

Approximating Cycles in Directed Graphs: Fast Algorithms for Girth and Roundtrip Spanners

Jakub Pachocki ^{*} Liam Roditty [†] Aaron Sidford [‡]
Roei Tov [§] Virginia Vassilevska Williams [¶]

Abstract

The *girth* of a graph, i.e. the length of its shortest cycle, is a fundamental graph parameter. Unfortunately all known algorithms for computing, even approximately, the girth and girth-related structures in directed weighted m -edge and n -node graphs require $\Omega(\min\{n^\omega, mn\})$ time (for $2 \leq \omega < 2.373$). In this paper, we drastically improve these runtimes as follows:

- **Multiplicative Approximations in Nearly Linear Time:** We give an algorithm that in $\tilde{O}(m)$ time computes an $\tilde{O}(1)$ -multiplicative approximation of the girth as well as an $\tilde{O}(1)$ -multiplicative roundtrip spanner with $\tilde{O}(n)$ edges with high probability (w.h.p).
- **Nearly Tight Additive Approximations:** For unweighted graphs and any $a \in (0, 1)$ we give an algorithm that in $\tilde{O}(mn^{1-a})$ time computes an $O(n^a)$ -additive approximation of the girth, w.h.p. We show that the runtime of our algorithm cannot be significantly improved without a breakthrough in combinatorial boolean matrix multiplication. We also show that if the girth is $O(n^a)$, then the same guarantee can be achieved via a deterministic algorithm.



Our main technical contribution to achieve these results is the first nearly linear time algorithm for computing roundtrip covers, a directed graph decomposition concept key to previous roundtrip spanner constructions. Previously it was not known how to compute these significantly faster than $\Omega(mn)$ time. Given the traditional difficulty in efficiently processing directed graphs, we hope our techniques may find further applications.

1 Introduction

The *girth* of a graph G is the length of the shortest cycle in G . It is a natural and fundamental graph parameter that has been extensively studied (see Diestel’s book [19] for a discussion) with research on its computation dating back to the 1960s. Perhaps, the most straightforward algorithm for the girth is simply to compute All-Pairs Shortest Paths (APSP). Surprisingly, this simple relationship leads to the best known algorithm for girth in

a n -node graph with nonnegative weights: the breakthrough APSP algorithm of Williams [55] can compute the girth in $n^3/2^{\Theta(\sqrt{\log n})}$ time. For sparse weighted graphs with m edges, an $O(mn)$ runtime was recently obtained by Orlin and Sedeño-Noda [38], improving upon an $O(mn + n^2 \log \log n)$ runtime that follows from using the best known sparse APSP algorithm [43].

When the graph can be dense, all known girth algorithms run in $n^{3-o(1)}$ time (unless the weights are small integers bounded in absolute value by M in which case an $\tilde{O}(Mn^\omega)$ time is known [46]). Vassilevska W. and Williams [53] explained this by showing that the girth in weighted graphs is equivalent to APSP in the sense that if one of the two problems has a “truly subcubic”, $O(n^{3-\varepsilon})$ time algorithm for some $\varepsilon > 0$, then both do. As it is a longstanding open problem whether APSP has a truly subcubic time algorithm, computing the girth in $O(n^{3-\varepsilon})$ time would be a huge breakthrough.

One might wonder whether very sparse graphs allow for much faster than $O(mn)$ time girth algorithms (the above discussion says that dense graphs probably do not). Very recently, Lincoln et al. [32] showed that if there exists any integer L such that for sparsity $m = \Theta(n^{1+1/L})$, one can obtain an $O(mn^{1-\varepsilon})$ time algorithm for the girth of weighted directed graphs, then Max-Weight k -Clique would have surprisingly fast algorithms. Weighted k -Clique is a notoriously difficult problem: it is the basis of several conditional lower bounds. Here, assuming its hardness implies that $mn^{1-o(1)}$ time is likely necessary for computing the girth exactly, for almost any sparsity.

Because of the strong barriers for exact computation, efficient *approximation* algorithms are of interest. Fast approximations for the girth in *undirected* graphs are possible and have been studied extensively. It is well known ([2]) that for any integer $k \geq 1$, every weighted undirected n -node graph G contains an $O(n^{1+1/k})$ edge $(2k-1)$ -spanner, i.e. a subgraph that $(2k-1)$ -approximates all pairwise distances in G . Such $(2k-1)$ -spanners can be computed in $\tilde{O}(mn^{1/k})$

^{*}OpenAI, merettm@gmail.com

[†]Bar Ilan University, liamr@macs.biu.ac.il

[‡]Stanford University, sidford@stanford.edu

[§]Bar Ilan University, roei81@gmail.com

[¶]MIT CSAIL, virgic@csail.mit.edu

time [51], and immediately imply efficient approximation algorithms for girth in undirected weighted graphs. In unweighted undirected graphs even better results are known. Itai and Rodeh [28] gave an $O(n^2)$ time additive 1-approximation algorithm, and follow-up work [47, 33] developed even more efficient, combinatorial, truly sub-quadratic, approximation algorithms.

Although impressive approximate girth algorithms are possible for undirected graphs, they all exploit properties particular to undirected graphs. In particular, not only do sparse spanners exist in undirected graphs, but it is also known [9] that undirected graphs with $\Omega(n^{1+1/k})$ edges must contain a $2k$ -cycle. This fact is at the heart of obtaining fast girth approximation algorithms.

In contrast, directed graphs do not always contain sparse spanners and dense digraphs might not have any cycles at all (a directed bipartite clique is an example of each). It is completely unclear what (if any) structure there is to exploit in directed graphs to obtain fast girth approximation algorithms. Due to the close relationship between girth and APSP [53], a-priori it could be that the girth problem in directed graphs suffers from the same problem as APSP in directed graphs and no finite approximation is even possible without resolving a major open problem about BMM.

A potential saving point is that while directed graphs may not contain sparse spanners, they do contain sparse *roundtrip spanners*. That is, if one uses $d(u, v) + d(v, u)$ for the distance between u and v instead of $d(u, v)$, then one can obtain a very similar result for directed graphs as in the undirected case: for all $k \geq 1$ and $\varepsilon > 0$, every n node directed graph G contains a $(2k + \varepsilon)$ -roundtrip spanner on $O(k^2/\varepsilon n^{1+1/k} \log n W)$ edges (for nonnegative integer weights bounded by W). Furthermore, since roundtrip distances form a metric on the vertices, there are emulators for roundtrip distances with the same sparsity quality trade-offs as the best spanners (although they might be expensive to compute as computing the metric itself solves APSP).

Unfortunately, while roundtrip spanners have been known to exist for over a decade (and roundtrip emulators for over 25 years [40, 41, 2] through the aforementioned reduction to spanners), the fastest algorithms for computing them run in $O(mn)$ time, essentially the time to solve APSP.

1.1 Our Results In this paper we provide the first non-trivial approximation algorithms for computing the girth and related properties on directed graphs that run substantially faster than the roughly $\Omega(mn)$ time currently needed to solve APSP on a n -node, m -edge directed graph with non-negative weights.

We show how to compute $\tilde{O}(1)$ -multiplicative approximations to the *girth* and construct *multiplicative roundtrip* spanners in nearly linear time (See Section 1.1.1), and we show how to compute additive approximations to the girth in time that is nearly tight under standard assumptions (See Section 1.1.2). To achieve these results we provide the first nearly linear time algorithms for computing roundtrip covers, a natural directed graph decomposition notion in prior work on roundtrip spanners [18, 45] (See Section 1.2).

1.1.1 Multiplicative Approximations in Nearly Linear Time [53] showed that the girth problem in weighted graphs is subcubically equivalent to APSP in general graphs. Thus, obtaining a truly subcubic algorithm for girth in weighted graphs would imply a major breakthrough, and is a daunting task for current techniques. In fact, *no* nontrivial combinatorial approximation algorithms were previously known even for the restricted case of unweighted directed graphs. On the other hand, we show how to obtain an $O(\log n)$ approximation to the girth in weighted graphs in slightly super-linear time (See Theorem 1.1). Setting $k := \log n$ in this theorem allows us to compute an $O(\log^2 n)$ approximation to the girth in *nearly linear* time.

THEOREM 1.1. (MULTIPLICATIVE GIRTH APPROXIMATION) *For any n -node, m -node directed graph with nonnegative integer edge weights, with unknown girth g and integer $k \geq 1$, in time $O(mn^{1/k} \log^5 n)$ we can compute an estimate \bar{g} such that $g \leq \bar{g} \leq O(k \log n) \cdot g$ with high probability.*




Thus, by suffering only a logarithmic loss in the accuracy, one can obtain a girth estimate much faster than mn , a runtime that is likely optimal for exact computation.

Using our new directed graph decomposition algorithm we also show how to compute multiplicative roundtrip spanners in nearly linear time. A *spanner* is a sparse subgraph that preserves the distances of the original graph with some multiplicative or additive approximation. Since even preserving the asymmetric reachability structure of directed graphs may require $\Omega(n^2)$ edges (e.g. the complete directed bipartite graph), no sparse spanner yielding a finite multiplicative approximation is possible. Instead, we consider spanners under the *roundtrip* distance metric, i.e. *roundtrip* spanners.

Given vertices u and v the *roundtrip* distance between u and v is the distance from u to v plus the distance from v to u . Roundtrip distances were studied by Cowen and Wagner [18] in the context of routing. Later Roditty, Thorup and Zwick [45] obtained roundtrip spanners for directed graphs that are almost as good

in terms of their sparsity/approximation tradeoff as the spanners of undirected graphs [2]: $(2k-\varepsilon)$ -multiplicative approximation with $\tilde{O}((k^2/\varepsilon)n^{1+1/k})$ edges for any integer $k \geq 1$ and $\varepsilon \in (0, 1)$. However, the construction of these spanners requires precomputing the roundtrip distances between *all* pairs of vertices, resulting in a running time of roughly $\Omega(mn)$. We show how to nearly match this size/approximation tradeoff while decreasing the time needed to construct them to nearly linear in the number of edges in the graph.

 **THEOREM 1.2. (MULTIPLICATIVE ROUNTRIP SPANNERS)** *Given any n -node, m -edge directed graph with nonnegative integer edge weights and any $k \geq 1$ in time $O(mn^{1/k} \log^5 n)$ we can compute an $O(k \log n)$ -multiplicative roundtrip spanner with $O(n^{1+1/k} \log^2 n)$ edges with high probability.*

Our techniques are inherently parallelizable, and we provide the first work-efficient parallel algorithms for computing both the approximate girth and the strongly connected components of an unweighted directed graph with depth linear in the diameter of the computed objects (See Section 5.2).

