Improved Local Computation Algorithms for Constructing Spanners

Rubi Arviv
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Lily Chung
MIT, Cambridge, MA, USA

Reut Levi
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Edward Pyne
MIT, Cambridge, MA, USA

Abstract

A spanner of a graph is a subgraph that preserves lengths of shortest paths up to a multiplicative distortion. For every k, a spanner with size $O(n^{1+1/k})$ and stretch $(2k+1)$ can be constructed by a simple centralized greedy algorithm, and this is tight assuming Erdős girth conjecture.

In this paper we study the problem of constructing spanners in a local manner, specifically in the Local Computation Model proposed by Rubinfeld et al. (ICS 2011).

We provide a randomized Local Computation Agorithm (LCA) for constructing $(2r-1)$-spanners with $\tilde{O}(n^{1+1/r})$ edges and probe complexity of $\tilde{O}(n^{1-1/r})$ for $r \in \{2,3\}$, where n denotes the number of vertices in the input graph. Up to polylogarithmic factors, in both cases, the stretch factor is optimal (for the respective number of edges). In addition, our probe complexity for $r=2$, i.e., for constructing a 3-spanner, is optimal up to polylogarithmic factors. Our result improves over the probe complexity of Parter et al. (ITCS 2019) that is $\tilde{O}(n^{1-1/2r})$ for $r \in \{2,3\}$. Both our algorithms and the algorithms of Parter et al. use a combination of neighbor-probes and pair-probes in the above-mentioned LCAs.

For general $k \geq 1$, we provide an LCA for constructing $O(k^2)$-spanners with $\tilde{O}(n^{1+1/k})$ edges using $O(n^{2/3}\Delta^2)$ neighbor-probes, improving over the $\tilde{O}(n^{2/3}\Delta^4)$ algorithm of Parter et al.

By developing a new randomized LCA for graph decomposition, we further improve the probe complexity of the latter task to be $O(n^{2/3-(1.5-\alpha)/k}\Delta^2)$, for any constant $\alpha > 0$. This latter LCA may be of independent interest.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Local Computation Algorithms, Spanners

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.42

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2105.04847v2`

Funding *Lily Chung*: Supported by an Akamai Presidental Fellowship.
Reut Levi: Supported by ISF Grant No. 1867/20
Edward Pyne: Supported by an Akamai Presidental Fellowship.

1 Introduction

A spanner is a sparse structure that is a subgraph of the input graph and preserves, up to a predetermined multiplicative factor, the pairwise distance of vertices. Formally, a k-spanner of a graph $G=(V,E)$ is a graph $G'=(V,E')$ such that $E' \subseteq E$, in which the distance between any pair of vertices in G' is at most k times longer than the corresponding distance in G. k is referred to as the *stretch* of the spanner.

Spanners have numerous applications in a wide variety of fields such as communication networks [4, 29, 30], biology [5] and robotics [11, 16]. Consequently, the problem of constructing spanners has been studied extensively in several models, such as the distributed model [6, 12, 13, 14, 15, 17, 31], streaming algorithms [1, 22] and dynamic algorithms [10, 9].

This problem was also considered in the realm of sublinear algorithms and in particular in the model of *Local computation algorithms* (LCAs) introduced by Rubinfeld et al. [32] (see also Alon et al. [2] and survey in [24]). In this model the goal is to avoid computing the entire output and instead to compute parts of the output on demand. This model is suitable for the case that not only the input is massive but also the output. Moreover, LCAs support queries from different users while preserving consistency with a single valid solution (although there might be several valid solutions) across different queries. The notion of computing the output locally goes back to local algorithms, locally decodable codes and local reconstruction algorithms. LCAs can be viewed as a generalization of these frameworks.

Recently, several works [26, 25, 23, 28] considered the problem of constructing spanners in the LCA model. The formulation of the problem in this model is as defined next.

▶ **Definition 1** ([2, 26]). *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.*

For specific details regarding the types of probes supported in the LCA model, we refer the reader to Section 2.

1.1 Our Results

We provide LCAs that with high probability construct the following spanners.

1. A 3-spanner with $\tilde{O}(n^{1+1/2})$ edges. The probe and time complexity of the algorithm is $\tilde{O}(n^{1/2})$ which is optimal up to polylogarithmic factors (and constitutes the first optimal algorithm for general graphs). The size-stretch trade-off is optimal as well (up to polylogarithmic factors). This improves over the algorithm of Parter et al. [28] whose probe and time complexity is $\tilde{O}(n^{3/4})$.
2. A 5-spanner with $\tilde{O}(n^{1+1/3})$ edges (the size-stretch trade-off is optimal up to polylogarithmic factors). The probe and time complexity of the algorithm is $\tilde{O}(n^{2/3})$. This improves over the algorithm of Parter et al. [28] whose probe and time complexity is $\tilde{O}(n^{5/6})$.
3. An $O(k^2)$-spanner with $\tilde{O}(n^{1+1/k})$ edges with high probability. The probe and time complexity of the algorithm is $O(n^{2/3}\Delta^2)$ where Δ denotes the maximum degree of the input graph. This improves over the algorithm of Parter et al. [28] whose probe and time complexity is $\tilde{O}(n^{2/3}\Delta^4)$. Our algorithm (and the algorithm of [28]) uses only neighbor probes for this task.
4. By additionally taking advantage of adjacency probes we further improve the probe and time complexity of the latter algorithm to be $O(n^{2/3-(1.5-\alpha)/k}\Delta^2)$, for any constant $\alpha > 0$. This result utilizes a new, efficient local computation algorithm for decomposing a graph into subgraphs with improved maximum degree that may be of independent interest.

1.2 Our algorithms and techniques

We next describe our algorithms in high-level. Our LCAs for constructing 3-spanners and 5-spanners share similarities with the LCAs in [28] (which are inspired by the algorithm of Baswana and Sen for constructing spanners [7]). The main novelty of our algorithms is in selecting several sets of centers, each designed to cluster different type of vertices. The basic idea is that for high-degree vertices we need to select less centers. Consequently, we can allow more edges per pair of vertex-cluster or cluster-cluster which decreases the probe complexity. To support this approach we also change the way each vertex finds its center. See more details in Subsections 1.3 and 1.4.

Our algorithm for constructing $O(k^2)$-spanners consists of two parts. The first, which is described in high-level is Subsection 1.5, closely follows the construction in [28]. The main novelty in this algorithm is in the way we partition the Voronoi cells, which are formed with respect to randomly selected centers, into clusters of smaller size. In addition, we make other adjustments in order to save an additional factor of Δ in the probe and time complexity. The second is a new LCA for decomposing a graph into subgraphs of smaller maximum degree. As the first algorithm depends quadratically on the degree, this allows for further savings. We elaborate on this algorithm, which may be of independent interest, in Subsection 1.6.

1.3 Algorithm for constructing 3-spanners

We begin with describing our algorithm for constructing 3-spanners from a global point of view. The local implementation of this global algorithm is relatively straight-forward.

The high level idea is as follows. We consider a partition of the vertices into *heavy* and *light* according to their degrees. All the edges incident to light vertices are added to the spanner. We now focus on the heavy vertices. As a first step, a random subset of vertices is selected. We refer to these vertices as *centers*. With high probability, every heavy vertex has a center in its neighborhood. Assuming this event occurs, each heavy vertex joins a *cluster* of at least one of the centers in its neighborhood. A cluster is composed from a center and a subset of its neighbors. On query $\{u, v\}$, where both u and v are heavy, we consider two cases.

1. u and v belong to the same cluster. In this case we add the edge $\{u, v\}$ to the spanner only in case u is the center of v or vice versa.
2. Otherwise, u and v belong to different clusters. Assume without loss of generality that the degree of v is not greater than the degree of u. We divide the edges incident to v into fixed size *buckets* and add the edge $\{u, v\}$ only if it has minimum rank amongst the edges that are incident to the cluster of u.

In order to make the above high-level description concrete we need to set up some parameters and describe how the centers are selected and how each vertex finds its center. We begin by defining vertices with degrees larger than $\sqrt{n}$ as heavy. Thus by adding all the edges incident to light vertices we add at most $O(n^{3/2})$ edges.

The selection of the centers proceeds as follows. We define $t = \Theta(\log \sqrt{n})$ sets of centers, which are picked uniformly at random, $S_1, \ldots, S_t$ such that the size of S_1 is $\Theta(\sqrt{n})$ and the size of S_{i+1} is roughly half of the size of S_i. Thus, overall, the number of centers is $\tilde{O}(\sqrt{n})$.

We next describe how each heavy vertex finds its center. We partition the heavy vertices into t sets, $V_1, \ldots, V_t$ according to their degrees. The set V_1 contains all the vertices with degree in $[\sqrt{n} + 1, 2\sqrt{n}]$ and in general for every $i \in [t]$, the set V_i contains all the vertices with degree in $[2^{i-1}\sqrt{n} + 1, 2^i\sqrt{n}]$. The centers for vertices in the set V_i are taken from the set S_i. With high probability, for every $i \in [t]$, each vertex $v \in V_i$ has at least one vertex from S_i in its neighborhood and at most $O(\log n)$. Thus, with high probability, each heavy vertex belongs to at least one cluster and at most $O(\log n)$ clusters.

