


Nearly Optimal Local Algorithms for Constructing Sparse Spanners of Clusterable Graphs

Reut Levi  

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

Moti Medina  

Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

Omer Tubul 

Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

Abstract

In this paper, we study the problem of locally constructing a sparse spanning subgraph (LSSG), introduced by Levi, Ron, and Rubinfeld (ALGO'20). In this problem, the goal is to locally decide for each $e \in E$ if it is in G' where G' is a connected subgraph of G (determined only by G and the randomness of the algorithm). We provide an LSSG that receives as a parameter a lower bound, ϕ , on the conductance of G whose query complexity is $\tilde{O}(\sqrt{n}/\phi^2)$. This is almost optimal when ϕ is a constant since $\Omega(\sqrt{n})$ queries are necessary even when G is an expander. Furthermore, this improves the state of the art of $\tilde{O}(n^{2/3})$ queries for $\phi = \Omega(1/n^{1/12})$.

We then extend our result for $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graphs and provide an algorithm whose query complexity is $\tilde{O}(\sqrt{n} + \phi_{\text{out}}n)$ for constant k and ϕ_{in} . This bound is almost optimal when $\phi_{\text{out}} = O(1/\sqrt{n})$.

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Mathematics of computing \rightarrow Graph algorithms; Theory of computation \rightarrow Graph algorithms analysis

Keywords and phrases Locally Computable Algorithms, Sublinear algorithms, Spanning Subgraphs, Clusterable Graphs

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.60

Category RANDOM

Funding *Reut Levi*: The author was supported by the Israel Science Foundation under Grant 1867/20.

Moti Medina: The author was supported by the Israel Science Foundation under Grants 867/19 and 554/23.

Omer Tubul: The author was supported by the Israel Science Foundation under Grants 867/19 and 554/23.

1 Introduction

When dealing with huge graphs, several practical constraints arise: (i) **Memory Limitations**: It is often impractical or infeasible to store the entire graph in the local memory of a processing unit. (ii) **Algorithmic Efficiency**: Due to the graph's size, running linear-time (or even slower) algorithms becomes challenging. (iii) **Parallel Computation**: Relying on a single processing unit for computations can be inefficient. The Centralized Local model, also called Locally Computable Algorithms (LCA), was introduced by Rubinfeld et al. [27] to address these challenges. This model treats the input graph as if it is stored in a (likely distributed) database. External processing units can query this database to perform computations efficiently. The system prohibits shared memory or communication between



© Reut Levi, Moti Medina, and Omer Tubul;

licensed under Creative Commons License CC-BY 4.0

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024).

Editors: Amit Kumar and Noga Ron-Zewi; Article No. 60; pp. 60:1–60:21



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

querying processes to reduce coordination overhead. Instead, shared randomness accompanies input access. The approach involves running sublinear-time algorithms to extract global graph properties or locally examining the input graph as needed by applications.

One of the problems studied in this model is locally constructing a sparse spanning subgraph of a connected input graph. It was introduced and formalized by Levi, Ron, and Rubinfeld [17] as follows.

► **Definition 1** ([17]). *An algorithm \mathcal{A} is a Local Sparse Spanning Graph (LSSG) algorithm if, given $n \geq 1$, $\varepsilon > 0$, a sequence of random bits $r \in \{0, 1\}^*$ and query access to the incidence-lists representation of a connected graph $G = (V, E)$ over n vertices,¹ it provides oracle access to a subgraph $G' = (V, E')$ of G such that:*

- (1) G' is connected, and
- (2) $|E'| \leq (1 + \varepsilon) \cdot n$ with probability at least $1 - 1/\Omega(n)$, where E' is determined by G and r .^{2 3}

As observed in [17], if we insist that G' should have the minimum number of edges sufficient to span G , namely, that G' be a spanning tree, then the task cannot be performed by an LCA in general without inspecting almost all of G . On the other hand, even under the above relaxation, [17] showed that this task requires $\Omega(\sqrt{n})$ queries.⁴ They complimented this negative result with an almost tight upper bound that works under the promise that the input graph expands extremely fast. In particular, their algorithm is tight when the growth rate of their graph is $\Omega(d)$ for small sets (of size roughly \sqrt{n}), where d is the maximum vertex degree. In fact, their result achieves a sublinear query complexity only when the growth rate (on small sets) is at least $d^{1/2+1/\log n}$.⁵ This raises the question *whether it is possible to obtain a similar result also for graphs whose growth rate is just a small constant?* (which, in particular, may not depend on d). Namely, for graphs, which are just good expanders. We answer this question affirmatively by showing almost tight results for expanders. Moreover, our upper bound works for general graphs and receives as a parameter a lower bound on the conductance of the graph, defined as follows.

