Improved Roundtrip Spanners, Emulators, and Directed Girth Approximation*

Alina Harbuzova† Ce Jin‡ Virginia Vassilevska Williams§ Zixuan Xu¶

Abstract

Roundtrip spanners are the analog of spanners in directed graphs, where the roundtrip metric is used as a notion of distance. Recent works have shown existential results of roundtrip spanners nearly matching the undirected case, but the time complexity for constructing roundtrip spanners is still widely open.

This paper focuses on developing fast algorithms for roundtrip spanners and related problems. For any n-vertex directed graph G with m edges (with non-negative edge weights), our results are as follows:

- **3-roundtrip spanner faster than APSP:** We give an $\tilde{O}(m\sqrt{n})$-time algorithm that constructs a roundtrip spanner of stretch 3 and optimal size $O(n^{3/2})$. Previous constructions of roundtrip spanners of the same size either required $\Omega(nm)$ time [Roditty, Thorup, Zwick SODA'02; Cen, Duan, Gu ICALP'20], or had worse stretch 4 [Chechik and Lifshitz SODA'21].
- **Optimal roundtrip emulator in dense graphs:** For integer $k \geq 3$, we give an $O(kn^2 \log n)$-time algorithm that constructs a roundtrip *emulator* of stretch $(2k-1)$ and size $O(kn^{1+1/k})$, which is optimal for constant k under Erdős' girth conjecture. Previous work of [Thorup and Zwick STOC'01] implied a roundtrip emulator of the same size and stretch, but it required $\Omega(nm)$ construction time. Our improved running time is near-optimal for dense graphs.
- **Faster girth approximation in sparse graphs:** We give an $\tilde{O}(mn^{1/3})$-time algorithm that 4-approximates the girth of a directed graph. This can be compared with the previous 2-approximation algorithm in $\tilde{O}(n^2, m\sqrt{n})$ time by [Chechik and Lifshitz SODA'21]. In sparse graphs, our algorithm achieves better running time at the cost of a larger approximation ratio.

*The full version of the paper can be accessed at `https://arxiv.org/abs/2310.20473`

†Massachusetts Institute of Technology. `hadought@mit.edu`

‡Massachusetts Institute of Technology. `cejin@mit.edu`. Partially supported by NSF Grant CCF-2129139.

§Massachusetts Institute of Technology. `virgi@mit.edu`. Supported by NSF Grants CCF-2129139 and CCF-2330048 and BSF Grant 2020356.

¶Massachusetts Institute of Technology. `zixuanxu@mit.edu`.

1 Introduction

A t-spanner of a graph is a subgraph that approximates all pairwise distances within a factor of t. Spanners are useful in many applications since they can be significantly sparser than the graphs they represent, yet are still a good representation of the shortest paths metric. As many algorithms are much faster on sparse graphs, running such algorithms on a spanner rather than the graph itself can be significantly more efficient, with only a slight loss in approximation quality.

For undirected graphs, the spanner question is very well understood. It is known that for all integers $k \geq 2$, every n-vertex undirected (weighted) graph contains a $(2k-1)$-spanner on $O(n^{1+1/k})$ edges [ADDJS93] and this is optimal under Erdős' girth conjecture [TZ01].

For directed graphs, however, there can be no non-trivial spanners under the usual shortest paths metric: consider for instance a complete bipartite graph, with edges directed from one partition to the other. Omitting a single edge (u, v) would cause the distance $d(u, v)$ to go from 1 to ∞.

Nevertheless, one can define a notion of a spanner in directed graphs based on the *roundtrip* metric defined by Cowen and Wagner [CW04]: $d(u \leftrightharpoons v) = d(u, v) + d(v, u)$. A roundtrip t-spanner of a directed graph is a subgraph that preserves all pairwise roundtrip distances within a factor of t.

Cen, Duan and Gu [CDG20] showed that basically the same existential results are possible for roundtrip spanners as in undirected graphs: for every integer $k \geq 2$ every n-vertex directed graph contains a $(2k-1)$-roundtrip spanner on $O(kn^{1+1/k} \log n)$ edges. For the special case of $k = 2$, it was known earlier that every n-vertex graph contains a 3-roundtrip spanner on $O(n\sqrt{n})$ edges [RTZ08].

The known results on algorithms for constructing spanners and roundtrip spanners differ drastically however. Baswana and Sen [BS07] presented a randomized linear time algorithm for computing an $O(kn^{1+1/k})$-edge $(2k-1)$-spanner of any n-vertex weighted graph (which was later derandomized [RTZ05]). Meanwhile, the algorithms for constructing roundtrip spanners are much slower.

The first construction of roundtrip spanners was given by Roditty, Thorup and Zwick in [RTZ08], where they gave the construction of $(2k+\varepsilon)$-roundtrip spanners on $\tilde{O}((k^2/\varepsilon)n^{1+1/k})$ edges for any graph with edge weights bounded by $\operatorname{poly} n$ ($\log nW$ dependence in the size otherwise) in $O(mn)$ time. Later, Zhu and Lam [ZL18] derandomized this construction and improved the sparsity of the spanner to contain $\tilde{O}((k/\varepsilon)n^{1+1/k})$ edges. Most recently, Chechik and Lifshitz constructed a 4-roundtrip spanner on $O(n^{3/2})$ edges in $\tilde{O}(n^2)$ time. All currently known results on constructions of roundtrip spanners are summarized in Table 1.

Notice that for all cases with running time faster than mn, the stretch is suboptimal for the used sparsity. This motivates the following:

Question: What is the best construction time for roundtrip spanners of optimal stretch-sparsity tradeoff?

Alongside the construction of roundtrip spanners, another closely related problem is approximating the girth (i.e. the length of the shortest cycle) in directed graphs. The first nontrivial algorithm is by Pachocki, Roditty, Sidford, Tov, Vassilevska Williams [PRSTV18], who gave an $O(k \log n)$ approximation algorithm running in $\tilde{O}(mn^{1/k})$ time. Further improvements by [CLRS20; DV20] followed. Most recently, Chechik and Lifshitz [CL21] obtained a 2-approximation in $\tilde{O}(\min\{n^2, m\sqrt{n}\})$ time, which is optimal for dense graphs. The current known results are summarized in Table 2.

While the 2-approximation result is optimal for dense graphs, and while a $2-\varepsilon$-approximation is (conditionally) impossible in $O((mn)^{1-\delta})$ time [DV20], it is unclear what other approximations (2.5? 3?) are possible with faster algorithms. This motivates the following question:

Question: what is the best running time-approximation tradeoff for the girth of directed graphs?

1.1 Our Results

Throughout this paper, we consider directed graphs on n vertices and m edges with non-negative edge weights. We use $\tilde{O}(\cdot)$ to hide $\operatorname{poly}\log(n)$ factors. All our algorithms are Las Vegas randomized.

THEOREM 1.1. *There is a randomized algorithm that computes a 3-roundtrip spanner of $O(n^{3/2})$ size in $\tilde{O}(m\sqrt{n})$ time.*

This can be compared with the 4-roundtrip spanner of $O(n^{3/2})$ size constructable in $O(n^2 \log n)$ time from [CL21].

Citation	Stretch	Sparsity	Time
Roditty, Thorup, Zwick [RTZ08] △	$2k+\varepsilon$	$\tilde{O}\left(\frac{k^2}{\varepsilon}n^{1+1/k}\right)$	$O(mn)$
Pachocki, Roditty, Sidford, Tov, Vassilevska W. [PRSTV18]	$O(k\log n)$	$\tilde{O}(n^{1+1/k})$	$\tilde{O}(mn^{1/k})$
Chechik, Liu, Rotem, Sidford [CLRS20]	$O(k\log k)$	$\tilde{O}(n^{1+1/k})$	$\tilde{O}(m^{1+1/k})$
Cen, Duan, Gu [CDG20]	$2k-1$	$\tilde{O}(kn^{1+1/k})$	$\tilde{O}(mn\log W)$
Chechik, Liu, Rotem, Sidford [CLRS20] △	$8+\varepsilon$	$\tilde{O}(n^{3/2}/\varepsilon^2)$	$\tilde{O}(m\sqrt{n})$
Dalirrooyfard and Vassilevska W. [DV20] △	$5+\varepsilon$	$\tilde{O}(n^{3/2}/\varepsilon^2)$	$\tilde{O}(m\sqrt{n})$
Chechik and Lifshitz [CL21]	4	$O(n^{3/2})$	$\tilde{O}(n^2)$
New	3	$O(n^{3/2})$	$\tilde{O}(m\sqrt{n})$

Table 1: Known results on constructions of roundtrip spanners on a weight directed graph on n vertices and m edges with edge weight bounded by W. Results marked with △ are subsumed by other results.

Alongside spanners, another important object of study are *emulators*: sparse graphs that approximate all pairwise distances; the difference here is that emulators are not required to be subgraphs, and can be weighted even if the original graph was unweighted. Similar to roundtrip spanners being analogs of spanners in directed graphs, we consider roundtrip emulators which are the analogs of emulators in directed graphs. While emulators are very well studied in undirected graphs [ACIM99; DHZ96; Woo06; Pet09; BKMP10; BV15; BV16; AB17; HP18; LVWX22; KP23], the authors are not aware of any results, for the roundtrip metric. The only known construction of roundtrip emulators is implied from using the roundtrip metric in Thorup-Zwick's distance oracle in [TZ01], which has $(2k-1)$-stretch and $O(kn^{1+1/k})$ edges but requires $\tilde{O}(mn)$ construction time.

We obtain a very fast algorithm that constructs essentially optimal roundtrip emulators (up to the Erdős girth conjecture).

THEOREM 1.2. *For integers $k \geq 3$, there is a randomized algorithm that computes a $(2k-1)$-roundtrip emulator of $O(kn^{1+1/k})$ size in $O(kn^2\log n)$ time.*

While the result is only for roundtrip emulators, rather than spanners, it achieves a much faster running time than any result on roundtrip spanners with optimal approximation-size tradeoff. This is the first algorithm that achieves a sub-mn running time for the problem.

We next focus on the closely related question of girth approximation. We prove:

THEOREM 1.3. *There is a randomized algorithm that computes a 4-multiplicative approximation of the girth of a directed graph in $\tilde{O}(mn^{1/3})$ time.*

Let us compare with the previous known directed girth approximation algorithms. Compared with the 2-approximation in $\tilde{O}(n^2, m\sqrt{n})$ time from [CL21], Theorem 1.3 achieves a better running time for $m \leq o(n^{5/3})$ while raising the approximation ratio to 4. Dalirrooyfard and Vassilevska W. [DV20] gave for every constant $\varepsilon > 0$, a $(4+\varepsilon)$-approximation algorithm running in $\tilde{O}(mn^{\sqrt{2}-1})$ time. Our algorithm removes the ε from the approximation factor and further improves the running time.

1.2 Paper organization After introducing useful notations and terminologies in Section 2, we give a high level overview of our techniques in Section 3. Then, in Section 4 we describe our 3-roundtrip spanner algorithm (Theorem 1.1). In Section 5 we describe our roundtrip emulator algorithm. In Section 6 we describe our girth approximation algorithm. We conclude with a few open questions in Section 7.

2 Preliminaries

We use $\tilde{O}(\cdot)$ to hide $\operatorname{poly}\log(n)$ factors, where n is the number of vertices in the input graph.

Citation	Approximation Factor	Time
Pachocki, Roditty, Sidford, Tov, Vassilevska W. [PRSTV18]	$O(k \log n)$	$\tilde{O}(mn^{1/k})$
Chechik, Liu, Rotem, Sidford [CLRS20] $\triangle$	3	$\tilde{O}(m\sqrt{n})$
Chechik, Liu, Rotem, Sidford [CLRS20]	$O(k \log k)$	$\tilde{O}(m^{1+1/k})$
Dalirrooyfard and Vassilevska W. [DV20] $\triangle$	$4+\varepsilon$	$\tilde{O}(mn^{\sqrt{2}-1})$
Dalirrooyfard and Vassilevska W. [DV20] $\triangle$	$2+\varepsilon$	$\tilde{O}(m\sqrt{n})$
Dalirrooyfard and Vassilevska W. [DV20] $\triangle$	2	$\tilde{O}(mn^{3/4})$ (unweighted)
Chechik and Lifshitz [CL21]	2	$\tilde{O}(\min\{n^2, m\sqrt{n}\})$
New	4	$\tilde{O}(mn^{1/3})$

Table 2: Known results on girth approximation on a weight directed graph on n vertices and m edges with edge weight bounded by W. Results marked with $\triangle$ are subsumed by other results.

In this paper, the input graph $G = (V, E)$ is always a weighted directed graph with vertex set V of size $|V| = n$ and edge set E of size $|E| = m$ with non-negative edge weights. Without loss of generality, we assume G does not have parallel edges. We use $\text{wt}(u, v)$ to denote the weight of the directed edge $(u, v) \in E$. For any two vertices $u, v \in V$, we use $d_G(u, v)$ to denote the distance (length of the shortest path) from u to v in G, and we use $d_G(u \leftrightharpoons v) := d_G(u, v) + d_G(v, u)$ to denote the *roundtrip distance* between u and v. When the context is clear, we simply use $d(u, v)$ and $d(u \leftrightharpoons v)$. For a subset of vertices $W \subseteq V$, we use $G[W]$ to denote the subgraph of G induced by the vertex set W.

The *girth* of G is the length (total edge weight) of the shortest cycle in G. We say a graph $H = (V, E')$ is an *α-roundtrip emulator* of graph $G = (V, E)$, if for every two vertices $u, v \in V$ it holds that $d_G(u \leftrightharpoons v) \leq d_H(u \leftrightharpoons v) \leq \alpha \cdot d_G(u \leftrightharpoons v)$. Furthermore, if H is a subgraph of G, we say H is an *α-roundtrip spanner* of G.

Without loss of generality, we may assume G is strongly-connected, since otherwise we can run the algorithm for girth approximation (or roundtrip spanner/emulator) on each strongly-connected component. In addition, we may assume the maximum degree of G is bounded by $O(m/n)$. This is due to the following regularization lemma shown in [CLRS20]. This assumption will be used in Section 6.

LEMMA 2.1. (REGULARIZATION [CLRS20]) *Given a directed weighted graph $G = (V, E)$ on n vertices and m edges, one can construct a graph H on $O(n)$ vertices and $O(m)$ edges with non-negative edge weights and maximum degree $O(m/n)$ in $O(m)$ time such that all of the following holds:*

1. *All roundtrip distances between pairs of vertices in G are the same in H as in G.*
2. *Given a cycle in H, one can find a cycle of the same length in G in $O(m)$ time.*
3. *Given a subgraph H' in H, one can find in $O(m)$ time a subgraph G' of G such that $|E(G')| \leq |E(H')|$ and the roundtrip distances in G' are the same as in H'.*

In our algorithms, we often use Dijkstra's algorithm to compute single-source distances. On a weighted directed graph $G = (V, E)$, we use *out-Dijkstra* from a source $s \in V$ to refer to Dijkstra algorithm computing distances $d(s, \cdot)$ from s, and use *in-Dijkstra* from s to refer to Dijkstra algorithm computing distances $d(\cdot, s)$ into s.

3 Technical Overview

3.1 Previous Work Throughout this paper, our techniques are based on the following key observation introduced in [CL21].

LEMMA 3.1. (KEY OBSERVATION [CL21]) *Let $G = (V, E)$ be a weighted directed graph with nonnegative edge weights. For vertices $u, v, r \in V$, if*

$$2 \cdot d(v, r) + d(r, u) \leq 2 \cdot d(v, u) + d(u, r), \tag{3.1}$$

then

$$d(u \leftrightharpoons r) \leq 2 \cdot d(u \leftrightharpoons v).$$

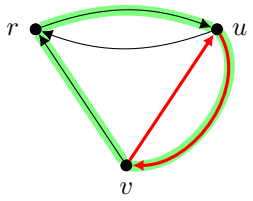

Figure 1: A illustration of Lemma 3.1 with $u, v, r \in V$ satisfying Eq. (3.1). The red cycle $u \rightsquigarrow v \rightsquigarrow u$ can be 2-approximated by the cycle $u \rightsquigarrow v \rightsquigarrow r \rightsquigarrow u$ highlighted green.