Given a heavy vertex $v \in V_i$, the centers of v are found by going over all the vertices in S_i, u, and checking if $\{u, v\}$ is an edge in the graph. Since the total number of centers is $\tilde{O}(\sqrt{n})$, the probe and time complexity of finding the center of a given vertex is $\tilde{O}(\sqrt{n})$.

It remains to set the size of the buckets. Let $\{u, v\} \in E$ be such that $v \in V_i$ and $\deg(u) \leq \deg(v)$. Since $v \in V_i$, it follows that $\deg(v) \leq 2^i\sqrt{n}$. Since $|S_i| \leq c\sqrt{n}\log n/2^i$ for some constant c, by setting the size of the buckets to be $\sqrt{n}$ we obtain that the total number of edges between heavy vertices that belong to different clusters is $\tilde{O}(n^{3/2})$, as desired.

From the fact that the size of the buckets is $\sqrt{n}$ it follows that the total probe and time complexity of our algorithms is $\tilde{O}(\sqrt{n})$. From the fact that the diameter of every cluster is 2 we obtain that for every edge $\{u, v\}$ which we remove from the graph, there exists a path of length at most 3 between u and v. Hence, the stretch factor of our spanner is 3, as desired.

1.4 Algorithm for constructing 5-spanners

We extend the ideas from the previous section to obtain our algorithm for constructing 5-spanners as follows. We partition the vertices in the graph into three sets: *heavy*, *medium*, and *light*. The set of light vertices is defined to be the set of all vertices with a degree at most $n^{1/3}$ and the set of heavy vertices is defined to be the set of all vertices of degree at least $n^{2/3}$. The set of the medium vertices is defined to be all vertices that are not light nor heavy.

As before, we add to the spanner all the edges incident to light vertices and cluster all the heavy vertices into a cluster of diameter 2. The difference is that now when we partition the heavy vertices into sets according to their degrees the first set consists of all vertices with a degree in $[n^{2/3} + 1, 2n^{2/3}]$.

We partition the set of medium vertices into two sets according to the following random process. Each medium vertex v samples uniformly at random $\Theta(\log n)$ of its neighbors. If one of the vertices in the sample is heavy then v joins the cluster of the heavy vertex in the sample that has minimum rank. Otherwise we say that v is *bad*. This forms clusters of diameter 4.

In a similar manner to the process described above we define another a new collection of sets of centers for the bad vertices such that the total number of such centers is $\tilde{O}(n^{2/3})$ and each bad vertex belongs to at least one cluster and at most $O(\log n)$ clusters. The new centers are selected (randomly) only from the set of vertices which are not heavy. We call the corresponding clusters *light-clusters* since they contain at most $n^{2/3}$ vertices and have diameter 2. Since the total number of light-clusters is $\tilde{O}(n^{2/3})$ we can afford to take an edge between every pair of adjacent light-clusters. Moreover, we partition each light-cluster into buckets of size $n^{1/3}$ and take an edge between every pair of adjacent buckets. Since each bad vertex belongs to $O(\log n)$ light-clusters, the total number of pairs of buckets is $\tilde{O}(n^{4/3})$. The time and probe complexity of finding all the edges incident to two buckets is $\tilde{O}(n^{2/3})$, as desired.

To analyse the stretch factor we partition the edges we remove into three types. The first type of edges are edges between vertices in the same cluster. The second type of edges are edges between a vertex v and a cluster C which is not light, in which case there at least one edge in the spanner which is incident to both v and C. The third type of edges are edges which are incident to a pair of light-clusters, in which case there exists at least one edge in the spanner which is incident to each pair of such clusters. Thus, overall the stretch factor is 5, as claimed.

1.5 Algorithm for constructing $O(k^2)$-spanners

The high-level idea of the algorithm for constructing $O(k^2)$-spanners, which we describe from a global point of view, follows that of [28]. The vertices of the graph are first partitioned into $\tilde{O}(n^{2/3})$ Voronoi cells which are formed with respect to a randomly selected set of $\tilde{O}(n^{2/3})$ centers. We can assume that each Voronoi cell has diameter $O(k)$ by using a separate algorithm to handle *remote* vertices which may not be close to a center. Each Voronoi cell is then partitioned into clusters of size $L = \tilde{O}(n^{1/3})$. In addition each Voronoi cell is marked with probability $1/n^{1/3}$ which respectively also marks all the clusters of the cell. Each non-marked cluster connects to all the adjacent marked clusters using a single edge. This forms *clusters-of-clusters* around marked clusters. Instead of connecting every pair of adjacent clusters A and B, which we can not afford, our goal is to connect A with the cluster-of-clusters of some marked cluster C adjacent to B. Since we can not afford to reconstruct the cluster-of-clusters of C, we instead find the identity of all the Voronoi cells which are adjacent to C and try to connect A with at least one of these cells. We show that this is indeed the case although A may not be connected directly to any one of these cells. By applying an inductive argument we show that the number of hops between A and B is $O(k)$, where traversing from one Voronoi cell to another is considered one hop. Since the diameter of each Voronoi cell is $O(k)$ we obtain an overall stretch factor of $O(k^2)$.

We improve on the LCA of [28] in two main ways. The first main improvement is a new method for partitioning the Voronoi cells into clusters of size L, allowing the cluster containing a vertex to be reconstructed using $O(\Delta^2 L^2)$ probes instead of $O(\Delta^3 L^2)$. The second improvement relates to the problem of connecting a cluster A to the cluster-of-clusters of some marked cluster C adjacent to B. In particular there is an issue of which marked cluster C should be chosen, since it is too expensive to reconstruct every marked cluster adjacent to B. The LCA of [28] processes a single cluster of each adjacent marked Voronoi cell to B, of which there may be as many as $\tilde{O}(\Delta)$. We instead devise a rule by which B is *engaged* with a single marked cluster adjacent to it, and show that in fact it suffices to only consider this one cluster. Combining these improvements reduces the total number of probes from $\tilde{O}(n^{2/3}\Delta^4)$ to $O(n^{2/3}\Delta^2)$.

1.6 Algorithm for graph decomposition

To further reduce the runtime of Theorem 31, we develop a new local computation algorithm to decompose graphs into subgraphs with smaller degree. Observe that for a graph G, for subgraphs $G_1, \ldots, G_t$, if we have k-spanners $H_i \subset G_i$ for every i, the union $\bigcup_{i\in[t]} H_i$ is a k-spanner for G. As the runtime of Theorem 31 depends on the maximum degree Δ, we develop an efficient LCA to break G into t graphs, each with maximum degree $O(\max\{\Delta/t, \log n\})$, where t is a parameter to be chosen. Given v and an index $i \in [t]$, the LCA returns in time $O(\Delta/\sqrt{t})$ all neighbors of v in G_i (i.e. it supports `ALL_NBR` queries to each subgraph). We believe this algorithm may have other applications.

To apply this algorithm in the spanner framework, we compose the LCA for $O(k^2)$ spanners with the LCA for graph decomposition. This is more subtle than generic sequential composition of algorithms, as we must ensure the per-query overhead is mild. We do this by observing the $O(k^2)$-spanner algorithm only ever makes all neighbor queries, and so the decomposition LCA spends $O(\Delta/\sqrt{t})$ work per query the spanner LCA makes to the graph. In particular, as the spanner LCA makes $O(n^{2/3}\Delta)$ all neighbor queries, our new runtime is $O(n^{2/3}\Delta^2/t^{3/2})$ given our choice of t. As decomposing G into t subgraphs increases the size of the output spanner by a factor of t, we ultimately balance parameters and obtain a probe and time complexity of $O(n^{2/3-(1.5-\alpha)/k}\Delta^2)$ for any $\alpha > 0$.

1.6.1 Decomposing the Graph

To build this graph, consider assigning each edge of $G = (V, E)$ two colors $i, j \in [R]$, one from each endpoint. In particular, v assigns its first Δ/R edges to receive color 1, the next Δ/R to receive color 2, etc. Then if an edge (u, v) has received colors i, j from both endpoints, we can let the overall edge color be (i, j) where we assume w.l.o.g that $u < v$. Observe that if each color corresponds to a subgraph, then this decomposition breaks G into R^2 subgraphs. Moreover, given i, j and a vertex v, we can quickly enumerate blocks i and j from v and determine which edges lie in the specified subgraph. However, as these blocks may be poorly aligned, this as described results in a maximum subgraph degree of Δ/R rather than Δ/R^2. Instead, we have each vertex choose a random shift, and assign labels to its blocks according to this shift. Then via standard concentration bounds the maximum degree of every subgraph is as desired. We remark that as we must enumerate every element of bucket i and j from v to find edges with label (i, j), the worst-case time for an individual neighbor query can be up to $\Omega(\Delta/R)$. However, we only need to do this once to answer an all-neighbors query, so as long as all the neighbors are desired we can efficiently amortize this cost.

1.7 The number of random bits

All our algorithms are randomized and hence use random bits. For results 1-2 we use randomness in the selection of centers and representatives. In result 3 we use randomness in the selection of centers, marked clusters, and random ranking of edges.