► **Definition 2** (Graph Conductance). *Let $G = (V, E)$ be an undirected graph with maximum vertex degree d . Let $S \subseteq V$ denote a nonempty subset of V , then the conductance of S w.r.t. G is $\phi_G(S) \stackrel{\text{def}}{=} \frac{e(S, V \setminus S)}{d|S|}$, where $e(A, B) \stackrel{\text{def}}{=} |\{\{a, b\} \in E \mid a \in A, b \in B\}|$, for $A, B \subseteq V$. The conductance of G , $\phi(G)$ is then defined as*

$$\phi(G) \stackrel{\text{def}}{=} \min_{\substack{S \subseteq V \\ |S| \leq |V|/2}} \phi_G(S). \quad (1)$$

To explain why conductance comes into play in our algorithm, we first describe a common paradigm for constructing sparse spanning subgraphs and spanners. In many cases, the paradigm is to select a random set of vertices, also referred to as *centers*, and to partition the vertices of the graphs into Voronoi cells according to this selection of centers. Usually, it is easy to span the Voronoi cells (assuming the centers are selected u.a.r.), and the challenging

¹ Namely, the algorithm can query each vertex $v \in V$ and an index i , who is the i th neighbor of v (where if v has less than i neighbors, then a special symbol is returned, e.g., “no neighbor”).

² E' is determined only by G and r and not from, e.g., the queries made to the oracle or their order.

³ We say that an event occurs *with high probability* (w.h.p) if it occurs with probability at least $1 - 1/\Omega(n)$.

⁴ One way to show this is by reducing the problem of testing cycle-freeness to the LSSG problem.

⁵ To illustrate this drawback, note that even for $d = 5$ and growth rate = 2 their algorithm is not guaranteed to be sublinear.

part is to preserve the connectivity between Voronoi cells. This is also the approach taken by [17] (and generalized in [12, 23, 2]). To bypass the challenging part of connecting the Voronoi cells, we aim to choose centers that are initially connected and distributed nearly uniformly. To this end, we select the centers by performing random walks from a single vertex and taking the endpoints of these walks to be the set of centers. We set the length of the random walks to be at least the mixing time of the graph, which, in turn, depends on the conductance of the graph, so the selected centers are distributed almost uniformly. We then add the edges traversed by the random walks to our constructed subgraph, so it only remains to span the Voronoi cells (which can be done efficiently, assuming the centers are distributed almost uniformly).

Since every graph can be partitioned into expanders (see, e.g., [22]), one may wonder if this approach can be extended to work in general graphs. To make this question concrete, consider an input graph composed of two expanders (of equal size) connected with a single edge between them. While the *inner conductance* of each expander is high, the overall conductance of the graph is $O(1/n)$, and consequently, the mixing time of the graph is high. So, the question that might come to mind is whether it is possible to extend the approach mentioned above so the length of the random walks will depend on the inner-conductance of the expanders (which is a constant) and not on the conductance of the entire graph. This question becomes more interesting as the connectivity between the expanders, which is referred to as the *outer-conductance* of the expanders, grows but not to the extent in which the overall conductance is high. We show that our approach can be extended to support a wide range of inner-conductance and outer-conductance. More specifically, we extend our result to clusterable graphs as defined in the work of Gharan and Trevisan [9] and Czumaj, Peng, and Sohler [6].

