Local Computation Algorithms for Spanners

Merav Parter[1]
Weizmann IS, Rehovot, Israel
merav.parter@weizmann.ac.il

Ronitt Rubinfeld[2]
CSAIL, MIT, Cambridge, MA, USA and TAU, Tel Aviv, Israel
ronitt@csail.mit.edu

Ali Vakilian[3]
CSAIL, MIT, Cambridge, MA, USA
vakilian@mit.edu

Anak Yodpinyanee[4]
CSAIL, MIT, Cambridge, MA, USA
anak@mit.edu

Abstract

A graph spanner is a fundamental graph structure that faithfully preserves the pairwise distances in the input graph up to a small multiplicative stretch. The common objective in the computation of spanners is to achieve the best-known existential size-stretch trade-off *efficiently*.

Classical models and algorithmic analysis of graph spanners essentially assume that the algorithm can read the input graph, construct the desired spanner, and write the answer to the output tape. However, when considering massive graphs containing millions or even billions of nodes not only the input graph, but also the output spanner might be too large for a single processor to store.

To tackle this challenge, we initiate the study of *local computation algorithms (LCAs)* for graph spanners in general graphs, where the algorithm should locally decide whether a given edge $(u, v) \in E$ belongs to the output (sparse) spanner or not. Such LCAs give the user the "illusion" that a *specific* sparse spanner for the graph is maintained, without ever fully computing it. We present several results for this setting, including:

- For general n-vertex graphs and for parameter $r \in \{2, 3\}$, there exists an LCA for $(2r - 1)$-spanners with $\widetilde{O}(n^{1+1/r})$ edges and sublinear probe complexity of $\widetilde{O}(n^{1-1/2r})$. These size/stretch trade-offs are best possible (up to polylogarithmic factors).
- For every $k \geq 1$ and n-vertex graph with maximum degree Δ, there exists an LCA for $O(k^2)$ spanners with $\widetilde{O}(n^{1+1/k})$ edges, probe complexity of $\widetilde{O}(\Delta^4 n^{2/3})$, and random seed of size polylog(n). This improves upon, and extends the work of [Lenzen-Levi, ICALP'18].

We also complement these constructions by providing a polynomial lower bound on the probe complexity of LCAs for graph spanners that holds even for the simpler task of computing a sparse connected subgraph with $o(m)$ edges.

To the best of our knowledge, our results on 3 and 5-spanners are the first LCAs with sublinear (in Δ) probe-complexity for $\Delta = n^{\Omega(1)}$.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners, Theory of computation → Sketching and sampling

[1] MP is supported by Minerva Foundation (124042) and ISF-2084/18.
[2] RR is supported by the NSF grants CCF-1650733, CCF-1733808, IIS-1741137 and CCF-1740751.
[3] AV is supported by the NSF grant CCF-1535851.
[4] AY is supported by the NSF grants CCF-1650733, CCF-1733808, IIS-1741137 and the DPST scholarship, Royal Thai Government.

© Merav Parter, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee;
licensed under Creative Commons License CC-BY
10th Innovations in Theoretical Computer Science (ITCS 2019).
Editor: Avrim Blum; Article No. 58; pp. 58:1–58:21

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases Local Computation Algorithms, Sub-linear Algorithms, Graph Spanners

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.58

1 Introduction

One of the fundamental structural problems in graph theory is to find a *sparse* structure which preserves the pairwise distances of vertices. In many applications, it is crucial for the sparse structure to be a *subgraph* of the input graph; this problem is called the *spanner* problem. For an input graph $G = (V, E)$, a *k-spanner* $H \subseteq G$ (for $k \geq 1$) satisfies that for any $v, u \in V$, the distance from v to u in H is at most k times the distance from v to u in G, where k is referred to as the *stretch* of the spanner. Furthermore, to reduce the *cost* of the solution, it is desired to output a minimum size/weight such subgraph H. The notion of spanners was introduced by Peleg and Schäffer [31] and has been used widely in different applications such as routing schemes [3, 30], synchronizers [2, 32], SDD's [37] and spectral sparsifiers [19].

It is folklore that for every n-vertex graph G, there exists a $(2k-1)$-spanner $H \subseteq G$ with $O(n^{1+1/k})$ edges. In particular, if the *girth conjecture* of Erdős [14] is true, then this size-stretch trade-off is optimal. Spanners have been considered in many different models such as distributed algorithms [5, 8–11, 13, 33] and dynamic algorithms [4, 6, 12].

Local computation of small stretch spanners. When the graph is so large that it does not fit into the main memory, the existing algorithms are not sufficient for computing a spanner. Instead, we aim at designing an algorithm that answers *queries* of the form "is the edge (u, v) in the spanner?" without computing the whole solution upfront. One way to get around this issue is to consider the *Local Computation Algorithms (LCAs)* model (also known as the *Centralized Local model*), introduced by Rubinfeld et al. [35] and Alon et al. [1]. There can be many different plausible k-spanners; however, the goal of LCAs for the k-spanner problem is to design an algorithm that, given access to *primitive* probes (i.e. Neighbor, Degree and Adjacency probes) on the input graph G, for each *query* on an edge $e \in E(G)$ *consistently* with respect to a *unique* k-spanner $H \subseteq G$ (picked by the LCA arbitrarily), outputs whether $e \in H$. The performance of the LCA is measured based on the *quality of solution* (i.e. number of edges in H) and the *probe complexity* (the maximum number of probes per each query) of the algorithm[5]. In other words, an LCA gives us the "illusion" as if we have query access to a precomputed *k-spanner* of G.

The study of LCAs with *sublinear* probe complexity for *nearly linear* size spanning subgraphs (or *sparsifiers*) is initiated by Levi, Ron, and Rubinfeld [24, 25] for some restricted families of graphs such as minor-closed families. However, their focus is mainly on designing LCAs that preserve the *connectivity* while allowing the stretch factor to be as large as n. Moreover, in their work, the input graph is sparse (has $O(n)$ edges), while the classical k-spanner problem becomes relevant only when the input graph is dense (with superlinear number of edges). Recently, Lenzen and Levi [21] designed the first sparsifier LCA in *general graphs* with $(1+\varepsilon)n$ edges, stretch $O(\log^2 n \cdot \text{poly}(\Delta/\varepsilon))$ and probe complexity of $O(\text{poly}(\Delta/\varepsilon) \cdot n^{2/3})$, where Δ is the maximum degree of the input graph.

[5] We may also measure the *time complexity* of an LCA. In our LCAs, the time complexities are clearly only a factor of $\text{poly}(\log n)$ higher than the corresponding probe complexities, so we focus our analysis on probe complexities.

In this work, we show that sublinear time LCAs for spanners are indeed possible in several cases. We give: (I) 3 and 5-spanners for general graphs with optimal trade-offs between the number of edges and the stretch parameter (up to polylogarithmic factors), and (II) general k-spanners, either in the dense regime (when the minimum degree is at least $n^{1/2-1/(2k)}$) or in the sparse regime (when the maximum degree is $n^{1/12-\varepsilon}$).

Broader scope and agenda: local computation algorithms for dense graphs. LCAs have been established by now for a large collection of problems, including Maximal Independent Set, Maximum Matching, and Vertex Cover [1,15,26,27,29,34,35]. These algorithms typically suffer from a probe complexity that is exponential in Δ and thus are efficient only in the sparse regime when $\Delta = O(1)$.

To this end, obtaining LCAs even with a polynomial dependency in Δ is a major open problem for many classical local graph problems, as noted in [16,26,28]. For instance, recently Ghaffari and Uitto [16] obtained an LCA for the MIS problem with probe complexity of $\Delta^{O(\log\log\Delta)} \cdot \log n$ improving upon a long line of results. Their result also illustrates the connection between LCAs with good dependency on Δ, and algorithms for the massively parallel computation model with sublinear space per machine. Recently, [26] and [21] provided LCAs with probe complexities *polynomial* in Δ for the problems of $(1-\varepsilon)$-maximum matching and sparse connected subgraphs, respectively. Note that in the context of spanners, such algorithms are still inefficient when the maximum degree is polynomial in n, which is precisely the setting where graph sparsification is applied.

1.1 Our results and techniques

In this paper we initiate the study of LCAs for graph spanners in *general graphs* which concerns with the following task: *How can we decide quickly (e.g., sublinear in n time) if a given edge e belongs to a sparse spanner (with fixed stretch) of the input graph, without preprocessing and storing any auxiliary information?* In the design of LCAs for graph problems, the set of defined *probes* to the input graph plays an important role. Here we consider the following common probes: Neighbor probes ("what is the i^{th} neighbor of u"?), Degree probes ("what is $\deg(u)$?") and Adjacency probes ("are u and v neighbors"?) [17,18]. We emphasize that the answer to an Adjacency probe on an ordered pair $\langle u, v\rangle$ is the index of v in $\Gamma(u)$ if[6] the edge exists and $\perp$ otherwise. Note that if the maximum degree in the input graph is $O(1)$, each Adjacency probe can be implemented by $O(1)$ number of Neighbor probes.

The problem of designing LCAs for spanners is closely related to designing LCAs for *sparse connected subgraphs* with $(1+\varepsilon)n$ edges which was first introduced by [24]. With the exception of [21], a long line of results for this problem usually concerns special *sparse* graph families, rather than general graphs. A summary of these results with a comparison to our results is provided in Table 1.

1.1.1 LCAs for 3 and 5-Spanners for General Graphs

Our first contribution is the local construction of 3 and 5-spanners for general graphs, while achieving the optimal trade-offs between the number of edges and the stretch factors (up to polylogarithmic factors)[7]. In particular, our LCAs have $n^{1-\varepsilon}$ probe complexity even when the input graph is dense with $\Delta = \Omega(n)$; note that in such a case, given a query

[6] $\Gamma(u)$ denotes the neighbor set of u, whereas $\Gamma^+(u) = \Gamma(u) \cup \{u\}$.
[7] Indeed, the girth conjecture of Erdős is resolved for these stretch factors; see e.g., [39].

