{in}}(i_{\text{in}}R)$ from G and recurse. This is exactly what occurs in line 8. The same argument now applies to line 10. Finally, note that all balls we create are of radius $2r + R = O(Rk \log \log n)$.

Total sizes of balls is $n^{1+O(1/k)}$. We show by induction that the total number of vertices among all balls in the roundtrip cover computed is at most $10n^{\frac{k}{k-1}}$ for an input graph G with n vertices. We show this by analyzing lines 6, 8, and 10. For line 6, note that because $\min(|V(B_v^{\text{in}}((i_{\text{in}}+1)R))|, |V(B_v^{\text{out}}((i_{\text{out}}+1)R))|) \geq \frac{3n}{4}$, we know that $|V(B_v^{\text{in}}((i_{\text{in}}+1)R)) \cap V(B_v^{\text{out}}((i_{\text{out}}+1)R))| \geq \frac{n}{2}$. Therefore, it suffices to verify

$$2n + 10\left(\frac{n}{2}\right)^{\frac{k}{k-1}} \leq 10n^{\frac{k}{k-1}}$$

which is clear. For line 8, for simplicity let $s = |V(B_v^{\text{in}}(i_{\text{in}}R))|$. Then by the condition of GOODCUT, it suffices to note that

$$10|V(B_v^{\text{in}}((i_{\text{in}}+1)R))|^{\frac{k}{k-1}} + 10(n-s)^{\frac{k}{k-1}}$$
$$\leq 10(s^{\frac{k-1}{k}} n^{\frac{1}{k}})^{\frac{k}{k-1}} + 10(n-s)^{\frac{k}{k-1}}$$
$$\leq 10sn^{\frac{1}{k-1}} + 10(n-s)n^{\frac{1}{k-1}} = 10n^{\frac{k}{k-1}}.$$

The same argument now applies to line 10.

Can be implemented to run in time $m^{1+O(1/k)}$. We can implement the algorithm to grow $B_v^{\text{in}}(i_{\text{in}}R)$ and $B_v^{\text{out}}(i_{\text{out}}R)$ at the same rate, i.e., we process a single inedge and outedge at a time, and increment i_{in} and i_{out} when we are sure that we've processed the whole inball or outball. This can be done with Dijkstra's algorithm. We stop growing a ball once it contains at least $\frac{3n}{4}$ vertices. This way, any time we recurse, the total amount of work we have done to this point is at most twice the number of edges in the piece we are recursing on in lines 6, 8, and 10. To bound the runtime, we imagine lines 8 and 10 as partitioning the graph into pieces of the form $B_v^{\text{in}}(i_{\text{in}}R)$ or $B_v^{\text{out}}(i_{\text{out}}R)$ and then recursing on $B_v^{\text{in}}((i_{\text{in}}+1)R)$ or $B_v^{\text{out}}((i_{\text{out}}+1)R)$. This way, the depth of the recursion is at most $O(\log n)$ because we know that $|V(B_v^{\text{in}}((i_{\text{in}}+1)R))|, |V(B_v^{\text{out}}((i_{\text{out}}+1)R))| \leq \frac{3n}{4}$ when we recurse.

We will now show that the total number of edges in level ℓ of the recursion is bounded by $\left(1 + \frac{2}{k}\right)^\ell m^{\frac{k}{k-1}}$, where the top level is level 0. We proceed by induction on ℓ. Say that the algorithm partitions G into $G = G_1 \cup G_2 \cup \cdots \cup G_j$, where each G_i is either of the form $B_v^{\text{in}}(i_{\text{in}}R)$ or $B_v^{\text{out}}(i_{\text{out}}R)$. For simplicity, let $s_i = |E(G_i)|$ and let $t_i = |E(B_v^{\text{in}}((i_{\text{in}}+1)R))|$ or $t_i = |E(B_v^{\text{out}}((i_{\text{out}}+1)R))|$ corresponding to what G_i was. We know by the condition of GOODCUT that $t_i \leq \max\left(\left(1 + \frac{1}{k}\right)s_i, s_i^{\frac{k-1}{k}} m^{\frac{1}{k}}\right)$. By induction, we know that the total number of edges processed in level ℓ is at most

$$\sum_i \left(1 + \frac{2}{k}\right)^{\ell-1} t_i^{\frac{k}{k-1}}$$
$$\leq \left(1 + \frac{2}{k}\right)^{\ell-1} \sum_i \max\left(\left(1 + \frac{1}{k}\right)s_i, s_i^{\frac{k-1}{k}} m^{\frac{1}{k}}\right)^{\frac{k}{k-1}}$$
$$\leq \left(1 + \frac{2}{k}\right)^{\ell-1} \sum_i \left(1 + \frac{2}{k}\right)s_i m^{\frac{1}{k-1}} \leq \left(1 + \frac{2}{k}\right)^\ell m^{\frac{k}{k-1}}$$

as $\sum_i s_i \leq m$ obviously.

Now, it is clear that the total work done on a graph G at some node of the recursion tree is $\tilde{O}(|E(G)|)$ as line 6 only occurs $O(\log n)$ times. Now taking $\ell = O(\log n)$ in the above claim completes the proof. □

4.4 Proofs of Theorem 3 and Theorem 4

Both theorems follow easily from Theorem 7.