1.1.2 Nearly Tight Additive Approximations

Our techniques can also be used to obtain fast combinatorial algorithms that achieve *additive approximations* of the girth on unweighted graphs as follows. Let $a \in (0, 1)$ be any constant and suppose the girth g is $< n^a / \log n$. Then, the algorithm from Theorem 1.1 will return w.h.p. in $\tilde{O}(m)$ time a cycle of length $O(n^a)$, which is (trivially) an additive $O(n^a)$ approximation of g . If on the other hand $g \geq n^a / \log n$, then if we take a random sample S of $Cn^{1-a} \log^2 n$ nodes for large enough constant C , then w.h.p. S will contain a vertex of the shortest cycle. Then, running BFS from each node of S will find the shortest cycle containing a node of S and hence compute g exactly:

COROLLARY 1.1. (ADDITIVE GIRTH APPROXIMATION) *For any unweighted n -node, m -edge directed graph with unknown girth g and $a \in (0, 1)$ in time $\tilde{O}(mn^{1-a})$ we can compute an estimate \bar{g} such that $g \leq \bar{g} \leq g + O(n^a)$ with high probability.*

Our algorithms for Theorem 1.1 and Corollary 1.1 are combinatorial, but randomized. It is unclear whether they can be derandomized without incurring a large runtime cost. In particular, our algorithms use sampling to crudely estimate the sizes of reachability sets for all vertices in the graph. As far as we know, there are no faster deterministic ways to do this in the worst case than explicitly computing the reachability

sets which requires $\Omega(\min\{n^\omega, mn\})$ time. We partially derandomize Corollary 1.1 using different techniques:

THEOREM 1.3. (DETERMINISTIC ADDITIVE GIRTH APPROXIMATION) *There is a deterministic combinatorial algorithm that for any unweighted n -node m -edge directed graph with unknown girth g and parameters $a, \epsilon \in (0, 1)$ computes in $\tilde{O}(\epsilon^{-2}mn^{1-a})$ time an estimate \bar{g} such that $g \leq \bar{g} \leq g + O(n^a)$ if $g \leq n^a$ and $g \leq \bar{g} \leq (1 + \epsilon)g$ if $g > n^a$.*

A natural question is whether the $\tilde{O}(mn^{1-a})$ runtime for n^a -additive approximation is necessary. Surprisingly, we show that when it comes to combinatorial algorithms, Theorem 1.3 and Corollary 1.1 are optimal up to constant factors in the additive error, barring a breakthrough in BMM algorithms:

THEOREM 1.4. (HARDNESS FOR IMPROVING ADDITIVE RUNNING TIME) *Suppose there is a combinatorial algorithm for some $\epsilon > 0$ and $a = 1/2$ that computes an additive $n^a - 1$ approximation to the girth of any unweighted n -node m -edge directed graph in $O(mn^{1-a-\epsilon})$ time. Then for some constant $\delta > 0$ there is an $O(n^{3-\delta})$ time combinatorial algorithm for $n \times n$ BMM.*

1.2 Algorithmic Techniques : Roundtrip Covers in Nearly Linear Time Our key technical contribution towards achieving the majority of our algorithmic results is the first nearly linear time algorithm for computing *roundtrip covers* of directed graphs. Informally, a roundtrip cover is a decomposition of a directed graph into an overlapping collection of *balls*, i.e. roundtrip distance induced subgraphs. It is required that the radius, or maximum roundtrip distance, in each ball be bounded and that any pair of vertices of bounded roundtrip distance appear together in some ball (See Section 5 for formal definition). Computing such covers naturally yields multiplicative roundtrip spanners and has been considered in previous work [18, 45].

Unfortunately, all known roundtrip cover computation algorithms prior to this paper ran in at least $\Omega(mn)$ time. It was not clear how to efficiently manipulate the roundtrip metric for the purposes of computing such decompositions. It seemed that, in the worst case, one would have to compute almost the entire roundtrip metric explicitly, i.e. solve APSP.

We overcome this difficulty through a careful application of a few techniques. The first is natural: cluster the graph by growing balls of exponentially distributed radii or using exponential distribution based clustering techniques. (Similar ideas were used recently for parallel algorithms for undirected graph decomposition [37] and directed maximum flow [22]). This does not distort too

many roundtrip distances, but may fail to produce clusters of significantly smaller size. The second is our key insight: we show that if we carefully seed such a clustering routine we can ensure that we either find a large cluster with small roundtrip diameter or we break the graph into significantly smaller pieces while sufficiently preserving roundtrip distances. Unfortunately, naively implementing such a procedure would be expensive (i.e. involve solving APSP). To circumvent this, we use another trick: a known sampling based approach to estimate the fraction of vertices that each vertex can reach or is reachable by within a given distance and show this suffices to pick seeds for clustering.

In short, we achieve our multiplicative approximations via a delicate combination of several powerful tools that have been defined and used before: (1) low diameter graph decompositions first introduced in [4], (2) using the exponential distribution for decomposition (e.g. in [34, 6, 37, 22]), (3) recursive graph decomposition (e.g. in [6, 24, 22]), and (4) sampling based reachability set estimation ([13]). However, despite the prevalence of this machinery, it was an open question whether or not it could be leveraged to yield *any* running time improvement for the directed problems we consider. It was unclear a-priori if there was structure to exploit to quickly decompose the roundtrip metric and if the problems we consider were as hard as APSP.

Our key contribution is to show that this is not the case and there is in fact a way to rapidly reveal non-trivial directed graph structure sufficient to achieve our results. There are several pitfalls that occur when naively applying standard machinery to this problem and we believe the strength of our result is to show how to methodically overcome them (see Section 3). The lack of fast combinatorial primitives for directed graphs is occasionally referenced as indicative of the gap between recent progress on approximate undirected network optimization problems [12, 35, 30, 48, 29, 42, 49] and directed problems [36, 31, 15]. We hope our results and the insights that underly them may find future use.

1.3 Additional Related Work For unweighted graphs, in the 1970s Itai and Rodeh [28] showed that the girth can be computed in $O(nm)$ time via BFS, or in $O(n^\omega) \leq O(n^{2.373})$ -time using fast matrix multiplication [17, 52, 26]. These are still the best run-times for the problem. Similarly to the relationship to APSP, [53] showed that the girth in unweighted graphs is subcubically equivalent to Boolean Matrix Multiplication (BMM). A large open question in BMM is whether there exist truly subcubic “combinatorial” algorithms, that can avoid the sophisticated but often impractical

tools for Strassen-like fast matrix multiplication (e.g. [17, 52, 26]). The reduction from [53] shows that either both BMM and girth have truly subcubic combinatorial algorithms, or neither of them does.

For the girth in undirected unweighted graphs, besides Itai and Rodeh’s [28] original $O(n^2)$ time additive 1-approximation algorithm, Roditty and Vassilevska W. [47] presented an $\tilde{O}(n^3/m)$ -time additive 3-approximation algorithm. The additive 1-approximation of [28] is also a multiplicative 4/3-approximation. Lingas and Lundell [33] presented the first algorithm that breaks the quadratic time bound of [28], at the price of a weaker approximation: their algorithm runs in $\tilde{O}(n^{3/2})$ time and returns a multiplicative 8/3-approximation. Roditty and Vassilevska W. [47] presented an $\tilde{O}(n^{5/3})$ -time deterministic multiplicative 2-approximation algorithm. They also showed how to obtain a less than 2 multiplicative approximation in truly subquadratic time for triangle-free graphs.

The history of combinatorial algorithms for BMM of $n \times n$ matrices is as follows. Bansal and Williams [5] obtained an $O(n^3/\log^{2.25} n)$ time combinatorial algorithm improving on the 40-year record of $O(n^3/\log^2 n)$ by the Four-Russians Algorithm [3]. The result of [5] was further improved by Chan [11] to $O((n^3/\log^3 n) \log^{O(1)} \log n)$ time and most recently by Yu [56] to $\tilde{O}((n^3/\log^4 n) \log^{O(1)} \log n)$.

Obtaining a truly subcubic time algorithm for APSP is among the most studied longstanding open problems in graph algorithms. In the 1970s Fredman [25] showed that the $O(n^3)$ time classical Floyd-Warshall algorithm is not optimal by giving an $O(n^3(\log \log n / \log n)^{1/3})$ time running time. Many polylogarithmic improvements followed, the last being $O(n^3 \log^3 \log n / \log^2 n)$ by Chan [10]. Two years ago, Williams [55] used techniques from circuit complexity, namely the polynomial method, to shave all polylogs, thus obtaining the current best bound for APSP, $n^3/2^{\Theta(\sqrt{\log n})}$.

Spanners in undirected weighted graphs were first studied by Awerbuch [4] and Peleg and Schäffer [39]. Althöfer et al. [2] showed that for every integer $k \geq 1$, every n -node graph, even if it is weighted, contains a multiplicative $(2k - 1)$ -spanner on $O(n^{1+1/k})$ edges. This result is optimal, conditioned on a well-known (and partially proven [54]) conjecture by Erdős [23] about the existence of graphs of high girth.

1.4 Organization The remainder of the paper is structure as follows. We introduce notation in Section 2, provide an overview of our approach in Section 3, and show how to compute roundtrip covers in Section 4. We then provide our algorithms for multiplicative approx-

imations in Section 5 and our algorithms for additive approximations in Section 6. We conclude with our lower bounds in Section 7.

2 Preliminaries

Here we introduce various terminology we use throughout the paper.

Graphs: Throughout this paper we let $G = (V, E, l)$ denote a directed graph with vertices V , edges $E \subseteq V \times V$, and non-negative edges lengths $l \in \mathbb{R}_{\geq 0}^E$. At times we consider unweighted graphs, that is graphs in which $l_e = 1$ for all $e \in E$ and in this case we will omit the l altogether.

Distances: We let $d_G(u, v)$ denote the (shortest path) distance from u to v in G and we abbreviate this as $d(u, v)$ when G is clear from context. At times we consider shortest path distances over edge subgraphs $F \subseteq G$ and write $d_F(u, v)$ to denote the length of the shortest path from u to v using only the edges in F . In all cases we define $d(u, v) = \infty$ if there is not a path from u to v .

Roundtrip Spanners: For $a, b \in V$ we refer to $d(a \rightrightarrows b) \stackrel{\text{def}}{=} d(a, b) + d(b, a)$ as the roundtrip distance between a and b . We call a subgraph $S \subseteq E$ an α -multiplicative roundtrip spanner if $d_S(a \rightrightarrows b) \leq \alpha \cdot (d_G(a \rightrightarrows b))$ for all $a, b \in V$.

Additive Approximation: We call an estimate \tilde{g} an α -additive approximation to the girth g of a directed graph if $g \leq \tilde{g} \leq g + \alpha$.

Distance Measures: For a directed graph $G = (V, E, l)$ we call $\min_{v \in V} \max_{v' \in V} d(v, v')$ the radius of G . We call $\max_{v, v' \in V} d(v, v')$ the diameter of G .

Balls: For a given metric, a *ball* of radius r around v is the set of vertices within distance r of v . We generally use the term ‘ball’ to refer to balls in the roundtrip metric. For a directed graph G , we use $\text{inball}_G(v, r)$ and $\text{outball}_G(v, r)$ to denote the subsets of vertices of G that can reach v within distance r or be reached from v within distance r , respectively.

Trees: Given a directed graph $G = (V, E, l)$ we call a $T_{\text{out}} \subseteq E$ an out-tree with root $r \in V$ if the edges form an undirected tree and are all oriented away from r (i.e. there is a r to v path for every node v in the tree). Similarly, we call T_{in} an in-tree with root $r \in V$ if the edges form an undirected tree and are all oriented towards $r \in V$ (i.e. there is a v to r path for every node v in the tree).

Paths and Cycles: A directed (simple) path $P = \langle u = v_1, v_2, \dots, v_k = v \rangle \subseteq V$ from u to v is an ordered set of vertices, where for every $i \in \{1, \dots, k-1\}$, $(v_i, v_{i+1}) \in E$. A cycle $C = \langle u = v_1, v_2, \dots, v_k = v \rangle$ is a direct (simple) path with an additional requirement that $(v_k, v_1) \in E$. If P is a path and $u, v \in P$ such

that u precedes v in P then we denote by $P(u, v)$ the subpath of P from u to v . If P_1 and P_2 are paths in G , then we denote by $P_1 \cdot P_2$ the concatenation of P_1 and P_2 .

Running Times: We use \tilde{O} -notation to hide logarithmic factors, i.e. $\tilde{O}(f(n)) = O(f(n) \log^c f(n))$.

Probability: We use *with high probability* (w.h.p) to denote that an event happens with probability at least $1 - 1/O(\text{poly}(n))$ where n is the size of the input to the problem.

3 Overview of the Approach

Our approach for computing multiplicative roundtrip spanners is broadly inspired by the following simple general strategy for computing spanners in undirected unweighted graphs:

1. Repeat until there are no vertices left:

- Grow a ball of random radius from a vertex.
- Add the edges in the computed shortest path tree to the spanner.
- Remove all vertices in the ball from the graph.