An important property of the above observation is that Eq. (3.1) is symmetric with respect to the roles of u and r. This symmetry is crucial to the analysis of the applications of Lemma 3.1 in the previous work [CL21] as well as in our new algorithms, so we first describe it in more details as follows.

The Symmetry Argument Consider the following routine that sparsifies a graph $G = (V, E)$ on n vertices using a random sample $S \subseteq V$. For every vertex $v \in V$, we check for every vertex $s \in S \cap N(v)$ and $u \in N(v)$ where $N(v)$ denotes the out neighborhood of v, if $2d(v, s) + d(s, u) \leq 2d(v, u) + d(u, s)$ then remove the edge (v, u). We say that we use the set S as *eliminators* to perform the sparsification since we are comparing the distance $d(v, u)$ using the distance information involving $s \in S$.

For any two neighbors $u, u' \in N(v)$ (possible $u = u'$), notice that the condition involving v, u, u' compares the distances $2d(v, u) + d(u, u')$ against $2d(v, u') + d(u', u)$, which is the same as if we switch the roles of u and u'. This means that either u eliminates u' or u' eliminates u. (We say "u eliminates u'" meaning that, if $u \in S$, then the edge (v, u') will be removed, namely u' is eliminated from $N(v)$.) So given a random $u \in N(v)$, in expectation half of the pairs (u, u') falls in the case where u can eliminate u' and additionally u can eliminate u itself. Thus, if $N(v) \cap S \neq \varnothing$, then in expectation the procedure will remove at least $|N(v)|/2$ edges. This implies that the graph sparsification can effectively remove a constant fraction of the edges adjacent to the vertices with high out-degree. More specifically, since on expectation, the sample S can hit vertex sets with size $\Omega(n/|S|)$, this procedure can remove a constant fraction of the outgoing edges adjacent to vertices with degree $\Omega(n/|S|)$. So if we repeat this process $\Theta(\log n)$ rounds, on expectation we can reduce the out-degree of every vertex to at most $O(n/|S|)$.

Applications of the key observation Now we are ready to explain how the above Lemma 3.1 is useful for constructing roundtrip spanners and approximating directed cycles.

1. **Girth approximation: reduce search space.** Suppose we can take a small random subset of vertices $S \subseteq V$ and for each vertex $s \in S$ and set the current girth estimate as the length of the shortest cycle passing through any vertex in S. Then if $r \in S$, Lemma 3.1 shows that we no longer have to consider the shortest cycle passing through v and u satisfying Eq. (3.1). This is because the shortest cycle passing through u and v can already be 2-approximated by the shortest cycle passing through r. Then if we want to search for cycles passing through v that cannot be 2-approximated, we would not need to consider the vertex u. Thus, using the sample S, we can compute a pruned vertex set $B(v) \subseteq V$ that contains all the vertices $u \in V$ such that Eq. (3.1) does not hold for any $r \in S$. By the symmetry argument, each sample that hits the set $B(v)$ can reduce the size of $B(v)$ by a constant fraction. So over $\Theta(\log n)$ rounds, we can obtain a pruned set of size roughly $O(n/|S|)$. This technique is used in the 2-approximation in [CL21] and will be used in our algorithm for computing a 4-approximation of the girth in Section 6.

2. **Roundtrip spanners: graph sparsification.** Suppose we take a random subset $S \subseteq V$ and add all the in/out shortest path tree from S to our spanner H. We apply Lemma 3.1 to the vertices $u, v, r \in V$ where

u, r are out-neighbors of v. If $r \in S$, Lemma 3.1 implies that we can delete the edge (v, u) since the shortest cycle containing (v, u) can be 2-approximated by the cycle passing through r and u, which is already added to the spanner H. As explained previously by the symmetry argument, in expectation we can reduce the out-degree of every vertex to roughly $O(n/|S|)$ if we repeat this process for $\Theta(\log n)$ rounds. This technique was used in the construction of 4-roundtrip spanners in [CL21] and will be used in our construction of 3-roundtrip spanner in Section 4 and our $(2k-1)$-roundtrip emulator in Algorithm 2.

3.2 Our Techniques Our techniques consist of a collection of extensions to the techniques introduced in [CL21]. We now highlight the novel components in each of our algorithms.

3-Roundtrip Spanner in $\tilde{O}(m\sqrt{n})$ Time Our algorithm follows from a modification of Chechik and Lifshitz's [CL21] 4-roundtrip spanner algorithm, which was based on the graph sparsification approach mentioned earlier. Our new idea lies in a more careful analysis of the stretch of the spanner: instead of directly bounding the roundtrip distance $d_H(u \leftrightharpoons v)$ between vertices u, v in the spanner H as Chechik and Lifshitz did, we separately bound the one-way distances $d_H(u, v), d_H(v, u)$ and add them up. After a slight change in their algorithm (namely, by computing distances in the original graph rather than in the sparsified graph in each round), this analysis enables us to improve the stretch from 4 to 3.

$(2k-1)$-Roundtrip Emulator in $\tilde{O}(n^2)$ Time The celebrated approximate distance oracle result of Thorup and Zwick [TZ01] immediately yields $(2k-1)$-emulators of $O(kn^{1+1/k})$ size for any metric. But a straightforward implementation of their generic algorithm in the roundtrip metric would require computing single source shortest paths from all vertices, in $\tilde{O}(mn)$ total time. For the easier case of undirected graphs, [TZ01] reduced the construction time to $O(kmn^{1/k})$, but unfortunately these techniques based on balls and bunches do not yield a speedup in our roundtrip distance setting.

Our faster roundtrip emulator algorithm combines Thorup and Zwick's technique [TZ01] with the graph sparsification approach of [CL21]. The intuition is that, since the bottleneck of the generic Thorup-Zwick algorithm lies in computing single source shortest paths, a natural idea is to use [CL21]'s approach to gradually sparsify the graph so that Dijkstra's algorithm can run faster. More specifically, recall that the Thorup-Zwick algorithm takes a sequence of nested vertex samples $S_1 \subseteq \cdots \subseteq S_k = V$ which serve as intermediate points for routing approximate shortest paths. In our case, these vertex samples also play the same role as in the graph sparsification approach described earlier, where short cycles going through these vertex samples can approximate the cycles we care about. This results in a multi-round algorithm that interleaves graph sparsification steps and running Dijkstra from vertices of S_i (with gradually increasing size) in $\tilde{O}(n^2)$ total time. It is not obvious that the $(2k-1)$-stretch of Thorup-Zwick still holds after adding these graph sparsification steps, but it turns out the stretch analysis of Thorup-Zwick fits nicely with the cycle approximation arguments, and with a careful analysis we are still able to show $(2k-1)$ stretch when $k \geq 3$.

For some technical reason related to the sampling argument of Thorup-Zwick, we had to slightly simplify the graph sparsification techniques of [CL21], in order to avoid an undesirable extra logarithmic factor in the sparsity bound of our roundtrip emulator. See the discussion in Remark 4.1 and the proof of Lemma 5.2.

4-Approximation of Girth in $\tilde{O}(mn^{1/3})$ Time Our algorithm vastly extends the technique of the 2-approximate girth algorithm in $\tilde{O}(\min\{n^2, m\sqrt{n}\})$ time by Chechik and Lifshitz [CL21]. In the 2-approximation algorithm, one takes a sample S of size $O(\sqrt{n})$ and uses in/out Dijkstra's to exactly compute the shortest cycle going through every $s \in S$. Then using S as eliminators, compute for every vertex $v \in V$ a pruned vertex set $B(v)$ of size $O(\sqrt{n})$, and search for short cycles from v on $G[B(v)]$. A natural attempt to improve the running time is to generalize this framework to multiple levels: take a sequence of vertex samples of increasing sizes $S_1, \ldots, S_{k-1}, S_k = V$ and compute a sequence of pruned vertex subsets $V = B_1(v), B_2(v), \ldots, B_k(v)$ of decreasing sizes for every v, so that one can run Dijkstra from/to every vertex in S_i on $G[B_i(v)]$ in $\tilde{O}(mn^{1/k})$ time. However, it is unclear how to do this since one can no longer check the condition Eq. (3.1) due to not having all the distance information from/to every vertex $s \in S_i$, and thus we cannot compute the sets $B_i(v)$ as desired.

In this work we are able to implement the above plan for $k = 3$, obtaining a 4-approximation girth algorithm in $\tilde{O}(mn^{1/3})$ time. We deal with the problem of not having enough distance information to compute $B_3(v)$ by using a certain distance underestimate obtained from the distance information from S_1, and enforcing a stricter set of requirements on the vertices that we explore, so that we always have their distance information available. We also apply more novel structural lemmas about cycle approximation that extend the key observation Lemma 3.1 of [CL21] in various ways, which may be of independent interest. As a result, our 4-approximation algorithm

becomes more technical than the previous 2-approximation algorithm in $\tilde{O}(m\sqrt{n})$ time.

Here, we highlight the key structural lemma (Lemma 6.15) that enabled us to overcome the above described difficulty. It is illustrated in the following Fig. 2, which can be viewed as an extension of Lemma 3.1 from 3 vertices to 4 vertices. As illustrated, if there exists some vertex r_2 that is in a short cycle with v but not in a short cycle with u, then we can find some vertex r_1 such that the cycle $v \rightsquigarrow r_2 \rightsquigarrow r_1 \rightsquigarrow u \rightsquigarrow v$ (highlighted in green) can approximate the shortest cycle passing through u and v (the cycle in red). Then similar to how we can use Lemma 3.1, we can ignore the vertex u in our search for the shortest cycle passing through v.

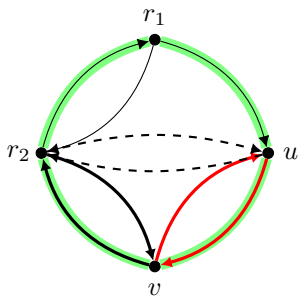

Figure 2: If there exists a vertex r_2 that is in a short cycle with v but not in a short cycle with u, then we can find a vertex r_1 such that the cycle passing through $v \rightsquigarrow r_2 \rightsquigarrow r_1 \rightsquigarrow u$ (the cycle highlighted in green) can approximate the shortest cycle passing through u and v (the cycle in red).

Furthermore, we note that we had to introduced a number of technicalities and a new structural theorem just to implement our proposed generalization for $k = 3$. So it is entirely unclear how to further generalize this approach for $k \geq 4$. Moreover, even if one successfully implements the proposed generalization naively, one would only obtain a 2^{k-1}-approximation in $\tilde{O}(mn^{1/k})$ time, which is far from being desirable.

4 3-Roundtrip Spanner

In this section, we present our algorithm for constructing a 3-roundtrip spanner with $O(n^{3/2})$ edges in time $\tilde{O}(m\sqrt{n})$ (Theorem 1.1). Our algorithm closely follows the previous $\tilde{O}(n^2)$-time 4-roundtrip spanner algorithm by Chechik and Lifshitz [CL21], but we use a more careful analysis to improve the stretch from 4 to 3.

4.1 Algorithm and stretch analysis Our algorithm (see pseudocode in Algorithm 1) has a similar structure as in [CL21]: We iteratively sample vertex subsets $S_i \subseteq V$ with geometrically increasing expected sizes $\mathbb{E}[|S_i|]$ up to $\sqrt{n}$. In each iteration i, we add the shortest path trees from/to every $s \in S_i$ into the spanner, and sparsify the input graph G using the method of [CL21] based on S_i (Line 8 – Line 11). Finally, we are able to sparsify the graph to contain only $O(n^{3/2})$ edges in expectation, and we will add these remaining edges to the spanner. Over all iterations, we add a total of $2n \cdot O(\sqrt{n}) = O(n^{3/2})$ edges to the spanner, and we only run $O(\sqrt{n})$ instances of Dijkstra which take $\tilde{O}(m\sqrt{n})$ total time.

The main difference from [CL21] lies in the sparsification rule at Line 10. Our rule is based on comparing distances in the original input graph G, while Chechik and Lifshitz's rule was based on distances in the sparsified graph G_i.

Remark 4.1. *Readers familiar with [CL21] may notice some other technical differences between Algorithm 1 and [CL21]: in order to remove a* $\log n$ *factor from the spanner size, Chechik and Lifshitz [CL21] had to resample* S_i *in case it is "unsuccessful" (i.e., Line 11 did not remove sufficiently many edges), whereas our Algorithm 1 achieves the same goal without resampling. Another difference is that we fix the sample rate of each iteration* i *at Line 5, while [CL21] determines sample rate based on the current size* $|E(G_i)|$.

These modifications are not essential for obtaining this 3-spanner result. In particular, our algorithm is equivalent to simply sampling $\sum_{i=0}^{\Delta-1} |S_i| = O(\sqrt{n})$ *vertices all at once. Nonetheless, we present it in this way because it leads to cleaner implementation and analysis. Furthermore, it will be useful later for our emulator algorithm in Section 5 (where we require* S_i *to be uniformly and independently sampled).*

Algorithm 1: 3-ROUNDTRIP-SPANNER(G)

Input: A weighted directed graph $G = (V, E)$
Output: a 3-roundtrip spanner $H \subseteq G$

1 $H \leftarrow (V(G), \varnothing)$
2 $G_0 \leftarrow G$
3 Let $\Delta := \lceil \log_{3/2} \sqrt{n} \rceil$, and $\alpha := (\sqrt{n})^{1/\Delta}$. // $\alpha \in [5/4, 3/2]$ when $\sqrt{n} \geq 2$
4 **for** $i \leftarrow 0, 1, \ldots, \Delta - 1$ **do**
5 Sample $S_i \subseteq V$ by including each vertex with probability α^i/n independently
6 Compute $d_G(s, v), d_G(v, s)$ for all $s \in S_i$ and $v \in V$ using Dijkstra
7 Add to H the shortest path trees in G from/to every vertex in $s \in S_i$
8 $G_{i+1} \leftarrow G_i$
9 **for** $(x, y), (x, s) \in E(G_i)$ *such that* $s \in S_i$ **do**
10 **if** $2d_G(x, s) + d_G(s, y) \leq 2\,\mathrm{wt}(x, y) + d_G(y, s)$ **then**
11 Remove the edge (x, y) from G_{i+1}
12 $H \leftarrow H \cup E(G_\Delta)$
13 **return** H

Now we prove the stretch of the spanner constructed by Algorithm 1. Our proof mostly follows [CL21]; the key difference is that [CL21] estimated $d_H(u \leftrightharpoons v)$ as a whole, while our improvement comes from separately estimating $d_H(u, v)$ and $d_H(v, u)$ and combine them to obtain an upper bound for $d_H(u \leftrightharpoons v)$.

LEMMA 4.2. *For any two vertices $u, v \in V$,*

$$d_H(u, v) \leq 2d_G(u, v) + d_G(v, u).$$

As a consequence, $d_H(u \leftrightharpoons v) \leq 3d_G(u \leftrightharpoons v)$ for any $u, v \in V$.