PROOF OF THEOREM 3. We first show the result for unweighted graphs. To show this, run

ROUNDTRIPCOVER$(G, O(k), 2^i)$ for $0 \leq i \leq O(\log n)$.

Now, set our estimate g' of the girth to be the smallest radius of any nontrivial ball that we had in a roundtrip cover. By the guarantees of ROUNDTRIPCOVER, it is clear that $g \leq g' \leq O(k \log \log n) \cdot g$

as desired. It is clear that the algorithm runs in time $\tilde{O}(m^{1+\frac{1}{k}})$ by Theorem 7. We can extend this to weighted graphs by instead taking $0 \le i \le O(\log nW)$, where W is the maximum edge weight. This can be improved to $O(\log n)$ by the same method as done in [16], where they give a general reduction by contracting small weight strongly connected components and deleting large weight edges (see Section 5.1 in [16] for more details). □

PROOF OF THEOREM 4. We first show the result for unweighted graphs. It is easy to see that

$$\bigcup_{i=0}^{O(\log n)} \text{RoundtripCover}(G, O(k), 2^i)$$

is an $O(k \log \log n)$ spanner with $\tilde{O}(n^{1+1/k})$ edges by Theorem 7. It is clear that the algorithm runs in time $\tilde{O}(m^{1+\frac{1}{k}})$. The extension to weighted graphs follows as in the above paragraph (proof of Theorem 3). □

5 AN $O(k \log k)$ APPROXIMATION IN $\tilde{O}(m^{1+1/k})$ TIME

In this section we explain how to combine the ideas from Algorithm 3 and Algorithm 1 to give an algorithm for $(O(k \log k), R)$-roundtrip covers with $\tilde{O}(n^{1+1/k})$ edges in time $\tilde{O}(m^{1+1/k})$. Then Theorem 5 and Theorem 6 follow from this in the same way that Theorem 3 and Theorem 4 followed from Theorem 7.

THEOREM 8 (IMPROVED RANDOMIZED ROUNDTRIP COVER). *For an n-vertex m-edge graph G, an execution of ROUNDTRIPCOVER2(G, k, R) returns a collection C of balls that form a $(O(k \log k), R)$ roundtrip cover of a weighted directed graph G in time $m^{1+O(1/k)}$ where $\sum_{B \in C} |V(B)| = n^{1+O(1/k)}$.*

The remainder of the section is organized as follows. We first give an overview for our approach, which combines the complementary approaches of sections Section 4 and Section 3. We then state our main algorithm, Algorithm 5. Afterwards, we analyze Algorithm 5 to prove Theorem 8 in Section 5.2. Finally, we apply Theorem 8 to prove Theorem 5 and Theorem 6.

Overview of approach. Throughout this section, we assume that we have applied Lemma 3.1 to make our graph G approximately regular. Here we give a high level overview for the ideas behind the algorithm. Let G be an n-vertex m-edge graph and let $K := 10k \log k$ for integer k. We start by generalizing Algorithm 1 and Algorithm 2 slightly, where we consider the case where the sampled sets S_i have size $\tilde{O}(n^{1/k})$ instead of $\tilde{O}(n^{1/2})$. To elaborate, we first view Algorithm 1 and Algorithm 2 as algorithms with the following guarantees. They add $\tilde{O}(n^{3/2})$ edges towards a spanner, and then for each vertex v which is not yet in a cycle of length $4R$ using the current spanner edges builds a data structure D_v (corresponding to Algorithm 2) which certifies that for all but at most $O(n^{1/2})$ other vertices u we have that $d(v \leftrightarrows u) > 4R$. We can generalize this as follows. There is a corresponding algorithm (Algorithm 6) which has the following guarantees. It adds $\tilde{O}(n^{1+1/k})$ edges towards a spanner, and then for each vertex v which is not yet in a cycle of length $2KR$ using the current spanner edges builds a data structure

D_v which certifies that for all but at most $O(n^{\frac{k-1}{k}})$ other vertices u we have that $d(v \leftrightarrows u) > KR$.

After running this generalized algorithm (Algorithm 6), for a vertex v, we can define *i-similar vertices* to v, which are intuitively the vertices that the data structure D_v thinks could still possibly be in a cycle of length kR with v and which are within distance iR of v. Then we define a sequence $E_v^0, E_v^1, \cdots, E_v^K$ of "balls" centered at v, where E_v^i is the outball from v consisting of i-similar vertices. The following important conditions hold: $v \in E_v^0$, and $E_v^i \subseteq E_v^{i+1}$ for all $0 \le i < K$. Finally, if $u \in E_v^i$ and $d(u \leftrightarrows u') \le R$, then $u' \in E_v^{i+1}$. This allows us to apply the ball-growing procedure in Algorithm 3 but using the balls E_v^i. Note that by our choice of K and a variant of Lemma 4.2, there exists a good cut. This is because

$$n^{1-\left(\frac{k-1}{k}\right)^K} \ge n^{1-\frac{1}{k}} = n^{\frac{k-1}{k}}.$$

Hence, we can make this good cut and then recurse. Here, our cutting condition is simpler (only checks vertices, not edges) because we have reduced to the case of regular graphs through Lemma 3.1.