2. Recurse on the subgraph induced by the edges that have endpoints in different balls.

If the radii are chosen appropriately one can show that the shortest path trees approximately preserve the distance between the endpoints of all edges inside a ball and that not too many edges are cut (i.e. have endpoints in different balls). While there is a great body of work on efficiently constructing spanners with many desirable properties [7, 14, 20, 21, 51], this simple strategy suffices to provide a polylogarithmic multiplicative spanner in nearly linear time.

Unfortunately there are two serious issues that prevent us from easily extending this approach to compute roundtrip spanners for directed graphs:

- Recursing on cut edges does not work (or even make sense).
- There may be problematic vertices that are at a small distance to (or from) all the vertices, but have a large roundtrip distance to every vertex.

We derive our algorithm by carefully addressing these two issues.

Issue #1: How to Recurse? The first issue is immediate. If we grow multiple balls in a directed graph and attempt to recurse on cut edges, it may be the case that we disconnect the graph and the roundtrip distances for the cut edges become infinite.



Consequently, if we recurse on the cut edges alone we simply do not have enough information to recover the path information we lost. Therefore, it is not clear if there is any reasonable way to recurse on the edges that we may distort.

To alleviate this issue we instead build a randomized scheme where we reason directly about the probability of cutting or distorting the roundtrip distance between any particular pair of vertices. Building on previous work on graph decomposition [34, 6, 37] and directed maximum flow [22] we use the fact that if we grow a ball where the radii are chosen from an exponential distribution then we can directly reason about the probability that removing that ball cuts a cycle. We then proceed to repeatedly grow balls of exponential radii, removing them in each iteration. Since the exponential distribution is memoryless, we can show that the probability that this approach cuts a cycle depends only on the parameter of the exponential distribution we use.

For completeness, in Appendix C we prove formally that repeated exponential ball growing works. For our algorithms we instead use a slightly more sophisticated clustering technique described in Section 4.1. This scheme works for similar reasons, but is more easily parallelizable. Ultimately, both techniques allow us to reason about the probability of destroying a cycle (rather than an edge) and repeat until all cycles corresponding to the roundtrip distances between vertex pairs are preserved with high probability. The difficulty remains in ensuring that we can actually terminate such a procedure in a small number of iterations (i.e. Issue #2).

Issue #2: How to Avoid Problematic Vertices?

The second issue seems even more troubling. Suppose that there is a graph with non-trivial cycle structure we would like to approximate. To create a harder instance one could simply create a new graph by adding many new vertices each of which has one short length edge to every original vertex and one long length edge from every vertex. Clearly these new vertices do not affect the cycle structure of the original graph that we wish to approximate. However, any shortest path query from these new vertices will quickly explore the entire graph. Consequently, starting any sort of clustering from these vertices could be quite expensive in terms of running time, yet reveals very little information about the graph's cycle structure.

Even if we take a different approach and simply attempt to improve the running time of constructing the roundtrip spanners from prior work [45], a similar issue arises. Here the immediate issue is that the algorithm computes balls in the roundtrip metric. However, to do

this, again we need to explore many vertices at a large distance in the roundtrip metric for analogous reasons.

To alleviate this issue, we use sampling to estimate, in near linear time, the fraction of vertices in $|V|$ of $O(r)$ -balls around all nodes, up to a small additive error ϵ . This can be done in nearly linear time due to a clever technique of Cohen [13] (For completeness, see Appendix however, we give a self-contained construction tailored to our purposes in Section A.) This allows us to find the problematic vertices that can reach many vertices at distance r yet are reachable by few at distance r (or vice versa). By using this technique to find problematic vertices we can better bias the seeds of our decomposition routines and make more progress in nearly linear time. This is crucial to our algorithm.

Building the Algorithm Combining our ideas to deal with these two issues yields our algorithm. We estimate for every vertex the number of vertices at distance $O(r)$ both from and to it. In one case there is a vertex that can both reach and is reachable by many vertices. In this case, we can compute a large enough ball (in the roundtrip metric) that contains vertices of small roundtrip distance, and then we recurse on the rest of the vertices outside the ball. In the other case, there are many vertices that either do not reach or are not reached by too many vertices at distance $O(r)$ and we can grow clusters from them and recurse on all the clusters. In either case we show that we do not need to recurse too many times and that ultimately, with constant probability (see Section 4.3), any particular pair of vertices at small roundtrip distance is together in a cluster. Repeating this procedure multiple times yields our nearly linear time roundtrip cover algorithm (see Section 4).

We use our roundtrip cover algorithm to compute multiplicative roundtrip spanners and obtain multiplicative estimates of the girth. A naive application of our procedure would yield a logarithmic dependence on the range of lengths in the graph. To avoid this, we show how to break a directed graph into smaller graphs reducing to subproblems where lengths vary only polynomially in the number of vertices. Furthermore, we show that our algorithm is inherently parallel and we obtain new work / depth tradeoffs for these problems. As discussed, this also yields faster additive approximation for the girth, though new insights are needed to obtain deterministic results (see Section 5). As discussed in the introduction, our roundtrip cover algorithm is also used to compute additive approximations to the girth (see Section 6).

4 Roundtrip Covers

In this section we provide our main results on graph partitioning. In particular we show how to efficiently construct roundtrip covers, first introduced in [45].

DEFINITION 4.1. (ROUNDTRIP COVERS, DEFINITION 2.4 IN [45]) *A collection \mathcal{C} of balls is a (k, R) -roundtrip-cover of a directed graph $G = (V, E, l)$ if and only if each ball in \mathcal{C} is of radius at most kR , and for every $u, v \in V$ such that $d_G(u \rightrightarrows v) \leq R$, there is a ball $B \in \mathcal{C}$ such that $u, v \in B$.*

The main result of this section is the following theorem, stating that we can construct a $(O(k \log n), R)$ -roundtrip-cover with high probability in $\tilde{O}(mn^{1/k})$ time.

THEOREM 4.1. (FAST ROUNDTRIP COVER) *The algorithm FAST-ROUNDTRIP-COVER(G, k, R), returns a collection \mathcal{C} that is an $(O(k \log n), R)$ -roundtrip-cover of directed graph $G = (V, E, l)$, w.h.p. in time $O(mn^{1/k} \log^4 n)$. Moreover, every vertex $v \in V$ belongs to $O(n^{1/k} \log n)$ elements of \mathcal{C} .*

Note that if G has integer edge lengths between 0 and U we can immediately apply Theorem 4.1 for every value of R that is a power of 2 and obtain $O(k \log n)$ -multiplicative roundtrip spanners with $O(n^{1+1/k} \log^2 n \log U)$ edges in time $O(mn^{1/k} \log^4 n \log U)$ as well as compute an $O(k \log n)$ multiplicative approximation to the girth in the same running time. Consequently, proving Theorem 4.1 encapsulates much of the difficulty in achieving our desired algorithmic results. However, in Section 5 we show how a more careful application of Theorem 4.1 yields even stronger results, completely removing the dependence on U .

The remainder of this section is dedicated to providing the algorithm FAST-ROUNDTRIP-COVER and proving Theorem 4.1. First in Section 4.1 we provide our main graph clustering tool, then in Section 4.2 we provide our technique for estimating the fraction of vertices reachable to and from each vertex at some radius. Finally, in Section 4.3 we put these tools together to provide FAST-ROUNDTRIP-COVER and prove Theorem 4.1.

4.1 Clustering Here we provide the primary clustering/partitioning technique we use for our algorithm. We provide an algorithm that partitions the vertices into regions of bounded radius of our choice centered around a chosen subset of the vertices so that the probability of separating any two vertices of bounded roundtrip distance is small. The ability to control the radii and choose the starting vertices is key to deriving our algorithm.

As discussed in the introduction we use an exponential-distribution-based clustering procedure so that we can argue directly about the probability of cutting any particular cycle. This allows us to apply this procedure multiple times and argue by union and Chernoff bounds that with high probability we do not cut any cycle that we want to approximate, and thus obtain a good approximation of any relevant cycle. However, rather than simply growing balls of exponentially distributed radius and repeating (as discussed in Section 3), we provide a different scheme in the flavor of [37, 22] that better parallelizes. For completeness we complement our analysis with a proof that this sequential ball growing scheme also works in Appendix C.

Our algorithms, CLUSTER-OUT and CLUSTER-IN are given in Figure 1. Given a graph $G = (V, E, l)$, a set of vertices $S \subseteq V$ and a target radius r , the algorithm uses the exponential distribution to assign vertices in G to clusters for each of the $v \in S$. The assignment is done in a way that ensures that these clusters each have bounded radius. By our choice of assignment rule and distribution we formally show that the probability that two vertices of small roundtrip distance are not in the same cluster is sufficiently small.

$(V_1, \dots, V_t) = \text{CLUSTER-OUT}(-\text{IN})(G, S, r)$, where $G = (V, E, l)$ is a directed graph, $S \subseteq V$ and $r > 0$.

1. Set $\beta := \log(n)/r$.
2. For every vertex $v \in S$, pick $x_v \sim \text{Exp}(\beta)$.
3. For each vertex $u \in V$, assign u to the cluster rooted at the vertex $v \in S$ which minimizes $-x_v + d_G(v, u)$, unless that quantity is positive; in that case, do not assign u to any cluster. (use $d_G(u, v)$ for CLUSTER-IN)
4. Let V_1, \dots, V_{t-1} be the clusters produced by the above step.
5. Return $(V_1, \dots, V_{t-1}, V \setminus \bigcup_i V_i)$.

Figure 1: The clustering algorithm.

In the remainder of this section we formally analyze this algorithm proving Lemma 4.1. The analysis we present is very similar to that of [37] and uses a subset of the ideas of [22]. The main difference is that we start only from a subset of the vertices S . Our analysis makes use of several facts regarding the exponential distribution which for completeness we prove in Appendix B.

LEMMA 4.1. *Let (V_1, V_2, \dots, V_t) be the partition of V returned by CLUSTER-OUT(G, S, r) (analogously of CLUSTER-IN). Then, for any $c \geq 1$ we have*

1. *with probability at least $1 - n^{1-c}$ for all $i < t$, the*

radius of the tree corresponding to V_i is at most $c \cdot r$,

2. for any pair of vertices u, v at roundtrip distance at most R in G , they are in the same set V_i with probability at least $\exp(-\log(n)R/r)$.

Furthermore, the algorithm runs in time $O(m \log n)$.

Proof. We prove the lemma for CLUSTER-OUT (the proof for CLUSTER-IN is analogous). We use various facts about the exponential distribution though this proof (See Section B for their proof). Note that the maximum radius of any cluster, V_i is upper bounded by $\max_i x_i$ by design. For every $i \in 1, \dots, t$, we have

$$\Pr[x_i \geq c \cdot r] \leq \exp(-c \cdot \beta r) = n^{-c}.$$

By a union bound the maximum radius of is at most $c \cdot r$ with probability at least $1 - n^{1-c}$.

To prove the remainder of the lemma, fix $u, v \in V$ with roundtrip distance at most R in G . Assume $s \in S$ is the vertex minimizing $-x_s + \min(d_G(s, u), d_G(s, v))$ and that this quantity is less than 0 (otherwise we have $u, v \in V_t$). Let T be the second smallest value of this quantity, or 0, whichever is smaller. Condition on the values of x_s and T and assume without loss of generality that $d_G(s, u) \leq d_G(s, v)$. Then u is assigned to the cluster rooted at s , and u and v can be separated only if $-x_s + d_G(s, v) > T$. By the triangle inequality, this would imply $-x_s + d_G(s, u) + R > T$. By assumption, we have $-x_s + d_G(s, u) < T$, or equivalently $x_s > d_G(s, u) - T$. By the memoryless property of the exponential distribution (See Lemma B.1), we see that the probability that the cluster rooted at s contains both vertices u and v is at least

$$\begin{aligned} \Pr[x_s > d_G(s, u) - T + R \mid x_s > d_G(s, u) - T] \\ &= \Pr[\text{Exp}(\beta) \geq R] \\ &= \exp(-\beta R), \end{aligned}$$

yielding the desired result. ■

4.2 Estimating Ball Sizes To compute part of a roundtrip cover, ideally we would just partition the graph using our decomposition scheme of the previous section and repeat until the clusters have good roundtrip diameter. Unfortunately, as discussed in Section 3 this approach fails as there may be problematic vertices that have a large low-radius ball in one direction, and a small low-radius ball in the other direction. In other words, calls to CLUSTER-IN and CLUSTER-OUT with the wrong set S might only yield trivial partitions, i.e. $V_1 = V$.