When the centers and representatives are selected independently, the arguments for proving the guarantees on the sparsity of the spanner follow from standard concentration bounds. As shown by Parter at al. [28], by using a less standard analysis one is able to prove that the same guarantees on the sparsity hold even when the random bits are only $\Theta(\log n)$-wise independent (which requires only $O(\log^2 n)$ truly random bits). Furthermore, by using an intricate analysis, they showed that the guarantees on the stretch factor continue to hold as well. In this writing, we do not repeat the analysis in [28] since it lends itself quite easily to our setting.

For result 4, similar techniques to those of [28] allow the result to be implemented using polylog(n)-wise independence as well [1].

1.8 Related work

As mentioned above, the work which is the most closely related to our work is by Parter et el. [28]. In addition to the upper bounds mentioned in Section 1.1 they also observe that it is possible to obtain an LCA for constructing 5-spanners with $\tilde{O}(n^{1+1/k})$ edges and probe complexity $\tilde{O}(n^{1-1/(2k)})$ for the special case in which the minimum degree is known to be at least $n^{1/2-1/(2k)}$ (this builds on the fact that by picking $\tilde{O}(n^{(1+1/k)/2})$ centers, w.h.p. each vertex has a center in its neighborhood). In addition to upper bounds, they also provide a lower bound of $\Omega(\min\{\sqrt{n}, \frac{n^2}{m}\})$ probes for the simpler task of constructing a spanning graph with $o(m)$ edges, where m denotes the number of edges in the input graph.

Our work also builds on the upper bound in [23], designed originally for bounded degree graphs, which provide a spanner with $(1 + \epsilon)n$ edges on expectation, where ϵ is a parameter, stretch factor $O(\log^2 n \cdot \text{poly}(\Delta/\epsilon))$ and probe complexity of $O(\text{poly}(\Delta/\epsilon) \cdot n^{2/3})$. The work in [23] is a follow-up of [26, 25] which initiated the study of LCAs for constructing ultra-sparse (namely, with $(1 + \epsilon)n$ edges) spanning subgraphs.

[1] More specifically, by using the concentration bound from Fact 5.3 in [28] on the sum of d-wise independent random variables.

2 Preliminaries

The input graph $G = (V, E)$ is a simple undirected graph with $|V| = n$ vertices and a bound on the degree Δ. Both parameters n and Δ are known to the algorithm. Each vertex $v \in V$ is represented as a unique ID from $[n]$.

A local algorithm has access to the *adjacency list oracle* $\mathcal{O}^G$ which provides answers to the following probes (in a single step):

- **Degree probe:** Given $v \in V$, returns the degree of v, denoted by $\deg(v)$.
- **Neighbour probe:** Given $v \in V$ and an index i, returns the i^{th} neighbor of v if $i \leq \deg(v)$. Otherwise, $\perp$ is returned. Additionally, for $v \in V$, we define the all-neighbors query, denoted by `ALL_NBR`(v), which returns all the neighbors of v. Clearly, this query can be implemented by $\deg(v) + 1$ neighbor probes.
- **Adjacency probe:** Given an ordered pair $\langle u, v\rangle$ where $u \in V$ and $v \in V$, if v is a neighbor of u then i is returned where v is the i^{th} neighbor of u. Otherwise, $\perp$ is returned.

We denote the distance between two vertices u and v in G by $\mathrm{d}(u, v)$ and the set of neighbours of v in G by $N_G(v)$. We denote by $N_G(v)[i]$ the i-th neighbour of v in G. For vertex $v \in V$ and an integer k, let $\Gamma_k(v, G)$ denote the set of vertices at distance at most k from v. When the graph G is clear from the context, we shall use the shorthand $d(u, v)$, $N(v)$ and $\Gamma_k(v)$ for $d_G(u, v)$, $N_G(v)$ and $\Gamma_k(v, G)$, respectively. We define a ranking r of the edges as follows: $r(u, v) < r(u', v')$ if and only if $\min\{u, v\} < \min\{u', v'\}$ or $\min\{u, v\} = \min\{u', v'\}$ and $\max\{u, v\} < \max\{u', v'\}$.

We shall use the following definitions in our algorithms for constructing 3-spanners and 5-spanners.

▶ **Definition 2.** *We say that a vertex $v \in V$ is in* class $i \in \mathbb{N}$ w.r.t. Δ *if* $\deg(v) \in [2^{i-1}\Delta + 1, 2^i\Delta]$.

▶ **Definition 3.** *We say that an index $i \in \mathbb{N}$ is in* bucket $j \in \mathbb{N}$ w.r.t. Δ *if* $i \in [(j-1) \cdot \Delta + 1, j \cdot \Delta]$.

2.1 Probes in the LCA model

Since the introduction of the model in [32], there have been several formulations concerning, mainly, the measure of performance, the way the input is accessed, and whether preprocessing is allowed. In particular, when the input is a graph, there is the question of whether the LCA can probe the graph anywhere (i.e. ask for the neighbors of an arbitrary vertex). In contrast to message-passing models such as CONGEST and distributed LOCAL algorithms, in LCAs the standard assumption is that indeed the LCA can access the graph anywhere and more specifically that each vertex in the input graph is represented as a unique ID from $[n] = \{1, \ldots, n\}$. To support this claim, we refer the reader to the ultra-formal definition in [20] (Definition 12.11) as well as [3, 18].

We note that the utility of making far-probes [2] was studied in [21], in which the authors showed that for a large family of problems, this extra power is not so useful. Indeed, this extra power is not always used by LCAs. For example, in the recent result of Ghaffari [19], which provides an LCA for the problem of Maximal Independent Set, the assumption is that the IDs are taken from $[n^{10}]$. Nonetheless, we stress that this extra power is an important feature of

[2] Namely, probing vertices for which we do not yet know a path from the query vertex.

the LCA model, which, in particular, distinguishes it from message-passing models (see more on the difference between LCAs and distributed LOCAL algorithms in Section 4.1 in [24]) and comes into play in problems which have a more global nature. For example, this extra power is utilized in Prop. 12.13 in [20] for graph coloring and in [27] for approximate Maximum-Matching. The latter LCA is used in Behnezhad et al. [8] to obtain a state-of-the-art sublinear algorithm for the extensively studied problem of approximate Maximum-Matching.

3 LCA for constructing 3-spanners

In this section, we prove the following theorem. Due to space limitations, we defer the claims regarding probe and time complexities as well as the stretch factor to the appendix.

▶ **Theorem 4.** *There exists an LCA that given access to an n-vertex simple undirected graph G, constructs a 3-spanner of G with $\tilde{O}(n^{1+1/2})$ edges whose probe complexity and time complexity are $\tilde{O}(n^{1/2})$.*

Our algorithm is listed as Algorithm 1. As mentioned-above our algorithm proceeds by forming clusters around centers and connecting the different clusters. To make the description of our algorithm complete we begin with describing the selection of centers.

We define $t \stackrel{\text{def}}{=} \log \sqrt{n}$ sets of centers $S_1, \ldots, S_t$. For every $i \in [t]$, we pick u.a.r. x_i vertices to be in S_i where $x_1 = \sqrt{n} \log n$ and $x_{i+1} = x_i/2$ for every $i \in [t-1]$. The rest of the details of the algorithm appear in Algorithm 1. We next prove the correctness of the algorithm.

Recall that we refer to a vertex whose degree is greater than $\sqrt{n}$ as heavy. The next claim states that with high probability every heavy vertex has at least one center and $O(\log n)$ centers in its neighborhood.

▷ Claim 5. With high probability, for every $i \in [t]$ and every vertex $v \in V$ that is in class i w.r.t. $\sqrt{n}$ it holds that $N(v) \cap S_i \neq \emptyset$ and that $|N(v) \cap S_i| = O(\log n)$.

Algorithm 1 LCA for constructing 3-spanners.

Input: Access to an undirected graph $G = (V, E)$ and a query $\{u, v\} \in E$ where we assume w.l.o.g. that $\deg(u) \geq \deg(v)$.
Output: Returns whether $\{u, v\}$ belongs to the spanner or not.

1. If $\deg(v) \leq n^{1/2}$ then return YES (recall that $\deg(u) \geq \deg(v)$).
2. Otherwise, let c denote the class of u w.r.t. $\sqrt{n}$ (see Definition 2).
3. If $v \in S_c$ then return YES.
4. Otherwise, let $\mathcal{C} \stackrel{\text{def}}{=} S_c \cap N(u)$. If $\mathcal{C} = \emptyset$ then return YES.
5. Let i denote the index of u in $N(v)$ and let b denote the *bucket* of i w.r.t. $\sqrt{n}$ (see Definition 3).
6. For each $x \in \mathcal{C}$:
   a. Go over every $j < i$ such that j is in bucket b and return YES if for every such j, $N(v)[j]$ does not belong to the cluster of x.
7. Return NO.

▷ Claim 6. With high probability, the stretch factor of the spanner constructed by Algorithm 1 is 3.

Proof. Let $\{u, v\}$ be an edge in E such that $\deg(u) \geq \deg(v)$. We will show that there exists a path of length at most 3 between u and v in the the spanner constructed by Algorithm 1 denoted by $G' = (V, E')$. If $\deg(v) \leq \sqrt{n}$ then $\{u, v\} \in E'$ and we are done. Otherwise, if there exists a cluster C such that u and v are both belong to C then in G' they are both connected by an edge to the center of C. Thus there exists a path of length at most 2 between u and v in G'. Otherwise, let C' be a cluster for which u belongs to. We claim that v is adjacent to C' in G'. This follows by induction on the index of u in $N(v)$ and Sub-Step 6a. ◁

▷ Claim 7. The probe and time complexity of Algorithm 1 is $O(\sqrt{n} \log n)$.