Algorithm 5 ROUNDTRIPCOVER2(G, k, R). Takes in a n-vertex m-edge graph G, parameter k, and distance R. Returns a $(O(k \log k), R)$ roundtrip cover $C = \{B_1, B_2, \ldots, \}$

1: $C \leftarrow \emptyset$.
2: $(G', C', D) \leftarrow$ BUILDSIMILAR(G, k, R)
3: $C \leftarrow C'$.
4: $C \leftarrow C \cup$ BALLGROW(G', k, R, D).
5: **return** C.

5.1 Explanation of algorithms

Explanation of Algorithm 5, Algorithm 6, Algorithm 7, Algorithm 8, Algorithm 9. Throughout, we let $\hat{n}$ be the number of vertices at the top level of recursion in the algorithms and we let $K := 10k \log k$ for integer k.

We start by explaining Algorithm 6 (BUILDSIMILAR), which builds a data structure which allows efficient similarity queries. It follows the same blueprint as Algorithm 2. The algorithm first selects sets S^i for $1 \le i \le 100 \log \hat{n}$, where $|S^i| = 100n^{1/k} \log^2 \hat{n}$ for all i. It then computes shortest path trees to and from all vertices in all S^i. The algorithm then adds roundtrip balls of radius $O(KR)$ centered at each $u \in S^i$ to our roundtrip cover. The algorithm then marks all vertices v from the graph that are within distance $2KR$ both to and from some vertex u in some S^i as not turned on anymore. Then for all vertices $v \in V(G)$ the algorithm builds sets T_v^i and a uniform sample S_v^i of T_v^i of size $O(\log \hat{n})$ that allow us to "test" whether another vertex u is similar to v, i.e. could potentially be in a cycle of length $O(KR)$ with v. Eventually, $|T_v^i|$ gets small, and the algorithm stops processing vertex v. Finally, it returns the graph G' of all still on vertices, the updated roundtrip cover, and the data structure D for similarity testing consisting of all the S_v^i for each vertex v and shortest path trees from all $u \in S^i$.

Now we explain Algorithm 8 (SIMILAR), which uses the data structure D computed by BUILDSIMILAR to decide whether vertex u

Algorithm 6 BuildSimilar(G, k, R). Takes in a n-vertex m-edge graph G, parameter k, and distance R. Returns a triple (G', C, D), where $G' \subseteq G$ is a subgraph which still needs to be processed, C is a set of balls to include in the roundtrip cover, and D is a data structure which supports similarity queries. $\hat{n}$ is the number of vertices at the top level of recursion.

1: $C \leftarrow \emptyset$.
2: $K \leftarrow 10k \log k$.
3: Select uniformly random subsets $S^1, S^2, \cdots, S^{100 \log \hat{n}} \subseteq V(G)$ where $|S^i| = 100n^{\frac{1}{k}} \log^2 \hat{n}$ for all i.
4: For all vertices $u \in S^i$ for some i, build a shortest path tree to and from u.
5: $C \leftarrow \bigcup_{i=1}^{100 \log \hat{n}} \bigcup_{u \in S^i} B_u((4K+1)R)$.
6: **for** $v \in V(G)$ **do**
7:     **if** $v \in \left(B_u^{\text{in}}(2KR) \cap B_u^{\text{out}}(2KR) \right)$ for some $u \in S^i$ for some i **then** on$[v] \leftarrow$ **false**.
8: **for** $v \in V(G)$ **do**
9:     **for** $i = 1$ to K **do**
10:        $T_v^i = \{u \in S^i : d(v, u) \leq KR$ and $d(u, w) \leq 2KR$ for all $w \in S_v^j$ for all $1 \leq j < i.\}$
11:        **if** $|T_v^i| \geq 50 \log \hat{n}$ **then**
12:          $S_v^i \leftarrow$ uniform sample of T_v^i of size $50 \log \hat{n}$.
13:        **else**
14:          Return to line 8.
15: Have D store all the S_v^i and shortest path trees from all vertices $u \in S^i$ for some i.
16: **return** $(G[\{v : \text{on}[v] = \textbf{true}\}], C, D)$.

Algorithm 7 BallGrow(G, k, R, D). Takes in a n-vertex m-edge graph G, parameter k, distance R, and data structure D supporting similarity queries. Returns a set C of balls to include in the roundtrip cover.

1: $C \leftarrow \emptyset$.
2: $K \leftarrow 10k \log k$.
3: on$[v] \leftarrow$ **true** for all $v \in V(G)$.
4: **while** there exists v with on$[v] = $ **true** **do**
5:     **for** $i = 0$ to $K - 1$ **do**
6:        $E_v^i \leftarrow \{u \in V(G) : \text{on}[u]$ and Similar(G, u, v, D, i, R) and u reachable from v through $E_v^i\}$.
           ▷ We elaborate on this definition Section 5.1.
7:        $E_v^{i+1} \leftarrow \{u \in V(G) : \text{on}[u]$ and Similar$(G, u, v, D, i+1, R)$ and u reachable from v through $E_v^{i+1}\}$.
8:        **if** GoodCut2(G, E_v^i, E_v^{i+1}) **then**
9:          $C \leftarrow C \cup$ RoundtripCover2(E_v^{i+1}, k, R).
10:          on$[v] \leftarrow$ **false** for all $v \in E_v^i$.
11:          Break loop and return to line 4.
12: **return** C.