► **Definition 3** (Graph Clusterability. Based by [6]). *Let $G = (V, E)$ be an undirected graph with maximum degree d . Let $n \stackrel{\text{def}}{=} |V|$. For any $S \subseteq V$, let $G[S]$ be the induced subgraph of G on the vertex set S . We say that a graph G is (k, ϕ) -clusterable, where $k \in \{1, \dots, n\}$, $\phi \in [0, 1]$, if there exists a partition of V into h sets C_1, \dots, C_h s.t. $1 \leq h \leq k$, and that for each $i \in \{1, \dots, h\}$ it holds that $\phi(G[C_i]) \geq \phi$. We refer to each C_i as a ϕ -cluster and the corresponding partition to h clusters as an (h, ϕ) -clustering. Similarly, We say that a graph G is $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable, if for each $i \in \{1, \dots, h\}$ it holds that $\phi(G[C_i]) \geq \phi_{\text{in}}$ and $\phi_G(C_i) \leq \phi_{\text{out}}$. We refer to each C_i as a $(\phi_{\text{in}}, \phi_{\text{out}})$ -cluster.⁶*

Aside from connectivity, another desirable property of G' is that it will preserve the pairwise distances between vertices. In particular, we say that the subgraph G' is an α -spanner of G if for every $u, v \in V$, $\text{dist}_{G'}(u, v) \leq \alpha \cdot \text{dist}_G(u, v)$ where $\text{dist}_{G'}(u, v)$ and $\text{dist}_G(u, v)$ denote the length of the shortest path from u to v in G' and G , respectively. We refer to α as the *stretch factor* of the spanner G' .

In the next section, we state the performances of our upper bounds in terms of their query complexity, the number of random bits they use, and the stretch factor of the obtained spanning subgraph, G' .

1.1 Our Results

Our first result is an LSSG that receives as a parameter a lower bound, ϕ , on the conductance of the graph, whose query complexity is $\tilde{O}(\sqrt{n}/\phi^2)$ for constant d , as stated next.

⁶ Note that requiring $\phi > 0$ implies that each induced cluster of G is also connected.

► **Theorem 4.** *There is an LSSG algorithm that given query access to a connected graph $G = (V, E)$ and a lower bound ϕ on $\phi(G)$, provides access to $G' = (V, E')$ such that the following holds. (1) The graph G' is a connected subgraph of G and with high probability $|E'| = n + O\left(\frac{\sqrt{n}\log^2 n}{\phi^2}\right)$. Moreover, the stretch factor of G' is $O\left(\frac{\log n}{\phi^2}\right)$. (2) The query complexity of the algorithm is $O\left(\sqrt{n} \cdot \left(\frac{\log^2 n}{\phi^2} + d^2\right)\right)$, and (3) the number of random bits it uses is $O\left(\frac{\log d \cdot \log^2 n}{\phi^2}\right)$, where d is a bound on the maximum degree of G .*

Therefore, for constant ϕ and constant d this upper bound is tight, up to polylogarithmic factors in n . Moreover, it improves the state-of-the-art upper bound for general bounded-degree graph of $\tilde{O}(n^{2/3})$ queries by Lenzen and Levi [12] if and only if $\phi > n^{-1/12}$. Consequently, we may assume that $\phi > n^{-1/12}$ throughout this paper. As a result, we obtain that $|E'| = n + o(n)$, which makes G' an ultra-sparse spanner of stretch $O(\log n/\phi^2)$.

Our next main result is stated in the next theorem.

► **Theorem 5.** *There is an LSSG algorithm that given query access to a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph $G = (V, E)$, where each cluster is of size at least $\beta \cdot n$, provides access to $G' = (V, E')$ such that the following holds. (1) The graph G' is a connected subgraph of G and with high probability $|E'| \leq n(1 + \varepsilon)$. Moreover, the stretch factor of G' is $\Theta\left(\frac{\log n}{\phi_{\text{in}}^2}\right)$. (2) The query complexity of the algorithm is $O\left(\log^2 n \cdot (\beta\phi_{\text{out}})^{-1} + n \log^2 n \cdot k^3 d^3 \phi_{\text{out}} (\varepsilon\phi_{\text{in}})^{-1}\right)$, and (3) the number of random bits it uses is $O\left(\frac{\log d \cdot \log^2 n}{\phi_{\text{in}}^2}\right)$, where d is a bound on the maximum degree of G .*

For instances where $\phi_{\text{in}}, k, \beta, \varepsilon$, and d are constants, we obtain the following corollary.