Table 1 Table of results on LCAs for the spanner problem. The symbol '$-$' indicates that the stretch is not analyzed. The input graph is a simple graph with n vertices, m edges, maximum degree Δ, and belongs to the indicated graph family. $\widetilde{O}$ hides a factor of $\text{poly}(\log n, k)$.

	Reference	Graph Family	# Edges	Stretch	Probe Complexity
Prior works	[24]	Bounded Degree Graphs	$(1+\varepsilon)n$	$-$	$\Omega(\sqrt{n})$
		Expanders	$(1+\varepsilon)n$	$-$	$O(\sqrt{n})$
		Subexponential growth	$(1+\varepsilon)n$	$-$	$O(\sqrt{n})$
	[23]	Minor-free	$(1+\varepsilon)n$	$\text{poly}(\Delta, 1/\varepsilon)$	$\text{poly}(\Delta, 1/\varepsilon)$
	[25]	Minor-free	$(1+\varepsilon)n$	$O((\log\Delta)/\varepsilon)$	$\text{poly}(\Delta, 1/\varepsilon)$
	[22]	Expansion $(1/\log n)^{1+o(1)}$	$(1+\varepsilon)n$	super-exponential in $1/\varepsilon$	super-exponential in $1/\varepsilon$
	[21]	General	$(1+\varepsilon)n$	$O(\log^2 n \cdot \text{poly}(\Delta/\varepsilon))$	$O(n^{2/3} \cdot \text{poly}(\Delta/\varepsilon))$
Here	Thm. 1	General	$\widetilde{O}(n^{1+1/r})$	$2r-1$ $(r \in \{2,3\})$	$\widetilde{O}(n^{1-1/(2r)})$
	Thm. 12	Min degree $O(n^{1/2-1/(2k)})$	$\widetilde{O}(n^{1+1/k})$	5	$\widetilde{O}(n^{1-1/(2k)})$
	Thm. 2	Max degree $O(n^{1/12-\varepsilon})$	$\widetilde{O}(n^{1+1/k})$	$O(k^2)$	$\widetilde{O}(n^{1-4\epsilon})$
	Thm. 3	General	$o(m)$	any $k \le n$	$\Omega(\min\{\sqrt{n}, n^2/m\})$

edge (u, v), the LCA should return yes or no without being able to inspect the neighbor lists $\Gamma(u)$ and $\Gamma(v)$. In what follows we show how to manipulate the common distributed construction by Baswana and Sen [5] to yield LCAs for 3-spanners and 5-spanners with sublinear probe complexity.

The common distributed approach. Most distributed spanner constructions are based on thinning the graph via clustering: construct a random set S of *centers* by adding each vertex to S independently with some fixed probability. For each vertex v sufficiently close to a center in S, include the edges of the shortest path connecting v to its closest member $s \in S$: this induces a *cluster* around each center $s \in S$, where every pair of vertices in the same cluster are connected by a short path. Then, add edges connecting pairs of neighboring clusters to ensure the desired stretch factor.

The following algorithm constructs a 3-spanner $H \subseteq G$ with $\widetilde{O}(n^{3/2})$ edges. First, add to H all edges incident to vertices of degree at most $\sqrt{n}$. Second, pick a collection S of centers by sampling each vertex independently with probability $\Theta(\log n/\sqrt{n})$. Each vertex v of degree at least $\sqrt{n}$ picks a *single* neighboring center $s \in S \cap \Gamma(v)$ (which exists w.h.p.) as its *center*, then adds (v, s) to H, forming a collection of $|S| = O(\sqrt{n})$ clusters (stars) around these centers. Lastly, every vertex u adds only one edge to each of its neighboring clusters – note that this last step may add edges whose endpoints are both non-centers. This results in a 3-spanner: For omitted edge (u, v) in G, if u and v are in the same cluster, then they have a path of length 2 through their shared center s. If u and v are in different clusters, an edge from u to some other vertex w in v's cluster would have been chosen, providing the path $\langle u, w, s, v\rangle$ of desired stretch 3 connecting u and v, where s is v's center.

The challenge and key ideas. Recall that our goal is to design an LCA for 3-spanners $H \subseteq G$ of size $\widetilde{O}(n^{3/2})$ and probe complexity of $\widetilde{O}(n^{3/4})$: the LCA is given an edge (u, v) and must answer whether $(u, v) \in E(H)$. First, if $\deg(u)$ or $\deg(v)$ is at most $\sqrt{n}$, then the algorithm can immediately say `YES`. This requires only two DEGREE probes for the endpoints u, v. Hence, the interesting case is where both u and v have degrees at least $\sqrt{n}$.

We start by sampling each vertex into the center set S with probability of $p = \Theta(\log n/\sqrt{n})$, thus w.h.p. guaranteeing that each high-degree vertex has at least one sampled neighbor. For clarity of explanation, assume that given the ID of a vertex v, the LCA algorithm can decide

(with no further probes) whether v is sampled. Upon selecting the set of centers S, the above mentioned distributed algorithm has two degrees of freedom (which our LCA algorithm will enjoy). First, for a high-degree vertex v, there could be potentially many sampled neighbors in S: the distributed algorithm lets v join the cluster of an arbitrarily sampled neighbor. The second degree of freedom is in connecting a high-degree vertex to neighboring clusters. In the distributed algorithm, a vertex connects to an arbitrarily chosen neighbor in each of its neighboring clusters. Since the answers of the LCA algorithm should be consistent, it is important to carefully fix these decisions to allow small probe complexity.

The naïve approach for 3-spanners and its shortcoming. The most naïve approach is as follows: for each v, traverse the list $\Gamma(v)$ in a fixed order and pick the *first* neighbor that satisfies the required conditions. That is, a vertex joins the cluster of its *first* sampled neighbor (center) and connects to its *first* representative neighbor in each of its neighboring clusters. To analyze the probe complexity of such a construction, consider a query edge (u, v) where $\deg(u), \deg(v) \geq \sqrt{n}$. By probing for the first $\sqrt{n}$ neighbors of u and v, one can compute the cluster centers c_u and c_v of u and v with high probability. The interesting case is where u and v belong to different clusters. In such a case, the LCA algorithm should say `YES` only if v is the first neighbor of u that belongs to the cluster of c_v. To check if this condition holds, the algorithm should probe for each of the neighbors w of u that appears before v in $\Gamma(u)$, and say `NO` if there exists such earlier neighbor w that belongs to the cluster of c_v. Here, it remains to show how this cluster-membership testing procedure is implemented.

A *cluster-membership test*, for a pair $\langle s, w\rangle$ with $s \in S$, must return `YES` iff w belongs to the cluster of the center s. The above mentioned algorithm thus makes $O(\deg(v))$ cluster-membership tests for each w preceding u in $\Gamma(v)$ and $s = c_v$. Since each center is sampled with probability $p = \log n/\sqrt{n}$, the probe complexity of a single cluster-membership test is $O(\sqrt{n})$ w.h.p., leading to a total probe complexity of $O(\deg(v) \cdot \sqrt{n})$.

Idea (I) – Multiple centers for efficient cluster-membership test. The key idea in our solution is to pick the cluster centers in a way that allows answering each cluster-membership test for a pair $\langle s, w\rangle$ using a single Neighbor probe! Towards this goal, we let each high-degree vertex join *multiple* clusters, instead of just one. In particular, for a vertex w, we look at the subset $\Gamma_1(w)$ consisting of its first $\sqrt{n}$ neighbors in $\Gamma(w)$. We then let w join the clusters of all sampled neighbors in $\Gamma_1(w) \cap S$. Since each vertex is a center with probability p, this implies that, w.h.p., w joins $\Theta(p \cdot |\Gamma_1(w)|) = \Theta(\log n)$ many clusters. Though this approach adds a multiplicative $O(\log n)$ factor to the size of our spanner, it will pay off dramatically in terms of the probe complexity of our LCA. In particular, this modification enables the algorithm to test cluster-membership with a single Adjacency probe: the vertex w belongs to the cluster of s, if the index of s in w's neighbor-list is at most $\sqrt{n}$ (the index is returned by the Adjacency probe on u and s). This idea alone decreases the probe complexity of our LCA to $\widetilde{O}(\deg(w))$.

Idea (II) – Neighborhood partitioning. The multiple center technique above allows our LCA to handle edges adjacent to a vertex u of degree at most $n^{3/4}$. For $\deg(u) > n^{3/4}$, our LCA cannot afford to look at all neighbors of u. To this end, we *partition* the neighbors of u into *blocks* of size $n^{3/4}$ each. Rather than adding only one edge between u to each neighboring cluster, we make the decision on which edges to keep for each block *independently*, by scanning only the block containing v and keeping (u, v) if v belongs to the cluster that was not previously seen in this block. Though this leads to an increase in the number of

edges by a factor of $\deg(u)/n^{3/4} \leq n^{1/4}$, we can now keep the probe complexity down to $\widetilde{O}(n^{3/4})$ as we only need to scan the block containing v given the query (u, v) instead of u's entire neighbor-list. To keep the size of the spanner small, e.g., $\widetilde{O}(n^{3/2})$, we use the fact that $O(n^{1/4} \log n)$ sampled vertices are enough to hit the neighborhoods of all vertices with degree more than $n^{3/4}$ with high probability. Since for each block of size $n^{3/4}$ in the neighborhood of u the algorithm adds $O(|S|)$ edges, the total number of edges added per vertex is $O(|S| \cdot \deg(u)/n^{3/4}) = \widetilde{O}(n^{3/2})$, as desired.