---

is i-similar to vertex v. It returns true if and only if

$$d(v, u) \leq iR \text{ and}$$

$$d(u, w) \leq (i + K)R \text{ for all } w \in S_v^j \text{ for all } 1 \leq j \leq 100 \log \hat{n}.$$

Intuitively, this contains a ball around v of distance iR that contains

Algorithm 8 Similar(G, u, v, D, i, R), Takes in a n-vertex m-edge graph G, vertices $u, v \in V(G)$, data structure D, parameter R, decides whether u is i-similar to v

1: $K \leftarrow 10k \log k$.
2: **if** $d(v, u) > iR$ **then**
3:     **return false**
4: **for** $1 \leq j \leq 100 \log \hat{n}$ **do**
5:     **for** $w \in S_v^j$ **do**
6:        **if** $d(u, w) > (i + K)R$ **then**
7:        **return false**
    return true

Algorithm 9 GoodCut2(G, B_1, B_2), takes a graph G with n vertices and m edges, balls $B_1 \subseteq B_2$, and determines whether recursing on B_2 and then deleting B_1 from our graph is good progress

1: **if** $V(B_2) \leq n^{\frac{1}{k}} |V(B_1)|^{\frac{k-1}{k}}$ **then**
2:     **return true**
3: **else**
4:     **return false**

---

all vertices which could potentially be in a cycle of length KR with v, according to the algorithm.

Now we explain Algorithm 9 (GoodCut2), which decides whether cutting out ball B_1 and recursing on B_2 constitutes good enough progress. This simply takes as input two balls B_1 and B_2 and decides whether recursing on B_2 and then deleting B_1 is good enough progress in trying to achieve a $\tilde{O}(n^{1+O(k^{-1})})$ total size of roundtrip covers. Here, we check only the vertex condition instead of the edge condition (different from Algorithm 4 GoodCut) because we have already reduced to the case where our graph G is approximately regular (Lemma 3.1).

Now we explain Algorithm 7 (BallGrow), which grows the balls E_v^i. on$[v] = $ **false** if vertex v has been resolved, i.e. we can ensure that for any u with $d(v \leftrightarrows u) \leq R$, that v and u are in a roundtrip ball of diameter $O(KR)$. Otherwise, on$[v] = $ **true**. We now grow balls $E_v^0, E_v^1, \cdots, E_v^K$ around v, up until line 8 is satisfied. Our main claim is that when we recurse on E_v^{i+1}, then we can safely remove all vertices in E_v^i. While the definition in line 6

$$E_v^i \leftarrow \{u \in V(G) : \text{on}[u] \text{ and Similar}(G, u, v, D, i, R)$$

$$\text{and } u \text{ reachable from } v \text{ through } E_v^i\}$$

may seem recursive, all we mean is to say that we run a search from v, only keeping vertices which are i-similar, i.e. Similar(G, u, v, D, i, R) is true.

Finally, our main algorithm Algorithm 5 (RoundtripCover2) first calls BuildSimilar to build the similarity data structure needed for BallGrow. It also removes vertices from G that were already resolved (i.e. in cycles of length $2KR$) to get a graph G'. Then it grows balls to partition G' and recurse.

5.2 Analysis

In this section we analyze the above algorithms. We first show that the number of similar vertices to any vertex v in G' (line 4) is at most $n^{\frac{k-1}{k}}$ with high probability.

LEMMA 5.1. *Consider an execution of* ROUNDTRIPCOVER2(G_0, k, r) *on an $\hat{n}$-vertex vertex graph G_0. Consider a recursive execution of* ROUNDTRIPCOVER2(G, k, r) *on an n-vertex m-edge graph G. Consider the resulting execution* BALLGROW(G', k, R, D) *(line 4). With probability at least $1 - \hat{n}^{-7}$ we have that for all $v \in V(G')$ that the number of vertices $u \in V(G')$ satisfying*

$$d(v, u) \leq KR \text{ and}$$

$$d(u, w) \leq 2KR \text{ for all } w \in S_v^j \text{ for all } 1 \leq j \leq 100 \log \hat{n}$$

is at most $n^{\frac{k-1}{k}}$.

PROOF. We follow the same approach as the proofs in Section 3. Consider a vertex $v \in V(G)$. Define

$$H_v^i := \{u \in V(G) : d(v, u) \leq KR \text{ and}$$

$$d(u, w) \leq 2KR \text{ for all } w \in S_v^j \text{ for all } 1 \leq j < i\},$$

i.e. all vertices $u \in V(G)$ which would "pass" the i-th level similarity test for v. Our main claim is that if $|H_v^i| \geq n^{\frac{k-1}{k}}$, then we have that $\frac{|H_v^{i+1}|}{|H_v^i|} \leq \frac{9}{10}$ with high probability. This implies the result, because if $|H_v^{100 \log \hat{n}}| \geq n^{\frac{k-1}{k}}$ still, then we have that

$$|H_v^{100 \log \hat{n}}| \leq \left(\frac{9}{10}\right)^{100 \log \hat{n}} n < 1,$$

an obvious contradiction.