Proof. Steps 1-2 can be implemented by making a single degree probe. Their time complexity is $O(1)$. Step 3 can be implemented by accessing the random coins. To implement Step 4 we need to go over all the vertices in S_c (we may assume w.l.o.g. that we generate all the centers in advance as there are only $O(\sqrt{n} \log n)$ centers) and check whether they are in $N(u)$ (by making a single adjacency probe). Thus the probe (and time) complexity of this step is $O(\sqrt{n} \log n)$. Step 5 can be implemented by a single adjacency probe. The total number of vertices we check in Sub-Step 6a is bounded by the size of $\mathcal{C}$ times the size of a bucket which is $\sqrt{n}$. For each vertex we check we make a single neighbor and then we check whether it belongs to the cluster of a specific center. The latter can be implemented by making a single degree probe and a single adjacency probe. By Claim 5, the size of $\mathcal{C}$ is bounded by $O(\log n)$, thus the probe (and time) complexity of Step 6 is $O(\sqrt{n} \log n)$. The claim follows. ◁

▷ Claim 8. With high probability, the number of edges of the spanner constructed by Algorithm 1 is $\tilde{O}(n^{1+1/2})$.

Proof. The number of edges added to E' due to Step 1 is at most $n^{3/2}$. By the bound on the number of centers, the number of edges added to E' due to Step 3 is $O(n^{3/2} \log n)$. To analyse the number of edges added to E' due to Step 6 consider an edge $\{u, v\}$ such that $\deg(u) \geq \deg(v)$, $\deg(v) > \sqrt{n}$ and $v \notin S_c$, where c denotes the class of u w.r.t. $\sqrt{n}$. Since u is in class c it follows that $\deg(u) \leq 2^c\sqrt{n}$. Since $\deg(v) \leq \deg(u)$ it follows that $N(v)$ has at most 2^c buckets. By Sub-Step 6a, for any cluster C, the number of edges in E' that are incident to v and a vertex from C is at most 2^c (since we add to E' at most a single edge for each bucket of $N(v)$). Since the number of clusters of class c is $O(\sqrt{n} \log n/2^c)$, the total number of clusters of class greater or equal to c is $O(\sqrt{n} \log n/2^c)$ as well. Therefore, the total number of edges that are incident to v and added to E' due to Step 6 is $O(\sqrt{n} \log n)$. By Claim 5, w.h.p., the number of edges that are added due to Step 4 is 0. We conclude that the $|E'| = O(n^{3/2} \log n)$, as desired. ◁

4 LCA for constructing 5-spanners

In this section, we prove the following theorem.

▶ **Theorem 9.** *There exists an LCA that given access to an n-vertex simple undirected graph G, constructs a 5-spanner of G with $\tilde{O}(n^{1+1/3})$ edges whose probe complexity and time complexity are $\tilde{O}(n^{2/3})$.*

Our algorithm for constructing 5-spanners also proceeds by forming clusters around centers and connecting the different clusters. For the sake of presentation, we first describe our local algorithm from a global point of view (see algorithm 2). In Section 4.1 we describe the local implementation of this algorithm.

As in the algorithm for constructing 3-spanners, the clusters are formed around randomly selected centers only that now we have two types of clusters (and centers), *heavy-clusters* and *light-clusters* that will be described in the sequel.

The selection of the first type of centers

The selection of the first type of centers proceeds as follows. We define $a \stackrel{\text{def}}{=} \log n^{1/3}$ sets of centers $S_1^1, \ldots, S_a^1$. For every $i \in [a]$, we pick u.a.r. y_i vertices to be in S_i^1 where $y_1 = n^{1/3} \log n$ and $y_{i+1} = y_i/2$ for every $i \in [a-1]$. The clusters which are formed around the first type of centers are the *heavy-clusters*. The formation of the heavy-clusters is described in Step 2 of Algorithm 2.

The selection of the second type of centers

The selection of the second type of centers proceeds as follows. We define $b \stackrel{\text{def}}{=} \log n^{1/3}$ sets of centers $S_1^2, \ldots, S_b^2$. For every $i \in [b]$, we pick u.a.r. x_i vertices to be in S_i^2 where $x_1 = n^{2/3} \log n$ and $x_{i+1} = x_i/2$ for every $i \in [b-1]$.

The clusters which are formed around the second type of centers are the *light-clusters*. The formation of the light-clusters is described in Step 3 of Algorithm 2.

The way we connect the different clusters is described in Steps 4 and 5.

In the next couple of claims we prove that with high probability every vertex v such that $\deg(v) > n^{1/3}$ joins at least one cluster and at most $O(\log n)$ clusters. To do so, we partition the vertices with degree greater than into $n^{1/3}$ into 3 sets. The first set, denoted by H, is the set of vertices, v, such that $\deg(v) \geq n^{2/3}$. The second set is the set of vertices, v, such that $n^{1/3} < \deg(v) < n^{2/3}$ for which at least half of the vertices in $N(v)$ have degree at least $n^{2/3}$. We denote this set by M_1. M_2 consists of the remaining vertices. Namely, M_2 is the set of vertices, v, such that $n^{1/3} < \deg(v) < n^{2/3}$ and for which less than half of the vertices in $N(v)$ have degree at least $n^{2/3}$.

The implication of the next claim is that w.h.p. every vertex in H joins at least one heavy-cluster and at most $O(\log n)$ heavy-clusters.

▷ Claim 10. With high probability, for every $v \in H$ it holds that $N(v) \cap S_c^1 \neq \emptyset$ and that $|N(v) \cap S_c^1| = O(\log n)$ where $c \in [a]$ is the class of v w.r.t. $n^{2/3}$.

The implication of the next claim (when combined with Claim 10) is that w.h.p. every vertex in M_1 joins, via a representative, at least one heavy-cluster and at most $O(\log n)$ heavy-clusters.

▷ Claim 11. With high probability, for every $v \in M_1$ it holds that v has a representative.

Proof. Let $v \in M_1$. Consider Step 2b of Algorithm 2. Since at least half of the neighbors of v have degree at least $n^{2/3}$, it follows that w.h.p. $R_v \neq \emptyset$ and so v has a representative. ◁

The implication of the next claim is that w.h.p. every vertex in M_2 that does not have a representative joins at least one light-cluster and at most $O(\log n)$ light-clusters.

▷ Claim 12. With high probability, for every $v \in M_2$ it holds that $N(v) \cap S_c^2 \neq \emptyset$ and that $|N(v) \cap S_c^2| = O(\log n)$ where $c \in [b]$ is the class of v w.r.t. $n^{1/3}$.

The following corollary follows directly from Claims 10–12.

Algorithm 2 Global algorithm for constructing 5-spanners.

Input: A graph $G = (V, E)$.

Output: Constructs a 5-spanner of G, $G' = (V, E')$.

1. For every v such that $\deg(v) \leq n^{1/3}$ add to E' all the edges that are incident to v.
2. Forming heavy-clusters:
   a. For each vertex v such that $\deg(v) \geq n^{2/3}$ we define the centers of v to be $N(v) \cap S_c^1$ where c is the class of v w.r.t. $n^{2/3}$ (see Definition 2). For every center s of v, v joins the cluster of s by adding the edge $\{s, v\}$ to E'.
   b. Each vertex v such that $n^{1/3} < \deg(v) < n^{2/3}$ sample u.a.r. $y \stackrel{\text{def}}{=} \Theta(\log n)$ of its neighbors. Let R_v denote this set. The *representative* of v is defined to be the vertex, r, of minimum id in R_v such that $\deg(r) \geq n^{2/3}$ (if such vertex exists). If v has a representative, r, then the edge $\{v, r\}$ is added to E' (and hence v joins all the clusters of r).
3. Forming light-clusters:
   a. For each vertex v such that $n^{1/3} < \deg(v) < n^{2/3}$ for which v does not have a representative we define the centers of v to be $N(v) \cap S_c^2$ where c is the class of v w.r.t. $n^{1/3}$ (see Definition 2). For every center s of v, v joins the cluster of s by adding the edge $\{s, v\}$ to E'.
4. Connecting vertices to adjacent heavy-clusters:
   a. Let $\{u, v\}$ be such that $\deg(u) \geq \deg(v)$ and u belongs to a heavy-cluster. For each cluster C that u belongs to, do:
      i. Partition the interval $[\deg(v)]$ into sequential intervals, which we refer to as buckets, of size $n^{2/3}$: $b_1, \ldots, b_s$ (where only b_s may have size which is smaller than $n^{2/3}$).
      ii. For each $i \in [s]$, go over every $j \in b_i$ in increasing order and check if $N(v)[j]$ belongs to C. If such j is found, add $\{v, N(v)[j]\}$ to E' and move to the next bucket.
5. Connecting adjacent light-clusters:
   a. Let $\{u, v\}$ be such that both u and v belong to different light-clusters. For each light clusters C_u and C_v that u and v belong to, respectively, do:
      i. Let s_u and s_v denote the centers of C_u and C_v, respectively. Let c_u and c_v denote the classes of u and v w.r.t. $n^{1/3}$, respectively.
      ii. Partition the vertices in $N(s_u)$ that belong to the cluster C_u (namely, the neighbors of s_u that are in class c_u w.r.t. $n^{1/3}$) into subsets of size $n^{1/3}$ greedily by their index in $N(s_u)$, $S_1^u, \ldots, S_t^u$ (all the subsets are of size $n^{1/3}$ except from perhaps S_t^u).
      iii. Repeat Step 5(a)ii for the vertices in $N(s_v)$ that belong to C_v and let $S_1^v, \ldots, S_r^v$ denote the resulting subsets.
      iv. For each $i \in [t]$ and $j \in [r]$, add the edge of minimum rank in $E(S_i^u, S_j^v)$ to E' (if such edge exists).