Proof. Let P denote the shortest path from u to v in G. If P is completely contained in the final G_Δ, then by Line 12 clearly $d_H(u, v) = d_G(u, v)$ and we are done. For the remaining case, consider any iteration i in which some edge (x, y) of P is removed from G_{i+1} at Line 11. By Line 10, there is a vertex $s \in S_i$ such that

$$2d_G(x, s) + d_G(s, y) \leq 2\,\mathrm{wt}(x, y) + d_G(y, s),$$

which means

$$\begin{aligned} d_G(x, s) + d_G(s, y) &\leq 2\,\mathrm{wt}(x, y) + d_G(y, s) - d_G(x, s) \\ &\leq 2\,\mathrm{wt}(x, y) + d_G(y, x). \end{aligned} \tag{4.2}$$

Since H contains the shortest path trees in G from s and to s (by Line 7), we have

$$\begin{aligned} d_H(u, v) &\leq d_H(u, s) + d_H(s, v) \\ &= d_G(u, s) + d_G(s, v) \\ &\leq d_G(u, x) + d_G(x, s) + d_G(s, y) + d_G(y, v) \\ &\leq d_G(u, x) + 2\,\mathrm{wt}(x, y) + d_G(y, x) + d_G(y, v). \end{aligned}$$

(by Eq. (4.2))

Then, using $d_G(y, x) \leq d_G(y, v) + d_G(v, u) + d_G(u, x)$, we immediately obtain

$$\begin{aligned} d_H(u, v) &\leq 2\big(d_G(u, x) + \mathrm{wt}(x, y) + d_G(y, v)\big) + d_G(v, u) \\ &= 2d_G(u, v) + d_G(v, u). \end{aligned}$$

∎

4.2 Analysis of sparsity and running time Now we analyze the expected size of H and the running time of Algorithm 1. We first prove the following lemma that bounds the expected number of edges in G_i. From now on we use $m_i := |E(G_i)|$. Recall from Line 3 that $\Delta = \lceil \log_{3/2} \sqrt{n} \rceil$, $\alpha = (\sqrt{n})^{1/\Delta}$, and note that $\alpha \in [5/4, 3/2]$ when $\sqrt{n} \geq 2$.

LEMMA 4.3. *For $i = 0, \ldots, \Delta$, we have*

$$\mathbb{E}[m_i] \leq 2n^2/\alpha^i.$$

Proof. In the i-th iteration, we sample $S_i \subseteq V$ by including each vertex independently with probability $p_i := \alpha^i/n$. In the following, we focus on a particular vertex $x \in V$, and let $\deg_i(x) = |N_{G_i}(x)|$ denote the out-degree of x in G_i.

For any two out-neighbors $v_s, v_y \in N_{G_i}(x)$, we say v_s *eliminates* v_y, if the inequality at Line 10 holds for $(s, y) := (v_s, v_y)$. Observe that the inequality at Line 10 is (essentially) symmetric with respect to y and s, and one immediately observes that for any two $v, v' \in N_{G_i}(x)$ (possibly $v = v'$), either v eliminates v', or v' eliminates v.[1] Then, Line 11 indicates that, for any $v_s, v_y \in N_{G_i}(x)$, if $v_s \in S_i$ and v_s eliminates v_y, then $v_y \notin N_{G_{i+1}}(x)$. Therefore, $\deg_{i+1}(x)$ is the number of out-neighbors of x that are not eliminated by anyone from S_i.

For every $v \in N_{G_i}(x)$, let e_v denote the number of $v' \in N_{G_i}(x)$ that eliminates v (including v itself). We have

$$1 \leq e_v \leq \deg_i(x) \tag{4.3}$$

and

$$\frac{1}{|N_{G_i}(x)|} \sum_{v \in N_{G_i}(x)} e_v = \frac{\deg_i(x) + 1}{2}. \tag{4.4}$$

Then, over a uniformly independently sampled set S_i of eliminators, we analyze the expected number of out-neighbors of x that are not eliminated, as follows:

$$\begin{aligned}
\mathop{\mathbb{E}}_{S_i}[\deg_{i+1}(x) \mid G_i] &= \sum_{v \in N_{G_i}(x)} (1 - p_i)^{e_v} \\
\text{(by convexity of } f(x) = (1-p_i)^x \text{, and Eqs. (4.3) and (4.4))} \quad &\leq \frac{|N_{G_i}(x)|}{2} \cdot (1 - p_i)^1 + \frac{|N_{G_i}(x)|}{2} \cdot (1 - p_i)^{\deg_i(x)} \\
&= \frac{\deg_i(x)}{2} \cdot (1 - p_i + (1 - p_i)^{\deg_i(x)}) \\
&\leq \frac{\deg_i(x)}{2} \cdot (1 + e^{-p_i \deg_i(x)}).
\end{aligned}$$

Multiplying both sides by p_{i+1},

$$\begin{aligned}
\mathop{\mathbb{E}}_{S_i}[p_{i+1} \deg_{i+1}(x) \mid G_i] &\leq \frac{p_{i+1} \deg_i(x)}{2} \cdot (1 + e^{-p_i \deg_i(x)}) \\
\text{(by } p_i = \alpha^i/n\text{)} \quad &= \frac{\alpha}{2}(p_i \deg_i(x) + p_i \deg_i(x) e^{-p_i \deg_i(x)}) \\
&< \frac{\alpha}{2}(p_i \deg_i(x) + 1).
\end{aligned}$$

Hence,

$$\mathbb{E}[p_{i+1} \deg_{i+1}(x)] \leq \frac{\alpha}{2}(\mathbb{E}[p_i \deg_i(x)] + 1).$$

Since $0 \leq p_0 \deg_0(x) \leq \frac{1}{n} \cdot (n-1) < 1$, by induction we obtain $\mathbb{E}[p_i \deg_i(x)] < \alpha/(2-\alpha)$ for all i (recall $\alpha \leq 3/2 < 2$). Summing over all $x \in V$, we obtain

$$\mathbb{E}[m_i] = \frac{1}{2} \sum_{x \in V} \mathbb{E}[\deg_i(x)] \leq \frac{n}{2} \cdot \frac{\alpha/(2-\alpha)}{p_i} = \frac{\alpha}{4 - 2\alpha} n^2/\alpha^i < 2n^2/\alpha^i.$$

□

[1] In more detail, by symmetry we can pick $(s, y) := (v, v')$ or (v', v) to satisfy $2d_G(x, s) + d_G(s, y) \leq 2d_G(x, y) + d_G(y, s)$. Then, the inequality at Line 10 holds due to $d_G(x, y) \leq \mathrm{wt}(x, y)$.

Now we are ready to present the analysis of the sparsity of H and the running time of our algorithm.

Sparsity For each iteration i of Algorithm 1, by definition (Line 5) we have expected sample size

$$\mathbb{E}[|S_i|] = \alpha^i,$$

and we add the shortest path trees from / to every vertex in $s \in S_i$ in G, which contain $|S_i| \cdot 2(n-1)$ edges. So summing over all iterations, the number of edges we add in expectation is at most (recall $\alpha \geq 5/4$)

$$\mathbb{E}\left[\sum_{i=0}^{\Delta-1} |S_i| \cdot 2(n-1)\right] = 2(n-1) \cdot \sum_{i=0}^{\log_\alpha \sqrt{n}-1} \alpha^i = \frac{(2n-1)(\sqrt{n}-1)}{\alpha - 1} < 8n^{3/2}.$$

In the last step (Line 12), we add all the edges in G_Δ to H. By Lemma 4.3, we have

$$\mathbb{E}[|E(G_\Delta)|] \leq 2n^2/\alpha^\Delta = 2n^{3/2}.$$

Thus, in expectation we add $O(n^{3/2})$ edges to H in total as desired.

Running Time In each iteration i, the bottleneck is at Line 6 where we run $|S_i|$ instances of Dijkstra on G, each taking $O(m + n \log n)$ time. The sparsification steps (Line 8 – Line 11) can be implemented in $O(|S_i| \cdot |E(G_i)|) \leq O(|S_i| \cdot m)$ time. So in expectation the total time taken by the algorithm is bounded by

$$\mathbb{E}\left[\sum_{i=0}^{\Delta-1} |S_i| \cdot O(m + n\log n)\right] = O(m + n\log n) \sum_{i=0}^{\log_\alpha \sqrt{n}-1} \alpha^i = O(m\sqrt{n} + n\sqrt{n}\log n).$$

5 $(2k-1)$-roundtrip emulator in nearly quadratic time

In this section, we give the construction of a $(2k-1)$-roundtrip emulator on $O(kn^{1+1/k})$ edges running in $O(kn^2 \log n)$ time for $k \geq 3$ (Theorem 1.2). Our algorithm does not work for $k = 2$. (For $k = 2$, our 3-roundtrip spanner algorithm from Section 4 has $\tilde{O}(m\sqrt{n})$ time complexity, which is slower than $\tilde{O}(n^2)$ for any nontrivial input size $m \gg n^{1.5}$.)

5.1 Algorithm

Our algorithm carefully combines ideas from Thorup-Zwick distance oracle [TZ01] and the graph sparsification technique introduced in [CL21]. The pseudocode of our algorithm is given in Algorithm 2. The main body contains $(k-1)\Delta = \Theta(\log n)$ iterations (indexed by $i = r\Delta + t$), divided into $(k-1)$ rounds (indexed by $r \in \{0, \ldots, k-2\}$), where each round consists of Δ iterations (indexed by the inner loop variable $t \in \{0, \ldots, \Delta - 1\}$). The i-th iteration samples a vertex subset S_i, whose expected size $\mathbb{E}[|S_i|]$ gradually increases from 1 in the 0-th iteration to $\Theta(n^{(k-1)/k})$ in the last iteration. In each iteration we run in/out-Dijkstra from every sampled vertex $s \in S_i$ on the current (sparsified) graph $G_i \subseteq G$. Using the obtained distance information from/to S_i, we not only perform the graph sparsification steps (Line 14–Line 17) as in [CL21], but also compute pivots $p_i(u) \in S_i$ and bunches $B_i(u) \subseteq S_i$ used in Thorup and Zwick's algorithm [TZ01] (in the roundtrip metric) and adds edges to the emulator H accordingly (Line 10 – Line 13). The main complication compared to [TZ01] is that we now have a sequence of (gradually sparsified) graphs G_i involved rather than a single graph G, and the pivots $p_i(u)$ are defined using the distances on the current graph G_i, while the bunches $B_i(u)$ are defined with respect to the pivot $p_{r\Delta-1}(u)$ on the graph $G_{r\Delta-1}$ from the *previous round* of the outer loop r.

By our parameter setting, we expect each round in the outer loop to roughly decrease the size of the current graph by a factor of $n^{1/k}$. After running all $(k-1)$ rounds, we can show the remaining graph $G_{(k-1)\Delta}$ has $O(n^{1+1/k})$ edges in expectation, and we add all of them to the emulator H.

5.2 Analysis of sparsity and running time

We can without loss of generality assume $k \leq \log n$, since otherwise we can run the algorithm for $k = \lfloor \log n \rfloor$ and still satisfy all the requirements. Recall from Line 3 that $\Delta = \lceil \log_{3/2} n^{1/k} \rceil$, $\alpha := (n^{1/k})^{1/\Delta}$, and note that $\alpha \in [5/4, 3/2]$.

Algorithm 2 has $(k-1)\Delta = \log_\alpha n^{1-1/k}$ iterations. It has a similar structure as our earlier Algorithm 1 for 3-roundtrip spanner (except for the additional Line 10 – Line 13 here). For each iteration $i = r \cdot \Delta + t$ where $r \in \{0, \ldots, k-2\}$ and $t \in \{0, 1, \ldots, \Delta - 1\}$, by Line 8 we have

$$\mathbb{E}[|S_i|] = \alpha^i.$$

Algorithm 2: $(2k-1)$-EMULATOR(G) (for $k \geq 3$)

Input: A weighted directed graph $G = (V, E)$
Output: A $(2k-1)$-roundtrip emulator H of G

1 $H \leftarrow (V(G), \varnothing)$
2 $G_0 \leftarrow G$
3 Let $\Delta := \lceil \log_{3/2} n^{1/k} \rceil$, and $\alpha := (n^{1/k})^{1/\Delta}$. // $\alpha \in [5/4, 3/2]$ when $n^{1/k} \geq 2$
4 Let G_{-1} = empty graph and $p_{-1}(u) := \perp$ for all $u \in V$. // $d_{G_{-1}}(u \leftrightharpoons p_{-1}(u)) = +\infty$.
5 **for** $r \leftarrow 0, \ldots, k-2$ **do**
6 **for** $t \leftarrow 0, \ldots, \Delta - 1$ **do**
7 Let $i := r\Delta + t$
8 Sample $S_i \subseteq V$ by including each vertex with probability α^i/n independently
9 Compute $d_{G_i}(s, v), d_{G_i}(v, s)$ for all $s \in S_i$ and $v \in V$ using Dijkstra
10 Define pivot $p_i(u) := \arg\min_{s \in S_i} d_{G_i}(u \leftrightharpoons s)$ for all $u \in V$
11 Define bunch $B_i(u) := \{s \in S_i : d_{G_i}(u \leftrightharpoons s) < d_{G_{r\Delta-1}}(u \leftrightharpoons p_{r\Delta-1}(u))\}$.
12 **for** $u \in V, s \in \{p_i(u)\} \cup B_i(u)$ **do**
13 Add edge (u, s) with weight $d_{G_i}(u, s)$ and edge (s, u) with weight $d_{G_i}(s, u)$ to H
14 $G_{i+1} \leftarrow G_i$
15 **for** $(x, y), (x, s) \in E(G_i)$ *such that* $s \in S_i$ **do**
16 **if** $2d_{G_i}(x, s) + d_{G_i}(s, y) \leq 2\,\mathrm{wt}(x, y) + d_{G_i}(y, s)$ **then**
17 Remove the edge (x, y) from G_{i+1}
18 $H \leftarrow H \cup G_{(k-1)\Delta}$
19 **return** H

Similar to the analysis of our 3-roundtrip spanner algorithm, we have the following lemma on the expected number edges $m_i := |E(G_i)|$.

LEMMA 5.1. *In Algorithm 2, for* $0 \leq i \leq (k-1)\Delta$ *we have*

$$\mathbb{E}[m_i] \leq 2n^2/\alpha^i.$$

The proof of Lemma 5.1 is identical to the proof of Lemma 4.3 for Algorithm 1, and is omitted here. Note that in Algorithm 2, Line 10 – Line 13 do not affect edges of G_i, and the remaining part of the algorithm is almost identical to Algorithm 1 except that the number of iterations is changed from Δ to $(k-1)\Delta$ (and α is changed accordingly), and the sparsification rule (Line 16) now depends on distances of G_i instead of G. These modifications do not affect the proof of Lemma 4.3.

Running Time Over all iterations of the inner **for** loop, for every $i = 0, \ldots, (k-1)\Delta - 1$, the bottleneck is to run $|S_i|$ instances of in/out-Dijkstras on G_i (Line 9), each taking $O(m_i + n \log n)$ time. The sparsification steps (Line 14 – Line 17) can be implemented in $O(|S_i| \cdot m_i)$ time. Thus by Lemma 5.1, the expected total running time of our algorithm can be bounded by (note that S_i and m_i are independent random variables)

$$\begin{aligned}
\mathbb{E}\Big[\sum_{i=0}^{(k-1)\Delta-1} |S_i| \cdot O(m_i + n\log n)\Big] &\leq \sum_{i=0}^{(k-1)\Delta-1} \alpha^i \cdot O\big(2n^2/\alpha^i + n\log n\big) \\
&= \sum_{i=0}^{(k-1)\Delta-1} \alpha^i \cdot O\big(2n^2/\alpha^i\big) \\
&= O(n^2 \cdot (k-1)\Delta) \\
&= O(n^2 \log n).
\end{aligned}$$

Sparsity Similar to in [TZ01], we first bound the expected size of the bunches defined in Line 11. As mentioned earlier in Remark 4.1, here we rely on the property that the vertex samples S_i are uniform and independent.