To alleviate this issue, we use a fast sampling approach to estimate the sizes of the $O(r)$ -balls of all vertices, that allows us to identify these problematic vertices efficiently.

LEMMA 4.2. ([13]) *For all $\epsilon \in (0, 1)$ there is an algorithm ESTIMATE-BALLS(G, r, ϵ) that in $O(m\epsilon^{-2} \log^2 n)$ time computes n -length vectors s^{out}, s^{in} , with the following property. For any vertex u , let \bar{s}_u^{out} be the fraction of vertices in V such that $d_G(u, v_i) \leq r$. Then, w.h.p., for all vertices u , $|\bar{s}_u^{out} - s_u^{out}| \leq \epsilon$, where s_u^{out} is the component of s^{out} corresponding to u . An analogous statement holds for s^{in} .*

For completeness, we provide a self-contained proof of Lemma 4.2 in Appendix A.

4.3 Fast Roundtrip Covers Combining the techniques of Section 4.1 and Section 4.2 here we provide our efficient algorithm for constructing roundtrip covers, i.e. FAST-ROUNDTRIP-COVER, and prove the main theorem of this section, Theorem 4.1, analyzing this algorithm.

We push much of the work of this algorithm to a subroutine PROBABILISTIC-COVER that performs the simpler task of constructing *probabilistic* roundtrip cover: that is a partition of the vertex set such that any two vertices close enough in the roundtrip metric are in the same cluster with at least some fixed probability. Our main roundtrip cover construction is then simply a union of sufficiently many probabilistic roundtrip covers computed by PROBABILISTIC-COVER.

The statement and analysis of PROBABILISTIC-COVER are the most technically involved results of this section. Our algorithm, PROBABILISTIC-COVER, takes as input a directed graph G , a target radius r , and proceeds as follows. First we use ESTIMATE-BALLS from Section 4.2 to estimate the fraction of vertices in all balls of radius $O(r)$ up to an additive $1/8$. Then we consider two cases. In the first case we find that there is some vertex that can reach a large fraction of the vertices at distance $O(r)$ and can be reached by a large fraction of the vertices at distance $O(r)$. In this case we know that many vertices have a small roundtrip distance to this vertex so we simply output a roundtrip metric ball around this vertex and recurse on the remaining vertices. Otherwise, we know that there are many vertices that do not reach (or are not reachable by) many vertices at distance $O(r)$ and we can cluster to or from these vertices using CLUSTER-OUT(-IN) analyzed in Section 4.1 and recurse on the clusters. In either case we recurse on subsets of vertices that are a constant fraction of the original size and hence only need to

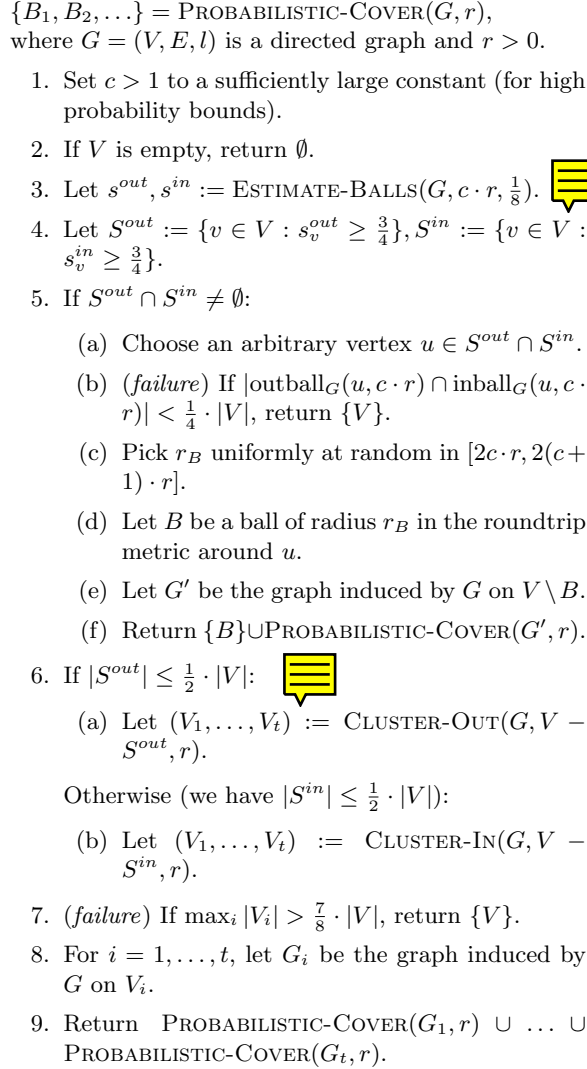


Figure 2: Single pass of cover construction.

recurse a for a logarithmic number of iterations. We formally analyze this algorithm and prove that it has the desired properties in the following Lemma 4.3.

LEMMA 4.3. Let $\mathcal{C} := \text{PROBABILISTIC-COVER}(G, r)$. Then:

1. each $B \in \mathcal{C}$ is a ball of radius $O(r)$ in the roundtrip metric, w.h.p.,
2. any pair of vertices u, v at roundtrip distance at most R in G are in the same element of \mathcal{C} with probability at least $\exp(-6 \log^2(n)R/r)$, and
3. every vertex $v \in V$ belongs to exactly one element of \mathcal{C} .

Furthermore, the algorithm runs in time $O(m \log^3 n)$.

Proof. Property 3. is easily verified.

To prove property 1., first note that for large enough c w.h.p. all calls to the subroutines CLUSTER-IN, CLUSTER-OUT and ESTIMATE-BALLS yield the guarantees described in Lemmas 4.1 and 4.2. Conditioning on this event, we show the failures in steps 5(b) and 7 of PROBABILISTIC-COVER never occur; this is enough to show the thesis.

First assume that there exists a vertex $u \in S^{out} \cap S^{in}$. Then, by assumption, we have $|\text{outball}_G(u, c \cdot r)| \geq (\frac{3}{4} - \frac{1}{8}) \cdot |V| \geq \frac{5}{8} \cdot |V|$ and $|\text{inball}_G(u, c \cdot r)| \geq \frac{5}{8} \cdot |V|$. Hence $|\text{outball}_G(u, c \cdot r) \cap \text{inball}_G(u, c \cdot r)| \geq \frac{1}{4} \cdot |V|$. Hence the failure in step 5(b) cannot occur. Thus, if the condition in step 5 holds, then whp the algorithm will return a cover. Let's assume then that $S^{out} \cap S^{in} = \emptyset$. Then it must be that either $|S^{in}| \leq |V|/2$, or that $|S^{out}| \leq |V|/2$. Assume that $|S^{out}| \leq \frac{1}{2} \cdot |V|$ (the case $|S^{in}| \leq \frac{1}{2} \cdot |V|$ is analogous). By assumption, the radii of all balls grown in calls to CLUSTER-OUT are at most $c \cdot r$, and so the sizes of the clusters constructed are at most $\frac{7}{8} \cdot |V|$ (by assumption on accuracy of ESTIMATE-BALLS). The last cluster cannot be larger than $\frac{1}{2} \cdot |V|$ by construction. Hence the failure in step 7 cannot occur.

To prove property 2., first note that in every recursive call, the size of the vertex set is multiplied by at most $7/8$. Therefore, there are at most $\lceil \log_{8/7} n \rceil$ levels of recursion. Now fix two vertices u and v at roundtrip distance at most $R \leq r$ in G . In each level of recursion, the vertex set is partitioned by a call to CLUSTER-IN or CLUSTER-OUT, or growing a roundtrip metric ball of radius chosen uniformly at random from $[2c \cdot r, 2(c+1) \cdot r]$. For the first two cases, the probability u and v are not separated if they have not been separated previously is at least $\exp(-\log(n)R/r)$ by Lemma 4.1. For the last case, the probability is easily seen to be at least $1 - R/(2 \cdot r) \geq \exp(-\log(n)R/r)$. Hence, the probability that u and v are not separated at all is at least

$$\exp(-\log(n)R/r)^{\lceil \log_{8/7} n \rceil} \geq \exp(-6 \log^2(n)R/r).$$

Note that computing the ball in the roundtrip metric in step 5(d) reduces to two single source shortest path computations from u . Consequently, the running time is dominated by the $O(\log n)$ calls to ESTIMATE-BALLS. ■

With PROBABILISTIC-COVER in hand, we are ready to present the complete efficient algorithm for constructing roundtrip covers. The algorithm, FAST-ROUNDRIP-COVER is given in Figure 3 and we conclude with its analysis, i.e. the proof of Theorem 4.1.

Proof of Theorem 4.1. By Lemma 4.3, w.h.p. all

$\{C_1, C_2, \dots\} = \text{FAST-ROUNTRIP-COVER}(G, k, R)$,
where $G = (V, E, l)$ is a directed graph and $k, R > 0$.

1. Let $r := 6Rk \log n$.
2. Let $c > 1$ to a sufficiently large constant (for high probability bounds).
3. Let $C_0 := \emptyset$.
4. For $i = 1, \dots, c \cdot \lceil n^{1/k} \rceil \cdot \lceil \log n \rceil$:

$$C_i := C_{i-1} \cup \text{PROBABILISTIC-COVER}(G, r).$$

5. Return C .

Figure 3: The fast roundtrip cover algorithm.

balls in \mathcal{C} have the desired radius properties and the size assertions are easily verified.

It remains to show that w.h.p. any two vertices at short roundtrip distance share at least one cluster in \mathcal{C} . Fix two vertices u and v at roundtrip distance at most R in G . By Lemma 4.3, the probability they are in the same cluster in any single cover pass is at least $\exp(-6 \log^2(n)R/r) = \exp(-\log(n)/k)$. Hence, the probability they are separated in $\lceil n^{1/k} \rceil = \lceil \exp(\log(n)/k) \rceil$ independent passes is at most $\exp(-1)$. Therefore, the probability they are separated in all the of $c \cdot \lceil n^{1/k} \rceil \cdot \lceil \log n \rceil$ passes is at most $\exp(-c \log n) = n^{-c}$. ■

5 Roundtrip Spanners and More

The analysis in Section 4 as encompassed by Theorem 4.1 yields our best result for unweighted graphs G ; the union of $\text{FAST-ROUNTRIP-COVER}(G, k, R)$ for $R = 2^0, 2^1, \dots, 2^{\lceil \log_2 n \rceil}$ is w.h.p. an $O(k \log n)$ -multiplicative roundtrip-spanner of G . More generally, for weighted graphs we obtain the following corollary:

COROLLARY 5.1. *Given a directed graph $G = (V, E, l)$ we can construct w.h.p. a $O(k \log n)$ -roundtrip-spanner of G with $O(n^{1+1/k} \log n \log(nW))$ edges in $O(mn^{1/k} \log^4 n \log(nW))$ time, where W is the ratio between the largest and the smallest length in G .*

The first aim of this section is to remove the dependence on W from both the size of the spanner and the running time (Section 5.1). This allows us to prove the following result on spanner construction, and Theorem 1.1 as its corollary.

THEOREM 5.1. *The algorithm $\text{FAST-ROUNTRIP-SPANNER}(G, k)$ in $O(mn^{1/k} \log^4 n)$ time computes w.h.p. an $O(k \log n)$ -roundtrip-spanner of G of size $O(n^{1+1/k} \log^2 n)$.*

Then we shall investigate parallel algorithms that result from our scheme, obtaining the following results (Section 5.2).