▶ **Corollary 13.** *With high probability every vertex v such that $\deg(v) > n^{1/3}$ joins at least one cluster and at most $O(\log n)$ clusters.*

▷ Claim 14. With high probability, $|E'| = \tilde{O}(n^{1+1/3})$.

Proof. The number of edges added to E' due to Step 1 is at most $n^{1+1/3}$. By Claims 10 and 12 the number of edges added to E' due to Steps 2a and 3a is $\tilde{O}(n)$. Since each vertex has at most one representative the number of edges added to E' due to Step 2b is at most n.

Consider $\{u, v\}$ such that $\deg(u) \geq \deg(v)$ and u belongs to a heavy-cluster C. According to Step 4(a)ii we connect v to C by adding to E' at most $\lceil \deg(v)/n^{2/3} \rceil$ edges (at most one edge for each bucket of $N(v)$).

If $\deg(u) \leq n^{2/3}$ then $\deg(v) \leq n^{2/3}$ as well and so $\lceil \deg(v)/n^{2/3} \rceil \leq 1$. Therefore the total number of edges that are incident to v and added to E' due to Step 4(a)ii is bounded by the total number of centers of the first type which is $\tilde{O}(n^{1/3})$.

Otherwise, let c denote the class of u w.r.t. $n^{2/3}$, then by definition $\deg(u) \leq 2^c \cdot n^{2/3}$. Therefore by our assumption $\deg(v) \leq 2^c \cdot n^{2/3}$ as well. The number of centers in S_c^1 is $n^{1/3} \log n / 2^c$ and so the total number of centers in $\bigcup_{c \leq i \leq a} S_i^1$ is $O(n^{1/3} \log n / 2^c)$. Observe that the number of edges which are incident to v and added to E' due to Step 4(a)ii is at most $\lceil \deg(v)/n^{2/3} \rceil$ times the number of centers in $\bigcup_{c \leq i \leq a} S_i^1$. Thus the total number of edges that are incident to v and added to E' due to Step 4(a)ii is $\tilde{O}(n^{1/3})$ in this case as well. Therefore, the total number of edges which are added to E' in Step 4(a)ii is $\tilde{O}(n^{1+1/3})$.

By Claim 12 it follows that the total number of subsets partitioning the light clusters is $\tilde{O}(n^{2/3})$ as the size of each subset is $n^{1/3}$ except for at most $\tilde{O}(n^{2/3})$ subsets, and since each vertex may belong to $O(\log n)$ different clusters. Since in Step 5(a)iv we add at most a single edge between a pair of subsets the total number of edges added to E' due to this step is $\tilde{O}(n^{1+1/3})$. This concludes the proof of the claim. ◁

▷ Claim 15. With high probability, the stretch factor of the spanner constructed by Algorithm 2 is 5.

Proof. Let $\{u, v\}$ be an edge which is not included in E'. By Step 1 of the algorithm it follows that the degree of both u and v is greater than $n^{1/3}$. By Corollary 13 w.h.p. all vertices with degree greater than $n^{1/3}$ join at least one cluster. In the rest of the proof we condition on the event that both u and v join at least one cluster.

Assume w.l.o.g. that $\deg(u) \geq \deg(v)$. If both u and v belong to the same cluster (either heavy or light) then there exists a path of length at most 4 in G' between u and v as the diameter of each cluster is at most 4.

If u belongs to a heavy cluster, C, then by Step 4(a)ii of the algorithm it follows that there exists at least one edge in E' which is incident to v and a vertex in C. Since the diameter of C is at most 4 it follows that there exists a path in G' from v to u.

Otherwise, both u and v belong to different light clusters C_u and C_v. By Step 5(a)iv, there exists at least one edge in E' which is incident to a vertex in C_u and a vertex in C_v. Since the diameter of a light cluster is at most 2 we obtain that there exists a path in G' from u to v of length at most 5. This concludes the proof of the claim. ◁

4.1 The local implementation

In this section we prove the following claim. In the proof of the claim we also describe the local implementation of Algorithm 2.

▷ Claim 16. The probe and time complexity of the local implementation of Algorithm 2 is $\tilde{O}(n^{2/3}\log n)$.

Proof. On query $\{u, v\}$ we first probe the degree of u and v and return YES if either u or v have degree which is at most $n^{1/3}$. Otherwise, assume w.l.o.g. that $\deg(u) \geq \deg(v)$. we consider the following cases.

First case: $\deg(u) \geq n^{2/3}$. In this case we find the centers of u by going over all the centers, s, in S^1_c where c is the class of u w.r.t. $n^{2/3}$ and preforming the adjacency probe $\langle u, s\rangle$. If v belongs to the set of centers of u then we return YES. Overall, since the number of centers of the first type is $\tilde{O}(n^{1/3})$, finding the centers of u requires $\tilde{O}(n^{1/3})$ probes and time.

We then find the bucket b of u in $N(v)$ w.r.t. $n^{2/3}$ (see Definition 3) by preforming the adjacency probe $\langle v, u\rangle$. Let i denote the index of u in $N(v)$. For each center of u, s and for each $j \in b$ such that $j < i$, we check if $N(v)[j]$ belongs to the cluster of s. In order to do so we first probe the degree of $y = N(v)[j]$. If $\deg(y) \geq n^{2/3}$ then v is in the cluster of s if and only if it is a neighbour of s and is in class c w.r.t. $n^{2/3}$ where c is such that s belongs to S^1_c. If $\deg(y) < n^{2/3}$ then we first find the representative of y and if it has a representative we check if it belongs to the cluster of s. Since we have to check this for at most $n^{2/3}$ vertices and for $O(\log n)$ centers, overall the probe and time complexity of preforming this task is $\tilde{O}(n^{2/3})$.

Second case: $n^{1/3} < \deg(u) < n^{2/3}$ and either u or v have a representative. In this case we proceed as in the previous case only that we preform all the checks with respect to the centers of the representative of u (and/or the representative of v). Since finding the representative of a vertex requires $O(\log n)$ probes and time the probe and time complexity in this case is $\tilde{O}(n^{2/3})$ as well.

Third case: $n^{1/3} < \deg(u) < n^{2/3}$ and both u and v do not have representatives. This corresponds to the case in which both u and v belong to light clusters. In order to find the centers of u we simply go over all vertices, y, in $N(u)$ and check if y is in S^2_c where c denotes the class of u w.r.t. $n^{1/3}$. We repeat the same process for v. Since checking if a vertex belongs to S^2_c can be done in $O(\log n)$ time (we can generate all the centers in advance and store them in a binary search tree) this task requires $\tilde{O}(n^{2/3})$ probes and time.

Finally, for each pair of centers s_u and s_v of u and v, respectively, we go over all the neighbours of s_u and s_v and determine for each one, according to its degree, whether it belongs to the cluster of s_u and s_v, respectively. We then find the subsets that u and v belong to as defined in Steps 5(a)ii and 5(a)iii and return YES if and only of $\{u, v\}$ is the edge of minimum rank that connects these subsets.

The above three cases cover all possible scenarios which implies that the time (and probe) complexity of the local implementation of Algorithm 2 is $\tilde{O}(n^{2/3})$ as claimed. ◁

5 Graph Decomposition Via Ranking

We give the formal statement of the LCA of Item 4. We note that our decomposition gives a stronger promise than the maximum degree of each subgraph being bounded, in that we actually bound the degree vertex-wise. In particular, up to poly-logarithmic factors, the average degree of every subgraph is equal to the overall average degree divided by the number of subgraphs with high probability.

▶ **Theorem 17.** *There exists an LCA that, given a parameter $R \leq \sqrt{\Delta}$ and access to an n-vertex simple undirected graph $G = (V, E)$ with maximum degree Δ, decomposes G into edge-disjoint subgraphs $G_1, \ldots, G_{R^2}$ such that:*

1. *Given $(u, v) \in E$, the $i \in [R^2]$ such that $(u, v) \in G_i$ can be computed in time and space $O(1)$.*
2. *Given $v \in V$ and $i \in [R^2]$, $\texttt{ALL\_NBR}_i(v)$ can be computed in time and space $O(d_G(v)/R)$.*
3. *With high probability, the degree of v in G_i is $O(\max\{\log(n), d_G(v)/R^2\})$ for every $v \in V$ and $i \in [R^2]$. In particular, for every i the maximum degree of G_i is $O(\max\{\log(n), \Delta/R^2\})$ with high probability.*

We first describe the decomposition in a global manner. We refer to each subgraph as a color. We assign each edge in G one of R^2 colors, such that the degree and ALL_NBR query times are as claimed. We will identify the set of colors with $[R] \times [R]$, and assume $R^2 \leq \Delta$ since otherwise the statement is trivial. For convenience, let $d(v) := d_G(v)$ and $d_i(v) := d_{H_i}(v)$.