LEMMA 5.2. *For each $i = r \cdot \Delta + t$ (where $r \in \{0, \dots, k-2\}, t \in \{0, \dots, \Delta - 1\}$) and each vertex $u \in V$, we have*

$$\mathbb{E}[|B_i(u)|] = \alpha^{t+1}.$$

Proof. By definition of B_i at Line 11, since $G_i \subseteq G_{r\Delta-1}$ and thus $d_{G_{r\Delta-1}}(\cdot,\cdot) \le d_{G_i}(\cdot,\cdot)$, we have

$$\begin{aligned} |B_i(u)| &= |\{s \in S_i : d_{G_i}(u \leftrightharpoons s) < d_{G_{r\Delta-1}}(u \leftrightharpoons p_{r\Delta-1}(u))\}| \\ &\le |\{s \in S_i : d_{G_{r\Delta-1}}(u \leftrightharpoons s) < d_{G_{r\Delta-1}}(u \leftrightharpoons p_{r\Delta-1}(u))\}|. \end{aligned}$$

Sort all $v \in V$ in increasing order of $d_{G_{r\Delta-1}}(u \leftrightharpoons v)$. Then $p_{r\Delta-1}(u) = \arg\min_{s \in S_{r\Delta-1}} d_{G_{r\Delta-1}}(u \leftrightharpoons s)$ is the first vertex in this ordering that is included in $S_{r\Delta-1}$, and $|B_i(u)|$ is bounded by the number of vertices included in S_i that occur before $p_{r\Delta-1}(u)$ in this ordering. Since $S_{r\Delta-1}$ and S_i are sampled uniformly and independently (conditioned on this ordering determined by $G_{r\Delta-1}$), the expected number of vertices included by $B_i(u)$ is at most

$$\sum_{j=1}^{n} \frac{\alpha^i}{n} \cdot \left(1 - \frac{\alpha^{r\Delta-1}}{n}\right)^j \le \frac{\alpha^i/n}{\alpha^{r\Delta-1}/n} = \alpha^{t+1}.$$

□

As a direct corollary, we can bound the expected total bunch size.

COROLLARY 5.3.

$$\mathbb{E}\left[\sum_{i=0}^{(k-1)\Delta-1} \sum_{u \in V} |B_i(u)|\right] \le O(kn^{1+1/k}).$$

Proof. For each $r \in \{0, \dots, k-2\}$, by Lemma 5.2 and linearity of expectation, we have

$$\mathbb{E}\left[\sum_{t=0}^{\Delta-1} |B_{r\Delta+t}(u)|\right] = \sum_{t=0}^{\log_\alpha n^{1/k} - 1} \alpha^{t+1} = O(n^{1/k})$$

for each $u \in V$. Summing over all $r \in \{0, \dots, k-2\}$ and $u \in V$,

$$\mathbb{E}\left[\sum_{i=0}^{(k-1)\Delta-1} \sum_{u \in V} |B_i(u)|\right] = \sum_{u \in V} \sum_{r=0}^{k-2} \mathbb{E}\left[\sum_{t=0}^{\Delta-1} |B_{r\Delta+t}(u)|\right] \le O(kn^{1+1/k}).$$

□

Now we can analyze the size of the emulator constructed by Algorithm 2.

LEMMA 5.4. *The emulator H returned by Algorithm 2 has expected size*

$$\mathbb{E}[|H|] \le O(kn^{1+1/k}).$$

Proof. By Corollary 5.3, the total number edges added at Line 13 has expectation at most

$$\sum_{i=0}^{(k-1)\Delta-1} \sum_{u \in V} 2(|B_i(u)| + 1) \le O(kn^{1+1/k}).$$

In the end at Line 18, we add all the edges in $G_{(k-1)\Delta}$ to H. By Lemma 5.1, we know that

$$\mathbb{E}[m_{(k-1)\Delta}] \le 2n^2/\alpha^{(k-1)\Delta} = 2n^2/\alpha^{(k-1)\log_\alpha n^{1/k}} = 2n^{1+1/k}.$$

Thus the expected size of H is $O(kn^{1+1/k})$ as desired. □

5.3 Stretch analysis By construction, it is clear that $d_H(u,v) \geq d_G(u,v)$ for all $u, v \in V$.

From now on we fix a pair of $u, v \in V$ and consider the shortest cycle C of length $g := d_G(u \leftrightharpoons v)$ containing the vertices u, v. We will prove $d_H(u \leftrightharpoons v) \leq (2k-1)d_G(u \leftrightharpoons v)$.

If C is included in the final sparsified graph $G_{(k-1)\Delta}$, then by Line 18 we know C is included in the emulator H and thus $d_H(u \leftrightharpoons v) = d_G(u \leftrightharpoons v)$. Hence, in the following we assume $C \not\subseteq G_{(k-1)\Delta}$, and let $0 \leq i < (k-1)\Delta$ be the first iteration in which C is destroyed by the sparsification steps, that is, $C \subseteq E(G_i)$ but $C \not\subseteq E(G_{i+1})$.

We first prove the following Lemma 5.5 (which is essentially from [CL21]), which shows that when C is destroyed in iteration i, it can be 2-approximated by a cycle going through some sampled vertex in iteration i.

LEMMA 5.5. *Then there exists some $s \in S_i$ such that*

$$d_{G_i}(v \leftrightharpoons s) \leq 2g, \text{ and } d_{G_i}(u \leftrightharpoons s) \leq 2g. \tag{5.5}$$

Proof. By definition of i, $d_{G_i}(u \leftrightharpoons v) = g = d_G(u \leftrightharpoons v)$. Let $(x,y) \in C \setminus E(G_{i+1})$ be an edge on the cycle that is removed. Assume without loss of generality that (x,y) lies on the shortest path from u to v (otherwise, we can swap the roles of u and v). By Line 16, this means that there exists some $s \in S_i$ where

$$2d_{G_i}(x,s) + d_{G_i}(s,y) \leq 2\,\mathrm{wt}(x,y) + d_{G_i}(y,s), \tag{5.6}$$

which implies the following estimate on the length of the shortest cycle going through u, s, v in G_i:

$$\begin{aligned}
&d_{G_i}(u,s) + d_{G_i}(s,v) + d_{G_i}(v,u) \\
\text{(triangle inequality)} \quad &\leq d_{G_i}(u,x) + d_{G_i}(x,s) + d_{G_i}(s,y) + d_{G_i}(y,v) + d_{G_i}(v,u) \\
\text{(by Eq. (5.6))} \quad &\leq d_{G_i}(u,x) + 2\,\mathrm{wt}(x,y) + d_{G_i}(y,s) - d_{G_i}(x,s) + d_{G_i}(y,v) + d_{G_i}(v,u) \\
&\leq d_{G_i}(u,x) + 2\,\mathrm{wt}(x,y) + (d_{G_i}(y,v) + d_{G_i}(v,u) + d_{G_i}(u,x) + d_{G_i}(x,s)) \\
\text{(expanding } d_{G_i}(y,s) \text{ using triangle inequality)} \quad &\\
&\quad - d_{G_i}(x,s) + d_{G_i}(y,v) + d_{G_i}(v,u) \\
&= 2d_{G_i}(u,x) + 2\,\mathrm{wt}(x,y) + 2d_{G_i}(y,v) + 2d_{G_i}(v,u) \\
((x,y) \text{ lies on shortest path from } u \text{ to } v) \quad &\\
&= 2d_{G_i}(u,v) + 2d_{G_i}(v,u) \\
&= 2g.
\end{aligned}$$

Thus

$$d_{G_i}(u \leftrightharpoons s) \leq d_{G_i}(u,s) + d_{G_i}(s,v) + d_{G_i}(v,u) \leq 2g$$

and the same holds for $d_{G_i}(v \leftrightharpoons s)$ as desired. ∎

By Lemma 5.5 and the definition of the pivots $p_i(u) := \arg\min_{s \in S_i} d_{G_i}(u \leftrightharpoons s), p_i(v) := \arg\min_{s \in S_i} d_{G_i}(v \leftrightharpoons s)$ (Line 10), we have

$$d_{G_i}(u \leftrightharpoons p_i(u)) \leq 2g, \text{ and } d_{G_i}(v \leftrightharpoons p_i(v)) \leq 2g. \tag{5.7}$$

We first consider the case when both $s \in B_i(u)$ and $s \in B_i(v)$ hold (where s is defined in Lemma 5.5). In this case, we have

$$\begin{aligned}
d_H(u \leftrightharpoons v) &\leq d_H(u \leftrightharpoons s) + d_H(s \leftrightharpoons v) \\
\text{(by Line 13)} \quad &\leq d_{G_i}(u \leftrightharpoons s) + d_{G_i}(s \leftrightharpoons v) \\
\text{(by Eq. (5.5))} \quad &\leq 4g \\
\text{(since } k \geq 3) \quad &\leq (2k-1)g
\end{aligned}$$

as desired.

Hence it remains to consider the case when either $s \notin B_i(u)$ or $s \notin B_i(v)$. In the following we only consider $s \notin B_i(v)$, and the other case where $s \notin B_i(u)$ follows from an analogous argument.

By definition of bunches at Line 11, $s \notin B_i(v)$ implies

$$d_{G_i}(v \leftrightharpoons s) \geq d_{G_{r\Delta-1}}(v \leftrightharpoons p_{r\Delta-1}(v)), \tag{5.8}$$

where $i = r\Delta + t$ ($r \in \{0, \ldots, k-2\}, t \in \{0, \ldots, \Delta-1\}$). Now we use an induction similar to [TZ01].

LEMMA 5.6. *Suppose integer $J \geq 0$ satisfies*

- $p_{(r-j)\Delta-1}(v) \notin B_{(r-j)\Delta-1}(u)$ *for all even* $0 \leq j < J$, *and*
- $p_{(r-j)\Delta-1}(u) \notin B_{(r-j)\Delta-1}(v)$ *for all odd* $0 \leq j < J$.

Then,

- *If J is even, then*
$$d_{(r-J)\Delta-1}(v \leftrightharpoons p_{(r-J)\Delta-1}(v)) \leq (J+2)g.$$
- *If J is odd, then*
$$d_{(r-J)\Delta-1}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) \leq (J+2)g.$$

Proof. We prove by induction on J. The base case $J = 0$ follows from

$$\begin{aligned} &\text{(by Eq. (5.8))} && d_{G_{r\Delta-1}}(v \leftrightharpoons p_{r\Delta-1}(v)) \leq d_{G_i}(v \leftrightharpoons s) \\ &\text{(by Eq. (5.5))} && \leq 2g. \end{aligned}$$

To prove the inductive case $J \geq 1$, we first consider the case with odd J. By the assumption for $j = J-1$, we have $p_{(r-J+1)\Delta-1}(v) \notin B_{(r-J+1)\Delta-1}(u)$. By definition of bunches at Line 11 (at iteration $i = (r-J+1)\Delta - 1 = (r-J)\Delta + (\Delta-1)$), this means

$$d_{G_{(r-J+1)\Delta-1}}(u \leftrightharpoons p_{(r-J+1)\Delta-1}(v)) \geq d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)). \tag{5.9}$$

Then,

$$\begin{aligned} &\text{(by Eq. (5.9))} && d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) \leq d_{G_{(r-J+1)\Delta-1}}(u \leftrightharpoons p_{(r-J+1)\Delta-1}(v)) \\ &\text{(triangle inequality)} && \leq d_{G_{(r-J+1)\Delta-1}}(v \leftrightharpoons p_{(r-J+1)\Delta-1}(v)) + d_{G_{(r-J+1)\Delta-1}}(v \leftrightharpoons u) \\ &\text{(by induction hypothesis)} && \leq (J-1+2)g + d_{G_{(r-J+1)\Delta-1}}(v \leftrightharpoons u) \\ &\text{(since } C \subseteq E(G_i) \subseteq E(G_{(r-J+1)\Delta-1})\text{)} && \leq (J-1+2)g + g \\ & && = (J+2)g, \end{aligned}$$

as desired.

The inductive proof for even J is similar, by switching the role of u and v. ∎

LEMMA 5.7. *Let $J \geq 0$ be the maximum integer for which the assumption in Lemma 5.6 holds. Then, $d_H(u \leftrightharpoons v) \leq (2J+5)g$.*

Proof. We prove the case where J is odd. The even case can be proved similarly by switching the role of u and v.

By the maximality of J, we have

$$p_{(r-J)\Delta-1}(u) \in B_{(r-J)\Delta-1}(v). \tag{5.10}$$

By the conclusion of Lemma 5.6, we have

$$d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) \leq (J+2)g. \tag{5.11}$$

Then,

$$\begin{aligned} d_H(u \leftrightharpoons v) &\leq d_H(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) + d_H(p_{(r-J)\Delta-1}(u) \leftrightharpoons v) \\ &\text{(by Line 13)} && \leq d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) + d_H(p_{(r-J)\Delta-1}(u) \leftrightharpoons v) \\ &\text{(by Line 13 and Eq. (5.10))} && \leq d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) + d_{G_{(r-J)\Delta-1}}(p_{(r-J)\Delta-1}(u) \leftrightharpoons v) \\ &\text{(triangle inequality)} && \leq 2d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons p_{(r-J)\Delta-1}(u)) + d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons v) \\ &\text{(by Eq. (5.11))} && \leq 2(J+2)g + d_{G_{(r-J)\Delta-1}}(u \leftrightharpoons v) \\ & && = (2J+5)g. \end{aligned}$$

∎

LEMMA 5.8. $d_H(u \leftrightharpoons v) \leq (2k-1)g$.

Proof. By Line 4 and Line 11, we know $B_{\Delta-1}(u) = B_{\Delta-1}(v) = S_{\Delta-1}$. In particular, this means J cannot satisfy the assumption of Lemma 5.6 if $J \geq r$.

Hence, the maximum J that could possibly satisfy the assumption of Lemma 5.6 is at most $r-1 \leq k-3$. Then, by Lemma 5.7, we have $d_H(u \leftrightharpoons v) \leq (2J+5)g \leq (2(k-3)+5)g = (2k-1)g$. ∎

6 4-Approximation of girth in $\tilde{O}(mn^{1/3})$ time

In this section, we present our algorithm for computing a 4-approximation of the girth in a weighted directed graph (Theorem 1.3).

In general, we follow the approach of Chechik and Lifshitz [CL21] which uses uniformly random vertex samples and certain elimination rules to prune the search space for each vertex $v \in V$. Our running time improvement comes from extending the framework of [CL21] by one more layer, using several novel structural and algorithmic ideas.

Throughout this section, $d(u,v)$ always means $d_G(u,v)$, where $G = (V,E)$ is the input directed graph.

6.1 Main Algorithm

By Lemma 2.1, we assume each vertex in G has degree at most $O(m/n)$.

Before describing our algorithm in detail, we first give a high-level overview of the structure of our algorithm. Our algorithm runs in three phases:

1. **Phase I.** Take a random sample $S_1 \subseteq V$ of $O(n^{1/3})$ vertices.

   For each $s_1 \in S_1$ use Dijkstra to find the shortest cycle going through s_1.

2. **Phase II.** Take a sample $S_2 \subseteq V$ of $O(n^{2/3})$ vertices. Based on the distance information from S_1 obtained in Phase I, for every $s_2 \in S_2$ we use the elimination rule from [CL21] (Lemma 3.1) to compute the pruned sets $B^{(2)}_{\text{out}}(s_2), B^{(2)}_{\text{in}}(s_2) \subseteq V$ of size $\tilde{O}(n^{2/3})$.

   For each $s_2 \in S_2$ use Dijkstra to find the shortest cycle going through s_2 and some $u \in B^{(2)}_{\text{out}}(s_2) \cap B^{(2)}_{\text{in}}(s_2)$.

3. **Phase III.** Based on the distance information obtained from Phase I and II, use our novel elimination rules (Definition 6.16 and Definition 6.19, which are more technical than [CL21]) to compute for every vertex $v \in V$ a pruned set $\tilde{B}'(v) \subseteq V$ of size $\tilde{O}(n^{1/3})$.

   For each $v \in V$ use Dijkstra to find the shortest cycle going through v in the induced subgraph $G[\tilde{B}'(v)]$.

Finally output the length of the shortest cycle encountered in the three phases as the girth estimate.

We present our main algorithm in Algorithm 3 as follows. It follows the three-phase structure described above (indicated by the comments), but involves more definitions and subroutines that will be explained in the following sections. The main statements for the correctness and running time of Algorithm 3 will be given in Theorem 6.23 and Theorem 6.25.