► **Corollary 6.** *There is an LSSG algorithm that given query access to a connected $(\Theta(1), \Theta(1), \phi_{\text{out}})$ -clusterable graph $G = (V, E)$, where each cluster is of size at least $\beta \cdot n$, provides access to $G' = (V, E')$ such that the following holds. (1) The graph G' is a connected subgraph of G and with high probability $|E'| \leq n(1 + \varepsilon)$. Moreover, the stretch factor of G' is $\Theta(\log n)$. (2) The query and time complexity of the algorithm is $\tilde{O}(\sqrt{n} + \phi_{\text{out}} n)$, and (3) the number of random bits it uses is $O(\log^2 n)$.*

Namely, our algorithm is nearly tight in this case as long as $\phi_{\text{out}} = O(1/\sqrt{n})$ and improves over the state-of-the-art for $\phi_{\text{out}} = O(1/n^{1/3})$.

1.2 Overview of Our Algorithms

1.2.1 The case of a single cluster

We begin by describing our algorithm for $k = 1$ for the case where the graph is rapidly mixing, namely, that its mixing time, τ , is $O(\log n)$. We first describe the algorithm from a global point of view. The algorithm picks an arbitrary vertex as a *primary-center*. It then performs $\tilde{\Theta}(\sqrt{n})$ lazy random walks of length τ from that center, where τ denotes the mixing time of the graph. The end-vertex of each one of these walks is added to the set of *secondary centers*. The edges traversed by these random walks are added to E' . Consequently, there is a path of length at most 2τ between every pair of secondary centers. In the second step, the graph's vertex set is partitioned into Voronoi-cells with respect to the selected secondary centers. Namely, each vertex joins the cell of its closest secondary center (breaking ties by ids). A spanning tree of each Voronoi-cell is then added to the spanner. The specific spanning tree added is rooted at the secondary center, where the path from each vertex to the root has the

least lexicographical order. As shown by [17], it is possible to reconstruct the edges incident to a vertex v in this tree at the same cost as performing the BFS exploration to find the secondary center of v . The resulting subgraph clearly spans the graph: for a pair of vertices u and v in the same Voronoi cell, there is a path between u and the respective secondary center of the cell and likewise for v ; If u and v are not in the same Voronoi cell, then their secondary centers are connected to the primary-center by paths of length at most τ . Thus the stretch-factor of the spanner is $O(\tau + \ell)$ where ℓ is an upper bound on the diameter of the Voronoi-cells, which is bounded by $O(\log n/\phi^2)$.

The local implementation proceeds as follows. On query $\{u, v\}$, the local algorithm simulates the first step of the global algorithm and returns YES if $\{u, v\}$ is an edge traversed by one of the random walks performed by the algorithm. Otherwise, it performs a BFS, layer by layer (that is, it reveals an entire layer in each step) from u until it finds the secondary center of u , likewise for v . If the centers of u and v are different, then the algorithm returns NO. Otherwise, it returns YES iff $\{u, v\}$ is an edge of the tree selected to span the Voronoi-cell of u and v (as described above).

The query and time complexity of performing the first step of the local algorithm is clearly $\tilde{O}(\sqrt{n} \cdot \tau)$. For the second step, since the length of the random walks performed in the first step is τ , the $\Theta(\sqrt{n})$ secondary centers are distributed almost uniformly in the graph. Therefore, with high probability, each vertex in the graph sees a secondary center after exploring $\tilde{O}(\sqrt{n})$ vertices. If this is not the case, we can afford to add all the edges incident to v to E' without harming the sparsity of G' while preserving the connectivity of G' .