Each vertex v draws a random value $r_v \sim [R]$. Furthermore, for every vertex v, let the first $\lceil d(v)/R \rceil$ neighbors of v be $B_1(v)$, the second be $B_2(v)$, etc. Note that this divides the out-edges into R blocks. For an edge (u, v) with $u < v$ in blocks $B_i(u), B_j(v)$ respectively, the color of the edge is the pair $(i + r_u \mod R,\ \ j + r_v \mod R)$. Let $G_{a,b}$ for $a, b \in [R]$ be the subgraph consisting of all edges with color (a, b).

We can then combine Theorem 17 with Theorem 31 to give the final result.

▶ **Corollary 18.** *There exists an LCA that given access to an n-vertex simple undirected graph G with maximum degree Δ, constructs an $O(k^2)$-spanner with $\tilde{O}(n^{1+1/k})$ edges whose probe complexity and time complexity are $O(n^{2/3-(1.5-\alpha)/k}\Delta^2)$, for any constant $\alpha > 0$.*

5.1 Decomposition Implementation

▷ Claim 19. Given $(u, v) \in G$, we can determine the color (a, b) of the edge in time and space $O(1)$.

Proof. By making two adjacency probes, we can determine the indices of edge (u, v) in u and v. Then we can compute which blocks contain this edge using two degree queries and a constant number of arithmetic operations, and then compute the final color by looking up the random shifts of u and v. ◁

▷ Claim 20. With high probability, for every $v \in V$ and $(a, b) \in [R] \times [R]$ we have $d_{a,b}(v) = O(\max\{\log(n), d(v)/R^2\})$.

Proof. Fix an arbitrary vertex $v \in V$ and color $(a, b) \in [R] \times [R]$. Fix its shift of r_v of v arbitrarily. Let $S = B_{a-r_v}(v) \cup B_{b-r_v}(v)$ be the set of all edges incident to v (in G) in blocks $a - r_v$ and $b - r_v$. Then S is a superset of the set of edges incident to v with color (a, b), and $|S| = 2d(v)/R$.

For an arbitrary edge $e = (v, u)$ in S, let X_e be the indicator random variable which is 1 exactly when the color of e is (a, b). Let k be the block index of e in u so that $e \in B_k(u)$. There are four cases to consider regarding e:

- Case 1: $v < u$ and $e \in B_{a-r_v}(v)$. Then e has color (a, b) if and only if r_u is equal to $b - k$, which occurs with probability exactly $1/R$.
- Case 2: $v < u$ and $e \notin B_{a-r_v}(v)$. In this case e never has color (a, b).
- Case 3: $v > u$ and $e \in B_{b-r_v}(v)$. Similarly to case 1, e has color (a, b) if and only if r_u is equal to $a - k$, which occurs with probability $1/R$.
- Case 4: $v > u$ and $e \notin B_{b-r_v(v)}$. Similarly to case 2, e never has color (a, b).

In any case, we have $P[X_e = 1] \leq 1/R$ for all $e \in S$. Furthermore, the variables in $\{X_e\}_{e \in S}$ are independent random variables: for distinct edges $e, e' \in S$ where $e = (u, v)$ and $e = (u', v)$, X_e and $X_{e'}$ are independent since the random variables $r_u, r_{u'}$ are independent.

Letting $X = \sum_{e \in S} X_e$ and picking an arbitrary constant $c \geq 2$, we find

$$E[X] \leq 2d(v)/R^2 \leq c \max\{\log n, d(v)/R^2\} =: \mu.$$

By the multiplicative Chernoff's bound we have

$$\Pr[X > 3\mu] \leq \exp(-\mu) \leq \exp(-c \log n) = n^{-c},$$

and so the total number of neighbors of v with color (a, b) is $O(\max\{\log n, d(v)/R^2\})$ with high probability. Finally, a union bound over all n vertices and $R^2 \leq \Delta$ colors completes the proof. ◁

▷ Claim 21. $\texttt{ALL\_NBR}_i(v)$ can be computed in time and space $O(d(v)/R)$.

Proof. Given $v \in V$ and a color $i = (a, b)$, let $S = B_{a-r_v}(v) \cup B_{b-r_v}(v)$ as before. Note that these correspond to the blocks which have received labels a and b respectively, given the random shift of vertex v. We make $2d(v)/R$ neighbor probes to determine all elements of S, then $2d(v)/R$ adjacency probes to determine the indices of every edge in the other endpoint. Then for each edge, we can check in time $O(1)$ (by examining the random shift of the other endpoint) if the label is (a, b). ◁

Proof of Corollary 18. Let $\beta \in (0, 1]$ be such that $3/(2 + \beta) = 1.5 - \alpha$. Given a query if edge $(u, v) \in G$ is in the spanner, we apply the LCA of Theorem 17 with parameter[3] $R = \lceil n^{1/(k(2+\beta))} \rceil$ to determine the i such that $(u, v) \in G_i$. We then apply Theorem 31 with parameter $k' = \lceil k(1 + \frac{2}{\alpha}) \rceil$ to the graph G_i and query if (u, v) is contained in the spanner, and return the answer. Note that we ultimately obtain a $O(k'^2) = O(k^2)$-spanner for every subgraph (and thus for the overall graph), and the number of edges is bounded as $\tilde{O}(R^2 n^{1+1/k'}) \leq \tilde{O}\left(n^{1+\frac{2}{k(2+\alpha)}+\frac{1}{k(1+2/\alpha)}}\right) = \tilde{O}(n^{1+1/k})$. Furthermore, the time complexity is $O(n^{2/3}\Delta^2 R^{-3}) \leq O\left(n^{\frac{2}{3}-\frac{1.5-\alpha}{k}}\Delta^2\right)$. ◀

References

1. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.
2. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139. SIAM, 2012. doi:10.1137/1.9781611973099.89.
3. Sepehr Assadi and Aditi Dudeja. Lecture 5 in advanced algorithms ii – sublinear algorithms. URL: https://people.cs.rutgers.edu/~sa1497/courses/cs514-s20/lec5.pdf.
4. Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.

[3] We have $R^2 \in O(n^{1/k})$; in order to apply Theorem 17 this should be at most Δ. We can assume this since if Δ is $O(n^{1/k})$ then the graph is already sparse to begin with.

5 Hans-Jürgen Bandelt and Andreas Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advances in applied mathematics*, 7(3):309–343, 1986.

6 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

7 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007. doi:10.1002/rsa.20130.

8 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. *CoRR*, abs/2211.15843, 2022. doi:10.48550/arXiv.2211.15843.

9 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

10 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.17.

11 Paul Chew. There is a planar graph almost as good as the complete graph. In Alok Aggarwal, editor, *Proceedings of the Second Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry, Yorktown Heights, NY, USA, June 2-4, 1986*, pages 169–177. ACM, 1986. doi:10.1145/10515.10534.

12 Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. *Theor. Comput. Sci.*, 399(1-2):83–100, 2008. doi:10.1016/j.tcs.2008.02.019.

13 Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In *Distributed Computing, 21st International Symposium, DISC 2007, Lemesos, Cyprus, September 24-26, 2007, Proceedings*, volume 4731 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2007. doi:10.1007/978-3-540-75142-7_16.

14 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. ACM, 2008. doi:10.1145/1400751.1400788.

15 Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. Local computation of nearly additive spanners. In *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*, volume 5805 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2009. doi:10.1007/978-3-642-04355-0_20.

16 David P Dobkin, Steven J Friedman, and Kenneth J Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.

17 Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 652–669. SIAM, 2017. doi:10.1137/1.9781611974782.41.

18 Guy Even, Moti Medina, and Dana Ron. Best of two local models: Centralized local and distributed local algorithms. *Inf. Comput.*, 262:69–89, 2018. doi:10.1016/j.ic.2018.07.001.

19 Mohsen Ghaffari. Local computation of maximal independent set. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 438–449. IEEE, 2022. doi:10.1109/FOCS54457.2022.00049.

20 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

21 Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing – 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 201–214. Springer, 2016. doi:10.1007/978-3-662-53426-7_15.

22 Michael Kapralov and David P. Woodruff. Spanners and sparsifiers in dynamic streams. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 272–281. ACM, 2014. doi:10.1145/2611462.2611497.

23 Christoph Lenzen and Reut Levi. A centralized local algorithm for the sparse spanning graph problem. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 87:1–87:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.87.

24 Reut Levi and Moti Medina. A (centralized) local guide. *Bull. EATCS*, 122, 2017. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/495.

25 Reut Levi, Dana Ron, and Ronitt Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 38:1–38:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.38.

26 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. *Algorithmica*, 82(4):747–786, 2020. doi:10.1007/s00453-019-00612-6.

27 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017. doi:10.1007/s00453-016-0126-y.