6.2 Phase I and II

In this subsection we describe Phase I and II of our Algorithm 3, which mostly follow the 2-approximation algorithm of [CL21] (with sample size $|S_1|$ changed from $O(\sqrt{n})$ to $O(n^{1/3})$). One piece missing from [CL21] but necessary for us is a certain closedness property of the pruned sets $B^{(2)}_{\text{out}}(v)$, which allows us to find all vertices in $B^{(2)}_{\text{out}}(v)$ by simply running Dijkstra from v (Lemma 6.6).[2]

Phase I (Line 2–Line 5) uniformly samples a set S_1 of $O(n^{1/3})$ vertices, and runs $O(n^{1/3})$ Dijkstra instances on G to find the shortest cycle going through any vertex in S_1.

OBSERVATION 6.1. *Phase I of Algorithm 3 runs in $\tilde{O}(mn^{1/3})$ total time.*

In Phase II we try to find other short cycles in G that are not 2-approximated by the estimate obtained in Phase I. The first step (Line 6) computes eliminators $R_{1,\text{out}}(v), R_{1,\text{in}}(v) \subseteq S_1$ of small size $|R_{1,\text{out}}(v)|, |R_{1,\text{in}}(v)| \leq O(\log n)$ for all $v \in V$. Intuitively these eliminators retain the usefulness of the sample S_1 in effectively pruning

[2] We need to compute these pruned sets $B^{(2)}_{\text{out}}(v)$ in order to prepare for the later Phase III, which was not required in [CL21]'s two-phase algorithm.

Algorithm 3: 4-Approximation-Girth(G)

Input: A strongly connected directed graph $G = (V, E)$ with maximum degree $O(m/n)$
Output: An estimate g' such that $g \le g' \le 4g$, where g is the girth of G

Initialize $g' \leftarrow \infty$
// Phase I
Sample $S_1 \subseteq V$ of size $|S_1| = O(n^{1/3})$
for $s_1 \in S_1$ **do**
    From s_1 run in- and out-Dijkstra on G
    $g' \leftarrow \min_{u \in V \setminus \{s_1\}} d(s_1 \leftrightharpoons u)$
// Phase II
Compute eliminators $R_{1,\text{out}}(v), R_{1,\text{in}}(v) \subseteq S_1$ of size $|R_{1,\text{out}}(v)|, |R_{1,\text{in}}(v)| = O(\log n)$ for all $v \in V$ using Algorithm 4
Sample $S_2 \subseteq V$ of size $|S_2| = O(n^{2/3})$
for $s_2 \in S_2$ **do**
    From s_2 run modified out-Dijkstra on $G[B^{(2)}_{\text{out}}(s_2)]$ and modified in-Dijkstra on $G[B^{(2)}_{\text{in}}(s_2)]$ (Lemma 6.6), where $B^{(2)}_{\text{out}}(\cdot), B^{(2)}_{\text{in}}(\cdot)$ are defined in Definition 6.2 // $B^{(2)}_{\text{out}}(\cdot)$ and $B^{(2)}_{\text{in}}(\cdot)$ depend on $R_{1,\text{out}}$ and $R_{1,\text{in}}$ respectively.
    $g' \leftarrow \min\{g', \min_{u \in B^{(2)}_{\text{out}}(s_2) \cap B^{(2)}_{\text{in}}(s_2) \setminus \{s_2\}} \big(d(s_2, u) + d(u, s_2)\big)\}$
// Phase III
Compute eliminators $R_{2,\text{in}}(v) \subseteq S_2$ of size $|R_{2,\text{in}}(v)| = O(\log n)$ for all $v \in V$ using Algorithm 5.
for $v \in V$ **do**
    From v run modified in-Dijkstra on $G[\tilde{B}'(v)]$, where $\tilde{B}'(v)$ is defined in Definition 6.19 // $\tilde{B}'(\cdot)$ depends on $R_{2,\text{in}}$ (and also $R_{1,\text{out}}$).
    $g' \leftarrow \min\{g', \min_{u \in G[\tilde{B}'(v)] \text{ and } (v,u) \in E} \big(d_{G[\tilde{B}'(v)]}(u, v) + \text{wt}(v, u)\big)\}$
return g'

the search space, while being small enough for the benefit of time efficiency. We defer the algorithm for computing eliminators (Algorithm 4) to the end of this subsection; instead we first present the following important definition that relies on these eliminators $R_{1,\text{out}}(v), R_{1,\text{in}}(v) \subseteq S_1$.

DEFINITION 6.2. ($B^{(2)}_{\text{out}}(v)$ AND $B^{(2)}_{\text{in}}(v)$, [CL21]) *For $v \in V$, given $R_{1,\text{out}}(v), R_{1,\text{in}}(v) \subseteq V$, we define vertex subsets*

$$B^{(2)}_{\text{out}}(v) = \{u \in V : 2d(v, r_1) + d(r_1, u) > 2d(v, u) + d(u, r_1) \text{ for all } r_1 \in R_{1,\text{out}}(v)\},$$

and symmetrically,

$$B^{(2)}_{\text{in}}(v) = \{u \in V : 2d(r_1, v) + d(u, r_1) > 2d(u, v) + d(r_1, u) \text{ for all } r_1 \in R_{1,\text{in}}(v)\}.$$

Definition 6.2 is motivated by the following lemma, which follows from the key observation (Lemma 3.1) of [CL21]. Intuitively it says $B^{(2)}_{\text{out}}(\cdot)$ captures cycles that cannot be 2-approximated by the estimate in phase I. [3]

LEMMA 6.3. (2-APPROXIMATION [CL21]) *If $u \notin B^{(2)}_{\text{out}}(v)$, then there exists $r_1 \in R_{1,\text{out}}(v) \subseteq S_1$ such that $d(r_1 \leftrightharpoons u) \le 2d(u \leftrightharpoons v)$.*

*The same statement holds if we replace "*out*" by "*in*".*

Proof. By Definition 6.2, since $u \notin B^{(2)}_{\text{out}}(v)$, there exists $r_1 \in R_{1,\text{out}}(v)$ such that

$$2d(v, r_1) + d(r_1, u) \le 2d(v, u) + d(u, r_1).$$

Then applying Lemma 3.1 to u, v, r_1, we have $d(r_1 \leftrightharpoons u) \le 2d(u \leftrightharpoons v)$.

The statement with "out" replaced by "in" can be proved symmetrically by reversing the edge directions. ∎

The following corollary of Lemma 6.3 shows that cycles passing through some $s_2 \in S_2$ are 2-approximated by Phase I and II of Algorithm 3. This is essentially how [CL21] obtained their 2-approximation.

COROLLARY 6.4. ([CL21]) *Let $s_2 \in S_2$ and C be the shortest cycle in G going through s_2. Then, the girth estimate g' obtained by the end of Phase II of Algorithm 3 satisfies $g' \le 2g$, where g denotes the length of C.*

Proof. We can assume $S_1 \cap C = \varnothing$, since otherwise the Phase I of Algorithm 3 can find C and hence $g' \le g$.

If $C \not\subseteq B^{(2)}_{\text{out}}(s_2)$, let $u \in C \setminus B^{(2)}_{\text{out}}(s_2)$. Then by Lemma 6.3 there exists $r_1 \in S_1$ such that $d(r_1 \leftrightharpoons u) \le 2d(u \leftrightharpoons s_2) = 2g$, so Phase I of Algorithm 3 will update g' with $d(r_1 \leftrightharpoons u) \le 2g$ (since $r_1 \in S_1$ and $u \ne r_1$).

Similarly, if $C \not\subseteq B^{(2)}_{\text{in}}(s_2)$, we also have $g' \le 2g$.

The remaining case is $C \subseteq B^{(2)}_{\text{out}}(s_2) \cap B^{(2)}_{\text{in}}(s_2)$. Then, Line 10 in Phase II of Algorithm 3 updates g' with g. ∎

The following key lemma (which will be proved later) states that the sets $B^{(2)}_{\text{in}}(v), B^{(2)}_{\text{out}}(v)$ defined using $R_{1,\text{in}}(v)$, $R_{1,\text{out}}(v)$ returned by COMPUTE-ELIMINATORS-1(G, S_1) (Algorithm 4) have small sizes. Intuitively, this is due to the symmetry of the elimination rule in Definition 6.2 and the sample size being $|S_1| = O(n^{1/3})$.

LEMMA 6.5. (SIZES OF $B^{(2)}_{\text{in}}(v), B^{(2)}_{\text{out}}(v)$, [CL21]) *With high probability[4] over the random sample $S_1 \subseteq V$, we have $|B^{(2)}_{\text{in}}(v)|, |B^{(2)}_{\text{out}}(v)| \le \tilde{O}(n^{2/3})$ for all $v \in V$.*

Then, the next step of Phase II is to uniformly sample a set S_2 of $O(n^{2/3})$ vertices (Line 7). We then run out-Dijkstra from every $s_2 \in S_2$ on the induced subgraph $G[B^{(2)}_{\text{out}}(s_2)]$, and update the girth estimate g' with the found cycles going through s_2 (Line 8–Line 10).

In order to implement the out-Dijkstra on $G[B^{(2)}_{\text{out}}(s_2)]$ at Line 9, we need the following Lemma 6.6 which states that the set $B^{(2)}_{\text{out}}(s_2)$ (as well as distances $d(s_2, u)$ for all $u \in B^{(2)}_{\text{out}}(s_2)$) can be efficiently computed given the eliminators $R_{1,\text{out}}(v)$ due to its special structure. Recall that $d(\cdot, \cdot)$ always denotes distances in the input graph G.

[3] We use superscript (2) in the notation of $B^{(2)}_{\text{out}}(v)$ for this reason, to distinguish it from the set $B^{(4)}_{\text{out}}(v)$ that will be introduced later in Section 6.3.

[4] We use "with high probability" to mean probability $1 - 1/n^c$ for arbitrary given constant $c \ge 1$.

LEMMA 6.6. (COMPUTE $B^{(2)}_{\text{out}}(v)$) *For any vertex $v \in V$, given $R_{1,\text{out}}(v)$ of size $O(\log n)$, there exists an algorithm running in $\tilde{O}(\frac{m}{n} \cdot |B^{(2)}_{\text{out}}(v)|)$ time that computes the set $B^{(2)}_{\text{out}}(v)$, and the distances $d(v,u)$ for all $u \in B^{(2)}_{\text{out}}(v)$.*

*The same statement holds if we replace "*out*" by "* in*" and replace $d(v,u)$ by $d(u,v)$.*

Proof. We run a modified out-Dijkstra from v on graph G, and let $D[u]$ denote the length of the shortest path from v to u found by this out-Dijkstra. The modification is that whenever we pop a vertex u from the heap, we relax the out-neighbors of u only if u satisfies

$$2d(v,r_1) + d(r_1,u) > 2D[u] + d(u,r_1), \text{ for all } r_1 \in R_{1,\text{out}}(v). \tag{6.12}$$

Comparing Eq. (6.12) with the definition of $B^{(2)}_{\text{out}}$ (Definition 6.2), the difference is that we use $D[u]$ in place of $d(v,u)$. Note that the other three terms in Eq. (6.12) are already computed in Phase I because $r_1 \in S_1$.

To show the correctness of the modified out-Dijkstra, the key claim is the following closedness property of $B^{(2)}_{\text{out}}(v)$:

CLAIM 6.7. *If $u \in B^{(2)}_{\text{out}}(v)$, then for every vertex x on the shortest path from v to u in G, it holds that $x \in B^{(2)}_{\text{out}}(v)$.*

Proof. For all $r_1 \in R_{1,\text{out}}(v)$, we have

$$\begin{aligned}
&\text{(by triangle inequality)} && 2d(v,r_1) + d(r_1,x) \geq 2d(v,r_1) + d(r_1,u) - d(x,u)\\
&\text{(by } u \in B^{(2)}_{\text{out}}(v)\text{)} && > 2d(v,u) + d(u,r_1) - d(x,u)\\
&\text{(by assumption on } x\text{)} && = 2d(v,x) + 2d(x,u) + d(u,r_1) - d(x,u)\\
&\text{(by triangle inequality)} && \geq 2d(v,x) + d(x,r_1).
\end{aligned}$$

Hence, we have $x \in B^{(2)}_{\text{out}}(v)$ by definition. ∎

By Claim 6.7, it is clear that our modified out-Dijkstra visits exactly all the vertices $u \in B^{(2)}_{\text{out}}(v)$, and correctly computes distances $D[u] = d(v,u)$ for all $u \in B^{(2)}_{\text{out}}(v)$.

Since $|R_{1,\text{out}}(v)| = O(\log n)$, checking the condition Eq. (6.12) for all $r_1 \in R_{1,\text{out}}(v)$ only takes $O(\log n)$ time per vertex $u \in V$. By our assumption that the degree of every vertex is at most $O(\frac{m}{n})$, it follows that the modified Dijkstra runs in time $\tilde{O}(\frac{m}{n} \cdot |B^{(2)}_{\text{out}}(v)|)$. ∎

Hence, we observe the following corollary:

COROLLARY 6.8. *Line 8–Line 10 of Algorithm 3 take total time $\tilde{O}(mn^{1/3})$.*

Proof. By Lemma 6.6, the modified out-Dijkstra from all $s_2 \in S_2$ takes total time

$$\tilde{O}(\frac{m}{n} \sum_{s_2 \in S_2} |B^{(2)}_{\text{out}}(s_2)|) \leq \tilde{O}(\frac{m}{n} \cdot |S_2| \cdot n^{2/3}) \leq \tilde{O}(mn^{1/3}),$$

where we used $|B^{(2)}_{\text{out}}(s_2)| \leq \tilde{O}(n^{2/3})$ from Lemma 6.5. The update step at Line 10 takes $O(m/n) \cdot |B^{(2)}_{\text{out}}(s_2)|$ time for each $s_2 \in S_2$, which also sums up to $\tilde{O}(mn^{1/3})$. ∎

Computing eliminators. Finally, we describe how to compute the eliminators $R_{1,\text{out}}(v), R_{1,\text{in}}(v) \subseteq S_1$ (Line 6 of Algorithm 3). This subroutine is basically the same as in [CL21], but we present it here using our notation for completeness. See the pseudocode of COMPUTE-ELIMINATORS-1(G, S_1) in Algorithm 4, which takes the uniform vertex sample $S_1 \subseteq V$, and returns $R_{1,\text{out}}(v) \subseteq S_1$ for all v. The algorithm for computing $R_{1,\text{in}}(v)$ is analogous: we simply run Algorithm 4 on the graph obtained by reversing the edge orientations of G, and we omit the detailed descriptions here.

Algorithm 4: COMPUTE-ELIMINATORS-1(G, S_1)

Input: The input graph $G = (V, E)$, and $S_1 = \{s_1, s_2, \ldots, s_{|S_1|}\} \subseteq V$ of size $|S_1| = O(n^{1/3})$ sampled uniformly and independently (with replacement)

Output: Sets $R_{1,\text{out}}(v) \subseteq S_1$ of size $O(\log n)$ for every vertex $v \in V$

1 $T^{(0)}(v), R^{(0)}(v) \leftarrow \varnothing$ for every $v \in V$
2 **for** $i \in \{1, \ldots, k\}$ *where* $k = 10 \log n$ **do**
3 $\quad S^{(i)} \leftarrow$ the next $10n^{1/3}/\log n$ samples from S_1
4 $\quad$ Run in- and out-Dijkstra from every $s \in S^{(i)}$ on G
5 $\quad$ **for** $v \in V$ **do**
6 $\quad\quad T^{(i)}(v) \leftarrow \{s \in S^{(i)} \mid \forall t \in R^{(i-1)}(v), 2d(v,s) + d(s,t) < 2d(v,t) + d(t,s)\}$
7 $\quad\quad$ **if** $T^{(i)}(v) \neq \varnothing$ **then**
8 $\quad\quad\quad t \leftarrow$ a random vertex $t \in T^{(i)}(v)$
9 $\quad\quad\quad R^{(i)}(v) \leftarrow R^{(i-1)}(v) \cup \{t\}$
10 $\quad\quad$ **else**
11 $\quad\quad\quad R^{(i)}(v) \leftarrow R^{(i-1)}(v)$
12 **return** $R_{1,\text{out}}(v) \leftarrow R^{(k)}(v)$ for each $v \in V$

Algorithm 4 runs in $k = 10 \log n$ iterations. In each iteration, it takes $10n^{1/3}/\log n$ fresh vertex samples (from S_1), and runs Dijkstra from them on G. Then, based on the obtained distance information, it possibly adds one sampled vertex t to each $R_{1,\text{out}}(v)$. By inspecting Algorithm 4, one immediately observes the following properties.