1.3 The case of k -clusterable graphs

We next describe our algorithm for k -clusterable graphs for the case that the mixing time of each cluster is $O(\log n)$, namely, that ϕ_{in} is a constant, and ϕ_{out} is $O(1/\sqrt{n})$. The first phase of the algorithm is similar to the algorithm for a single cluster. The only difference is that we start with $\Theta(\log n)$ primary-centers (chosen uniformly at random) rather than a single one. Thus, with high probability we hit every cluster with at least one primary-center. Therefore, after the first phase, our spanner consists of edges of $\tilde{\Theta}(\sqrt{n})$ random walks, traversed from each one of the primary centers, and the edges of the spanning trees of the Voronoi-cells which are constructed with respect to the set of secondary centers (which are now originated from several primary centers). Let us call the set of secondary centers originating from the same primary-center and their respective Voronoi-cells an *artificial-cluster*. Clearly, after the first step of the algorithm, the spanner spans each of the artificial clusters (from the same reasoning as above). Our goal is to ensure that every pair of artificial-clusters with an edge in their cut in the original graph will have an edge in their cut in the spanner. To this end, we sample u.a.r. a set of $\Theta(\log n/\epsilon)$ edges and add these edges to the spanner. Let us denote this set of edges by \mathcal{T} . For every pair of artificial-clusters whose cut does not intersect the set \mathcal{T} , we add all the edges in their cut to the spanner. The rationale is that if the respective cut is large, it will intersect \mathcal{T} ; otherwise, we can afford to add its edges to the spanner.

The analysis. The challenging part in analyzing this algorithm is to show that the secondary centers are distributed (almost) uniformly in each one of the clusters. This is crucial for proving that the query complexity remains $\tilde{O}(\sqrt{n})$. More specifically, this is crucial for claiming that with high probability, each vertex sees a secondary center after exploring $\tilde{O}(\sqrt{n})$ vertices. This is where the requirement on the outer-conductance of each cluster comes into play. We show that if the outer-conductance is $O(1/\sqrt{n})$ then for a constant

fraction of the vertices in the cluster, v , the end-vertex of a random walk from v of length $O(\log n)$ is likely to be any vertex in the cluster w.p. at least $\Omega(1/n)$ except for a small set of vertices of size $O(\sqrt{n})$. This is sufficient to upper bound the query complexity of the BFS exploration to find the center.

1.4 Related Work

1.4.1 LSSG Algorithms

The problem of finding a sparse spanning subgraph in the Centralized Local model was first studied in [16, 17], where the authors show a lower bound of $\Omega(\sqrt{n})$ queries for constant ε and Δ (see also survey by Rubinfeld [26]). They also present an upper bound with nearly tight query complexity for graphs with very good expansion properties.

In [17], the authors also provide an efficient algorithm for minor-free graphs that was later improved in [19]. The algorithm presented in [19] achieves a polynomial query complexity in Δ and $1/\varepsilon$ and is independent of n . The stretch factor of this algorithm is also independent of n and depends only on Δ , $1/\varepsilon$, and the size of the excluded minor h . A more general family of graphs, hyperfinite graphs, is studied in [14]. They show an upper bound (which builds on an algorithm in [17]) that has a query complexity that is independent of n (however, super-exponential in $1/\varepsilon$). Informally, both minor-free graphs and hyperfinite graphs are families of graphs that are, roughly speaking, sufficiently non-expanding everywhere. On the other hand, they show that, for a family of graphs with expansion properties that are slightly better, any local algorithm must have a query complexity that depends on n .

The first LSSG algorithm for general (bounded degree) graphs was introduced in [12], presenting a query complexity of $\tilde{O}(n^{2/3} \cdot \text{poly}(1/\varepsilon, d))$ and a stretch factor of $O(\log^2 n \cdot \text{poly}(d/\varepsilon))$. Recently, Bodwin and Fleischmann [4] introduced an Adjacency Oracle for a Spanning Subgraph of $(1 + \varepsilon)n$ edges for general (non-bounded degree) graphs that works in $\tilde{O}(n/\varepsilon)$ time, hence sublinear in the number of edges on a dense graph. Adjacency Oracles are closely related to LCA, except that Adjacency Oracles are allowed to perform a centralized pre-processing but demand a query time of $\tilde{O}(1)$. Their Adjacency Oracle implies an LSSG algorithm for general (non-bounded degree) graphs in $\tilde{O}(n)$ time, which works by constructing an Adjacency Oracle and uses it once for each query.

For more related work on LCAs for spanners, graph clustering, and LCAs for other graph problems, see Appendix A.