28 Merav Parter, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Local computation algorithms for spanners. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 58:1–58:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.58.

29 David Peleg and Jeffrey D Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 77–85, 1987.

30 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989.

31 Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Comput.*, 22(3):147–166, 2010. doi:10.1007/s00446-009-0091-7.

32 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Innovations in Computer Science – ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011.

A LCA for constructing $O(k^2)$-spanners

In this section, we present our LCA for constructing $O(k^2)$-spanners.

A.1 The algorithm that works under a promise

We begin by describing a global algorithm for constructing an $O(k^2)$-spanner which works under the following promise on the input graph $G = (V, E)$. Let $L \stackrel{\text{def}}{=} cn^{1/3} \log n$, where c is a constant that will be determined later. For every $v \in V$, let $i_v \stackrel{\text{def}}{=} \min_r\{|\Gamma_r(v)| \geq L\}$. We are promised that $\max_{v \in V}\{i_v\} \leq k$. In words, we assume that the k-hop neighborhood of every vertex in G contains at least L vertices.

In addition, we assume without loss of generality that $k = O(\log n)$ as already for $k = \log n$ our construction yields a spanner with $\tilde{O}(n)$ edges on expectation.

Our algorithm builds on the partition of V which is described next.

Centers. Pick a set $S \subset V$ by independently including each vertex v in S with probability $n^{-1/3} \log n$, so that $|S| = \Theta(n^{2/3} \log n)$ w.h.p. We shall refer to the vertices in S as *centers*. For each vertex $v \in V$, its *center*, denoted by $c(v)$, is the center which is closest to v amongst all centers (break ties between centers according to the id of the center).

Voronoi cells. The *Voronoi cell* of a vertex v, denoted by $\mathrm{Vor}(v)$, is the set of all vertices u for which $c(u) = c(v)$. Additionally, we assign to each cell a random rank, so that there is a uniformly random total order on the cells; note carefully that the rank of a cell thus differs from the rank of its center (which is given by its identifier, which is not assigned randomly). We remark that we can determine the rank of the cell from the shared randomness and the cell's identifier, for which we simply use the identifier of its center.

The Voronoi cells are partitioned into clusters which are classified into a couple of categories as described next.

Singleton Clusters. For each Voronoi cell, consider the BFS tree spanning it, which is rooted at the respective center. For every $v \in V$, let $p(v)$ denote the *parent* of v in this BFS tree. If v is a center then $p(v) = v$. For every $v \in V \setminus S$, let $T(v)$ denote the subtree of v in the above-mentioned BFS tree when we remove the edge $\{v, p(v)\}$; for $v \in S$, $T(v)$ is simply the entire tree. Now consider a Voronoi cell. If the cell contains at most L vertices, then the *cluster* of all the vertices in the Voronoi cell is the cell itself. Otherwise, there are two cases. If $T(v)$ contains more than L vertices, then we say that v is *heavy* and define the cluster of v to be the singleton $\{v\}$. Otherwise, we say that v is *light* and its cluster is defined as follows.

Non-singleton clusters. Observe that if v is light then it has a unique ancestor u (including v) such that u is not heavy and $p(u)$ is heavy. We define the cluster of v to consist of $T(u)$ and possibly additional subtrees, $T(u')$, where u' is a also a child of $p(u)$ (in $T(p(u))$, as described next.

We begin with some definitions and notations. In order to determine the cluster of u (which is also the cluster of v) consider transforming the heavy vertex $r = p(u)$ into a binary tree which we call *the auxiliary tree of* r, B_r, as follows. B_r is rooted at r and has i complete layers where i is such that $2^i < \deg(r)$ and $2^{i+1} \geq \deg(r)$. These layers consist of *auxiliary vertices*, namely they do not correspond to vertices in G. We then add another layer to B_r consisting of the neighbors of r, sorted from left to right according to their index in $N(r)$. Note that except from the root and the vertices at the last layer of B_r, all vertices in B_r are auxiliary vertices. This completes the definition of B_r. For each vertex $x \in B_r$ we define $B_r(x)$ to be the subtree of B_r rooted at x. We define $S(x) \stackrel{\text{def}}{=} B_r(x) \cap N(r)$, namely $S(x)$ is the set of vertices of $N(r)$ which are in the subtree of B_r rooted at x. The descendants of x, denoted by the set $D(x)$, are defined to be the union of the vertices in $T(y)$ for every $y \in S(x)$, namely $D(x) \stackrel{\text{def}}{=} \bigcup_{y \in S(x)} T(y)$. The weight of x is defined to be the number of vertices in $D(x)$, namely, $w(x) \stackrel{\text{def}}{=} |D(x)|$.

We are now ready to define the cluster of u. Let $z(u)$ be the unique ancestor of u in B_r (including r), z, for which $w(z) \leq L$ and $w(p(z)) > L$ (where $p(z)$ denotes the parent of z in B_r). The cluster of u (and v) is defined to be the set $D(z)$. This completes the description of how the Voronoi cell is partitioned into clusters.

Special vertices. In order to bound the number of clusters (see Section A.3) we shall use the following definitions.

▶ **Definition 22** (Special vertex). *We say that a vertex u is* special *if $|T(u)| > L$ and for every child of u in $T(u)$, t, it holds that $|T(t)| \leq L$.*

Analogously we define special auxiliary vertex as follows.

▶ **Definition 23** (Special auxiliary vertex). *We say that an auxiliary vertex y is a* special auxiliary vertex *if either of the following conditions hold:*

1. *y is a parent of a (non auxiliary) vertex v which is heavy. In this case we say that y is a type (a) special vertex.*
2. *$w(y) > L$ and for every child of y, t, it holds that $w(t) \leq L$. In this case we say that y is a type (b) special vertex.*

For a cluster C, let $c(C)$ denote the center of the vertices in C (all the vertices in the cluster have the same center). Let $\mathrm{Vor}(C)$ denote the Voronoi cell of the vertices in C.

A.2 The Edge Set

Our spanner, $G' = (V, E')$, initially contains, for each Voronoi cell, Vor, the edges of the BFS tree that spans Vor, i.e., the BFS tree rooted at the center of Vor spanning the subgraph induced by Vor. Clearly, the spanner spans the subgraph induced on every Voronoi cell. Next, we describe which edges we add to E' in order to connect adjacent clusters of different Voronoi cells.

Marked Clusters and Clusters-of-Clusters

Each center is *marked* independently with probability $p \stackrel{\text{def}}{=} 1/n^{1/3}$. If a center is marked, then we say that its Voronoi cell is marked and all the clusters in this cell are marked as well.

Cluster-of-clusters. For every marked cluster, C, define the *cluster-of-clusters* of C, denoted by $\mathcal{C}(C)$, to be the set of clusters which consists of C and all the clusters which are adjacent to C. Let B be a non-marked cluster which is adjacent to at least one marked cluster. Let Y denote the set of all edges such that one endpoint is in B and the other endpoint belongs to a marked cluster. The cluster B is *engaged* with the marked cluster C which is adjacent to B and for which the edge of minimum rank in Y has its other endpoint in C.

The Edges between Clusters

By saying that we *connect* two adjacent subsets of vertices A and B, we mean that we add the minimum ranked edge in $E(A, B)$ to E'. For a cluster A, define its *adjacent centers* $\mathrm{Cen}(\partial A) \stackrel{\text{def}}{=} \{c(v) \,|\, u \in A \wedge \{u, v\} \in E\} \setminus \{c(A)\}$, i.e., the set of centers of Voronoi cells that are adjacent to A. This definition explicitly excludes $c(A)$, as there is no need to connect A to its own Voronoi cell.

We next describe how we connect the clusters. The high-level idea is to make sure that for every adjacent clusters A and B we connect A with the cluster engaging B (perhaps not directly) and vise versa. For clusters which are not adjacent to any marked cluster and hence not engaged with any cluster we make sure to keep them connected to all adjacent Voronoi cells. Formally:

1. We connect every cluster to every adjacent marked cluster.

2. Each cluster A that is not engaged with any marked cluster (i.e., no cell adjacent to A is marked) we connect to each adjacent cell.
3. Suppose cluster A is adjacent to cluster B, where B is adjacent to a marked cell. Denote by C the (unique) marked cluster that B is engaged with. We connect A with B if the following conditions hold:
   a. the minimum ranked edge in $E(A, \mathrm{Vor}(B))$ is also in $E(A, B)$
   b. $c(B)$ is amongst the $n^{1/k} \log n$ lowest ranked centers in $\mathrm{Cen}(\partial A) \cap \mathrm{Cen}(\partial C)$

A.3 Sparsity

▷ Claim 24. The number of clusters, denoted by s, is at most $|S| + O(nk \log \Delta)/L)$.

▷ Claim 25. The number of edges in E' is $O(n^{1+1/k} \cdot k^2 \log^3 n)$ with high probability.

Proof. Deferred to full version. ◁

A.4 Connectivity and Stretch

▷ Claim 26. G' is connected.

▷ Claim 27. Denote by G_{Vor} the graph obtained from G by contracting Voronoi cells and by G'_{Vor} its subgraph obtained when doing the same in G'. If the cells' ranks are uniformly random, w.h.p. G'_{Vor} is a spanner of G_{Vor} of stretch $O(k)$.