OBSERVATION 6.9. *Algorithm 4 runs in time $\tilde{O}(mn^{1/3})$, and outputs sets $R_{1,\text{out}}(v) \subseteq S_1$ for all $v \in V$ of size $|R_{1,\text{out}}(v)| = O(\log n)$.*

Proof. First note that the total number of vertex samples required at Line 3 is $|S^{(1)} \uplus \cdots \uplus S^{(k)}| = k \cdot 10n^{1/3}/\log n = 100n^{1/3} \leq |S_1|$. In each iteration $1 \leq i \leq k$, the algorithm only adds at most one sampled vertex $t \in S_1$ to the set $R^{(i)}(v)$ for each $v \in V$ (Line 7–Line 11), so each output set $R_{1,\text{out}}(v) = R^{(k)}(v) \subseteq S_1$ and has size $|R_{1,\text{out}}(v)| \leq k \leq O(\log n)$.

In each iteration, the Dijkstra instances at Line 4 take time $|S^{(i)}| \cdot O(m + n \log n) \leq \tilde{O}(mn^{1/3})$. Then, to compute $T^{(i)}(v) \subseteq S^{(i)}$ at Line 6 for each $v \in V$, we check for every $s \in S^{(i)}$ whether $s \in T^{(i)}(v)$, by simply going over all $t \in R^{(i-1)}(v)$ and checking the condition $2d(v,s) + d(s,t) < 2d(v,t) + d(t,s)$. Note that all four terms in this inequality have already been computed by the in- and out-Dijkstras since $s \in S^{(i)}$ and $t \in S^{(1)} \cup \cdots \cup S^{(i-1)}$. So $T^{(i)}(v)$ can be computed in time $O(|S^{(i)}| \cdot |R^{(i-1)}(v)|) = O((n^{1/3}/\log n) \cdot \log n) = O(n^{1/3})$ for each $v \in V$. Thus each iteration runs in time $O(mn^{1/3})$ time and over all $k = O(\log n)$ iterations, Algorithm 4 runs in total time $\tilde{O}(mn^{1/3})$. ∎

Now we prove the key Lemma 6.5, which states that Algorithm 4 guarantees $B^{(2)}_{\text{out}}(v)$ and $B^{(2)}_{\text{in}}(v)$ to have small size with high probability.

Proof. [Proof of Lemma 6.5] The proof more or less follows from Section 6 in [CL21]. For purpose of the proof, we define the sets

$$B_i(v) = \{u \in V \mid 2d(v,u) + d(u,r) < 2d(v,r) + d(r,u)\, \forall r \in R^{(i)}(v)\}.$$

Then note that by definition $B^{(2)}_{\text{out}}(v) = \{u \in V \mid 2d(v,u) + d(u,r) < 2d(v,r) + d(r,u)\, \forall r \in R_{1,\text{out}}(v)\} = B_k(v)$. We want to show that

$$\Pr\left[|B_k(v)| > n^{2/3} \log n\right] \leq \frac{1}{n^2}.$$

We first show that if $|B_i(v)| > n^{2/3} \log n$, then

$$\mathbb{E}\left[|B_i(v)| \,\middle|\, |B_{i-1}(v)|\right] \leq \frac{3}{4}|B_{i-1}(v)|.$$

By symmetry [5] of the condition $2d(v,u) + d(u,s) < 2d(v,s) + d(s,u)$ with respect to $u \in B_{i-1}(v)$ and $s \in S^{(i)} \cap B_i(v)$, for any pair of vertices $u, u' \in B_{i-1}(v)$, either u can eliminate u' or u' can eliminate u. Thus given a random $s \in S^{(i)} \cap B_i(v)$, on expectation s can eliminate half of the vertices in $B_{i-1}(v)$. So conditioned on the event that $S^{(i)} \cap B_i(v) \neq \varnothing$, we have the expected size of $B_i(v)$ is at most half the size of $B_{i-1}(v)$. Specifically we have

$$\mathbb{E}\left[|B_i(v)| \,\middle|\, S^{(i)} \cap B_i(v) \neq \varnothing\right] \leq \frac{1}{2}|B_{i-1}(v)|.$$

Now since $|S^{(i)}| = 10n^{1/3}/\log n$ is a uniform random sample, we can compute $\Pr[S^{(i)} \cap B_i(v) = \varnothing]$ as

$$\begin{aligned}\Pr\left[S^{(i)} \cap B_i(v) = \varnothing\right] &= \left(1 - \frac{|B_i(v)|}{n}\right)^{10n^{1/3}/\log n} \approx \exp\left(-\frac{|B_i(v)| \cdot 10n^{1/3}}{n \log n}\right) \\ &\leq \left(\frac{1}{4}\right)^{\frac{|B_i(v)|}{n^{2/3}\log n}} \leq \frac{1}{4}.\end{aligned}$$

Thus we have

$$\begin{aligned}\mathbb{E}\left[|B_i(v)| \,\middle|\, |B_{i-1}(v)|\right] &= \mathbb{E}\left[|B_i(v)| \,\middle|\, |B_{i-1}(v)|, S^{(i)} \cap B_i(v) \neq \varnothing\right] \cdot \Pr\left[S^{(i)} \cap B_i(v) \neq \varnothing\right] \\ &\quad + \mathbb{E}\left[|B_i(v)| \,\middle|\, |B_{i-1}(v)|, S^{(i)} \cap B_i(v) = \varnothing\right] \cdot \Pr\left[S^{(i)} \cap B_i(v) = \varnothing\right] \\ &\leq \frac{1}{2}|B_{i-1}(v)| + \frac{1}{4}|B_{i-1}(v)| = \frac{3}{4}|B_{i-1}(v)|\end{aligned}$$

as desired.

Now we can easily finish the proof by applying Markov's inequality.

$$\Pr\left[|B_k(v)| > n^{2/3}\log n\right] \leq \frac{\mathbb{E}[|B_k(v)|]}{n^{2/3}\log n} \leq \frac{\left(\frac{3}{4}\right)^k n}{(n^{2/3}\log n)} \leq \left(\frac{3}{4}\right)^k n^{1/3} \leq \frac{1}{n^2}.$$

□

PROPOSITION 6.1. *Phase II of Algorithm 3 runs in $\tilde{O}(mn^{1/3})$ total time.*

Proof. Follows from Observation 6.9 and Corollary 6.8. □

6.3 New lemmas for 4-approximation In this section we describe our new structural lemmas that are useful for 4-approximation.

We start with the following Lemma 6.10, which naturally extends the 2-approximation lemma (Lemma 6.3) for $B^{(2)}_{\text{in}}(v)$ from one layer to two layers by exploiting the second sample set S_2.

LEMMA 6.10. *Let $u, v \in V$ $(u \neq v)$ and $r_2 \in S_2$. Suppose*

$$2d(r_2, v) + d(u, r_2) \leq 2d(u, v) + d(r_2, u).$$

Then, the girth estimate g' obtained by the end of Phase II of Algorithm 3 satisfies $g' \leq 4d(u \leftrightharpoons v)$.

Proof. Apply Lemma 3.1 (*with edge direction reversed*) to u, v, r_2, and obtain

$$d(r_2 \leftrightharpoons u) \leq 2d(u \leftrightharpoons v).$$

If $u \notin B^{(2)}_{\text{out}}(r_2)$, then by Lemma 6.3 there exists $r_1 \in R_{1,\text{out}}(r_2) \subseteq S_1$ such that

$$d(r_1 \leftrightharpoons u) \leq 2d(u \leftrightharpoons r_2) \leq 4d(u \leftrightharpoons v).$$

[5] For more details, refer to the proof of Lemma 3.3 in [CL21]

This implies $g' \le 4d(u \leftrightharpoons v)$ due to the update at Line 5 in Phase I of Algorithm 3 for $r_1 \in S_1$.[6]

Similarly, if $u \notin B^{(2)}_{\text{in}}(r_2)$, then we also have $g' \le 4d(u \leftrightharpoons v)$.

It remains to consider the case where $u \in B^{(2)}_{\text{in}}(r_2) \cap B^{(2)}_{\text{out}}(r_2)$. In this case, Line 10 of Algorithm 3 updates g' with $d(r_2 \leftrightharpoons u) \le 2d(u \leftrightharpoons v)$ (here we need to assume $u \neq r_2$; the $u = r_2$ case is already covered by Corollary 6.4).

Hence, we always have $g' \le 4d(u \leftrightharpoons v)$. ∎

In light of Lemma 6.10, a natural attempt for a 4-approximation algorithm is to imimate Phase II and focus on for each $v \in V$ the pruned vertex set $\{u \in V : 2d(r_2, v) + d(u, r_2) > 2d(u, v) + d(r_2, u)$ for all $r_2 \in R(v)\}$ for some suitably defined $R(v) \subseteq S_2$. As mentioned in the technical overview, this attempt would require distance information for all $r_2 \in S_2$, which is infeasible to compute efficiently enough due to the large size $|S_2| = O(n^{2/3})$. Thus, we need to use more structural lemmas for our algorithm, described as follows.

First, we generalize the key observation (Lemma 3.1) of [CL21] to the following Lemma 6.11. Note that Lemma 3.1 corresponds to the $k = 2$ case of Lemma 6.11. See Fig. 1 (the same figure as Lemma 3.1) for an illustration.

LEMMA 6.11. (GENERALIZED KEY OBSERVATION) *For any $k \ge 1$ and vertices u, v, r, if*

$$k \cdot d(v, r) + d(r, u) \le k \cdot d(v, u) + (k - 1) \cdot d(u, r),$$

then

$$d(r \leftrightharpoons u) \le k \cdot d(u \leftrightharpoons v).$$

Proof. Note that by triangle inequality, we have $d(u, v) \ge d(u, r) - d(v, r)$, so

$$\begin{aligned} k \cdot d(v, u) + k \cdot d(u, v) &\ge k \cdot d(v, u) + k \cdot d(u, r) - k \cdot d(v, r) \\ &\ge \big(k \cdot d(v, r) + d(r, u) - (k - 1) \cdot d(u, r)\big) + k \cdot d(u, r) - k \cdot d(v, r) \\ &= d(r, u) + d(u, r). \end{aligned}$$

∎

Lemma 6.11 inspires the following definition of $B^{(4)}_{\text{out}}(v)$ and a 4-approximation lemma (Corollary 6.13), which are analogous to $B^{(2)}_{\text{out}}(v)$ (Definition 6.2) and the 2-approximation lemma (Lemma 6.3).

DEFINITION 6.12. ($B^{(4)}_{\text{out}}(v)$) *For $v \in V$, given $R_{1,\text{out}}(v) \subseteq V$, we define vertex subsets*

$$B^{(4)}_{\text{out}}(v) = \{u \in V : 4d(v, r_1) + d(r_1, u) > 4d(v, u) + 3d(u, r_1) \text{ for all } r_1 \in R_{1,\text{out}}(v)\}.$$

COROLLARY 6.13. (4-APPROXIMATION) *If $u \notin B^{(4)}_{\text{out}}(v)$, then there exists $r_1 \in R_{1,\text{out}}(v)$ such that $d(r_1 \leftrightharpoons u) \le 4d(u \leftrightharpoons v)$.*

Proof. By Definition 6.12, since $u \notin B^{(4)}_{\text{out}}(v)$, there exists $r_1 \in R_{1,\text{out}}(v)$ such that

$$4d(v, r_1) + d(r_1, u) \le 4d(v, u) + 3d(u, r_1).$$

Then applying Lemma 6.11 with $k = 4$ to u, v, r_1, we have $d(r_1 \leftrightharpoons u) \le 4d(u \leftrightharpoons v)$. ∎

We also have the following relationship between $B^{(4)}_{\text{out}}(v)$ and $B^{(2)}_{\text{out}}(v)$.

LEMMA 6.14. *For all $v \in V$, $B^{(4)}_{\text{out}}(v) \subseteq B^{(2)}_{\text{out}}(v)$.*

As a consequence, the algorithm of Lemma 6.6 for computing $B^{(2)}_{\text{out}}(v)$ can also compute $B^{(4)}_{\text{out}}(v)$ in the same running time.

[6] This argument requires $r_1 \neq u$. This can be ensured by assuming $u \notin S_1$ without loss of generality: if $u \in S_1$, then Phase I of Algorithm 3 will update g' using $d(u \leftrightharpoons v)$.

Proof. If $u \in B^{(4)}_{\text{out}}(v)$, then by Definition 6.12 for all $r_1 \in R_{1,\text{out}}(v)$,

$$\begin{aligned} 2d(v,r_1) + d(r_1,u) &> 4d(v,u) + 3d(u,r_1) - 2d(v,r_1) \\ &= 2d(v,u) + d(u,r_1) + 2\big(d(v,u) + d(u,r_1) - d(v,r_1)\big) \\ &\geq 2d(v,u) + d(u,r_1). \end{aligned}$$

So $u \in B^{(2)}_{\text{out}}(v)$ by Definition 6.2. ∎

Now we state and prove our main novel technical lemma, which is a key ingredient of our 4-approximation algorithm.

LEMMA 6.15. (4-APPROXIMATION FILTERING LEMMA) *Consider vertices $r_2, v, u \in V$ such that $v \in B^{(4)}_{\text{out}}(r_2)$ and $u \notin B^{(2)}_{\text{out}}(r_2)$. Then there exists $r_1 \in R_{1,\text{out}}(r_2)$ such that $d(v \leftrightharpoons r_1) \leq 4d(v \leftrightharpoons u)$.*

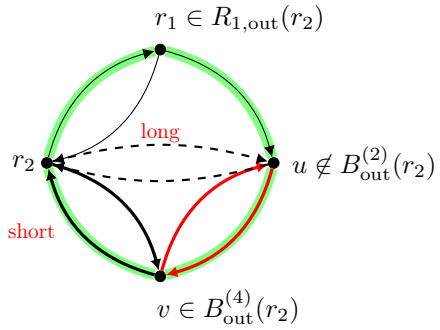

Figure 3: Illustration of the relationship between the vertices involved in Lemma 6.15. The two bold black cycle is relatively short and the dashed cycle is relatively long, the goal is to approximate the red cycle using the cycle highlighted green. As labeled, $v \in B^{(4)}_{\text{out}}(r_2)$ meaning that v and r_2 are in a short cycle, $u \notin B^{(2)}_{\text{out}}(r_2)$ meaning that u and r_2 are in a relatively long cycle. Then we can find some r_1 in the set of eliminators for r_2 such that the cycle passing through v and r_1 (highlighted green) approximate the red cycle passing through v and u.