Overview of the LCA for 5-spanners. For 5-spanners, the desired number of edges is $\widetilde{O}(n^{4/3})$. This allows us to immediately add to the spanner all edges incident to low-degree vertices u with $\deg(u) = \widetilde{O}(n^{1/3})$. The common distributed construction for 5-spanners computes $O(n^{2/3})$ clusters by sampling each center independently with probability $\Theta(\log n/n^{1/3})$. By letting each high-degree vertex (i.e., with $\deg(u) = \Omega(n^{1/3})$) join the cluster of one of its sampled neighbors, the spanner contains a collection of $O(n^{2/3})$ (vertex-disjoint) clusters that, w.h.p., cover all high-degree vertices. Finally, each pair of neighboring clusters C_1, C_2 are connected by adding an edge $(u, v) \in (C_1 \times C_2) \cap E$ to the spanner H. It is straightforward to verify that H is a 5-spanner of size $\widetilde{O}(n^{4/3})$.

Designing LCAs for the 5-spanner problem turns out to be significantly more challenging than the 3-spanner case. The reason is that deciding whether an edge (u, v) is in the 5-spanner requires information from the *second neighborhoods* of v and u, which is quite cumbersome when one cannot even read the entire neighborhood of a vertex. Our solution extends the 3-spanner construction in two ways: some of the edges added to our 5-spanner are between *cluster* pairs, instead of edges between a vertex and a cluster as in the 3-spanner solution. Another set of edges added to the 5-spanner is between pairs of vertex and cluster, but unlike the 3-spanner case, these clusters have now *radius two*.

Idea (III) – Cluster partitioning (bucketing). The standard clustering-based construction of 5-spanners adds an edge between every pair of neighboring clusters (stars). This clustering-based construction cannot be readily implemented with the desired probe complexity. To see why, consider clusters centered at s and t, containing u and v respectively. A naïve attempt spends $\deg(s) \cdot \deg(t)$ probes for vertices between these clusters, as to consistently pick a unique edge between the two clusters.

One of our tools extends the idea of neighborhood partitioning from 3-spanner into cluster partitioning. Each of the $O(n^{2/3})$ clusters is partitioned into balanced *buckets* of size $\Theta(n^{1/3})$.[8] The algorithm then picks only one edge between any pair of neighboring buckets. Since the number of buckets can be shown to be $\widetilde{O}(n^{2/3})$, the spanner size still remains $\widetilde{O}(n^{4/3})$. Unlike partitioning *neighbor-lists*, partitioning a *cluster* requires the full knowledge of its members – which are no longer nicely indexed in a list. To be able to efficiently partition a clusters, the algorithm allows only vertices with degree at most $n^{5/6}$ to be chosen as cluster centers. The benefit of this restriction is that one can inspect the entire neighborhood of a center in $O(n^{5/6})$. The drawback of this approach is that it only clusters vertices that have sufficiently many neighbors (i.e., at least $n^{1/3}$) with degree less than $n^{5/6}$. The remaining vertices are handled via their high-degree neighbors (i.e., of degree at least $n^{5/6}$) as described next.

[8] Note that each cluster may have at most one bucket of size $o(n^{1/3})$.

Idea (IV) – Representatives. Using the neighborhood-partitioning idea from 3-spanner, all vertices with degree at least $n^{5/6}$ can be clustered by sampling $\widetilde{O}(n^{1/6})$ cluster centers. By partitioning the neighborhood of each high-degree vertex into disjoint blocks each of size $\widetilde{O}(n^{5/6})$, one can construct a 3-spanner for all edges incident to these high-degree vertices with probe complexity of $\widetilde{O}(n^{5/6})$ while using $\widetilde{O}(n^{4/3})$ edges. To take care of vertices of degrees less than $n^{5/6}$ that have many high-degree neighbors, we let them join the cluster of their high-degree neighbors, hence creating *clusters of depth* 2.

To choose which cluster to join (in the second level), our vertex, which has many high-degree neighbors, simply chooses and connects itself to one or more high-degree neighbors, called its *representatives*. To determine the representatives of a vertex u, we simply pick $\Theta(\log n)$ random neighbors of u, and w.h.p. one of them will have high-degree, and hence is chosen as u's representative.

We implement our LCA by first picking $|S| = \widetilde{O}(n^{1/6})$ centers. Consider the query edge (u, v) where $\deg(u), \deg(v) \geq n^{1/3}$ and u has many high-degree neighbors. Here, u has $\Theta(\log n)$ representatives, each of which has $\Theta(\log n)$ centers in S w.h.p., so u belongs to $O(\log^2 n)$ clusters. As in the 3-spanner case, we keep (u, v) if v is the first neighbor of u in the cluster that v belongs to. We find the representatives of each neighbor of u by making $O(\log n)$ probes, and for all these $\deg(u) \cdot O(\log n) = \widetilde{O}(n^{5/6})$ representatives, check if they belong to any of v's $O(\log^2 n)$ clusters with $\widetilde{O}(n^{5/6})$ total probes.

▶ **Theorem 1** (3 and 5-spanners). *For every n-vertex simple undirected graph G, there exists an LCA for $(2r-1)$-spanners with $\widetilde{O}(n^{1+1/r})$ edges and probe complexity $\widetilde{O}(n^{1-1/(2r)})$ for $r \in \{2, 3\}$. Moreover, the algorithm only uses a seed of $O(\log^2 n)$ random bits.*

In fact, if G has *minimum* degree $\omega(n^{1/3})$, we may apply the 5-spanner construction (with modified parameters) to obtain 5-spanners with even smaller number of edges as indicated in Table 1 (Theorem 12): this minimum degree assumption indeed allows even sparser spanners, bypassing the girth conjecture that holds for *general* graphs. We also remark that, in the somewhat related setting of dynamic computation, spanner algorithms with worst-case *sublinear* update time are currently known only for 3 and 5-spanners as well (see Bodwin and Krinninger, [6]).

1.1.2 LCA for $O(k^2)$-spanners

Our second contribution is the local construction of $O(k^2)$-spanners with $O(n^{1+1/k})$ edges for any $k \geq 1$, which has sub-linear probe complexity for graphs of maximum degree $\Delta = O(n^{1/12-\varepsilon})$. Our approach improves upon and extends the recent work of Lenzen and Levi [21]. The work of [21] aims at locally constructing a spanning subgraph with $O(n)$ edges, but the stretch parameter of their subgraph might be as large as $O(\text{poly}(\Delta)\log^2 n)$. In addition, this construction requires a random seed of polynomial size. In our construction, we reduce the stretch parameter of the constructed subgraph to $O(k^2)$, independent of both n and Δ, while using only $\widetilde{O}(n^{1+1/k})$ edges. In addition, we implement our randomized constructions using $\text{poly}(\log n)$ independent random bits, whereas [21] uses $\text{poly}(n)$ bits. We remark that for the LCAs with large stretch parameter considered in [21], our techniques can still be applied to exponentially reduce the required amount of random bits, and save a factor of Δ in the probe complexity. The details of $O(k^2)$-spanners are omitted in this version: please refer to the longer version of this paper for the missing details of this result.

▶ **Theorem 2** ($O(k^2)$-spanners). *For every integer $k \geq 1$ and every n-vertex simple undirected graph G with maximum degree Δ, there exists a (randomized) LCA for $O(k^2)$-spanner with $\widetilde{O}(n^{1+1/k})$ edges and probe complexity $\widetilde{O}(\Delta^4 n^{2/3})$. Moreover, the algorithm only uses $O(\log^2 n)$ random bits.*

The high level structure is as in [21]: for a given stretch parameter k, partition the edges in G into the *sparse* set E_{sparse} and the *dense* set E_{dense}. Roughly speaking, the sparse set E_{sparse} only consists of edges (u, v) for which the k-neighborhood in G of either u or v contains at most $O(n^{2/3})$ vertices. For this sparse region in the graph, we can simulate a standard distributed algorithm for spanners [5, 7] (using only a poly-logarithmic number of random bits), with small probe complexity. This yields an LCA handling the sparse edges with $O(\Delta^2 n^{2/3})$ probe complexity.

To take care of the dense edges, we sample a collection of $O(n^{2/3} \log n)$ centers and partition the (dense) vertices into *Voronoi cells* around these centers.

The main challenge is in connecting the Voronoi cells, keeping in mind that taking an edge between every pair of cells adds too many edges to the spanner. To get around it, the main contribution of [21] was in designing a set of rules for connecting bounded-size sub-structures in Voronoi cells, called *clusters*. The high-level description of the rules are as follows[9]: *mark* a random subset of $O(n^{1/3} \log n)$ Voronoi cells (among the $n^{2/3}$ Voronoi cells), then connect[10] them according to the following rules using $\widetilde{O}(n)$ edges each. Rule (1): connect every marked Voronoi cells to each of its neighboring Voronoi cells. Rule (2): if a Voronoi cell has no neighboring marked Voronoi cells, then connect it to all its neighboring Voronoi cells as well. Rule (3): For each pair of (not necessarily adjacent) Voronoi cell a and marked Voronoi cell c sharing common neighboring Voronoi cells $\Gamma(\mathsf{a}) \cap \Gamma(\mathsf{c})$, keep an edge from a to a single Voronoi cell $\mathsf{b}^* \in \Gamma(\mathsf{a}) \cap \Gamma(\mathsf{c})$ (i.e., b^* has the minimum ID in $\Gamma(\mathsf{a}) \cap \Gamma(\mathsf{c})$). This last rule handles the edges of (unmarked) Voronoi cells that have some neighboring marked Voronoi cell.

Idea (V) – Establishing the $O(k^2)$ stretch guarantee. In our implementation, the radius of each Voronoi cell is $O(k)$ (as opposed to $O(\Delta \log n)$ in [21]). Thus, it suffices to show that the spanner path from Voronoi cell supervertices a to b only visits $O(k)$ other Voronoi cells. To this end, we impose a random ordering of the Voronoi cells, by assigning them distinct random *ranks*. We then make the following modification to Rule (3): add an edge from a to b if there exists a marked Voronoi cell c such that the rank $r(\mathsf{b})$ of b is among the $O(n^{1/k} \log n)$ lowest ranks in $\Gamma(\mathsf{a}) \cap \Gamma(\mathsf{c})$, restricted to those discovered by the LCA. This modified rule allows us to extend the inductive connectivity argument of [21] to show that every pair of adjacent cells are connected by a path that goes through $O(k)$ cells – since each cell has radius $O(k)$, the final stretch is $O(k^2)$.