THEOREM 5.2. *Given an unweighted directed graph G and an upper bound R on the maximum diameter of any strongly connected component of G , we can w.h.p. compute the strongly connected components of G in $\tilde{O}(m)$ work and $\tilde{O}(R)$ depth.*

THEOREM 5.3. *Given an unweighted directed graph G , we can w.h.p. compute an $O(k \log n)$ approximation to the girth of G in $\tilde{O}(mn^{1/k})$ work and $\tilde{O}(\text{girth}(G))$ depth.*

5.1 Removing the Dependence on Edge

Lengths Our algorithm for constructing the spanner will remain based on the idea of taking a union of $(O(k \log n), R)$ -roundtrip-covers over $R \in \mathcal{R}$, for some set \mathcal{R} such that every roundtrip distance in G is a constant factor smaller than some element of \mathcal{R} .

The main idea in removing the dependence on the lengths of the edges is that for a fixed value of R , we do not have to consider all the edges in G when constructing a $(O(k \log n), R)$ -roundtrip-cover. First, note that we can remove all the edges longer than R , as that does not change any roundtrip distance smaller than R . Simultaneously, for any strongly connected component of edges shorter than R/n , we can replace it by a single vertex. Indeed, uncontracting all such vertices after obtaining a roundtrip cover will increase the length of any path found by only an additive R . Finally, we can remove all the edges that do not participate in any strongly connected component, as that does not impact any roundtrip distances. The idea is similar to those given in [16, 22]; the main difference from the scheme of [22] is in preserving only edges that are parts of connected components of edges shorter than R . The described process is formalized in Definition 5.1.

DEFINITION 5.1. *Let $G = (V, E, l)$ be a directed graph and $x_L, x_R \in \mathbb{R}$ be such that $0 < x_L < x_R$. We construct G collapsed to $[x_L, x_R]$ by:*

- *merging any vertices that can reach each other while following only edges of length at most x_L ,*
- *removing all edges longer than x_R ,*
- *removing all edges whose endpoints cannot reach each other while following only edges of length at most x_R , and*
- *removing all vertices of degree 0 remaining after the above operations.*

To simplify notation, we define the L_∞ -roundtrip distance $d_G^\infty(u, v)$:

DEFINITION 5.2. For a directed graph $G = (V, E, l)$ and a pair of vertices $u, v \in V$, we define the L_∞ -roundtrip distance between u and v , denoted $d_G^\infty(u, v)$, as the minimum value of d such that there is a cycle C containing u and v such that $l_e \leq d$ for all $e \in C$.

We now show that performing the process for all $R \in \{2^t : t \in \mathbb{Z}\}$ results in a collection of graphs with a bounded size.

LEMMA 5.1. Let $G = (V, E, l)$ be a directed graph. For every $t \in \mathbb{Z}$, let $G^{(t)}$ be G collapsed to $[2^t/n, 2^t]$. Then the total number of edges in all $G^{(t)}$ is $O(m \log n)$, and the total number of vertices in all $G^{(t)}$ is $O(n \log n)$.

Proof. Fix an edge $e = (u, v) \in E$ and t such that e is an edge in $G^{(t)}$. Note that since e is not contracted, we must have $d_G^\infty(u, v) > 2^t/n$. Simultaneously, since e is part of a strongly connected component in G consisting of edges of length at most 2^t , we must have $d_G^\infty(u, v) \leq 2^t$. Hence $t - \log_2 d_G^\infty(u, v) \in [0, \log_2 n)$. Therefore e is included in $O(\log n)$ of the graphs $G^{(t)}$.

Assume that u is a vertex of $G^{(t)}$. By construction, it must be part of a nontrivial strongly connected component in $G^{(t)}$, and so it is merged with another vertex in $G^{(t')}$ for all $t' \geq t + \log_2 n$. Since there are only $O(n)$ possible vertices that can result from merging vertices in V , and each of them appears in $O(\log n)$ graphs $G^{(t)}$, we obtain the thesis. ■

If we can construct all the graphs $G^{(t)}$ efficiently, we can simply run FAST-ROUNDTRIP-COVER on each of them to obtain a spanner for G . Following the idea of the proof of Lemma 5.1, to construct all of $G^{(t)}$, is enough to compute for each edge $(u, v) \in E$ the value of $d_G^\infty(u, v)$. This is obtained by the algorithms ROUNDTRIP- L_∞ -SPANNER and FIND-COLLAPSE-TIMES, described below. The algorithm FIND-COLLAPSE-TIMES computes $d_G^\infty(u, v)$ for every edge (u, v) in G , assuming that all the edges have distinct weights from 1 to m .

LEMMA 5.2. Let $G = (V, E)$ be a directed graph and (e_L, \dots, e_R) be a sequence of edges on V . Assume that every edge in E is contained in a strongly connected component of $(V, \{e_L, \dots, e_R\})$. Let $s = \text{FIND-COLLAPSE-TIMES}(G, (e_L, \dots, e_R))$. Then for every $e \in E$, it holds that s_e is the minimum i such that e is contained in a strongly connected component of $(V, \{e_L, \dots, e_{s_e}\})$. Moreover, the algorithm runs in $O((|V| + |E| + (R - L + 1)) \log(R - L + 1))$ time.

$s = \text{FIND-COLLAPSE-TIMES}(G, (e_L, \dots, e_R))$, where $G = (V, E)$ is a directed graph, $L, R \in \mathbb{N}$ with $1 \leq L \leq R$.

1. If $L = R$, set $s_e = L$ for all $e \in E$ and return s .
2. Let $M = \lfloor (L + R)/2 \rfloor$.
3. Let

$$E' := \{e \in E \mid e \text{ is contained inside a SCC of the graph } (V, \{e_L, \dots, e_M\})\}.$$

4. Let V' be V with edges in E' contracted.
5. Let $s' := \text{FIND-COLLAPSE-TIMES}((V, E'), (e_L, \dots, e_M))$.
6. Let $s'' := \text{FIND-COLLAPSE-TIMES}((V', E \setminus E'), (e_{M+1}, \dots, e_R))$.
7. Return s' merged with s'' .

Figure 4: The recursive algorithm for computing collapse times.

Proof. Correctness is easily proven by induction. To bound the running time, it is enough to observe that every recursive call halves $(R - L + 1)$, and every edge in E is only passed to one recursive call. ■

The algorithm ROUNDTRIP- L_∞ -SPANNER constructs an $O(n)$ -sized subset F of the edges of G that preserves the L_∞ -roundtrip distances between vertices of G . It also returns a tree T containing all the vertices that can result from collapsing cycles of maximum edge length lower than some bound, with edges of T describing the hierarchical structure on them. Lowest common ancestor queries on T enable us to efficiently compute $d_G^\infty(u, v)$ for any u, v .

LEMMA 5.3. Let $G = (V, E, l)$ be a directed graph. Let $(F, T) = \text{ROUNDTRIP-}L_\infty\text{-SPANNER}(G)$. Then:

1. $F \subseteq E$ is such that for any pair of vertices u, v contained in a cycle in G with maximum edge length R , there exists a cycle in (V, F) containing u and v with maximum edge length R ,
2. $|F| = O(n)$, and
3. for any two vertices $u, v \in V$, the label of the lowest common ancestor of u and v in T is equal to $d_G^\infty(u, v)$.

Moreover, the algorithm works in $O(m \log n)$ time.

Proof. By Lemma 5.2, the application of FIND-COLLAPSE-TIMES computes for each edge $(u, v) \in E$ the value of $d_G^\infty(u, v)$. The claims of the

$(F, T) = \text{ROUNDTRIP-}L_\infty\text{-SPANNER}(G)$,
where $G = (V, E, l)$ is a directed graph.

1. Remove from G any edges that are not part of a strongly connected component.
2. Let e_1, \dots, e_m be the edges of G , ordered by increasing length.
3. Let $s := \text{FIND-COLLAPSE-TIMES}(G, (e_1, \dots, e_m))$.
4. Let $V_0 = V, F_0 = \emptyset$.
5. Let $T = (V, \emptyset)$.
6. For i in $1, \dots, m$:
 - (a) Let E_i be the set of edges e for which $s_e = i$.
 - (b) Let E'_i be union of any out- and in-trees for the strongly connected components of (V_{i-1}, E_i) .
 - (c) Let $F_i := F_{i-1} \cup E'_i$.
 - (d) Let V_i be the set of vertices obtained from V_{i-1} by contracting all of E_i (equivalently E'_i).
 - (e) Label every vertex of $V_i \setminus V_{i-1}$ by $l(e_i)$.
 - (f) Add all vertices of $V_i \setminus V_{i-1}$ to T .
 - (g) For every vertex $u \in V_{i-1}$ that was contracted into a vertex $v \in V_i \setminus V_{i-1}$, add an edge between v to u to T .
7. Return (F_m, T) .

Figure 5: The algorithm for computing a small subset of edges that preserves L_∞ -roundtrip distance.

lemma follow by construction. \blacksquare

Finally, we describe our complete algorithm for computing roundtrip distance spanners of weighted graphs. The algorithm first computes all graphs $G^{(t)}$ using a call to $\text{ROUNDTRIP-}L_\infty\text{-SPANNER}$ and the ideas of the proof of Lemma 5.1. It then invokes $\text{FAST-ROUNDTRIP-COVER}$ for each of $G^{(t)}$ and returns the union of the results, together with a L_∞ -roundtrip distance spanner for G to account for the collapsed clusters in $G^{(t)}$.

Reminder of Theorem 5.1 *The algorithm $\text{FAST-ROUNDTRIP-SPANNER}(G, k)$ in $O(mn^{1/k} \log^4 n)$ time computes w.h.p. an $O(k \log n)$ -roundtrip-spanner of G of size $O(n^{1+1/k} \log^2 n)$.*

Proof of Theorem 5.1. First note that for each t , F_0 provides a low-cost spanner for every collapsed vertex of $G^{(t)}$. By uncollapsing the collapsed vertices of $G^{(t)}$ and adding in edges from F_0 , the length of any path in the roundtrip cover at radius 2^t increases by at most an additive 2^t . Since edges larger than 2^t have no influence

$F = \text{FAST-ROUNDTRIP-SPANNER}(G, k)$,
where $G = (V, E, l)$ is a directed graph and $k \geq 1$.

1. Let $(F_0, T) := \text{ROUNDTRIP-}L_\infty\text{-SPANNER}(G)$.
2. For all $t \in \mathbb{Z}$, let $G^{(t)}$ be G collapsed to $[2^t/n, 2^t]$.
3. Let $i := 0$.
4. For every t such that $G^{(t)}$ is nonempty:
 - (a) $C_i := \text{FAST-ROUNDTRIP-COVER}(G^{(t)}, k, 2^t)$.
 - (b) $F_{i+1} := F_i \cup$ shortest path trees to and from roots of each ball in C_i .
 - (c) $i := i + 1$.
5. Return F_i .

Figure 6: The roundtrip spanner algorithm.

on roundtrip distances not larger than 2^t , we see that the roundtrip covers computed for each $G^{(t)}$ are also roundtrip covers for G (after adding the edges of F_0).

To obtain the claimed running time, we need to show that the nonempty graphs $G^{(t)}$ can be computed efficiently. Following the idea of the proof of Lemma 5.1, we see that it is sufficient to compute for every edge (u, v) the value of $d_G^\infty(u, v)$. By Lemma 5.3, this is easily done using lowest common ancestor queries on T . \blacksquare

Theorem 1.1 is an immediate corollary of Theorem 5.1.