2 Preliminaries

In this section, we describe our main technical tools. Omitted proofs appear in Appendix B.

Notation. Throughout this paper, we consider a bounded degree (undirected) simple graph $G = (V, E)$, where $V = [n]$ and its maximum degree is $d = \max_{v \in V} d_G(v)$, where $d_G(v)$ denotes the degree of v w.r.t. the graph G . The identifier (ID in short) of a vertex $v \in V$ is simply v . For each $A \subseteq V$, we define $N_G(A)$ to be the number of neighbors of A *outside* of A in G , i.e., $N_G(A) \stackrel{\text{def}}{=} |\{v \in V \setminus A \mid \{u, v\} \in E, u \in A\}|$. When the graph G is clear from the context, we omit the subscript G . For a vertex v , we call the set of vertices of distance at most ℓ from v the ℓ -ball around v . Let $\Gamma_h(v)$ denote the minimum size ball around v that contains at least h vertices. Since the maximum degree of the graph is bounded by d , it holds that $h \leq |\Gamma_h(v)| \leq (h - 1)d$.

The total order over the vertices induces a total order (ranking) ρ over the edges of the graph in the following straightforward manner: for any $\{u, v\}, \{u', v'\} \in E$, $\rho(\{u, v\}) < \rho(\{u', v'\})$ if and only if $\min\{u, v\} < \min\{u', v'\}$ or $\min\{u, v\} = \min\{u', v'\}$ and $\max\{u, v\} < \max\{u', v'\}$. The total order over the vertices also induces an order over those vertices visited by a Breadth First Search (BFS) starting from any given vertex v , and whenever we refer to a BFS, we mean that it is performed according to this order. Instead of writing $\log_2(\cdot)$, we use $\log(\cdot)$. We use \tilde{O} for $\tilde{O}(x) = O(x) \cdot \text{poly}(\log x)$.

2.1 Mixing-Time of Regular Graphs

Let $G = (V, E)$ be an undirected and d -regular graph (where self-loops and multiple edges are allowed). The *Adjacency Matrix of G* , denoted by $A(G)$ is real and symmetric and so it has n real eigenvalues $\lambda_1 \geq \lambda_2 \dots \geq \lambda_n$ where $\lambda_1 = d$ and $\lambda_1 > \lambda_2$ iff G is connected. A d -regular graph on n vertices is called an (n, d, α) -graph if $|\lambda_2|, |\lambda_n| \leq \alpha d$. Define $\lambda(G) = \max(|\lambda_2|, |\lambda_n|)$ and $\alpha(G) = \lambda(G)/d$. Thus, every d -regular graph on n vertices is an $(n, d, \alpha(G))$ -graph. Let $\hat{A}(G) = \frac{1}{d}A(G)$ denote the *normalized adjacency matrix of G* . Note that $\hat{A}^t \vec{p}$ is the distribution on the vertices resulting from random walks of length t w.r.t. the initial vertex distribution \vec{p} . We shall use the following (folklore; see, e.g., [11, 10]) theorem and lemmas.

► **Theorem 7.** *Let G be an (n, d, α) -graph with the normalized adjacency matrix \hat{A} . Then for any distribution vector \vec{p} and any positive integer t : $\|\hat{A}^t \vec{p} - \vec{u}\|_1 \leq \sqrt{n} \cdot \alpha^t$, where \vec{u} denotes the uniform distribution vector, i.e., $\vec{u} \stackrel{\text{def}}{=} \frac{1}{n} \cdot \vec{1}$.*

One can obtain from Theorem 7 an upper bound, t , such that random walks of length at least t (independent of the initial vertex distribution) yield a distribution on the vertices that is almost uniform. We denote this upper bound by $\tau(G)$ and refer to it as the *mixing time of G* . When the graph G is evident from the context, we denote the mixing time by τ .