Idea (VI) – Graph connectivity with bounded independence. One of our key technical contributions is in showing that one can implement the above randomized random rank assignment using small number of random bits. We show that the ranks of Voronoi cells can be computed using $T = \Theta(k)$ hash functions $h_1, \cdots, h_T$ chosen uniformly at random form a family of $O(\log n)$-wise independent hash functions of the form $\{0,1\}^{\log n} \to \{0,1\}^{O((\log n)/k)}$. We define our rank function as a concatenation of h_i's on the ID of the Voronoi cell's center: for the Voronoi cell centered at v, $r(v) = h_1(\mathsf{ID}(v)) \circ \ldots \circ h_T(\mathsf{ID}(v))$. We then carefully adopt the inductive stretch argument to this randomized rank assignment with limited independence so that in the i^{th} step, our analysis only relies on the hash function h_i.

---

[9] Here, we state a simplified version of the rules. In particular, the rules are expressed in terms of clusters whose exact definitions are skipped for now. Refer to the longer version of our paper for the precise definitions of the rules.

[10] We *connect* two vertex sets by adding the unique lexicographically-first edge between the two vertex sets (if any exists) based on the vertex IDs of the endpoints.

1.1.3 Lower Bounds

To establish the lower bound, we construct two distributions over undirected d-regular graph instances that contain a designated edge e. For graphs in the first family, it holds that after removing e, w.h.p., they remain connected while in the second family, removing e disconnects the endpoints of e and leave them in separate connected components. We show that for the edge e, any LCA that makes $o(\min\{\sqrt{n}, n/d\}) = o(\min\{\sqrt{n}, n^2/m\})$ probes can only distinguish whether the underlying graph is from the first family or the second family with probability $1/2 + o(1)$.

Our approach mainly follows from the analysis of Kaufman, Krivelevich, and Ron [20], on the lower bound construction of [24]. While [20] studies a rather different problem of bipartiteness testing, we consider similar probe types and obtain a similar lower bound as those of [20]. On the other hand, the construction of [24] shows the probe complexity of $\Omega(\sqrt{n})$ for LCAs for spanning graphs that only use NEIGHBOR probes, not ADJACENCY probes.

▶ **Theorem 3** (Lower Bound). *Any local randomized LCA that computes, with success probability at least* $2/3$*, a spanner of the simple undirected* m*-edge input graph* G *with* $o(m)$ *edges, has probe complexity* $\Omega(\min\{\sqrt{n}, n^2/m\})$.

The details of this result are deferred to the longer version of this paper.

1.2 Model Definition and Preliminaries

Graph notation. Throughout, we consider simple unweighted undirected graphs $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges. Each vertex v is labeled by a unique $O(\log n)$-bit value $\mathsf{ID}(v)$[11]. For $u \in V$, let $\Gamma(u, G) = \{v : (u, v) \in E\}$ be the neighbors of u, $\deg(u, G) = |\Gamma(u, G)|$ be its degree, and define $\Gamma^+(u, G) = \Gamma(u, G) \cup \{u\}$. Denote $V_I = \{v \in V : \deg(v, G) \in I\}$ where I is an interval. For $u, v \in V$, let $\mathsf{dist}(u, v, G)$ be the shortest-path distance between u and v in G. Let $\Gamma^k(u, G) = \{v : \mathsf{dist}(u, v, G) \leq k\}$ be the k^{th}-neighborhood of u, and denote its size $\deg_k(u, G) = |\Gamma^k(u, G)|$. For subsets $V_1, V_2 \subseteq V$, let $E(V_1, V_2) = E \cap (V_1 \times V_2)$. The parameter G may be omitted for the input graph.

We assume that the input graph has an adjacency list representation: each neighbor set has a fixed ordering, $\Gamma(u) = \{v'_1, \ldots, v'_{\deg(u)}\}$; this ordering may be arbitrarily (e.g., not necessarily sorted by vertex IDs). Many of the algorithms in this paper are based on partitioning the neighbor-list into balanced-size blocks. For $\Delta \in [n]$ and $u \in V$ such that $\deg(u) \geq \Delta$, let $\Gamma_{\Delta,1}(u), \ldots, \Gamma_{\Delta,\Theta(\deg(u)/\Delta)}(u)$ be blocks of neighbors obtained by partitioning $\Gamma(u)$ into consecutive parts. Each block is of size Δ, except possibly for the last block that is allowed to contain up to 2Δ vertices.

Local Computation Algorithms. We adopt the definition of LCAs by Rubinfeld et al. [35]. A local algorithm has access to the *adjacency list oracle* $\mathcal{O}^G$ which provides answers to the following probes (in a single step):

- **Neighbor probes**: Given a vertex $v \in V$ and an index i, the i^{th} neighbor of v is returned if $i \leq \deg(v)$. Otherwise, $\perp$ is returned. The orderings of neighbor sets are fixed in advance, but can be arbitrary.
- **Degree probes**: Given a vertex $v \in V$, return $\deg(v)$. This probe type is defined for convenience, and can alternatively be implemented via a binary search using $O(\log n)$ NEIGHBOR probes.

[11] We do *not* require IDs to be a bijection $V \to [n]$ as in other LCA papers.

- **Adjacency probes**: Given an ordered pair $\langle u, v\rangle$, if $v \in \Gamma(u)$ then the index i such that v is the i^{th} neighbor of u. Otherwise, $\perp$ is returned.

▶ **Definition 4** (LCA for Graph Spanners)**.** An *LCA* $\mathcal{A}$ for graph *spanners* is a (randomized) algorithm with the following properties. $\mathcal{A}$ has access to the adjacency list oracle $\mathcal{O}^G$ of the input graph G, a tape of random bits, and local read-write computation memory. When given an input (query) edge $(u, v) \in E$, $\mathcal{A}$ accesses $\mathcal{O}^G$ by making probes, then returns `YES` if (u, v) is in the spanner H, or returns `NO` otherwise. This answer must only depend on the query (u, v), the graph G, and the random bits. For a fixed tape of random bits, the answers given by $\mathcal{A}$ to all possible edge queries, must be consistent with one particular sparse spanner.

The main complexity measures of the LCA for graph spanners are the size and stretch of the output spanner, as well as the probe complexity of the LCA, defined as the maximum number of probes that the algorithm makes on $\mathcal{O}^G$ to return an answer for a single input edge. Informally speaking, imagine m instances of the same LCA, each of which is given an edge of G as a query, while the *shared* random tape is broadcasted to all. Each instance decides if its query edge is in the subgraph by making probes to $\mathcal{O}^G$ and inspecting the random tape, but may not communicate with one another by any means. The LCA succeeds for the input graph G and the random tape if the collectively-constructed subgraph is a desired spanner. All the algorithms in this paper are randomized and, for any input graph, succeed with high probability $1 - 1/n^c$ over the random tape.

Paper Organization. In Section 2 and 3 we describe our results for 3 and 5-spanners in general graphs. For simplicity, we first describe all our randomized algorithms as using full independence, then in Section 4, we explain how these algorithms can be implemented using a seed of poly-logarithmic number of random bits).

Clarification. Throughout we use the term "spanner construction" when describing how to construct our spanners. These construction algorithms are used only to define the unique spanner, based on which the LCA makes its decisions: we never construct the full, global spanner at any point.

2 LCA for 3-Spanners

In this section, we present the 3-spanner LCA with probe complexity of $\widetilde{O}(n^{3/4})$. We begin in Section 2.1 by establishing some observations that allow us to "take care" of different types of edges separately based on the degrees of their endpoints. In Section 2.2-2.3 we provide constructions that take care of each type of edges; the analysis of stretch, probe complexity and spanner size for each case is included in their respective sections. We establish our final LCA for 3-spanners in Section 2.4.

2.1 Edge classification

▶ **Definition 5** (Subgraphs taking care of edges)**.** For stretch parameter k and set of edges $E' \subseteq E$, we say that the subgraph $H' \subseteq G$ *takes care* of E' if for every $(u, v) \in E'$, $\mathsf{dist}(u, v, H') \leq k$.

Observe that if we have a collection of subgraphs H_i's such that every edge in $(u, v) \in E$ is taken care by at least one H_i, then the union H of the H_i's constitutes a k-spanner for G.

▶ **Observation 6** (Spanner construction by combining subgraphs). *For a collection of subsets $E_1, \ldots, E_\ell \subseteq E$ where $\cup_{i\in[\ell]} E_i = E$, if H_i is a subgraph of G that takes care of E_i, then $H = \cup_{i\in[\ell]} H_i$ is a k-spanner of G. Further, if we have an LCA $\mathcal{A}_i$ for computing each H_i (i.e., deciding whether the query edge $(u, v) \in H_i$ and reporting* YES *or* NO *accordingly), we may construct a final LCA that runs every $\mathcal{A}_i$ and answer* YES *precisely when at least one of them does so. The performance of our overall LCA (number of edges, probes, or random bits) can then be bounded by the respective sum over that of $\mathcal{A}_i$'s.*

Note that H_i may contain edges of E that are not in E_i, thus it is necessary that the overall LCA invokes every $\mathcal{A}_i$ even if $\mathcal{A}_i$ does not take care of the query edge.