Reminder of Theorem 1.1 *For any n -node, m -node directed graph with nonnegative integer edge weights, with unknown girth g and integer $k \geq 1$, in time $O(mn^{1/k} \log^5 n)$ we can compute an estimate \bar{g} such that $g \leq \bar{g} \leq O(k \log n) \cdot g$ with high probability.*

Proof of Theorem 1.1. It is sufficient to execute $\text{FAST-ROUNDTRIP-SPANNER}(G, k)$. The smallest diameter of any cluster computed in calls to $\text{FAST-ROUNDTRIP-COVER}$ will be no larger than $O(k \log n) \cdot \text{girth}(G)$. \blacksquare

5.2 Parallel Strongly Connected Components and Girth Estimation

Our main subroutine, $\text{FAST-ROUNDTRIP-COVER}$, is inherently parallelizable. This enables us to obtain a new parallel algorithm for computing strongly connected components in nearly linear work, and depth proportional to the maximum diameter of a strongly connected component (assuming access to a known upper bound). To our knowledge, no previous guarantees of this type have been known, despite the classical status of analogous guarantees for problems such as parallel u - v reachability in directed graphs.

Reminder of Theorem 5.2 *Given an unweighted directed graph G and an upper bound R on the maximum diameter of any strongly connected component of G , we can w.h.p. compute the strongly connected components of G in $\tilde{O}(m)$ work and $\tilde{O}(R)$ depth.*

To prove this result, we first formally state the parallel runtime guarantees of FAST-ROUNDTrip-COVER.

LEMMA 5.4. *For unweighted graphs, a parallel version of FAST-ROUNDTrip-COVER(G, k, R) can be implemented to work in $\tilde{O}(mn^{1/k})$ work and $\tilde{O}(R)$ depth.*

Proof. Since PROBABILISTIC-ROUNDTrip-COVER has only $O(\log n)$ levels of recursion, and the separate calls to it can be made in parallel, the bottleneck for depth is computing shortest paths. Since for unweighted graphs any paths computed in calls to ESTIMATE-BALLS, CLUSTER-OUT and CLUSTER-IN are of length $\tilde{O}(R)$, the thesis follows by employing standard parallel breadth first search (cf. [37]). ■

We now proceed to prove Theorem 5.2.

Proof of Theorem 5.2. We start by computing $\mathcal{C} := \text{FAST-ROUNDTrip-COVER}(G, \log n, R)$. Now note that w.h.p., for any pair of vertices u and v , they are part of the same cluster in \mathcal{C} if and only if they are in the same strongly connected component of G . Hence, it is enough to compute weakly connected components of the relation of being part of the same cluster in \mathcal{C} ; this is achieved with classical parallel algorithms [1, 44]. ■

Another corollary is that we can parallelize our girth estimation algorithm for unweighted graphs.

Reminder of Theorem 5.3 *Given an unweighted directed graph G , we can w.h.p. compute an $O(k \log n)$ approximation to the girth of G in $\tilde{O}(mn^{1/k})$ work and $\tilde{O}(\text{girth}(G))$ depth.*

Proof of Theorem 5.3. It suffices to invoke FAST-ROUNDTrip-COVER(G, k, R) for $R \in 2^0, 2^1, \dots$ until it returns a nonempty result. The work and depth bounds follow from Lemma 5.4. ■

6 Additive Approximation for the Girth

As discussed in the introduction, combining Theorem 1.1 with the BFS computation of the lengths of shortest cycles through all nodes in a random sample of size $\tilde{O}(n^{1-a})$, yields the following corollary:

Reminder of Corollary 1.1 *For all $a \in (0, 1)$, there is an $\tilde{O}(mn^{1-a})$ time combinatorial algorithm that w.h.p. returns an $O(n^a)$ additive approximation to the girth of an unweighted directed graph.*

It is unclear whether the algorithm from the above corollary can be derandomized. The algorithm uses randomization in many places: (1) it uses a random sample to hit long cycles that we don't have a handle on otherwise, (2) it uses sampling quite heavily to obtain estimates of the sizes of reachability sets of all vertices, (3) it grows random neighborhoods according to an exponential distribution.

We are not aware of any deterministic approach that achieves running time $O(mn^{1-\varepsilon})$ for $\varepsilon > 0$ for any of the above cases. In fact, as far as we know, the only way to achieve (2) deterministically is to compute the reachability trees explicitly. Despite this, we show that the result can be partially derandomized using different techniques:

Reminder of Theorem 1.3 *Let $G = (V, E)$ be a directed unweighted graph with unknown girth g , and let $0 < a, \varepsilon < 1$ be parameters. There is a deterministic combinatorial algorithm that computes in $\tilde{O}((1/\varepsilon^2)mn^{1-a})$ time a cycle whose length is*

1. *an $O(n^a)$ additive approximation of g if $g \leq n^a$, and*
2. *a $(1+\varepsilon)$ multiplicative approximation of g if $g > n^a$.*

In the remainder of this section we prove Theorem 1.3.

Roughly speaking, our algorithm works in iterations, where each iteration takes $\tilde{O}((1/\varepsilon)m)$ time. Let C be a shortest cycle in $G = (V, E)$. The idea of the algorithm is as follows. In each iteration we consider a shortest path of $\lceil \varepsilon \cdot n^a \rceil$ vertices. If no such path exists, then the diameter of G must be smaller than εn^a , and we can pick any cycle and return it as our approximation. Assume now that there is a shortest path P with at least $\lceil \varepsilon \cdot n^a \rceil$ vertices. Either $P \cap C \neq \emptyset$, or we can remove P from G and recurse on the remaining graph. If $P \cap C \neq \emptyset$, our algorithm finds an approximation for C by constructing a new weighted graph G' and a shortest path P' between two nodes s and t in G' whose second shortest simple path length is a good approximation to the length of C .

If we could compute this second shortest path exactly, then we would be done. Unfortunately, the fastest known algorithm for second shortest path takes $O(mn + n^2 \log \log n)$ time [27], and moreover Vassilevska W. and Williams [53] showed that the problem is sub-cubically equivalent to APSP, so that a truly subcubic algorithm for it would be a breakthrough. Our goal is to obtain an almost linear time algorithm, however, since we might need to repeat the procedure n^{1-a} times (removing n^a nodes in each iteration).

Fortunately, Bernstein[8] developed an $\tilde{O}(m/\varepsilon)$ time algorithm that computes a $(1 + \varepsilon)$ multiplicative

approximation for the second shortest simple path in directed weighted graphs. We use this algorithm for our $\tilde{O}(mn^{1-a})$ mixed approximation algorithm.

Before we formally describe our algorithm, we note that a cycle in a directed graph must be contained in a strongly connected component (SCC). We can assume that G is strongly connected, as otherwise we compute in $O(m)$ -time its SCCs and run the algorithm in every non-singleton SCC. If all SCCs are singletons, then the graph is a directed acyclic graph and has no cycles.

We start by taking an arbitrary vertex z of G and using BFS in $O(m)$ to find the longest shortest paths Q_{in}, Q_{out} , in and out of z , respectively. Let Q be the longer of Q_{in} and Q_{out} . By the triangle inequality, Q must have length at least half of the diameter of G (notice that since G is strongly-connected, the diameter is well-defined). Let $d = \lceil \varepsilon \cdot n^a \rceil$. If the length of Q is $< d$, then the diameter is $< 2d$, and any vertex of G is on a cycle of length at most $2d$: take the edge (x, y) on a shortest cycle \tilde{C} through x ; the length of \tilde{C} is $1 + d(y, x) \leq 1 + (2d - 1) = 2d$. Therefore, by running a BFS from an arbitrary vertex and stopping when the first backward edge is detected, we find a cycle that is an $O(\varepsilon n^a)$ additive approximation to the shortest cycle.

Otherwise, the diameter is at least $2d$. Let $P = \langle v_d, \dots, v_0 \rangle$ be a portion of d edges from the path that we have computed. We construct a new directed weighted graph G' as follows.

1. Initialize G' to be G . Set all weights to 1.
2. Add the following vertices and edges to G' .
 - (a) For each $v_i \in P$, where $i \in \{0, \dots, d\}$, create new nodes u_i and u'_i .
 - (b) For each $i \in \{0, \dots, d\}$ add an edge from u_i to u'_i , and for each $i \in \{0, \dots, d-1\}$ add an edge from u'_i to u_{i+1} . All edges are of weight 1.
3. For each $v_i \in P$, where $i \in \{0, \dots, d\}$, add the following new edges to G' .
 - (a) Add a new edge of weight $4d - 3i$ from u_i to v_i .
 - (b) For each outgoing edge $(v_i, x) \in E$ of v_i , add a new edge of weight $4d - 3i$ from u_i to x .
 - (c) For each incoming edge $(y, v_i) \in E$ of v_i , add a new edge of weight $3i$ from y to u'_i .

From the above construction it follows that there is a path $P' = \langle u_0, u'_0, u_1, u'_1, \dots, u_d, u'_d \rangle$ in G' of length $2d + 1$. Moreover, P' is the shortest path from u_0 to u'_d . To see this, notice first that there is no edge from u to v , where $u, v \in P'$ are not consecutive. Therefore,

any path from u_0 to u'_d other than P' , contains a vertex $v \notin P'$. The length of such a path is at least $4d > 2d + 1$ since it must use an edge of weight $4d - 3i$ to leave P' and an edge of weight $3j$ to return P' , where $0 \leq i \leq j \leq d$.

Next we apply Bernstein's algorithm to find a second shortest path for P' in G' . Similarly to prior work on replacement paths, given a shortest path Q we say that a path $D(u, v)$ is a $\langle u, v \rangle$ -detour of Q if $D(u, v)$ is a simple path for which $D(u, v) \cap Q = \{u, v\}$ and u precedes v on Q . It is easy to show that the second shortest path Q' of $Q = \langle v_0, \dots, v_k \rangle$ has the following form: $Q' = Q(v_0, u) \cdot D(u, v) \cdot Q(v, v_k)$, where $Q(v_0, u)$ and $Q(v, v_k)$ are the subpaths of Q from v_0 to u and from v to v_k , respectively, and $D(u, v)$ is a $\langle u, v \rangle$ -detour of Q (e.g. see Lemma 2.1 in [8]).

The following fact follows easily from the construction of G' and P' above.

FACT 6.1. *If P' has a $\langle u_i, u'_j \rangle$ -detour then it has the following structure: $(u_i, x) \cdot Q' \cdot (y, u'_j)$, where Q' is a path from x to y in G , and x is an out-neighbor of u_i in G and y is an in-neighbor of v_j in G . Notice Q' might be an empty path.*

In the next lemmas we establish the relationship between a shortest cycle that intersects P in G and a second shortest path for P' in G' .

LEMMA 6.1. *Let $0 \leq i \leq j \leq d$. If P' has a $\langle u_i, u'_j \rangle$ -detour $D(u_i, u'_j)$, then there is a cycle in G that contains $P(v_j, v_i)$ and has length $\leq |D(u_i, u'_j)| + |P(v_j, v_i)| = |D(u_i, u'_j)| + (j - i)$.*

Furthermore, if G has a simple cycle C that contains $P(v_j, v_i)$, then P' has a $\langle u_i, u'_j \rangle$ -detour of length at most $|C| - |P(v_j, v_i)| + 1$.

Proof. Let Q be a $\langle u_i, u'_j \rangle$ -detour of P' . From the construction of G' it follows that $Q = (u_i, x) \cdot Q' \cdot (y, u'_j)$ where Q' is a path from x to y in G^1 .

We show that $C = (v_i, x) \cdot Q' \cdot (y, v_j) \cdot P(v_j, v_i)$ is a cycle in G . From the construction of G' it follows that the edges (u_i, x) and (y, u'_j) in G' correspond to the edges (v_i, x) and (y, v_j) in G , respectively, and since the path Q' is also in G it follows that C is a cycle in G .