Proof. Since $u \notin B^{(2)}_{\text{out}}(r_2)$, by Definition 6.2 there exists $r_1 \in R_{1,\text{out}}(r_2)$ such that

$$2d(r_2,u) + d(u,r_1) \geq 2d(r_2,r_1) + d(r_1,u). \tag{6.13}$$

Since $v \in B^{(4)}_{\text{out}}(r_2)$ and $r_1 \in R_{1,\text{out}}(r_2)$, by Definition 6.12 we have

$$4d(r_2,r_1) + d(r_1,v) > 4d(r_2,v) + 3d(v,r_1). \tag{6.14}$$

Adding Eq. (6.13) multiplied by 2 with Eq. (6.14), and cancelling $4d(r_2,r_1)$ on both sides, we get

$$4d(r_2,u) + 2d(u,r_1) + d(r_1,v) > 2d(r_1,u) + 4d(r_2,v) + 3d(v,r_1).$$

Combining with $4d(r_2,v) + 4d(v,u) \geq 4d(r_2,u)$ (triangle inequality), this implies

$$4d(v,u) + 2d(u,r_1) + d(r_1,v) > 2d(r_1,u) + 3d(v,r_1).$$

Adding $4d(u,v)$ to both sides gives

$$\begin{aligned} 4d(u \leftrightharpoons v) + 2d(u,r_1) + d(r_1,v) &> \big(2d(r_1,u) + 2d(u,v)\big) + \big(2d(u,v) + 2d(v,r_1)\big) + d(v,r_1) \\ &\geq 2d(r_1,v) + 2d(u,r_1) + d(v,r_1), \end{aligned}$$

which immediately simplifies to

$$4d(u \leftrightharpoons v) > d(r_1,v) + d(v,r_1) = d(v \leftrightharpoons r_1).$$

∎

6.4 Phase III Now we are ready to describe Phase III, the most technical part of our Algorithm 3. It has a similar structure as Phase II: we first compute eliminators $R_{2,\text{in}}(v) \subseteq S_2$ of size $|R_{2,\text{in}}(v)| = O(\log n)$ for all $v \in V$, and then use these eliminators to define pruned vertex sets $\tilde{B}'(v)$ (which is a subset of $B'(v) \cup \{v\}$ which we will define shortly) to search for short cycles. In light of the 4-approximation filtering lemma (Lemma 6.15), we will ensure the eliminators satisfy the following property (it will be later shown in Observation 6.24):

$$\text{For every } v \in V \text{ and } r_2 \in R_{2,\text{in}}(v), \text{ we have } v \in B^{(4)}_{\text{out}}(r_2). \tag{6.15}$$

Again, we defer the algorithm for computing the eliminators $R_{2,\text{in}}(v)$ to the end of this subsection.

We first make the following technical definition of pruned vertex sets $B'(v)$, which is directly motivated by the structural lemmas from Section 6.3.

DEFINITION 6.16. ($B'(v)$) *For $v \in V$, let $B'(v)$ denote the set of vertices $s \in V$ that satisfy all the following conditions:*

1. $v \in B^{(4)}_{\text{out}}(s)$, *and*
2. $s \in B^{(2)}_{\text{out}}(r_2)$ *for all* $r_2 \in R_{2,\text{in}}(v)$, *and*
3. $2d(s,v) + d(r_2,s) < 2d(r_2,v) + \underline{d}(s,r_2)$ *for all* $r_2 \in R_{2,\text{in}}(v)$. *(where $\underline{d}$ is defined in Lemma 6.17)*

In this definition, condition 1 is motivated by the 4-approximation lemma (Corollary 6.13), condition 2 is motivated by our 4-approximation filtering lemma (Lemma 6.15) and Eq. (6.15), and condition 3 is motivated by Lemma 6.10. For technical reason, condition 3 involves a certain distance underestimate that is easier to compute, defined as follows (readers are encouraged to think of the underestimate as the original distance, and skip this definition at first read):

LEMMA 6.17. (UNDER-ESTIMATE OF $d(u,r_2)$) *For all $u \in V$ and $r_2 \in V$, define $\underline{d}(u,r_2)$ as follows:*

- ***Case*** $u \in B^{(2)}_{\text{in}}(r_2)$***:***
  Let $\underline{d}(u,r_2) := d(u,r_2)$.
- ***Case*** $u \notin B^{(2)}_{\text{in}}(r_2)$***:***
  Let
  $$\underline{d}(u,r_2) := \frac{1}{2} \min_{r_1 \in R_{1,\text{in}}(r_2)} (2d(r_1,r_2) + d(u,r_1) - d(r_1,u)). \tag{6.16}$$

Then, $\underline{d}(u,r_2) \le d(u,r_2)$ holds.

Proof. In order to prove $\underline{d}(u,r_2) \le d(u,r_2)$, it suffices to focus on the second case, $u \notin B^{(2)}_{\text{in}}(r_2)$. By definition of $B^{(2)}_{\text{in}}(r_2)$ (Definition 6.2), there exists $r_1 \in R_{1,\text{in}}(r_2)$ such that

$$2d(r_1,r_2) + d(u,r_1) \le 2d(u,r_2) + d(r_1,u).$$

This immediately implies $\underline{d}(u,r_2)$ as defined in Eq. (6.16) satisfies $2\underline{d}(u,r_2) \le 2d(u,r_2)$. ∎

The following key lemma (analogous to Lemma 6.5 from Phase II) bounds the size of $B'(v)$.

LEMMA 6.18. (SIZE OF $B'(v)$) *With high probability over the random samples $S_1, S_2 \subseteq V$, we have $|B'(v)| \le \tilde{O}(n^{1/3})$ for all $v \in V$.*

Intuitively, this is due to the symmetry of the elimination rule (Condition 3 in Definition 6.16 of $B'(v)$), and because the sample size is $|S_2| = O(n^{2/3})$. We will prove Lemma 6.18 later after describing the algorithm computing eliminators $R_{2,\text{in}}(v)$.

Our actual algorithm performs a modified in-Dijkstra from every $v \in V$ on the induced subgraph $G[\tilde{B}'(v)]$ (Line 13), where $\tilde{B}'(v)$ is a slight variant of $B'(v)$, which we shall define shortly. The reason for not using $B'(v)$

is because our modified in-Dijkstra algorithm does not know the true distance $d(s,v)$ needed for checking the condition 1 and 3 in the definition of $B'(v)$.[7] Instead, we use the current distance found by the in-Dijkstra to replace $d(s,v)$. The formal definition is as follows (again, readers are encouraged to skip this definition at first read, and think of $\tilde{B}'(v)$ as the same as $B'(v)$ for intuition):

DEFINITION 6.19. (MODIFIED IN-DIJKSTRA AND $\tilde{B}'(v)$) *For $v \in V$, consider the following modified in-Dijkstra algorithm starting from v on graph G, where we let $D[u]$ denote the length of the shortest path from u to v found by this in-Dijkstra.*

The modification is that whenever we pop a vertex $s \neq v$ from the heap, we relax the in-neighbors of s only if s satisfies all the following three conditions:

1. $4d(s,r_1) + d(r_1,v) > 4D[s] + 3d(v,r_1)$ *for all* $r_1 \in R_{1,\text{out}}(s)$, *and*
2. $s \in B^{(2)}_{\text{out}}(r_2)$ *for all* $r_2 \in R_{2,\text{in}}(v)$, *and*
3. $2D[s] + d(r_2,s) < 2d(r_2,v) + \underline{d}(s,r_2)$ *for all* $r_2 \in R_{2,\text{in}}(v)$.

Let $\tilde{B}'(v)$ denote the set of vertices s that are popped out from the heap and satisfy all the three conditions above, and additionally we also let $v \in \tilde{B}'(v)$. (Note that the source vertex v always relaxes all its in-neighbors in the beginning of in-Dijkstra)

OBSERVATION 6.20. $\tilde{B}'(v) \subseteq B'(v) \cup \{v\}$ *for all* $v \in V$.

Proof. Note that the three conditions in Definition 6.19 are the same as the three conditions in Definition 6.16 except that the terms $d(s,v)$ in condition 1 and 3 are replaced by $D[s]$. Since the distance $D[s]$ found by the in-Dijkstra from v must be greater than or equal to the true distance $d(s,v)$, we see that both condition 1 and 3 are strengthened. Hence, $\tilde{B}'(v) \subseteq B'(v)$. ∎

Phase III of our algorithm (Line 13) is implemented by the modified in-Dijkstra described in Definition 6.19. It remains to show that we can implement it efficiently. In particular, we need to show that checking the three conditions in Definition 6.19 is efficient. We first show that the underestimate $\underline{d}(u,r_2)$ from Lemma 6.17 can be computed efficiently.

LEMMA 6.21. (COMPUTE $\underline{d}(u,r_2)$) *For $r_2 \in V$, assume we know $B^{(2)}_{\text{in}}(r_2)$ and $d(x,r_2)$ for all $x \in B^{(2)}_{\text{in}}(r_2)$. Then $\underline{d}(u,r_2)$ can then be computed for any $u \in V$ in $O(\log n)$ time.*

Proof. According to the definition in Lemma 6.17, we first check whether $u \in B^{(2)}_{\text{in}}(r_2)$. In the first case where $u \in B^{(2)}_{\text{in}}(r_2)$, the answer is $d(u,r_2)$, which we know by assumption. In the second case where $u \notin B^{(2)}_{\text{in}}(r_2)$, we need to compute Eq. (6.16) by going over all $O(\log n)$ many $r_1 \in R_{2,\text{in}}(r_2)$. The expression of Eq. (6.16) only involves distances $d(r_1,\cdot)$ and $d(\cdot,r_1)$ for $r_1 \in R_{1,\text{in}}(r_2) \subseteq S_1$, which are already computed in Phase I of Algorithm 3. So we can compute the answer in $O(\log n)$ time. ∎

Now we show $\tilde{B}'(v)$ can be computed efficiently.

LEMMA 6.22. *The modified in-Dijkstra of Definition 6.19 computes $\tilde{B}'(v)$ in $\tilde{O}(\frac{m}{n} \cdot |\tilde{B}'(v)|)$ time.*

Proof. Suppose the modified in-Dijkstra pops vertex s from the heap.

- The condition 1 of Definition 6.19 can be checked in $O(1)$ time because we already know $d(r_1,\cdot), d(\cdot,r_1)$ for all $r_1 \in S_1$ from Phase I of Algorithm 3.
- The condition 2 can be checked in $O(|R_{2,\text{in}}(v)|) \leq O(\log n)$ time since we already computed $B^{(2)}_{\text{out}}(r_2)$ for all $r_2 \in S_2$ in Phase II of Algorithm 3.
- For condition 3, we need to check $2D[s] + d(r_2,s) < 2d(r_2,v) + \underline{d}(s,r_2)$ for all $r_2 \in R_{2,\text{in}}(v)$.

[7] Note that we introduced the under-estimate $\underline{d}(s,r_2)$ in condition 3 of Definition 6.16 for the same reason.

– Since the check for condition 2 has passed, we have $s \in B^{(2)}_{\text{out}}(r_2)$. So we know the value of $d(r_2, s)$ from Phase II of Algorithm 3 (note that $r_2 \in S_2$).

– Since $r_2 \in R_{2,\text{in}}(v)$, we have $v \in B^{(4)}_{\text{out}}(r_2) \subseteq B^{(2)}_{\text{out}}(r_2)$ by Eq. (6.15). So we know the value of $d(r_2, v)$ from Phase II of Algorithm 3.

– We can compute $\underline{d}(s, r_2)$ in $O(\log n)$ time due to Lemma 6.21 and Phase II of Algorithm 3.

Hence, we can check whether $s \in \tilde{B}'(v)$ in $O(\log^2 n)$ time. ∎

Now we are ready to prove that our Algorithm 3 achieves 4-approximation.

THEOREM 6.23. (CORRECTNESS OF ALGORITHM 3) *Algorithm 3 returns g' satisfying $g \le g' \le 4g$, where g is the girth of the input directed graph G.*

Proof. Let C be the shortest cycle of G with length g. Consider an arbitrary vertex v on C. If $v \in S_1$, then C is found in Phase I of Algorithm 3 and hence $g' = g$. If all vertices on C are contained in $\tilde{B}'(v)$, then it is eventually found at Line 14 in Algorithm 3, and $g' = g$. Hence, in the following we assume $v \notin S_1$, and there is some vertex $u \in C$ that is not included in $\tilde{B}'(v)$. We choose u to be the first ancestor of v on the cycle that is not in $\tilde{B}'(v)$ (in particular, u is a minimizer of $d(u, v)$ among $u \in C \setminus \tilde{B}'(v)$). Note that $u \neq v$ because $v \in \tilde{B}'(v)$ by definition.

Let $x \in C$ denote the out-neighbor of u on the cycle C. By our definition of u, we know the entire shortest path from x to v on C are contained in $\tilde{B}'(v)$. Then, x must have relaxed its in-neighbor u during the modified in-Dijkstra, which makes $D[u]$ equal to the true distance $d(u, v)$. The fact that $u \notin \tilde{B}'(v)$ then means some of the three conditions in Definition 6.19 is violated for u, which then implies $u \notin B'(v)$, as these conditions are equivalent to the three conditions in the definition of $B'(v)$ (Definition 6.16) due to $D[u] = d(u, v)$.

As $u \notin B'(v)$, we now divide into three cases depending on which condition in Definition 6.16 fails for u.

- Condition 1 fails, i.e., $v \notin B^{(4)}_{\text{out}}(u)$.

  Then by Corollary 6.13, there exists $r_1 \in R_{1,\text{out}}(u)$ such that $4d(u \leftrightharpoons v) \ge d(r_1 \leftrightharpoons v) \ge g'$ (due to the update at Line 5 for $r_1 \in R_{1,\text{out}}(u) \subseteq S_1$ during Phase I of Algorithm 3; note that $v \neq r_1$ since $v \notin S_1$).

- Condition 2 fails, i.e., $u \notin B^{(2)}_{\text{out}}(r_2)$ for some $r_2 \in R_{2,\text{in}}(v)$.

  Since $r_2 \in R_{2,\text{in}}(v)$, by Eq. (6.15) we have $v \in B^{(4)}_{\text{out}}(r_2)$. Then, by the 4-approximation filtering lemma (Lemma 6.15), there exists $r_1 \in R_{1,\text{out}}(r_2)$ such that $4d(v \leftrightharpoons u) \ge d(v \leftrightharpoons r_1) \ge g'$ (due to the update at Line 5 in Phase I of Algorithm 3).

- Condition 3 fails, and Conditions 1,2 hold. This is saying that there exists $r_2 \in R_{2,\text{in}}(v)$ such that

  $$2d(u, v) + d(r_2, u) \ge 2d(r_2, v) + \underline{d}(u, r_2). \tag{6.17}$$

  And, we have $v \in B^{(4)}_{\text{out}}(u)$ (by Condition 1) and $u \in B^{(2)}_{\text{out}}(r_2)$ (by Condition 2).

  We further divide into two cases:

  – Case $u \in B^{(2)}_{\text{in}}(r_2)$:
  In this case we have $\underline{d}(u, r_2) = d(u, r_2)$ by Lemma 6.17. So we apply Lemma 6.10 to Eq. (6.17) and obtain $g' \le 4d(u \leftrightharpoons v)$.

  – Case $u \notin B^{(2)}_{\text{in}}(r_2)$:
  Plugging the definition $\underline{d}(u, r_2) = \min_{r_1 \in R_{1,\text{in}}(r_2)} \frac{1}{2}(2d(r_1, r_2) + d(u, r_1) - d(r_1, u))$ (from Lemma 6.17) into Eq. (6.17), we obtain that there exists $r_1 \in R_{1,\text{in}}(r_2)$ such that

  $$2d(u, v) + d(r_2, u) \ge 2d(r_2, v) + \frac{1}{2}(2d(r_1, r_2) + d(u, r_1) - d(r_1, u)).$$

Multiplying both sides by 2, and then adding $4d(v,u) - 2d(r_2,u)$ to both sides, we get

$$\begin{aligned}
4d(u,v) + 4d(v,u) &\geq 4d(r_2,v) + 2d(r_1,r_2) + d(u,r_1) - d(r_1,u) + 4d(v,u) - 2d(r_2,u) \\
&\text{(by triangle inequality } d(r_2,u) \leq d(r_2,v) + d(v,u)) \\
&\geq 2d(r_2,v) + 2d(r_1,r_2) + d(u,r_1) - d(r_1,u) + 2d(v,u) \\
&\text{(by triangle inequality } d(r_1,u) \leq d(r_1,r_2) + d(r_2,v) + d(v,u)) \\
&\geq d(r_2,v) + d(r_1,r_2) + d(u,r_1) + d(v,u) \\
&\geq d(v \leftrightharpoons r_1).
\end{aligned}$$

Hence, $4d(v \leftrightharpoons u) \geq d(v \leftrightharpoons r_1) \geq g'$ (due to the update at Line 5 in Phase I of Algorithm 3).