Graph partitioning. A vertex v is low-degree if $\deg(v) \leq \sqrt{n}$, it is high-degree if $\deg(v) \geq \sqrt{n}$ and it is super-high degree if $\deg(v) \geq n^{3/4}$. Our LCA for 3-spanner assigns each edge of E into one or more of the subsets E_{low}, E_{high}, or E_{super} based on the degrees of its endpoints, where

$$E_{\text{low}} = \{(u, v) \in E \mid \min\{\deg(u), \deg(v)\} \leq \sqrt{n}\},$$

$$E_{\text{high}} = \{(u, v) \in E \mid \sqrt{n} < \min\{\deg(u), \deg(v)\} \leq n^{3/4}\}, \text{ and } E_{\text{super}} = E \setminus (E_{\text{low}} \cup E_{\text{high}}).$$

Because vertices of degree at most $\sqrt{n}$ have $O(n \cdot \sqrt{n}) = O(n^{3/2})$ incident edges in total, we may afford to keep all these edges, letting $H_{\text{low}} = (V, E_{\text{low}})$. Thus, an LCA simply needs to check the degrees of both endpoints (via DEGREE probes), and answer YES precisely when both (or in fact, even one) have degrees at most $\sqrt{n}$. From now on, assume that $\deg(u), \deg(v) \geq \sqrt{n}$.

2.2 3-spanner for the edges E_{high}

We pick a random *center* set S of size $O(\sqrt{n} \log n)$ by sampling vertex $v \in V$ into S independently with probability $p = \Theta((\log n)/\sqrt{n})$. For now, we assume that given an ID of a vertex v, we can decide in $O(1)$ time if $v \in S$. At the end of the section, we describe how to implement this using a seed of $O(\log n)$ random bits. For each endpoint v of E_{high}, let $S(v) = \Gamma'(v) \cap S$ where $\Gamma'(v)$ is the set of the first $\sqrt{n}$ neighbors of v in $\Gamma(v)$. By Chernoff bound we have that $|S(v)| = \Theta(\log n)$ (and in particular, $S(v)$ is non-empty). We call $S(v)$ the *multiple-center* set of v. The algorithm adds to H_{high} the edges (v, s) connecting v to each of its centers $s \in S(v)$. This adds a total of $O(n \log n)$ edges.

Next, for every v with $\deg(v) = O(n^{3/4})$, the algorithm traverses its neighbor list $\Gamma(v) = \{u_1, \ldots, u_\ell\}$ and adds the edges $(u_i, v) \in E_{\text{high}}$ to the spanner H_{high} only if u_i belongs to a *new* cluster; i.e., u_i has a center $s \in S(u_i)$ that no previous neighbor u_j, $j < i$, has as its center in $S(u_j)$. Since the algorithm adds an edge whenever a new center is revealed and there are $O(\sqrt{n} \log n)$ centers, the total number of edges added to the spanner is $O(n^{3/2} \log n)$.

We next describe the LCA that, given an edge $(u, v) \in E_{\text{high}}$, says YES iff $(u, v) \in H_{\text{high}}$. We assume throughout that $\deg(v) \leq \deg(u)$, so $\deg(v) = O(n^{3/4})$. First, by probing for the first $\sqrt{n}$ neighbors of u and v, one can compute the center-sets $S(u)$ and $S(v)$ each containing $O(\log n)$ centers in S. Next, the algorithm probes for all of v's neighbors $\Gamma(v) = \{u_1, \ldots, u_j = u, \ldots, u_\ell\}$. For every neighbor u_i appearing before u in $\Gamma(v)$, i.e., for every $i < j$, and for every center $s \in S(u)$, the algorithm makes a *cluster-membership test* for s and u_i. This cluster-membership test can be answered by making a single ADJACENCY probe on the pair $\langle u_i, s\rangle$, namely $s \in S(u_i)$ only if s is among the first $\sqrt{n}$ neighbors of u_i. Eventually, the algorithm $\mathcal{A}_{\text{high}}$ answers YES only if there exists $s' \in S(u)$ such that $s' \notin \bigcup_{i=1}^{j-1} S(u_i)$. It is straightforwards to verify that the probe complexity is $\widetilde{O}(\deg(u) + \sqrt{n}) = O(n^{3/4})$.

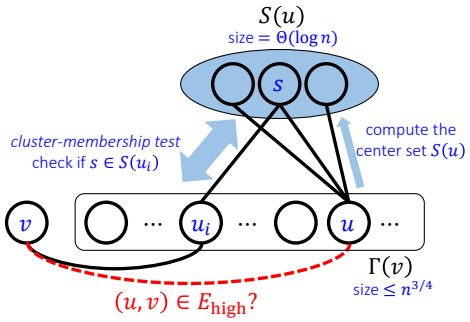

Figure 1 Illustration for the local construction of H_{high}.

Finally, we show that H_{high} is indeed a 3-spanner. For every edge (u, v) not added to the spanner, let $s \in S(u)$ and let u_i be the first vertex in $\Gamma(v)$ satisfying $s \in S(u_i)$. By construction, $(u_i, v) \in H_{\text{high}}$ and also the edges (u_i, s) and (u, s) are in the spanner H_{high}, providing a path of length 3 in H_{high}.

2.3 3-spanner for the edges E_{super}

We proceed by describing the construction of the 3-spanner H_{super} that takes care of the edges E_{super}. Let S' be a collection of $O(n^{1/4} \log n)$ centers obtained by sampling each $v \in V$ independently with probability $p' = \Theta((\log n)/n^{3/4})$. For each vertex v, define its center set $S'(v)$ to be the members of S' among the first $n^{3/4}$ neighbors of v, and if $\deg(v) \leq n^{3/4}$, then $S'(v) = S' \cap \Gamma(v)$. First, as in the construction of H_{high}, the algorithm connects each v to each of its centers by adding the edges (u, s) for every u and $s \in S'(u)$ to the spanner H_{super}.

Consider a vertex v and divide its neighbor list into consecutive *blocks* $\Gamma_1(v), \ldots, \Gamma_\ell(v)$, each of size $n^{3/4}$ (expect perhaps for the last block). In every block $\Gamma_i(v) = \{u_{i,1}, \ldots, u_{i,\ell'}\}$, the algorithm adds the edge $(v, u_{i,j})$ to the spanner H_{super} only if $u_{i,j}$ belongs to a *new* cluster with respect to all other vertices that appear before it in that block. Formally, the edge $(v, u_{i,j})$ is added iff there exists $s \in S'(u_{i,j})$ such that $s \notin \bigcup_{q \leq j-1} S'(u_{i,q})$. This completes the description of the construction. Observe that within each block, the LCA adds an edge for each new center. W.h.p., there are $O(n/n^{3/4}) = O(n^{1/4})$ blocks and $|S'| = O(n^{1/4} \log n)$ centers, so $O(\sqrt{n} \log n)$ edges are added for each v, yielding a spanner of size $O(n^{3/2} \log n)$.

The LCA $\mathcal{A}_{\text{super}}$ is very similar to $\mathcal{A}_{\text{high}}$: the main distinction is that given an edge (u, v) with $\deg(u) \geq n^{3/4}$, the algorithm $\mathcal{A}_{\text{super}}$ will probe only for the block $\Gamma_i(v) = \{u_{i,1}, \ldots, u_{i,j} = u, u_{i,\ell'}\}$ to which v belongs, ad will make its decision only based on that block. By probing for the degree of v, and the index j such that u is the j^{th} neighbor of v, one can compute the block $\Gamma_i(v)$ by making $n^{3/4}$ NEIGHBOR probes. In addition, by probing for the first $n^{3/4}$ neighbors of both u and v, one can compute the multiple-center sets $S'(u)$ and $S'(v)$. Finally, the algorithm applies a cluster-membership test for each pair $s \in S'(u)$ and $u_{i,q}$ for $q \leq j - 1$. It returns `YES` only if there exists $s \notin \bigcup_{q \leq j-1} S'(u_{i,q})$. Hence, the number of probes made by the LCA is w.h.p. bounded by $|\Gamma_i(v)| \cdot |S'(u)| = O(n^{3/4} \log n)$.

We now show that H_{super} is a 3-spanner for the edges H_{super}. Let (u, v) be such that $\deg(u) \geq n^{3/4}$ and let $\Gamma_i(v)$ be the block in $\Gamma(v)$ to which u belongs. Since $\deg(u) \geq n^{3/4}$, w.h.p. $|S'(u)| = \Theta(\log n)$. Assume that $(u, v) \notin H_{\text{super}}$. Fix $s \in S'(u)$ and let $u_{i,q}$ be the first vertex in $\Gamma_i(v)$ that belongs to the cluster of s. Since $(u, v) \notin H_{\text{super}}$, such a vertex $u_{i,q}$ is guaranteed to exist. The spanner H_{super} contains the edges $(s, u), (s, u_{i,q})$ and $(v, u_{i,q})$, thus containing a path of length 3 between u and v.

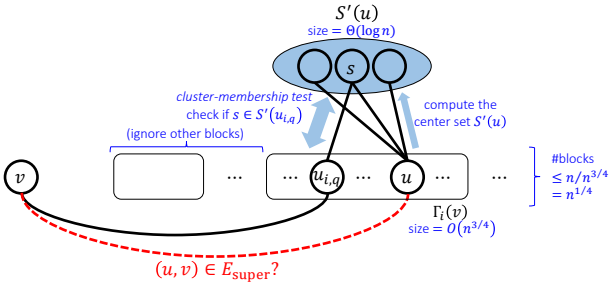

Figure 2 Illustration for the local construction of H_{super}.

2.4 The Final LCA

Given an edge (u, v) the algorithm says YES if one of the following holds:

- $\deg(u), \deg(v) \leq \sqrt{n}$.
- $u \in S(v) \cup S'(v)$ (or vice versa).
- the local algorithm $\mathcal{A}_{\text{high}}$ says YES on edge (u, v).
- the local algorithm $\mathcal{A}_{\text{super}}$ says YES on edge (u, v).

This completes the 3-spanner LCA from Theorem 1.