Now we show that if $|H_v^i| \geq n^{\frac{k-1}{k}}$, then we have that $\frac{|H_v^{i+1}|}{|H_v^i|} \leq \frac{9}{10}$ with high probability. Note that by definition that $T_v^i = S^i \cap H_v^i$. It is direct to verify by a Chernoff bound that $|T_v^i| \geq 50 \log \hat{n}$ with probability at least $1 - \hat{n}^{-10}$ assuming that $|H_v^i| \geq n^{\frac{k-1}{k}}$. By the definition of S_v^i (a uniformly random subset of T_v^i of size $50 \log \hat{n}$) and symmetry we can think of S_v^i simply as a uniformly random subset of H_v^i of size $50 \log \hat{n}$.

We now argue that for at least $\frac{9}{10}$ fraction of vertices in $w \in H_v^i$ we have that

$$\Pr_{w' \in H_v^i} [d(w, w') \leq 2KR] \leq \frac{4}{5},$$

i.e. only $\frac{4}{5}$ fraction of vertices $w' \in H_v^i$ satisfy $d(w, w') \leq 2KR$. Assume the contrary for contradiction. By the Pigeonhole principle, there are at least

$$\left(\frac{9}{10} \cdot \frac{4}{5} - \frac{1}{2}\right) |H_v^i|^2 = .22|H_v^i|^2$$

(unordered) pairs of vertices $w, w' \in H_v^i$ such that both $d(w, w') \leq 2KR$ and $d(w', w) \leq 2KR$. By the Pigeonhole principle again, there must be a vertex $w \in H_v^i$ for which at least $.44|H_v^i|$ vertices w' satisfy both $d(w, w') \leq 2KR$ and $d(w', w) \leq 2KR$, so $d(w, w') \leq 4KR$. Now, note that $.44|H_v^i| \geq .44n^{\frac{k-1}{k}}$ by our condition. We argue that this is impossible because v should have been marked as not on with high probability in line 7. Indeed, the probability that v failed to get marked as not on is at most

$$\left(1 - \frac{.44n^{\frac{k-1}{k}}}{n}\right)^{\sum_{i=1}^{100 \log \hat{n}} |S^i|} \leq 1 - \hat{n}^{-20}$$

as desired.

Now, consider the $\frac{9}{10}$ fraction of vertices $w \in H_v^i$ with

$$\Pr_{w' \in H_v^i} [d(w, w') \leq 2KR] \leq \frac{4}{5}.$$

For each of these vertices, the probability that $d(w, w')$ for all $w' \in S_v^i$ is at most $\left(\frac{4}{5}\right)^{50 \log \hat{n}} \leq 1 - \hat{n}^{-10}$. By definition then, we have that $\frac{|H_v^{i+1}|}{|H_v^i|} \leq \frac{1}{10}$ by definition, as the $\frac{9}{10}$ fraction of vertices in H_v^i discussed in this paragraph will with high probability not be in H_v^{i+1}. □

We next claim that the ball growing scheme of Algorithm 7 satisfies some important conditions, which intuitively make the E_v^i look like balls of radius iR.

LEMMA 5.2. *Consider an execution of* ROUNDTRIPCOVER2(G_0, k, R) *on n-vertex m-edge graph G_0. Now, consider the resulting execution of* BALLGROW(G, k, R, D) *on graph G. We have that in the execution for all $v \in V(G)$ that*

1. $v \in E_v^0$.
2. $E_v^i \subseteq E_v^{i+1}$ *for $0 \leq i \leq K - 1$.*
3. *For $0 \leq i \leq K - 1$, if $u \in E_v^i$ and $d(u \leftrightarrows u') \leq R$, then $u' \in E_v^{i+1}$.*

PROOF. For the first claim, note that vertices $w \in S_v^j$ for all j satisfy $d(v, w) \leq KR$ by line 11 of BUILDSIMILAR. Therefore, v satisfies all conditions of being in E_v^i on line 6 of BALLGROW.

The second claim is obvious from looking at the the definition of E_v^i in line 6 of BALLGROW.

For the third claim, note that if $u \in E_v^i$ and $d(u \leftrightarrows u') \leq R$ then $d(v, u') \leq d(v, u) + R = (i + 1)R$. Also, for any $w \in S_v^j$ for $1 \leq j \leq 100 \log \hat{n}$ we have that $d(u', w) \leq d(u, w) + R \leq (i + 1 + K)R$. Hence $u' \in E_v^{i+1}$ as desired. □

We now show the analogue to Lemma 4.2, specifically that for some iteration $0 \leq i \leq K - 1$ in BALLGROW, we have that the condition in line 8 triggers.