Hence we have established $g' \leq 4d(v \leftrightharpoons u)$ in all three cases. ∎

Computing eliminators. Finally, we describe how to compute the eliminators $R_{2,\text{in}}(v) \subseteq S_2$ (Line 11 of Algorithm 3). The algorithm has a similar overall structure as the eliminator computation in Phase II (and [CL21]) described earlier (Algorithm 4). The main idea is to exploit the symmetry in the definition of $B'(v)$ (Definition 6.16), but here it involves more conditions and we need to be slightly more careful to make sure the running time is $\tilde{O}(mn^{1/3})$. See the pseudocode of COMPUTE-ELIMINATORS-2(G, S_2) in Algorithm 5, which takes the uniform vertex sample $S_2 \subseteq V$, and returns $R_{2,\text{in}}(v) \subseteq S_2$ for all v.

Algorithm 5: COMPUTE-ELIMINATORS-2(G, S_2)

Input: The input graph $G = (V, E)$, and $S_2 = \{s_1, s_2, \ldots, s_{|S_2|}\} \subseteq V$ of size $|S_2| = O(n^{2/3})$ sampled uniformly and independently (with replacement)

Output: The sets $R_{2,\text{in}}(v) \subseteq S_2$ of size $O(\log n)$ for every vertex $v \in V$

1 $T^{(0)}(v), R^{(0)}(v) \leftarrow \varnothing$ for every $v \in V$
2 **for** $i \in \{1, \ldots, k\}$ *where* $k = 10 \log n$ **do**
3 $S^{(i)} \leftarrow$ the next $10n^{2/3}/\log n$ samples from S_2.
4 **for** $s \in S^{(i)}$ **do**
5 Compute $B^{(2)}_{\text{out}}(s)$, $B^{(2)}_{\text{in}}(s)$ and $B^{(4)}_{\text{out}}(s)$, and distances $d(s,v)$ for all $v \in B^{(2)}_{\text{out}}(s)$, $d(v,s)$ for all $v \in B^{(2)}_{\text{in}}(s)$, using Lemma 6.6
6 **for** $v \in V$ **do**
7 $T^{(i)}(v) \leftarrow \{s \in S^{(i)} \mid v \in B^{(4)}_{\text{out}}(s)$ and $\forall t \in R^{(i-1)}(v), s \in B^{(2)}_{\text{out}}(t)$ and $2d(s,v) + d(t,s) < 2d(t,v) + \underline{d}(s,t)\}$, where $\underline{d}(\cdot,\cdot)$ is defined in Lemma 6.17.
8 **if** $T^{(i)}(v) \neq \varnothing$ **then**
9 $t \leftarrow$ a random vertex $t \in T^{(i)}(v)$
10 $R^{(i)}(v) \leftarrow R^{(i-1)}(v) \cup \{t\}$
11 **else**
12 $R^{(i)}(v) \leftarrow R^{(i-1)}(v)$
13 **return** $R_{2,\text{in}}(v) \leftarrow R^{(k)}(v)$ for each $v \in V$

By inspecting Algorithm 5, we observe the following properties (analogous to Observation 6.9 for Algorithm 4 from Phase II).

OBSERVATION 6.24. *Algorithm 5 runs in time $\tilde{O}(mn^{1/3})$, and outputs sets $R_{2,\text{in}}(v) \subseteq S_2$ of size $|R_{2,\text{in}}(v)| = O(\log n)$ for all $v \in V$. Moreover, for every $v \in V$ and $s \in R_{2,\text{in}}(v)$, we have $v \in B^{(4)}_{\text{out}}(s)$.*

Proof. First note that the total number of vertex samples required at Line 3 is $|S^{(1)} \uplus \cdots \uplus S^{(k)}| = k \cdot 10n^{2/3}/\log n = 100n^{2/3} \leq |S_2|$. In each iteration $1 \leq i \leq k$, the algorithm only adds at most one sampled vertex $t \in S_2$ to the set $R^{(i)}(v)$ for each $v \in V$, so each output set $R_{2,\text{in}}(v) = R^{(k)}(v) \subseteq S_2$ and has size $|R_{2,\text{in}}(v)| \leq k \leq O(\log n)$.

To prove the moreover part, note that by definition of $T^{(i)}(v)$ at Line 7, $v \in B^{(4)}_{\text{out}}(s)$ holds for all $s \in T^{(i)}(v)$ and thus for all $s \in R_{2,\text{in}}(v)$.

It remains to bound the running time. In each iteration, Line 5 takes time $\tilde{O}(\frac{m}{n} \cdot n^{2/3})$ for each $s \in S^{(i)}$ by Lemma 6.6 (recall that $|B^{(2)}_{\text{out}}(s)|, |B^{(2)}_{\text{in}}(s)|, |B^{(4)}_{\text{out}}(s)| \leq \tilde{O}(n^{2/3})$ by Lemma 6.5 and Lemma 6.14), which sums to $\tilde{O}(n^{2/3}) \cdot \tilde{O}(\frac{m}{n} \cdot n^{2/3}) = \tilde{O}(mn^{1/3})$ in total.

To implement Line 7 efficiently, for any given $v \in V$ we want to quickly go over all $s \in S^{(i)}$ such that $v \in B^{(4)}_{\text{out}}(s)$. This can be achieved by a preprocessing stage that iterates over $s \in S^{(i)}$ and inserts s to the v-th bucket for every $v \in B^{(4)}_{\text{out}}(s)$, in $\tilde{O}(n^{2/3} \cdot n^{2/3}) = \tilde{O}(n^{4/3})$ total time. Then, we show that all four terms in the inequality at Line 7 are known from the computation at Line 5 or can be computed efficiently from there: $d(s, v)$ is known because $v \in B^{(4)}_{\text{out}}(s), s \in S^{(i)}$, $d(t, s)$ is known because $s \in B^{(2)}_{\text{out}}(t)$ and $t \in S_2$, $d(t, v)$ is known because $v \in B^{(4)}_{\text{out}}(t)$ and $t \in S_2$, and $\underline{d}(s, t)$ can be computed by Lemma 6.21 in $O(\log n)$ time because we know $B^{(2)}_{\text{in}}(t)$ and $d(x, t)$ for all $x \in B^{(2)}_{\text{in}}(t)$.

Thus overall $k = O(\log n)$ iterations, Algorithm 5 takes $\tilde{O}(mn^{1/3})$ time. ∎

Now we prove the key Lemma 6.18, which states that Algorithm 5 guarantees $B'(v)$ to have small size with high probability.

Proof. [Proof of Lemma 6.18]

The proof is based on symmetry of elimination, which is similar to the earlier proof of Lemma 6.5.

Fix the $v \in V$ from Definition 6.16. Due to Item 1 of Definition 6.16, here we only need to consider vertices from $C_v := \{s \in V : v \in B^{(4)}_{\text{out}}(s)\}$. We make the following definition motivated by Item 2 and Item 3 of Definition 6.16: for two vertices $s, t \in C_v$, we say t *eliminates* s, if $s \notin B^{(2)}_{\text{out}}(t)$ or $2d(s, v)+d(t, s) \geq 2d(t, v)+\underline{d}(s, t)$. Then, observe that $B'(v)$ consists of exactly the vertices $s \in C_v$ that are not eliminated by any vertex in $R_{2,\text{in}}(v)$.

Now we show that for any $s, t \in C_v$, either s eliminates t or t eliminates s. Suppose to the contrary that s does not eliminate t, and t does not eliminate s. Then we have inequalities

$$2d(s, v) + d(t, s) < 2d(t, v) + \underline{d}(s, t) \leq 2d(t, v) + d(s, t)$$

and

$$2d(t, v) + d(s, t) < 2d(s, v) + \underline{d}(t, s) \leq 2d(s, v) + d(t, s),$$

which are contradicting each other.

Having proved this symmetry property, the rest of the arguments is the same as in Lemma 6.5, and we omit it here. ∎

Finally, we can state the time complexity of the entire Algorithm 3.

THEOREM 6.25. (RUNNING TIME OF ALGORITHM 3) *Algorithm 3 runs in $\tilde{O}(mn^{1/3})$ time with high probability.*

Proof. The running time of Phase I is $\tilde{O}(mn^{1/3})$ by Observation 6.1. The running time of Phase II is $\tilde{O}(mn^{1/3})$ by Proposition 6.1.

For Phase III, Line 11 (computing eliminators $R_{2,\text{in}}(v)$ for all $v \in V$) takes $\tilde{O}(mn^{1/3})$ time by Observation 6.24. Then, the **for** loop takes $\tilde{O}(\frac{m}{n} \cdot |\tilde{B}'(v)|)$ time for each $v \in V$. Since $\tilde{B}'(v) \subseteq B'(v) \cup \{v\}$ (by Observation 6.20) and $|B'(v)| \leq \tilde{O}(n^{1/3})$ (by Lemma 6.18), the total time for this loop is $n \cdot \tilde{O}(\frac{m}{n} \cdot n^{1/3}) = \tilde{O}(mn^{1/3})$.

Thus, the overall running time of Algorithm 3 is $\tilde{O}(mn^{1/3})$. ∎

7 Conclusion

We conclude with a few open questions:

1. Can we compute 3-roundtrip spanner in $\tilde{O}(n^2)$ time (or even faster)?
2. Can we compute $(2k-1)$-approximate roundtrip emulators faster on sparse graphs?
3. For the $O(mn^{1/k})$-time roundtrip spanner (or directed girth) algorithm of [CLRS20], can we improve its $O(k \log k)$ approximation ratio to $O(k)$? Can our technique be combined with the divide-and-conquer techniques of [PRSTV18; CLRS20; DV20]?
4. Can we show fine-grained lower bounds for the task of computing roundtrip spanners? In particular, can we rule out $\tilde{O}(m)$-time algorithms for computing $(2k-1)$-roundtrip spanners of sparsity $O(n^{1+1/k})$?

References

[AB17] Amir Abboud and Greg Bodwin. "The 4/3 Additive Spanner Exponent Is Tight". *J. ACM* 64.4 (Sept. 2017). ISSN: 0004-5411. DOI: 10.1145/3088511. URL: https://doi.org/10.1145/3088511.

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. "Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication)". *SIAM Journal on Computing* 28.4 (1999), pp. 1167–1181. DOI: 10.1137/S0097539796303421. eprint: https://doi.org/10.1137/S0097539796303421. URL: https://doi.org/10.1137/S0097539796303421.

[ADDJS93] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. "On Sparse Spanners of Weighted Graphs". *Discret. Comput. Geom.* 9 (1993), pp. 81–100.

[BKMP10] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. "Additive Spanners and (α,β)-Spanners". *ACM Trans. Algorithms* 7.1 (Dec. 2010). ISSN: 1549-6325. DOI: 10.1145/1868237.1868242. URL: https://doi.org/10.1145/1868237.1868242.

[BS07] Surender Baswana and Sandeep Sen. "A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs". *Random Struct. Algorithms* 30.4 (2007), pp. 532–563.

[BV15] Gregory Bodwin and Virginia Vassilevska Williams. "Very Sparse Additive Spanners and Emulators". In: *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*. ITCS '15. Rehovot, Israel: Association for Computing Machinery, 2015, pp. 377–382. ISBN: 9781450333337. DOI: 10.1145/2688073.2688103. URL: https://doi.org/10.1145/2688073.2688103.

[BV16] Greg Bodwin and Virginia Vassilevska Williams. "Better Distance Preservers and Additive Spanners". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '16. Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 855–872. ISBN: 9781611974331.

[CDG20] Ruoxu Cen, Ran Duan, and Yong Gu. "Roundtrip Spanners with (2k-1) Stretch". In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 24:1–24:11. DOI: 10.4230/LIPIcs.ICALP.2020.24. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.24.

[CL21] Shiri Chechik and Gur Lifshitz. "Optimal Girth Approximation for Dense Directed Graphs". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. SIAM, 2021, pp. 290–300. DOI: 10.1137/1.9781611976465.19. URL: https://doi.org/10.1137/1.9781611976465.19.

[CLRS20] Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. "Constant girth approximation for directed graphs in subquadratic time". In: *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. ACM, 2020, pp. 1010–1023. DOI: 10.1145/3357713.3384330. URL: https://doi.org/10.1145/3357713.3384330.

[CW04] Lenore Cowen and Christopher G. Wagner. "Compact roundtrip routing in directed networks". *J. Algorithms* 50.1 (2004), pp. 79–95. DOI: 10.1016/j.jalgor.2003.08.001. URL: https://doi.org/10.1016/j.jalgor.2003.08.001.

[DHZ96] Dorit Dor, Shay Halperin, and Uri Zwick. "All Pairs Almost Shortest Paths". In: vol. 29. Aug. 1996, pp. 452–461. DOI: 10.1137/S0097539797327908.

[DV20] Mina Dalirrooyfard and Virginia Vassilevska Williams. "Conditionally Optimal Approximation Algorithms for the Girth of a Directed Graph". In: *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. Vol. 168. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 35:1–35:20. DOI: 10.4230/LIPIcs.ICALP.2020.35. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.35.

[HP18] Shang-En Huang and Seth Pettie. "Lower Bounds on Sparse Spanners, Emulators, and Diameter-reducing shortcuts". In: *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*. 2018.

[KP23] Shimon Kogan and Merav Parter. "New Additive Emulators". In: *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Ed. by Kousha Etessami, Uriel Feige, and Gabriele Puppis. Vol. 261. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 85:1–85:17. ISBN: 978-3-95977-278-5. DOI: 10.4230/LIPIcs.ICALP.2023.85. URL: https://drops.dagstuhl.de/opus/volltexte/2023/18137.

[LVWX22] Kevin Lu, Virginia Vassilevska Williams, Nicole Wein, and Zixuan Xu. "Better Lower Bounds for Shortcut Sets and Additive Spanners via an Improved Alternation Product". In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2022, pp. 3311–3331.

[Pet09] Seth Pettie. "Low distortion spanners". *ACM Transactions on Algorithms (TALG)* 6.1 (2009), pp. 1–22.

[PRSTV18] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. "Approximating Cycles in Directed Graphs: Fast Algorithms for Girth and Roundtrip Spanners". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. SIAM, 2018, pp. 1374–1392. DOI: 10.1137/1.9781611975031.91. URL: https://doi.org/10.1137/1.9781611975031.91.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. "Deterministic Constructions of Approximate Distance Oracles and Spanners". In: *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*. Vol. 3580. Lecture Notes in Computer Science. Springer, 2005, pp. 261–272. DOI: 10.1007/11523468_22. URL: https://doi.org/10.1007/11523468%5C_22.

[RTZ08] Liam Roditty, Mikkel Thorup, and Uri Zwick. "Roundtrip spanners and roundtrip routing in directed graphs". *ACM Trans. Algorithms* 4.3 (2008), 29:1–29:17. DOI: 10.1145/1367064.1367069. URL: https://doi.org/10.1145/1367064.1367069.

[TZ01] Mikkel Thorup and Uri Zwick. "Approximate distance oracles". In: *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*. ACM, 2001, pp. 183–192. DOI: 10.1145/380752.380798. URL: https://doi.org/10.1145/380752.380798.

[Woo06] David P. Woodruff. "Lower Bounds for Additive Spanners, Emulators, and More". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 389–398. DOI: 10.1109/FOCS.2006.45.

[ZL18] Chun Jiang Zhu and Kam-yiu Lam. "Deterministic improved round-trip spanners". *Inf. Process. Lett.* 129 (2018), pp. 57–60. DOI: 10.1016/j.ipl.2017.09.008. URL: https://doi.org/10.1016/j.ipl.2017.09.008.