Missing piece: computing centers in the LCA model. In the LCA model, we do not generate the entire set S (or S') up front. Instead, we may verify whether $v \in S$ on-the-fly using v's ID by, e.g., applying a random map (chosen according to the given random tape) from v's ID to $\{0, 1\}$ with expectation p. In fact, this hitting set argument does not require full independence – the discussion on reducing the amount of random bits is given in Section 4, but for now we formalize it as the following observation.

▶ **Observation 7** (Local Computation of Centers). *Let S be a center set obtained by placing each vertex into S independently with probability $p = \Theta(\log n/\Delta)$. W.h.p., S forms a hitting set for the collection of neighbor sets of all vertices of degree at least Δ. Further, under the LCA model, we may check whether $v \in S$ locally without making any probes.*

3 LCA for 5-Spanners

We now consider LCAs for 5-spanners, aiming for spanners of size $\widetilde{O}(n^{4/3})$ with probe complexity $\widetilde{O}(n^{5/6})$. We start by noting that the construction of H_{super} for the 3-spanners in fact gives for every $r \geq 1$, a 3-spanner of size $\widetilde{O}(n^{1+1/r})$ for the subset of edges (u, v) with $\min\{\deg(u), \deg(v)\} \geq n^{1-1/(2r)}$: this is achieved by instead setting the threshold for super-high degree at $n^{1-1/(2r)}$, pick $|S'| = \widetilde{O}(n^{1/(2r)})$ centers, and use block size $n^{1-1/(2r)}$. The probe complexity for querying the spanner is $\widetilde{O}(n^{1-1/(2r)})$. For 5-spanner, by taking $r = 3$, one takes care of all edges (u, v) with $\max\{\deg(u), \deg(v)\} \geq n^{5/6}$.

Let $\Delta_{\text{low}} = n^{1/r}$, $\Delta_{\text{med}} = n^{1/2-1/(2r)}$ and $\Delta_{\text{super}} = n^{1-1/(2r)}$. For the purpose of constructing 5-spanners for general graphs, we let $r = 3$, simplifying the thresholds to $\Delta_{\text{low}} = \Delta_{\text{med}} = n^{1/3}$ and $\Delta_{\text{super}} = n^{5/6}$.) Again, we may afford to keep all edges incident to some vertex of degree at most Δ_{low}.

For integers $a \leq b$, let $V_{[a,b]} = \{v \in V(G) \mid \deg(v) \in [a, b]\}$. We will design a subgraph $H \subseteq G$ that will take care of the remaining edges $E_{\text{med}} = E(V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}, V_{[\Delta_{\text{med}},\Delta_{\text{super}}]})$.

Table 2 Edge categorization for the construction of 5-spanners.

Subset	Criteria	# Edges	Probe Complexity
E_{low}	$(u,v) \in E(V, V_{[1,\Delta_{\text{low}}]})$	$O(n \cdot \Delta_{\text{low}}) = O(n^{1+\frac{1}{r}})$	$O(1)$
E_{bckt}	$(u,v) \in E(V_{\text{dsrt}}, V_{\text{dsrt}})$	$O(\frac{n^2 \log^2 n}{\Delta_{\text{med}}^2}) = O(n^{1+\frac{1}{r}} \log^2 n)$	$O((\Delta_{\text{super}} + \Delta_{\text{med}}^2) \log^2 n) = O(n^{1-\frac{1}{2r}} \log^2 n)$
E_{rep}	$(u,v) \in E(V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}, V_{\text{crwd}})$	$O(\frac{n^2}{\Delta_{\text{super}}} \cdot \log n) = O(n^{1+\frac{1}{r}} \log n)$	$O(\Delta_{\text{super}} \log^3 n) = O(n^{1-\frac{1}{2r}} \log^3 n)$
E_{super}	$(u,v) \in E(V, V_{[\Delta_{\text{super}},n)})$	$O(\frac{n^3 \log n}{\Delta_{\text{super}}^2}) = O(n^{1+\frac{1}{r}} \log n)$	$O(\Delta_{\text{super}} \log n) = O(n^{1-\frac{1}{2r}} \log n)$

▶ **Definition 8** (Deserted and Crowded vertices)**.** A vertex $v \in V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$ is *deserted* if at least half of its neighbors in $\Gamma_{\Delta_{\text{med}},1}(v)$ are of degree at most Δ_{super}; i.e., $|\Gamma_{\Delta_{\text{med}},1}(v) \cap V_{[1,\Delta_{\text{super}}]}| \geq \Delta_{\text{med}}/2$. Otherwise, the vertex is *crowded*.

Criteria for edges. We aim to take care of edges for which both endpoints are in $V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$. To categorize our edges for the purpose of constructing 5-spanners, we need the following partition of these vertices.

Let V_{dsrt} (resp., V_{crwd}) be the set of deserted (resp., crowded) vertices in $V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$. Given a vertex, we can verify whether it is in any of these sets using $O(\Delta_{\text{med}})$ probes by checking the degrees of v and each vertex in $\Gamma_{\Delta_{\text{med}},1}(v)$. We then assign each $(u,v) \in E$ into one of the four cases $\{\text{low}, \text{bckt}, \text{rep}, \text{super}\}$ as given in Table 2 It is straightforward to verify that when $\Delta_{\text{low}} = \Delta_{\text{med}}$ (namely when we choose $r = 3$, which also yields the required performance), these four cases take care of all edges in E. We note that H_{rep} assumes that H_{super} is included: E_{rep} is taken care by $H_{\text{rep}} \cup H_{\text{super}}$, not by H_{rep} alone.

LCA for E_{bckt}: the cluster partitioning method. The algorithm is as follows.

- Only vertices of degree at most Δ_{super} are chosen to be in S with probability $p = \Theta((\log n)/\Delta_{\text{med}})$. Since at least half the vertices in $\Gamma_{\Delta_{\text{med}},1}(v)$ for any $v \in V_{\text{dsrt}}$ have degree smaller than Δ_{super}, we have that w.h.p. $|S(v)| = \Theta(\log n)$ the cluster-membership test can be done with constant number of probes. Let us denote by $\mathsf{C}(s) = \{s\} \cup \{v : s \in S(v)\}$ the cluster of center s.
- The partitioning of clusters into buckets is defined in a consistent way (regardless of the given query edge); for instance, create a list of vertices in the cluster, sort them according to their IDs, divide the list into buckets of size Δ_{med} possibly except for the last one. Note that we partition $\mathsf{C}(s)$ and $\mathsf{C}(t)$ separately – we do not combine their elements. Similarly, once we obtain buckets containing u and v, the order in which we check the adjacency of u' and v' must be consistent. To this end, define the ID of an edge (u,v) as $(\mathsf{ID}(u),\mathsf{ID}(v))$, where the comparison between edge IDs is lexicographic. Thus, this step only adds the edge of minimum ID between the two clusters.
- We also set the precondition $(u,v) \in E(V_{[\Delta_{\text{med}},n)}, V_{[\Delta_{\text{med}},n)})$, and consistently only allow candidate pairs $(u',v') \in E(V_{[\Delta_{\text{med}},n)}, V_{[\Delta_{\text{med}},n)})$, to ensure that the lexicographically first edge of this exact specification is added if one exists. We do not restrict to E_{bckt}, which require both endpoints to be deserted vertices, because checking whether $(u',v') \in E_{\text{bckt}}$ would take $\Theta(\Delta_{\text{med}})$ probes instead of constant probes. We restrict to edges whose endpoints have degrees at least Δ_{med} instead of considering the entire E so that S would be well-defined.

Local construction of $H_{\mathbf{bckt}}$. Each $v \in V_{[1,\Delta_{\text{super}}]}$ is added to S with probability $p = \Theta(\log n/\Delta_{\text{med}})$.

(A) If $u \in S(v)$ or $v \in S(u)$, answer YES.

(B) If $(u,v) \in E(V_{[\Delta_{\text{med}},n)}, V_{[\Delta_{\text{med}},n)})$:

- Compute $S(u)$ and $S(v)$ by iterating through $\Gamma_{\Delta_{\text{med}},1}(u)$ and $\Gamma_{\Delta_{\text{med}},1}(v)$.
- For each pair of $s \in S(u)$ and $t \in S(v)$:
  - Partition each of the clusters $\mathsf{C}(s)$ and $\mathsf{C}(t)$ into buckets of size (mostly) Δ_{med}. Denote the buckets containing u and v by $\mathsf{Bucket}(u,s)$ and $\mathsf{Bucket}(v,t)$, respectively.
  - Iterate through each pair of $u' \in \mathsf{Bucket}(u,s)$ and $v' \in \mathsf{Bucket}(v,t)$ and check if $(u',v') \in E(V_{[\Delta_{\text{med}},n)}, V_{[\Delta_{\text{med}},n)})$. Answer YES if the edge of minimum ID found is $(u',v') = (u,v)$.

▶ **Lemma 9.** *For $1 \le \Delta_{\text{med}} \le \sqrt{n} \le \Delta_{\text{super}} \le n$, there exists a subgraph $H_{\text{bckt}} \subseteq G$ such that w.h.p.:*

(i) *H_{bckt} has $O(\frac{n^2 \log^2 n}{\Delta_{\text{med}}^2})$ edges,*

(ii) *H_{bckt} takes care of E_{bckt}; that is, for every $(u,v) \in H_{\text{bckt}}$, $\mathsf{dist}(u,v,H_{\text{bckt}}) \le 5$, and*

(iii) *for a given edge $(u,v) \in E$, one can test if $(u,v) \in H_{\text{bckt}}$ by making $O((\Delta_{\text{super}} + \Delta_{\text{med}}^2)\log^2 n)$ probes.*

Proof.