LEMMA 5.3. *Consider an execution of* ROUNDTRIPCOVER2(G_0, k, R) *on n-vertex m-edge graph G_0. Now, consider the resulting execution of* BALLGROW(G, k, R, D) *on graph G. For some $0 \leq i \leq K - 1$ we have that the condition in line 8 is true, i.e. we get a good cut.*

PROOF. The computation proceeds the same way as in Lemma 4.2. Assume for contradiction that the condition in line 8 is never true, so

$$|V(E_v^{i+1})| > n^{\frac{1}{k}} |V(E_v^i)|^{\frac{k-1}{k}}.$$

By Lemma 5.2, we know that $|V(E_v^0)| \geq 1$, as $v \in E_v^0$. Therefore, one can check by induction that

$$|V(E_v^i)| \geq n^{1 - \left(\frac{k-1}{k}\right)^i}.$$

For $K = 10k \log k$ we have that

$$|V(E_v^K)| \geq n^{1 - \left(\frac{k-1}{k}\right)^K} > n^{\frac{k-1}{k}},$$

which contradicts Lemma 5.1. □

We turn to proving Theorem 8.

PROOF. We break the analysis into pieces. We show that executing ROUNDTRIPCOVER2(G, k, R) on a n-vertex m-edge graph G returns a $(O(k \log k), R)$ roundtrip cover C with total size of all balls at most $n^{1+O(\frac{1}{k})}$ in time $m^{1+O(\frac{1}{k})}$ with high probability.

Returns a $(O(k \log k), R)$ roundtrip cover. We first argue that in a call to BUILDSIMILAR(G, k, R) that for all vertices v where we marked on$[v] = $ **false** that v is properly resolved, i.e. for any vertex u with $d(v \leftrightarrows u) \le R$ that u and v are in a roundtrip ball of radius at most $(4K + 1)R$. Indeed, note that if we mark on$[v] = $ **false**, then there must have been a vertex w for which $w \in S^j$ for some j, and $d(v \leftrightarrows w) \le 2KR + 2KR = 4KR$. Then $d(u \leftrightarrows w) \le (4K + 1)R$. We have added the ball $B_w((4K + 1)R)$ to our roundtrip cover C, as desired (line 5).

The only other piece to verify is that when we mark on$[v] = $ **false** in an execution of BALLGROW(G, k, R, D) that v is properly resolved, i.e. that in some recursive subproblem we have that for all u with $d(v \leftrightarrows u) \le R$ that v and u are in a roundtrip ball of radius at most $O(KR)$. But this holds immediately by Lemma 5.2: if $v \in E^i_w$ for some w, and $d(v \leftrightarrows u) \le R$, then $u \in E^{i+1}_w$ as desired.

Total size of balls in C is $n^{1+O(\frac{1}{k})}$ with high probability. The analysis here follows closely to the corresponding paragraph in Section 4. We will show that the total size of all graphs processed in a single level of recursion the algorithm is at most $n^{\frac{k}{k-1}}$, where our initial call was ROUNDTRIPCOVER2(G, k, R) for a n-vertex m-edge graph G. Then, the total size of all graphs processed in the recursion is $\tilde{O}(n^{\frac{k}{k-1}})$, as the recursion depth is at most logarithmic. Then the bound on total size of balls in C follows as for a graph G with n vertices, the total size of balls added to C during BUILDSIMILAR(G, k, R) is at most

$$\sum_{j=1}^{100 \log n} |S^j| = \tilde{O}(n^{1+\frac{1}{k}}).$$

To show that the total size of all graphs processed at recursion depth ℓ in the algorithm is at most $n^{\frac{k}{k-1}}$, we use induction. Indeed, this holds at the bottom level of recursion. Consider an execution of BALLGROW(G, k, R, D) on an n-vertex m-edge graph G. Let $F_1^1, F_2^1, \cdots, F_t^1$ be all the balls E^i_v for which line 8 was satisfied (and we know that line 8 is satisfied for some i by Section 5.2). Let $F_1^2, F_2^2, \cdots, F_t^2$ be the corresponding balls E^{i+1}_v. We have that $\sum_{i=1}^{t} |V(F_i^1)| \le n$, as we marked all vertices in E^i_v as not on anymore if line 8 was satisfied for E^i_v and E^{i+1}_v. Additionally, by the condition of GOODCUT2, we have that $|V(F_i^2)| \le n^{\frac{1}{k}} |V(F_i^1)|^{\frac{k-1}{k}}$. By induction, the total sizes of all graphs processed at depth ℓ through recursion on $F_1^2, F_2^2, \cdots, F_t^2$ is at most

$$\sum_{i=1}^{t} |V(F_i^2)|^{\frac{k}{k-1}} \le \sum_{i=1}^{t} \left(n^{\frac{1}{k}} |V(F_i^1)|^{\frac{k-1}{k}} \right)^{\frac{k}{k-1}} \le \sum_{i=1}^{t} n^{\frac{1}{k-1}} |V(F_i^1)| \le n^{\frac{k}{k-1}}.$$

Can be implemented to run in time $m^{1+O(\frac{1}{k})}$ with high probability. The analysis in the above section on the total size of balls, we know that the total number of vertices in all graphs processed during the algorithm ROUNDTRIPCOVER2(G, k, R) is at most $n^{1+O(\frac{1}{k})}$. As we have reduced to the case of regular graphs through Lemma 3.1, the total number of edges in all graphs processed during

ROUNDTRIPCOVER2(G, k, R) is at most
$\tilde{O}(\delta n^{1+O(\frac{1}{k})}) \le \tilde{O}(m^{1+O(\frac{1}{k})})$ for $\delta = O(m/n)$.