Let C be a simple cycle such that $P(v_j, v_i) \subseteq C$ for some i, j (possibly equal). If $C = \langle v_j, \dots, v_i \rangle$ (i.e. $(v_i, v_j) \in E$), then $i \neq j$ and we have the following $\langle u_i, u'_j \rangle$ -detour: $D(u_i, u'_j) = \langle u_i, v_i, u'_j \rangle$. Otherwise,

¹Notice it is possible that $x = y$, and then $\langle v_i, x \rangle \cdot Q' \cdot \langle y, v_j \rangle$ is actually $\langle v_i, x, v_j \rangle$. It is also possible, in addition, $v_i = x = y$, and then $\langle v_i, x \rangle \cdot Q' \cdot \langle y, v_j \rangle$ is actually $\langle v_i, v_j \rangle$; this is the reason for adding the edges (u_i, v_i) in G' . For simplicity of the presentation, we assume the concatenation notation subsume all these cases.

we have a shortest path B from v_i to v_j , such that $B \cap P = \{v_i, v_j\}$ and $B \neq \{v_i, v_j\}$. Let x be the vertex that follows v_i in B and y the vertex the precedes v_j in B (it might be that $x = y$), then we have the following $\langle u_i, u'_j \rangle$ -detour: $D(u_i, u'_j) = (u_i, x) \cdot B(x, y) \cdot (y, u'_j)$. ■

LEMMA 6.2. *Let C^* be a shortest cycle in G . If $P(v_j, v_i) \subseteq C^*$, where $j \geq i$, then the length of a second shortest path of P' is at most $6d - 2 + |C^*|$.*

Proof. It follows from Lemma 6.1 that P' has a $\langle u_i, u'_j \rangle$ -detour. From Fact 6.1 it has the following structure: $(u_i, x) \cdot Q'(x, y) \cdot (y, u'_j)$. Consider the path $P'(u_0, u_i) \cdot (u_i, x) \cdot Q'(x, y) \cdot (y, u'_j) \cdot P'(u'_j, u'_d)$. Its length is $2i + (4d - 3i) + d_G(x, y) + 3j + 2(d - j) = 6d + (j - i) + d_G(x, y) = 6d - 2 + d_G(x, y) + 2 + (j - i) \leq 6d - 2 + d_G(v_i, v_j) + d_G(v_j, v_i) = 6d - 2 + |C^*|$. ■

According to Lemma 6.1, a second shortest path implies a cycle C in G consisting of the detour of the second shortest path together with the path in G corresponding to the subpath of P' that was circumvented. Notice it is easy to derive from C a simple cycle in G , which might be shorter. Denote by L the length of a second shortest path. The length of C is then at most $d + L$.

Let C^* be a shortest cycle in G . If $P \cap C^* \neq \emptyset$, according to Lemma 6.2, $L \leq 6d - 2 + |C^*|$. Since we are using a $(1 + \varepsilon)$ -approximation for the second shortest path ($\varepsilon < 1$), we get a cycle of length at most:

$$\begin{aligned} d + (1 + \varepsilon)L &= d + (1 + \varepsilon)(6d - 2 + |C^*|) \\ &= 7d - 2 + \varepsilon(6d - 2) + (1 + \varepsilon)|C^*| \\ &\leq O(d) + (1 + \varepsilon)|C^*|. \end{aligned}$$

If $g \leq n^a$, then we found a cycle of size $O(n^a)$. If $g > n^a$, then since $d \leq O(\varepsilon n^a)$, we have a $1 + O(\varepsilon)$ multiplicative approximation for the girth.

Thus, if an approximate second shortest path of length $\leq 16d$ is found, we can conclude that $L \leq 16d$ and hence $|C^*| \leq O(d)$, so we can stop and return. Otherwise, we can conclude that none of the vertices of P are on cycles of length $\leq d$ in G , as otherwise the algorithm would return an approximate second shortest path of length $7d - 2 + \varepsilon(6d - 2) + (1 + \varepsilon)|C^*| < 13d - 4 + 2d < 16d$. We can thus remove all the vertices of P from G and repeat the process above on the new graph.

Consider the first iteration in which P' contains a vertex of C^* . Since up to this iteration no vertices of P' are removed, the graph will contain a detour corresponding to the portion of C^* not on P , and the approximate cycle returned will be of length \leq

$O(d) + (1 + \varepsilon)|C^*|$, as argued above. If the girth is $\leq 2d$, the approximation is additive $O(d)$, and otherwise it is multiplicative $1 + O(\varepsilon)$.

The correctness of the algorithm follows from the discussion above. The runtime is as follows. The time to decompose the graph to its strongly-connected components is $O(m + n)$ [50]. The time to construct G' is $O(m)$, and the running time of Bernstein's algorithm for the second shortest path on G' is $\tilde{O}(m/\varepsilon)$. We conclude that the running time for a single iteration is $\tilde{O}(m/\varepsilon)$. The number of iterations we have in the algorithm is at most $\lceil (1/\varepsilon)n^{1-a} \rceil$, since we are removing $\lceil \varepsilon \cdot n^a \rceil$ vertices from G in each iteration. It follows that the total running time of the algorithm is $\tilde{O}((1/\varepsilon^2)mn^{1-a})$, thus proving Theorem 1.3.

7 Lower Bounds

In this section we provide a conditional lower bound for the problem of computing additive approximations for the girth of a directed unweighted graph.

Let us begin with our plausible hypothesis:

HYPOTHESIS 7.1. *Any combinatorial (possibly randomized) algorithm for triangle detection in n -node m -edge graphs with $m = \Theta(n^2)$ requires (expected) $n^{3-o(1)}$ time.*

Combinatorial algorithms informally do not use Strassen-like matrix multiplication, and hopefully do not hide high constants in the big- O . The current best combinatorial algorithms for triangle detection run in time $\min\{O(n^3/\log^4 n), O(m^{3/2})\}$ time [56, 28]. It is a major open problem to design a truly subcubic, i.e. an $O(n^{3-\varepsilon})$ time combinatorial algorithm for constant $\varepsilon > 0$ for triangle detection. Triangle detection is known [53] to be *subcubically equivalent* to Boolean Matrix Multiplication (BMM) under combinatorial fine-grained reductions, and thus the above hypothesis is equivalent to the hypothesis that combinatorial BMM of $n \times n$ matrices requires $n^{3-o(1)}$ time.

We now state our result:

THEOREM 7.1. *Under Hypothesis 7.1, any combinatorial algorithm that computes an additive $n^{1/2} - 1$ approximation to the girth of all directed n -node, $m = O(n)$ -edge graphs requires $mn^{1/2-o(1)}$ time.*

Proof. Let $G = (V, E)$ be an n -node, m -edge directed graph for $m = \Theta(n^2)$, so that we want to detect the presence of a 3-cycle in G . We now create a new directed H as follows:

- H has n^2 vertices: for every $v \in V$ we create n copies v_1, \dots, v_n .
- For every edge $(u, v) \in E$ of G , we create directed edges (u_n, v_1) and (u_i, v_{i+1}) for all $i \in \{1, 2\}$.

- For every vertex $v \in V$, create directed edges (v_i, v_{i+1}) for all $i \in \{3, \dots, n-1\}$.

Every triangle $a^1 \rightarrow a^2 \rightarrow a^3 \rightarrow a^1$ in G is represented by an n -cycle in H : $a_n^1 \rightarrow a_1^2 \rightarrow a_2^3 \rightarrow a_3^1 \rightarrow a_4^1 \rightarrow \dots \rightarrow a_n^1$.

Every n -cycle in H must correspond to a 3-cycle in G , as there is a path from v_3 to w_n if and only if $v = w$. Moreover, any cycle in H has length that is a multiple of n as each cycle must go through all n partitions of the graph over and over until it lands at the same node. The girth of H is thus either n if G has a 3-cycle, or at least $2n$ otherwise. H has $N = n^2$ vertices and $M \leq 3m + n^2 = \Theta(n^2)$ edges.

Suppose that there is some constant $\varepsilon > 0$ such that for all a there is an $O(MN^{1/2-\varepsilon})$ time algorithm that achieves an additive $N^{1/2} - 1$ approximation to the girth of M -edge, N -node directed graphs. Let's apply this algorithm to H . If it finds an additive $N^{1/2} - 1 = n - 1$ approximation to the girth of H , it will be able to detect whether G contains a triangle. The running time of the algorithm would be

$$O(MN^{1/2-\varepsilon}) = O(n^2 \cdot n^{1-2\varepsilon}) = O(n^{3-2\varepsilon}),$$

which contradicts Hypothesis 1. ■

Considering multiplicative approximation for the girth in directed unweighted graphs, it is known that any truly subcubic combinatorial algorithm that computes a $2 - \varepsilon$ approximation ($0 < \varepsilon < 1$) for the girth in directed unweighted graphs, implies a truly subcubic time combinatorial algorithm for triangle detection. This is formalized in the next probably folklore theorem. A formal proof of it appears in [47].

THEOREM 7.2. (FOLKLORE) *Under Hypothesis 7.1, any combinatorial algorithm that for $\varepsilon \in (0, 1)$ computes a multiplicative $2 - \varepsilon$ approximation for the girth of a directed n -node, m -edge graph requires $n^{3-o(1)}$ time.*

References

- [1] An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms* 3, 1 (1982), 57–67.
- [2] ALTHOFER, I., DAS, G., DOBKIN, D., JOSEPH, D., AND SOARES, J. On sparse spanners of weighted graphs. *Discrete & Computational Geometry* 9, 1 (1993), 81–100.
- [3] ARLAZAROV, V. L., DINIC, E. A., KRONROD, M. A., AND FARADZEV, I. A. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl.* 11 (1970), 1209–1210.
- [4] AWERBUCH, B. Complexity of network synchronization. *J. ACM* 32, 4 (Oct. 1985), 804–823.
- [5] BANSAL, N., AND WILLIAMS, R. Regularity lemmas and combinatorial algorithms. *Theory of Computing* 8, 1 (2012), 69–94.
- [6] BARTAL, Y. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on* (1996), pp. 184–193.
- [7] BASWANA, S., KAVITHA, T., MEHLHORN, K., AND PETTIE, S. New constructions of (α, β) -spanners and purely additive spanners. In *Proc. 16th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2005), pp. 672–681.
- [8] BERNSTEIN, A. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010* (2010), pp. 742–755.
- [9] BONDY, J., AND SIMONOVITS, M. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B* 16, 2 (1974), 97–105.
- [10] CHAN, T. M. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. STOC* (2007), pp. 590–598.
- [11] CHAN, T. M. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015* (2015), pp. 212–217.
- [12] CHRISTIANO, P., KELNER, J. A., MADRY, A., SPIELMAN, D. A., AND TENG, S.-H. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing* (New York, NY, USA, 2011), STOC '11, ACM, pp. 273–282.
- [13] COHEN, E. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences* 55, 3 (1997), 441–453.
- [14] COHEN, E. Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM J. Comput.* 28, 1 (Feb. 1999), 210–236.
- [15] COHEN, M. B., MADRY, A., SANKOWSKI, P., AND VLADU, A. Negative-weight shortest paths and unit capacity minimum cost flow in $O(m^{10/7} \log W)$ time. *CoRR abs/1605.01717* (2016).
- [16] COHEN, M. B., MILLER, G. L., PACHOCKI, J. W., PENG, R., AND XU, S. C. Stretching stretch. *CoRR abs/1401.2454* (2014).
- [17] COPPERSMITH, D., AND WINOGRAD, S. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280.
- [18] COWEN, L. J., AND WAGNER, C. G. Compact roundtrip routing in directed networks. *Journal of Algorithms* 50, 1 (2004), 79–95.
- [19] DIESTEL, R. *Graph theory*, 2 ed. Springer-Verlag, 2000.
- [20] DOR, D., HALPERIN, S., AND ZWICK, U. All-pairs almost shortest paths. *SIAM J. Comput.* 29, 5 (Mar.