(i) Size. In (A) we add $|S(v)| = \Theta(\log n)$ edges for each $v \in V_{\text{dsrt}}$, which constitutes to $O(n \log n)$ edges in total. In (B), we add one edge between each pair of buckets. We now compute the total number of buckets. The total size of clusters $\sum_{s \in S} |\mathsf{C}(s)| \le |S| + \sum_{v \in V_{[\Delta_{\text{med}},n)}} |S(v)| = O(n \log n)$, so there can be up to $O((n \log n)/\Delta_{\text{med}})$ full buckets of size Δ_{med}. As buckets are formed by partitioning $|S|$ clusters, there are up to $|S| = \Theta((n \log n)/\Delta_{\text{med}})$ remainder buckets of size less than Δ_{med}. Thus, there are $\Theta((n \log n)/\Delta_{\text{med}})$ buckets, and $O(((n \log n)/\Delta_{\text{med}})^2)$ edges are added in (B).

(ii) Stretch. Suppose that (u,v) is omitted. Fix centers $s \in S(u)$ and $t \in S(v)$, then the lexicographically-first edge $(u',v') \in E(\mathsf{Bucket}(u,s), \mathsf{Bucket}(v,t))$ must have been added to H_{bckt}, forming the path $\langle u,s,u',v',t,v\rangle$ (or shorter, if there are repeated vertices), yielding $\mathsf{dist}(u,v,H_{\text{bckt}}) \le 5$.

(iii) Probes. Computing $S(u)$ and $S(v)$ takes $O(\Delta_{\text{med}})$ probes. For each pairs of centers, we scan through the entire neighbor-lists $\Gamma(s)$ and $\Gamma(t)$ and collect all vertices in their respective clusters. This takes $O(\Delta_{\text{super}})$ probes each because we restrict to centers of degree at most Δ_{super}. Given the clusters, we identify the buckets containing u and v each of size $O(\Delta_{\text{med}})$. We then check through candidates (u',v') between these buckets, taking $O(\Delta_{\text{med}}^2)$ ADJACENCY probes. So, each pair of centers requires $O(\Delta_{\text{super}} + \Delta_{\text{med}}^2)$ total probes. We repeat the process for $|S(u)| \cdot |S(v)| = O(\log^2 n)$ pairs of centers w.h.p., yielding the claimed probe complexity. ◀

LCA for E_{rep}: the Representative method. We first explain the computation of the representative set $\mathsf{Reps}(v)$ for a croweded vertex $v \in V_{\text{crwd}}$, i.e., a collection of neighbors of v that have degree at least $n^{5/6}$. Using the random bits and the vertex ID, we sample a set R_v of $\Theta(\log n)$ (not necessarily distinct) indices in $[\Delta_{\text{med}}]$ at random (for details, see Sec. 4). Denote the neighbor-list of v by $\{x'_1, \ldots, x'_{\deg(v)}\}$, then define $\mathsf{Reps}(v) = \{x'_i : i \in R_v \text{ and } \deg(x'_i) \ge \Delta_{\text{super}}\}$. Then since at least half of the vertices in $\Gamma_{\Delta_{\text{med}},1}(v)$ are of degree

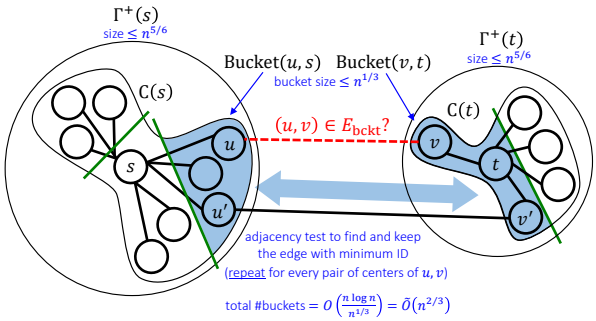

Figure 3 Illustration for the local construction of H_{bckt}. Green lines show the partition of clusters into buckets.

at least Δ_{super}, w.h.p. $\mathsf{Reps}(v) \neq \emptyset$. For consistency, we allow the same definition for $\mathsf{Reps}(v)$ for any $v \in V_{[\Delta_{\text{med}},n)}$ as well, even if it may result in empty sets of representatives. Hence computing $\mathsf{Reps}(v)$ takes $O(\log n)$ probes[12].

Let $E_{\text{super}} = \{(u,v) \in E \mid \max\{\deg(u), \deg(v)\} \geq n^{5/6}\}$ and apply the 3-spanner algorithm algorithm of Sec. 2 to construct a subgraph H_{super} that takes care of the edges E_{super}. To construct H_{super} the algorithm (fully described[13] in Sec. 2) samples a set S' of centers by picking each $v \in V$ independently with probability $O(\log n/n^{5/6})$. For every v with $\deg(v) \geq n^{5/6}$, let $S'(v)$ be the sampled neighbors in $S' \cap \Gamma_1(v)$ where $\Gamma_1(v)$ is the first block of size $n^{5/6}$ in $\Gamma(v)$. This allows us to check membership to a cluster of $s \in S'$ using a single adjacency probe. The idea would be to extend the 1-radius clusters of S' by one additional layer consisting of the crowded vertices connected to the cluster via their representatives.

For convenience, for a crowded v, define $RS(v) = \cup_{x' \in \mathsf{Reps}(v)} S'(x')$, the set of (multiple) centers of any of v's representatives. Observe that by adding the edge (v,x') to H_{rep} for every $x' \in \mathsf{Reps}(v)$, it yields that $\mathsf{dist}(v,s,H_{\text{rep}} \cup H_{\text{super}}) \leq 2$ for any $s \in RS(v)$.

Consider the query (u,v), and suppose that $v = v'_i$ is the i^{th} neighbor in u's neighbor-list, $\Gamma(u) = \{v'_1, \ldots, v'_{\deg(u)}\}$. We then add (u,v) to H_{rep} if and only if v introduces a new center through some representative; that is, $RS(v'_i) \setminus \cup_{j<i} RS(v'_j) \neq \emptyset$. To verify this condition locally, we first compute $RS(v)$, and for each of $\{v'_j\}_{j<i}$, $\mathsf{Reps}(v'_j)$. Then, we discard (u,v) if for every center $s \in RS(v)$, there exists x and v'_j where $x \in \mathsf{Reps}(v'_j)$ and $s \in S'(x)$; the last condition takes constant probes to verify. This gives the full LCA for constructing H_{rep} below.

Local construction of H_{rep}. Each $v \in V$ is added to S' with probability $p = \Theta((\log n)/\Delta_{\text{super}})$.

(A) If $v \in V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$ and $u \in \mathsf{Reps}(v)$, answer `YES`.

(B) If $u,v \in V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$:

- Compute $RS(v)$.
- Denote the neighbor-list of u by $\{v'_1, \ldots, v'_{\deg(u)}\}$; identify i such that $v = v'_i$.
- For each vertex $w \in \{v'_1, \ldots, v'_{i-1}\}$, if $w \in V_{[\Delta_{\text{med}},\Delta_{\text{super}}]}$, compute $\mathsf{Reps}(w)$.
- For each $s \in RS(v)$, iterate to check for a vertex x in any of the $\mathsf{Reps}(w)$'s obtained above, such that $s \in S'(x)$. Answer `YES` if there exists a vertex s where no such x exists.

[12] The naïve solution traverses the entire Δ_{med} first neighbors of v which is too costly.
[13] Upon replacing the degree threshold of $n^{3/4}$ with $n^{5/6}$.

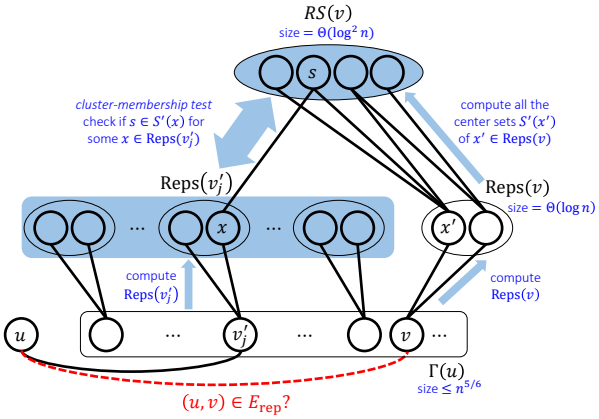

Figure 4 Illustration for the local construction of H_{rep}.

▶ **Lemma 10.** *For $1 \leq \Delta_{\text{med}} \leq \Delta_{\text{super}} \leq n$, there exists a subgraph $H_{\text{rep}} \subseteq G$ such that w.h.p.:*

- **(i)** *H_{rep} has $O(n^2/\Delta_{\text{super}} \cdot \log n)$ edges,*
- **(ii)** *$H_{\text{rep}} \cup H_{\text{super}}$ takes care of E_{rep}; that is, for every $(u, v) \in E_{\text{rep}}$, $\mathsf{dist}(u, v, H_{\text{rep}} \cup H_{\text{super}}) \leq 3$, and*
- **(iii)** *for a given edge $(u, v) \in E$, one can test if $(u, v) \in H_{\text{rep}}$ by making $O(\Delta_{\text{super}} \log^3 n)$ probes.*

Proof.

(i) Size. W.h.p., in (A) we add at most $\sum_{v \in V_{[\Delta_{\text{med}}, \Delta_{\text{super}}]}} |\mathsf{Reps}(v)| \leq n \cdot O(\log n) = O(n \log n)$. Similarly to the analysis of H_{high}, in (B) we add $|S'| = O((n \log n)/\Delta_{\text{super}})$ edges per vertex u, so $|E(H_{\text{rep}})| = O(n^2/\Delta_{\text{super}} \cdot \log n)$.

(ii) Stretch. This claim follows from the argument given in the overview, and is similar to the analysis of H_{high}.