We now argue that the non-recursive runtime of ROUNDTRIPCOVER2(G, k, R) on a graph G with n vertices and m edges is $\tilde{O}(m^{1+1/k})$. We start by analyzing Algorithm 6 (BUILDSIMILAR). We have in BUILDSIMILAR(G, k, R) that $\sum_{i=1}^{100 \log \hat{n}} |S^i| \le \tilde{O}(n^{1/k})$, where $\hat{n}$ is the number of vertices in the graph at the top level of recursion. Therefore, building a shortest path tree to and from all vertices in $\bigcup_i S^i$ takes $\tilde{O}(m^{1+1/k})$ time using Dijkstra's algorithm. Computing all the sets T^i_v clearly takes time

$$n \cdot \sum_{i=1}^{100 \log \hat{n}} |S^i| \cdot K = \tilde{O}(m^{1+1/k}).$$

We proceed to analyze Algorithm 8 (SIMILAR). This clearly takes $\tilde{O}(1)$ time per call, as we have precomputed all shortest path trees and distances to and from all vertices $u \in S^i$ in Algorithm 6. Also, Algorithm 9 (GOODCUT2) also obviously takes $O(1)$ time per call.

Now we analyze Algorithm 7 (BALLGROW). We can build the sets E^i_v by running any search from v, only keeping vertices u that satisfy SIMILAR(G, u, v, D, i, R). This takes time proportional to $\tilde{O}(\delta |E^{i+1}_v|)$, where we have used that each call to SIMILAR takes $\tilde{O}(1)$ time. In accounting for this runtime, we can push the contribution to the next recursion level (as we are recursing on E^{i+1}_v). Therefore, the total non-recursive runtime used is $\tilde{O}(m^{1+1/k})$ as claimed for an input graph with m edges.

Now, as the total number of edges over all graphs is $\tilde{O}(m^{1+O(\frac{1}{k})})$, the total runtime would also be $\tilde{O}(m^{1+O(\frac{1}{k})})$ as desired. □

We now use Theorem 8 to get multiplicative girth approximation and roundtrip spanners, proving Theorem 5 and Theorem 6.

PROOF OF THEOREM 5. We first show the result for unweighted graphs. To show this, run

ROUNDTRIPCOVER2($G, O(k), 2^i$) for $0 \le i \le O(\log n)$.

Now, set our estimate g' of the girth to be the smallest radius of any nontrivial ball that we had in a roundtrip cover. By the guarantees of ROUNDTRIPCOVER2, it is clear that $g \le g' \le O(k \log k) \cdot g$ as desired. It is clear that the algorithm runs in time $\tilde{O}(m^{1+\frac{1}{k}})$ by Theorem 8. We can extend this to weighted graphs by instead taking $0 \le i \le O(\log nW)$, where W is the maximum edge weight. This can be improved to $O(\log n)$ by the same method as done in [16], where they give a general reduction by contracting small weight strongly connected components and deleting large weight edges (see Section 5.1 in [16] for more details). □

PROOF OF THEOREM 6. We first show the result for unweighted graphs. It is easy to see that

$$\bigcup_{i=0}^{O(\log n)} \text{ROUNDTRIPCOVER2}(G, O(k), 2^i)$$

is an $O(k \log k)$ spanner with $\tilde{O}(n^{1+\frac{1}{k}})$ edges by Theorem 8. It is clear that the algorithm runs in time $\tilde{O}(m^{1+\frac{1}{k}})$. The extension to weighted graphs follows as in the above paragraph (proof of Theorem 5). □

6 CONCLUSION AND OPEN PROBLEMS

In this paper we provided multiple results on computing round-trip spanners and multiplicative approximations to the girth of an arbitrary directed graph. Our results all either improve running times, decrease the use of randomness, or improve the approximation quality of previous results. Ultimately, this work brings the state-of-the art performance of roundtrip spanners algorithms on directed graphs closer to matching that for undirected graphs.

An immediate open problem left open by our work is to fully close the gap between algorithmic guarantees for spanners of undirected graphs and roundtrip spanners of directed graphs and provide a deterministic algorithm which for all k in $\tilde{O}(mn^{1/k})$ time computes a $O(k)$ roundtrip spanner with $\tilde{O}(n^{1+1/k})$ edges. This paper resolves this problem for $k = \Omega(\log n)$ and makes progress on it for smaller values of k; it is still open to resolve it for all k.