- 2000), 1740–1759.
- [21] ELKIN, M., AND PELEG, D. $(1 + \epsilon, \beta)$ -spanner constructions for general graphs. *SIAM J. Comput.* 33, 3 (Mar. 2004), 608–631.
 - [22] ENE, A., MILLER, G., PACHOCKI, J., AND SIDFORD, A. Routing under balance. In *to appear in STOC 2016* (2016).
 - [23] ERDŐS, P. Extremal problems in graph theory. *Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)* (1963), 29–36.
 - [24] FAKCHAROENPHOL, J., RAO, S., AND TALWAR, K. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.* 69, 3 (2004), 485–497.
 - [25] FREDMAN, M. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing* 5 (1976), 49–60.
 - [26] GALL, F. L. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23–25, 2014* (2014), pp. 296–303.
 - [27] GOTTHILF, Z., AND LEWENSTEIN, M. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Inf. Process. Lett.* 109, 7 (2009), 352–355.
 - [28] ITAI, A., AND RODEH, M. Finding a minimum circuit in a graph. *SIAM J. Comput.* 7, 4 (1978), 413–423.
 - [29] KELNER, J. A., LEE, Y. T., ORECCHIA, L., AND SIDFORD, A. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014* (2014), pp. 217–226.
 - [30] LEE, Y. T., RAO, S., AND SRIVASTAVA, N. A new approach to computing maximum flows using electrical flows. In *Proceedings of the Forty-fifth Annual ACM Symposium on the Theory of Computing* (New York, NY, USA, 2013), STOC '13, ACM, pp. 755–764.
 - [31] LEE, Y. T., AND SIDFORD, A. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\text{vrank})$ iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18–21, 2014* (2014), pp. 424–433.
 - [32] LINCOLN, A., VASSILEVSKA WILLIAMS, V., AND WILLIAMS, R. Hypercliques and conditional hardness for sparse graph problems. Unpublished manuscript, submitted., 2017.
 - [33] LINGAS, A., AND LUNDELL, E. Efficient approximation algorithms for shortest cycles in undirected graphs. *Inf. Process. Lett.* 109, 10 (2009), 493–498.
 - [34] LINIAL, N., AND SAKS, M. Decomposing graphs into regions of small diameter. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1991), SODA '91, Society for Industrial and Applied Mathematics, pp. 320–330.
 - [35] MADRY, A. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS* (2010), IEEE Computer Society, pp. 245–254.
 - [36] MADRY, A. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26–29 October, 2013, Berkeley, CA, USA* (2013), pp. 253–262.
 - [37] MILLER, G. L., PENG, R., AND XU, S. C. Parallel graph decompositions using random shifts. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures* (New York, NY, USA, 2013), SPAA '13, ACM, pp. 196–203.
 - [38] ORLIN, J. B., AND SEDEÑO-NODA, A. An $O(mn)$ 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, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16–19* (2017), pp. 1866–1879.
 - [39] PELEG, D., AND SCHÄFFER, A. Graph spanners. *Journal of Graph Theory* 13, 1 (1989), 99–116.
 - [40] PELEG, D., AND SCHÄFFER, A. A. Graph spanners. *Journal of Graph Theory* 13, 1 (1989), 99–116.
 - [41] PELEG, D., AND ULLMAN, J. D. An optimal synchronizer for the hypercube. *SIAM J. Comput.* 18, 4 (1989), 740–747.
 - [42] PENG, R. A note on cut-approximators and approximating undirected max flows. *CoRR abs/1411.7631* (2014).
 - [43] PETTIE, S., AND RAMACHANDRAN, V. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.* 34, 6 (2005), 1398–1431.
 - [44] REIF, J., AND SPIRAKIS, P. Expected parallel time and sequential space complexity of graph and digraph problems. *Algorithmica* 7, 1 (1992), 597–630.
 - [45] RODITTY, L., THORUP, M., AND ZWICK, U. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms* 4, 3 (2008), 29:1–29:17.
 - [46] RODITTY, L., AND WILLIAMS, V. V. Minimum weight cycles and triangles: Equivalences and algorithms. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22–25, 2011* (2011), pp. 180–189.
 - [47] RODITTY, L., AND WILLIAMS, V. V. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17–19, 2012* (2012), pp. 833–845.
 - [48] SHERMAN, J. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26–29 October, 2013, Berkeley, CA, USA* (2013), pp. 263–269.
 - [49] SHERMAN, J. Generalized preconditioning and network flow problems. *CoRR abs/1606.07425* (2016).
 - [50] TARJAN, R. E. Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160.
 - [51] THORUP, M., AND ZWICK, U. Approximate distance oracles. *J. ACM* 52, 1 (2005), 1–24.

- [52] VASSILEVSKA WILLIAMS, V. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012* (2012), pp. 887–898.
- [53] VASSILEVSKA WILLIAMS, V., AND WILLIAMS, R. Sub-cubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA* (2010), pp. 645–654.
- [54] WENGER, R. Extremal graphs with no C_4 's, C_6 's, or C_{10} 's. *Journal of Combinatorial Theory, Series B* 52, 1 (1991), 113 – 116.
- [55] WILLIAMS, R. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014* (2014), pp. 664–673.
- [56] YU, H. An improved combinatorial algorithm for boolean matrix multiplication. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I* (2015), pp. 1094–1105.

A Ball Size Estimation

Here we present a routine that can estimate the sizes of neighborhoods of all vertices. The approach is similar to that of [13]. We provide the algorithm, ESTIMATE-BALLS, that when invoked on a graph G with radius parameter r and error parameter ϵ computes w.h.p. for every vertex the fraction of vertices that it can reach at distance at r and the fraction of vertices that can reach it.

$(s^{out}, s^{in}) = \text{ESTIMATE-BALLS}(G, r, \epsilon)$,
where $G = (V, E, l)$ is a directed graph and $r, \epsilon > 0$.

1. Sample $t = \lceil 20 \cdot \log n / \epsilon^2 \rceil$ vertices v_1, \dots, v_t independently uniformly at random from V with replacement.
2. Compute the distances between every vertex in V and each of v_1, \dots, v_t .
3. For each $u \in V$, let s_u^{out} be the fraction of v_1, \dots, v_t such that $d_G(u, v_i) \leq r$.
4. For each $u \in V$, let s_u^{in} be the fraction of v_1, \dots, v_t such that $d_G(v_i, u) \leq r$.
5. Return (s^{out}, s^{in}) .

Figure 7: The algorithm for estimating the sizes of out- and inballs at radius r for a given graph.

Our algorithm simply samples vertices with replacement and computes distances to and from them to estimate the ball sizes. The analysis of ESTIMATE-BALLS reduces to a simple application of Chernoff bounds and

union bound. We prove that it works in Lemma A.1.

LEMMA A.1. Let s^{out}, s^{in} be the output of ESTIMATE-BALLS(G, r, ϵ). For any vertex u , let \bar{s}_u^{out} be the fraction of vertices in V such that $d_G(u, v_i) \leq r$. Then, whp., for all vertices u it holds that $|\bar{s}_u^{out} - s_u^{out}| \leq \epsilon$. An analogous statement holds for s^{in} . The algorithm runs in time $O(m\epsilon^{-2} \log^2 n)$.

Proof. By a standard Chernoff bound, we have

$$\begin{aligned} \Pr[|\bar{s}_u^{out} - s_u^{out}| > \epsilon] &\leq 2 \exp(-2t\epsilon^2) \\ &\leq 2 \exp(-40 \log n) \\ &= 2n^{-40}. \end{aligned}$$

An analogous bound holds for s^{in} . ■

B Exponential Distributions

Here we recall some basic facts about the exponential distribution we use in the paper.

LEMMA B.1. (EXPONENTIAL DISTRIBUTION FACTS) We let $\text{Exp}(\alpha)$ denote the exponential distribution with parameter α . This distribution is supported on $\mathbb{R}_{\geq 0}$ with a pdf given $p(x) = \alpha \cdot \exp(-\alpha x)$. This distribution has the following properties:

- **CDF:** $\Pr[\text{Exp}(\alpha) \leq x] = 1 - \exp(-\alpha x)$ for $x \geq 0$.
- **Expected Value:** $\mathbb{E}\text{Exp}(\alpha) = \frac{1}{\alpha}$
- **Memoryless:** $\Pr[\text{Exp}(\alpha) \geq s + t \mid \text{Exp}(\alpha) \geq s] = \Pr[\text{Exp}(\alpha) \geq t]$
- **High Probability:** The maximum of n independent r.v.s drawn from $\text{Exp}(\alpha)$ is $O(\frac{\log n}{\alpha})$ with high probability.

Proof. Direct calculation reveals that

$$\begin{aligned} \Pr[\text{Exp}(\alpha) \leq x] &= \int_{-\infty}^x \alpha \exp(-\alpha x) \\ &= -\exp(-\alpha x) + \exp(0) \\ &= 1 - \exp(-\alpha x) \end{aligned}$$

giving the formula for the CDF. Furthermore, integration by parts yields that

$$\begin{aligned} \mathbb{E}\text{Exp}(\alpha) &= \int_0^{\infty} \alpha x \exp(-\alpha x) dx \\ &= [-x \exp(-\alpha x)]_0^{\infty} - \int_0^{\infty} -\exp(-\alpha x) dx \\ &= -\frac{1}{\alpha} \exp(-\alpha \infty) + \frac{1}{\alpha} \exp(-\alpha 0) = \frac{1}{\alpha} \end{aligned}$$

giving the expected value formula. Direct calculation again yields

$$\begin{aligned}\Pr[\text{Exp}(\alpha) \geq s+t \mid \text{Exp} \geq s] &= \frac{\Pr[\exp(\alpha) \geq s+t]}{\Pr[\exp(\alpha) \geq s]} \\ &= \frac{\exp(-\alpha(s+t))}{\exp(-\alpha s)} \\ &= \exp(-\alpha t) \\ &= \Pr[\text{Exp}(\alpha) \geq t]\end{aligned}$$

proving the memoryless property. Finally the high probability bound is immediate from the CDF and the definition of high probability. ■

C Sequential Clustering Algorithm

Here we formalize and prove the alternative approach to clustering described in Section 3.

$(V_1, V_2, \dots) =$
 SEQUENTIAL-CLUSTER-OUT(-IN)($G, (v_1, \dots, v_{t-1}), r$),
 where $G = (V, E, l)$ is a directed graph, $v_1, \dots, v_{t-1} \in V$,
 and $r > 0$.

1. Set $\beta := \log(n)/r$.
2. Let $G_0 := G$.
3. For i in $1, \dots, t$:
 - (a) If v_i is not in G_{i-1} , let $G_i := G_{i-1}, V_i := \emptyset$ and continue the loop. Otherwise:
 - (b) Pick $x_i \sim \text{Exp}(\beta)$.
 - (c) Let $V_i := \text{outball}_{G_{i-1}}(v_i, x_i)$.
 (inball $_{G_{i-1}}(v_i, x_i)$ for CLUSTER-IN)
 - (d) Let $G_i := G_{i-1}$ with V_i and incident edges removed.
4. Return $(V_1, V_2, \dots, V_{t-1}, V \setminus \bigcup_i V_i)$

Figure 8: The sequential clustering algorithm.

LEMMA C.1. Let $(V_1, V_2, \dots, V_t) = \text{SEQUENTIAL-CLUSTER-OUT}(G, (v_1, \dots, v_t), r_0, r)$ (analogously of SEQUENTIAL-CLUSTER-IN). Then for any $c \geq 1$ we have

1. with probability at least $1 - n^{1-c}$ for all $i < t$, the radius of the tree corresponding to V_i is at most $c \cdot r$, whp.,
2. for any pair of vertices u, v at roundtrip distance at most R in G , they are in the same set V_i with probability at least $\exp(-\log(n)R/r)$.

Proof. Note that the maximum radius of any constructed tree is upper bounded by $\max_i x_i$. For every

$i \in 1, \dots, t$, we have

$$\Pr[x_i \geq c \cdot r] \leq \exp(-c \cdot \beta r) = n^{-c},$$

and so by union bound the maximum radius is at most $c \cdot r$ with probability at least $1 - n^{1-c}$.

To prove the remainder of the lemma, fix two vertices u and v at roundtrip distance at most R in G . Assume the i -th cluster is the first one to contain an element of the set $\{u, v\}$. By the memoryless property of the exponential distribution we see that conditioned on this event the probability that cluster i contains both vertices u and v is at least

$$\Pr[\text{Exp}(\beta) \geq R] = \exp(-\beta R),$$

yielding the desired result. ■