(iii) Probes. Computing $RS(v)$ takes $O(\log n) \cdot \Delta_{\text{super}} = O(\Delta_{\text{super}} \log n)$ (recall that we only check $\Gamma_{\Delta_{\text{super}},1}$ of each reprsentative). Note also that $|RS(v)| = O(\log^2 n)$ since v has $O(\log n)$ representative, each of which belongs to $\Theta(\log n)$ clusters. Computing Reps for each neighbor $w \in \{v'_j\}_{j<i}$ of u takes $O(\log n)$ probes each, which is $O(\Delta_{\text{super}} \log n)$ in total since $\deg(u) \leq \Delta_{\text{super}}$. This also introduces up to $\Delta_{\text{super}} \cdot O(\log n)$ representatives in total. Checking whether each of the $O(\log^2 n)$ centers in $RS(v)$ is a center of each of these $O(\Delta_{\text{super}} \log n)$ representative takes, in total w.h.p., $O(\Delta_{\text{super}} \log^3 n)$ probes. ◀

Final 5-spanner results. To obtain an LCA for 5-spanners, we again invoke all of our LCAs for the four cases. Applying Lemma 9 and 10, we obtain the following LCA result for 5-spanner in general graphs.

▶ **Theorem 11.** *For every n-vertex simple undirected graph $G = (V, E)$ there exists an LCA for 5-spanner with $O(n^{4/3} \log^2 n)$ edges and probe complexity $O(n^{5/6} \log^3 n)$.*

Again, by combining results for larger degrees, we obtain an LCA for 5-spanners with smaller sizes on graphs with minimum degree at least $n^{1/2-1/(2r)}$.

▶ **Theorem 12.** *For every $r \geq 1$ and n-vertex simple undirected graph $G = (V, E)$ with minimum degree at least $n^{1/2-1/(2r)}$, there exists a (randomized) LCA for 5-spanner with $O(n^{1+1/r} \log^2 n)$ edges and probe complexity of $O(n^{1-1/(2r)} \log^3 n)$.*

4 Bounded Independence

In this section, we show that all our LCA constructions succeed w.h.p. using $\Theta(\log n)$-*wise independent hash functions* which only require $\Theta(\log^2 n)$ random bits. We use the following standard notion of d-wise independent hash functions as in [38]. In particular, our algorithms use the explicit construction of $\mathcal{H}$ by [38], with the parameters as stated in Lemma 14.

▶ **Definition 13.** For $N, M, d \in \mathbb{N}$ such that $d \leq N$, a family of functions $\mathcal{H} = \{h : [N] \to [M]\}$ is d-wise independent if for all distinct $x_1, ..., x_d \in [N]$, the random variables $h(x_1), ..., h(x_d)$ are independent and uniformly distributed in $[M]$ when h is chosen randomly from $\mathcal{H}$.

▶ **Lemma 14** (Corollary 3.34 in [38]). *For every $\gamma, \beta, d \in \mathbb{N}$, there is a family of d-wise independent functions $\mathcal{H}_{\gamma,\beta} = \left\{h : \{0,1\}^{\gamma} \to \{0,1\}^{\beta}\right\}$ such that choosing a random function from $\mathcal{H}_{\gamma,\beta}$ takes $d \cdot \max\{\gamma, \beta\}$ random bits, and evaluating a function from $\mathcal{H}_{\gamma,\beta}$ takes time* $\text{poly}(\gamma, \beta, d)$.

Then, we exploit the following result to show the concentration of d-wise independent random variables:

▶ **Fact 15** (Theorem 5(III) in [36]). *If X is a sum of d-wise independent random variables, each of which is in the interval $[0, 1]$ with $\mu = \mathbb{E}(X)$, then:*

- **(I)** *For $\delta \leq 1$ and $d \leq \lfloor \delta^2 \mu e^{-1/3} \rfloor$, it holds that $\Pr[|X - \mu| \geq \delta\mu] \leq e^{-\lfloor d/2 \rfloor}$.*
- **(II)** *For $\delta \geq 1$ and $d = \lceil \delta\mu \rceil$, it holds that: $\Pr[|X - \mu| \geq \delta\mu] \leq e^{-\delta\mu/3}$.*

Bounded independence for hitting set procedures. Most of our algorithms are based on the following hitting set procedure. For a given threshold $\Delta \in [1, n]$, each vertex flips a coin with probability $p = (c \log n)/\Delta$ of being head and the set of all vertices with head outcome join the set of centers S. Assuming the outcome of coin flips are fully independent, by the Chernoff bound, the followings hold w.h.p.:

(HI) There are $\Theta(pn)$ sampled vertices S.

(HII) For each vertex of degree at least Δ, it has $\Theta(\log n)$ centers among its first Δ neighbors.

Here we show that to satisfy properties (HI) and (HII), it is sufficient to assume that the outcomes of the coin flips are d-wise independent. By Lemma 14, to simulate d-wise independent coin flips for all vertices, the algorithm only requires $t = \Theta(d(\log n + \log 1/p))$ random bits: more precisely, setting $\gamma = \Theta(\log n)$ and $\beta = \log 1/p$ (for simplicity, lets assume that $\log 1/p$ is an integer), there exits a family of d-wise independent functions $\mathcal{H} = \left\{h : \{0,1\}^{\Theta(\log n)} \to \{0,1\}^{\log(1/p)}\right\}$ such that a random function $h \in \mathcal{H}$ can be specified by a string of random bits of length t. In other words, each function $h \in \mathcal{H}$ maps the ID of each vertex to the outcome of its coin flip according to a coin with bias p. Then, from a string $\mathcal{R}$ of t random bits, the algorithm picks a function $h_{\mathcal{R}} \in \mathcal{H}$ at random to simulate the coin flips of the vertices accordingly: the outcome of the coin flip of v is head if $h_{\mathcal{R}}(\mathsf{ID}(v)) = 0$ (which happens with probability p) and the coin flips are d-wise independent. Setting $d = c \log n$ for some constant $c > 1$, we prove the following:

▶ **Claim 16.** *If the coin flips are d-wise independent then properties (HI) and (HII) holds. Furthermore, the sequence of n d-wise independent coin flips can be simulated using a string of $O(\log^2 n)$ random bits.*

Construction of representatives in Section 3. The analysis above also extends to the process of computing Reps. Each crowded vertex chooses values $c \log n$ random indices (of its neighbor-list) in $[\Delta_{\text{med}}]$, each of which has probability $1/2$ of hitting a neighbor of degree at least Δ_{super}. Let $\{Z_i\}_{i \in [c \log n]}$ be indicators for these events and Z denote their sum, then the expected sum $\mathbb{E}(Z) \geq (c/2) \log n$. Imposing d-wise independence, Fact 15(I) implies that w.h.p., $Z > 0$, so the representative set is non-empty. We apply the union bound to show that $\mathsf{Reps}(v) \neq \emptyset$ for every $v \in V_{\text{crwd}}$, as desired.

References

1. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 23rd ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1132–1139, 2012.
2. Baruch Awerbuch and David Peleg. Network synchronization with polylogarithmic overhead. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 514–522, 1990.
3. Baruch Awerbuch and David Peleg. Routing with polynomial communication-space tradeoff. *SIAM J. Discrete Math.*, 5(2):151–162, 1992.
4. Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms (TALG)*, 8(4):35, 2012.
5. Surender Baswana and Sandeep Sen. A Simple and Linear Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs. *Random Structures and Algorithms*, 30(4):532–563, 2007.
6. Greg Bodwin and Sebastian Krinninger. Fully Dynamic Spanners with Worst-Case Update Time. In *Proc. 24th Annu. European Sympos. Algorithms* (ESA), pages 17:1–17:18, 2016.
7. Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing Local Distributed Algorithms under Bandwidth Restrictions. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 11:1–11:16, 2017.
8. Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. *Theoretical Computer Science*, 2008.
9. Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In *Proc. 21st Int. Symp. Dist. Comp.* (DISC), pages 179–192, 2007.
10. Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 273–282, 2008.
11. Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. Local computation of nearly additive spanners. In *Proc. 23rd Int. Symp. Dist. Comp.* (DISC), 2009.
12. Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms (TALG)*, 7(2):20, 2011.
13. Michael Elkin and Ofer Neiman. Efficient Algorithms for Constructing Very Sparse Spanners and Emulators. In *Proc. 28th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 652–669, 2017.
14. P Erdös. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3(2):113–116, 1965.
15. Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Proc. 22nd Annu. European Sympos. Algorithms* (ESA), pages 394–405, 2014.

16 Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. *Proc. 30th ACM-SIAM Sympos. Discrete Algs.* (SODA), 2019.

17 Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 465–469. Springer, 2011.

18 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

19 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 393–398, 2012.

20 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004.

21 Christoph Lenzen and Reut Levi. A Centralized Local Algorithm for the Sparse Spanning Graph Problem. In *Proc. 45th Int. Colloq. Automata Lang. Prog.* (ICALP), pages 87:1–87:14, 2018.

22 Reut Levi, Guy Moshkovitz, Dana Ron, Ronitt Rubinfeld, and Asaf Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, 50(2):183–200, 2017.

23 Reut Levi and Dana Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Transactions on Algorithms (TALG)*, 11(3):24, 2015.

24 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local Algorithms for Sparse Spanning Graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 826–842, 2014.

25 Reut Levi, Dana Ron, and Ronitt Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. *arXiv preprint*, 2016. `arXiv:1604.07038`.

26 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local Computation Algorithms for Graphs of Non-constant Degrees. *Algorithmica*, 77(4):971–994, 2017.

27 Yishay Mansour, Boaz Patt-Shamir, and Shai Vardi. Constant-time local computation algorithms. In *International Workshop on Approximation and Online Algorithms*, pages 110–121, 2015.

28 Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 653–664. Springer, 2012.

29 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273. Springer, 2013.

30 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. SIAM, 2000.

31 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

32 David Peleg and Jeffrey D Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on computing*, 18(4):740–747, 1989.

33 Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010.

34 Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Sciences*, 82(7):1180–1200, 2016.

35 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast Local Computation Algorithms. In *Innovations in Computer Science - ICS 2010*, pages 223–238, 2011.

36 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

37 Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

38 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

39 Rephael Wenger. Extremal graphs with no C4's, C6's, or C10's. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991.