Another key open problem is to further clarify the complexity of approximating the girth of a directed graph. Currently the only algorithms which provably outperform APSP for approximating the girth of a graph are Pachocki et. al. [16] and this paper. Consequently, all known girth approximation algorithms for directed graphs leverage techniques immediately applicable for spanner computation (with the sole possible exception of the algorithms of Section 3). Therefore, beyond improving roundtrip spanner routines to obtain an algorithm which can compute an $O(k)$-multiplicative approximation to the girth in $\tilde{O}(mn^{1/k})$, this suggests the even more challenging open problem of circumventing this "spanner barrier" to obtaining even faster running times. For undirected graphs, it possible to overcome this barrier in certain cases [9, 11, 13, 20]. However, some of the techniques used in these results are known not to extend to directed graphs, see e.g. [16, 20]. Consequently, further clarifying the complexity of girth approximation beyond the spanner barrier with either improved algorithms or new conditional lower bounds remains an difficult and interesting frontier.

One final open problem is to improve the parallel complexity of these routines. Previous work on the efficient construction of roundtrip spanners [16] provided such a result. Further, there have been recent advances in the efficient parallel computation of reachability in directed graphs [10, 12] and commute times of random walks. The combination of the ideas from these works with the results of this paper could be useful for obtaining further improvements for the efficient parallel computation of girth and roundtrip spanners [7].

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. 1993. On sparse spanners of weighted graphs. *Discrete Comput. Geom.* 9, 1 (1993), 81–100. https://doi.org/10.1007/BF02189308

[2] Surender Baswana and Sandeep Sen. 2003. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Automata, languages and programming*. Lecture Notes in Comput. Sci., Vol. 2719. Springer, Berlin, 384–396. https://doi.org/10.1007/3-540-45061-0_32

[3] Béla Bollobás. 1998. *Modern graph theory*. Graduate Texts in Mathematics, Vol. 184. Springer-Verlag, New York. xiv+394 pages. https://doi.org/10.1007/978-1-4612-0619-4

[4] L. Paul Chew. 1989. There are planar graphs almost as good as the complete graph. *J. Comput. System Sci.* 39, 2 (1989), 205–219. https://doi.org/10.1016/0022-0000(89)90044-5 Computational geometry.

[5] Edith Cohen. 1997. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.* 55, 3 (1997), 441–453. https://doi.org/10.1006/jcss.1997.1534 35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994).

[6] Michael B. Cohen, Jonathan A. Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. 2018. Solving Directed Laplacian Systems in Nearly-Linear Time through Sparse LU Factorizations. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. 898–909. https://doi.org/10.1109/FOCS.2018.00089

[7] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. 2017. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. 410–419.

[8] Lenore Cowen and Christopher G. Wagner. 2004. Compact roundtrip routing in directed networks. *J. Algorithms* 50, 1 (2004), 79–95.

[9] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. 2017. Finding Even Cycles Faster via Capped K-walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing* (Montreal, Canada) *(STOC 2017)*. ACM, New York, NY, USA, 112–120. https://doi.org/10.1145/3055399.3055459

[10] Jeremy T. Fineman. 2018. Nearly work-efficient parallel algorithm for digraph reachability. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. 457–470.

[11] Alon Itai and Michael Rodeh. 1977. Finding a Minimum Circuit in a Graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing* (Boulder, Colorado, USA) *(STOC '77)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/800105.803390

[12] Arun Jambulapati, Yang P. Liu, and Aaron Sidford. 2019. Parallel Reachability in Almost Linear Work and Square Root Depth. *CoRR* abs/1905.08841 (2019). arXiv:1905.08841 http://arxiv.org/abs/1905.08841

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

[14] Gary L. Miller, Richard Peng, and Shen Chen Xu. 2013. Parallel graph decompositions using random shifts. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*. 196–203. https://doi.org/10.1145/2486159.2486180

[15] James B. Orlin and Antonio Sedeño-Noda. 2017. An $O(nm)$ time algorithm for finding the min length directed cycle in a graph. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, PA, 1866–1879. https://doi.org/10.1137/1.9781611974782.122

[16] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. 2018. Approximating cycles in directed graphs: fast algorithms for girth and roundtrip spanners. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, PA, 1374–1392. https://doi.org/10.1137/1.9781611975031.91

[17] David Peleg. 2000. *Distributed computing*. SIAM Monographs on Discrete Mathematics and Applications, Vol. 5. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA. xvi+343 pages. A locality-sensitive approach.

[18] Iam Roditty, Mikkel Thorup, and Uri Zwick. 2008. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms* 4, 3 (2008), Art. 29, 17. https://doi.org/10.1145/1367064.1367069

[19] Liam Roditty, Mikkel Thorup, and Uri Zwick. 2005. Deterministic constructions of approximate distance oracles and spanners. In *Automata, languages and programming*. Lecture Notes in Comput. Sci., Vol. 3580. Springer, Berlin, 261–272. https://doi.org/10.1007/11523468_22

[20] Liam Roditty and Virginia Vassilevska Williams. 2012. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, New York, 833–845.

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

[22] Virginia Vassilevska Williams and Ryan Williams. 2010. Subcubic equivalences between path, matrix, and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science—FOCS 2010*. IEEE Computer Soc., Los Alamitos, CA, 645–654.