Proof. Deferred to the full version. ◁

▷ Claim 28. W.h.p., G' is a spanner of G of stretch $O(k^2)$.

Proof. Due to the promise on G, w.h.p. the spanning trees on Voronoi cells have depth $O(k)$. Hence, the claim holds for any edge within a Voronoi cell. Moreover, for an edge connecting different Voronoi cells, by Lemma 27, w.h.p. there is a path of length $O(k)$ in G'_{Vor} connecting the respective cells. Navigating with at most $O(k)$ hops in each traversed cell, we obtain a suitable path of length $O(k^2)$ in G'. ◁

A.5 The algorithm for general graphs

We use a combination of the algorithm in Section A with the algorithm by Baswana and Sen [7] which has the following guarantees.

▶ **Theorem 29** ([7]). *There exists a randomized k-round distributed algorithm for computing a $(2k-1)$-spanner $G' = (V, E')$ with $O(kn^{1+1/k})$ edges for an unweighted input graph $G = (V, E)$. More specifically, for every $\{u, v\} \in E'$, at the end of the k-round procedure, at least one of the endpoints u or v (but not necessarily both) has chosen to include $\{u, v\}$ in E'.*

We call a vertex v *remote* if the k-hop neighborhood of v contain less than L vertices. We denote by $\bar{R} \stackrel{\mathrm{def}}{=} V \setminus R$ the set of vertices which are not remote.

First Step. Run the algorithm from Section A on the subgraph induced by $\bar{R}$, i.e., $\{u, v\} \in E$ with $u, v \in \bar{R}$ is added to E' if and only if the algorithm outputs the edge.

Second Step. Run the algorithm of Baswana and Sen [7] on the subgraph $H = (V, \{\{u, v\} \in E \mid u \in R \text{ or } v \in R\})$, i.e., $\{u, v\} \in E$ with $u \in R$ or $v \in R$ is added to E' if and only if the algorithm outputs the edge.[4]

A.6 Stretch Factor

Consider any edge $e = \{u, v\} \in E \setminus E'$ we removed. If both u and v are in $\bar{R}$, then e was removed by the Algorithm from Section A, which was applied to the subgraph induced by $\bar{R}$. Applying Claim 27 to the connected component of e, we get that w.h.p. there is a path of length $O(k^2)$ from u to v in G'. If u or v are in R, by Theorem 29 there is a path of length $O(k)$ from u to v in G'.

▶ **Corollary 30.** *The above algorithm guarantees stretch $O(k^2)$ w.h.p. and satisfies that the expected number of edges in E' is $O(n^{1+1/k} \cdot k^2 \log^3 n)$*

A.7 The local implementation

In this section we prove the following theorem.

▶ **Theorem 31.** *There exists an LCA that given access to an n-vertex simple undirected graph G, with high probability constructs a $O(k^2)$-spanners with $\tilde{O}(n^{1+1/k})$ edges in expectation. The probe complexity and time complexity are $O(n^{2/3}\Delta^2)$. Moreover, the algorithm access the graph only by* `ALL_NBR` *queries (and performs $O(n^{2/3}\Delta)$ such queries).*

Proof. The local implementation of the algorithm which is described in the previous section is listed in Algorithm 3. The correctness of the algorithm follows from the previous sections. We shall prove that its complexity is as claimed.

The local implementation for remote vertices. For Step 1, we need to determine for both u and v if they are remote. Recall that a vertex u is remote if its k-hop neighborhood contains less than L vertices. Therefore, we can decide for any vertex u whether it is in R with at most L `ALL_NBR` probes. Thus the probe and time complexity is $O(L\Delta)$. If either u or v are remote then we need to determine for each vertex in their k-hop neighborhood whether it is remote or not. If either u or v are remote then the k-hop neighborhood of each of them contain at most $L\Delta$ vertices. This follows from the fact that the size of the k-hop neighborhood of v is at most Δ factor bigger from the k-hop neighborhood of u and vice versa. Thus, we need to call `ALL_NBR` at most $L^2\Delta$ times for this step. Hence, we obtain that the probe and time complexity of this step is $O(L^2\Delta^2)$, in total.

If $u, v \in \bar{R}$, namely, when both u and v are non-remote, the algorithm proceeds as in Section A.1. We next describe the local implementation of the algorithm for this case.

Finding the center and reconstructing the BFS tree. We first analyse the probe and time complexity of determining the center of a vertex. Given a vertex v we perform a BFS from v layer by layer and stop at the first layer in which we find a center or after exploring at least L vertices. Let i denote the layer in which the execution of the BFS stops. It follows that up to layer $i-1$ we explored strictly less than L vertices. Thus this step can be implemented by $O(L)$ calls to `ALL_NBR`. In particular, the probe and time complexity of finding the center is $O(L\Delta)$.

[4] The algorithm is described for connected graphs; we simply apply it to each connected component of H.

Algorithm 3 LCA for constructing $O(k^2)$-spanners.

Input: $\{u, v\} \in E$

Output: whether $\{u, v\}$ is in E' or not.

1. If u or v are in R, simulate the algorithm of Baswana and Sen at u and v when running it on the connected component of u and v in the subgraph H (see Section A.5). Return **YES** if either u or v has chosen to include $\{u, v\}$ and **NO** otherwise.
2. Otherwise, $u, v \in \bar{R}$ and we proceed according to Section A.1, where all nodes in R are ignored:
   a. If $\mathrm{Vor}(u) = \mathrm{Vor}(v)$, return **YES** if $\{u, v\}$ is in the BFS tree of $\mathrm{Vor}(u)$ and **NO** otherwise.
   b. Otherwise, let Q and W denote the clusters of u and v, respectively. Return **YES** if at least one of the following conditions hold for $A = Q$ and $B = W$, or symmetrically, for $A = W$ and $B = Q$, and **NO** otherwise.
      i. A is a marked cluster and $\{u, v\}$ has minimum rank amongst the edges in $E(A, B)$.
      ii. A is not engaged with any marked cluster. Namely, all the clusters which are adjacent to A are not marked. In this case, we take $\{u, v\}$ if it has minimum rank amongst the edges in $E(A, \mathrm{Vor}(B))$.
      iii. There exists a marked cluster C such that B is engaged with C, and the following holds:
         - $\{u, v\}$ has minimum rank amongst the edges in $E(A, \mathrm{Vor}(B))$.
         - The cell $\mathrm{Vor}(B)$ is amongst the $n^{1/k} \log n$ minimum ranked cells in $\mathrm{Cen}(\partial A) \cap \mathrm{Cen}(\partial C)$

We observe that at the same cost we also determine the path from $c(v)$ to v in the BFS tree rooted at $c(v)$ as follows. The parent of v in the tree is the neighbour of v that has minimum id amongst all neighbour of v that are closer than v to $c(v)$. Similarly, we can determine the parent of the parent of v and so on until we reach $c(v)$.

Determining if a vertex is heavy. In order to reconstruct the clusters we need to be able to determine if a vertex is heavy or not. Recall that a vertex v is heavy if $|T(v)| > L$. We explore $T(v)$ by performing a find center procedure on all the neighbours of v and then continuing recursively on all the neighbours of v that belong to $\mathrm{Vor}(v)$. Since finding the center takes $O(L)$ calls to `ALL_NBR`, we conclude that we can determine whether v is heavy or light by using $O(L^2\Delta)$ calls to `ALL_NBR`. This follows from the fact that when we partially or completely reveal $T(v)$, we need to find the center of at most $L\Delta$ vertices. Thus, the overall probe and time complexity for this step is $O(L^2\Delta^2)$.

Reconstructing the clusters. Given a vertex v we reconstruct its cluster as follows. First, perform a find-center operation on v and let $v := u_0, u_1, \ldots, u_d$ be the path to the center. We then determine if v is heavy using the prior procedure (and if so we are done). Otherwise, iteratively find $T(u_i)$ for $i \in [d]$ (where we do not search down the path that we have already explored), terminating the search when $T(u_i) > L$. In this case, construct the tree of special vertices below u_i and again find the first ancestor of u_{i-1} in this tree that is heavy, and let the cluster be the children of the predecessor special vertex. As we ultimately explore only $O(L\Delta)$ vertices, this results in $O(L\Delta)$ calls to find-center, which results in at most $O(L^2\Delta)$ calls to `ALL_NBR`.

Determining the cells adjacent to clusters. For Step 2 we need to reconstruct the cluster of u, the cluster of v, and the clusters that u and v are engaged with; this takes $O(L^2\Delta)$ calls to `ALL_NBR`. In addition, for each of these clusters C, we need to determine the center of each vertex adjacent to C. Since the size of the clusters is bounded by L, the number of vertices adjacent to C is at most $L\Delta$. Therefore the number of calls to find-center is at most $L\Delta$. This likewise requires $O(L^2\Delta)$ calls to `ALL_NBR` and overall $O(L^2\Delta^2)$ probes and time.

We conclude that we can perform all necessary checks to decide whether $\{u, v\} \in E'$ or not using $O(L^2\Delta)$ calls to `ALL_NBR` which invokes $O(L^2\Delta^2)$ neighbour probes. By the analysis above, the time complexity is $O(L^2\Delta^2)$ as well. ◀