► **Corollary 8.** *Let G be an (n, d, α) -graph with the normalized adjacency matrix \hat{A} . For any $\tau \geq \frac{\log(2n^{3/2})}{\log(1/\alpha)}$ and for any distribution vector \vec{p} it holds that $(\hat{A}^\tau \vec{p})_i \in [\frac{1}{2n}, \frac{3}{2n}]$, for every $i \in [n]$.*

2.2 Mixing-Time of General Bounded Degree Graphs

Given a d -bounded degree graph $G = (V, E)$, we would like to relate its mixing time to $\phi(G)$. For this purpose, we define $(G)_{\text{reg}}^{2d} = (V, R)$ where R contains all the edges in E and self-loops. In particular we add $2d - d_G(v)$ self-loops to each vertex $v \in V$. Thus, $(G)_{\text{reg}}^{2d}$ is a $2d$ -regular graph. When d is clear from the context, we use $(G)_{\text{reg}}$ to denote $(G)_{\text{reg}}^{2d}$.

We shall use the following classical results to relate $\phi(G)$ and the mixing-time of $(G)_{\text{reg}}$.

► **Theorem 9** (Perron-Frobenius, Symmetric Case (see e.g., [30])). *Let G be a connected (weighted) graph. Let $A(G)$ be its adjacency matrix, and let $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ be its eigenvalues. Then,*

- (1) λ_1 has a strictly positive eigenvector,
- (2) $\lambda_1 \geq -\lambda_n$, and
- (3) $\lambda_1 > \lambda_2$.

► **Lemma 10** (Cheeger's Inequality [5]). *Let G be a d -regular graph with eigenvalues $\lambda_1 \geq \lambda_2 \dots \geq \lambda_n$. Then, $\frac{d-\lambda_2}{2d} \leq \phi(G) \leq \frac{\sqrt{2d(d-\lambda_2)}}{d}$.*

We first prove (following [29, P. 106]) that the maximum eigenvalue in absolute value of $A((G)_{\text{reg}})$ is λ_2 .

▷ **Claim 11.** Let G be a connected, d -bounded degree graph. Let $\{\lambda_i\}_i$ denote the eigenvalues of $A((G)_{\text{reg}})$, then $\max(|\lambda_2|, |\lambda_n|) = \lambda_2$.

The following claim, combined with Corollary 8 provides an upper bound on the mixing-time of $(G)_{\text{reg}}$.

▷ **Claim 12.** Let $G = (V, E)$ be a connected d -bounded degree graph on n vertices, then $(G)_{\text{reg}}$ is an $\left(n, 2d, \left(1 - \frac{\phi^2(G)}{2}\right)\right)$ -graph.

The next Corollary follows from Corollary 8 and Claim 12.

► **Corollary 13.** Let $G = (V, E)$ be a connected d -bounded degree graph on n vertices and let $v \in V$. If we perform a random-walk in $(G)_{\text{reg}}$, starting from v , of length at least $\tau(G) \stackrel{\text{def}}{=} \frac{\log(2n^{3/2})}{\log\left(\left(1 - \frac{\phi^2(G)}{2}\right)^{-1}\right)}$, then the probability this walk ends in u is at least $1/(2n)$ for every $u \in V$.

3 LSSG for a Single Cluster

In this section, we prove Theorem 4, in particular, we present and prove correct our LSSG algorithm for the case where the input graph is a single cluster, i.e., a $(1, \phi)$ -clusterable graph. As such, our algorithm receives ϕ as a parameter where ϕ is a lower bound on the conductance of the input graph. We first describe our algorithm from a global point of view (Subsection 3.1) and then explain how this global algorithm can be implemented locally (Subsection 3.3).

3.1 The Global Algorithm for a Single Cluster

The algorithm has two stages. In the first stage, it picks an arbitrary vertex, \mathcal{P} , as the *primary-center*. It then performs $r \stackrel{\text{def}}{=} \tilde{\Theta}(\sqrt{n})$ s -wise independent lazy random walks from \mathcal{P} and takes the end-vertices of these walks to be the set of *secondary centers* (Steps 4-5). As defined in Section 2, when performing a lazy random walk, in each step, an edge is selected uniformly and independently with probability $1/2d$ (recall that we add $2d - d_G(v)$ self-loops to each vertex v to obtain a $2d$ -regular graph). The edges traversed by the random walks (which are not self-loops) are added to the spanner (Step 6). This guarantees that all the secondary centers are connected in the spanner. The length of the random walks τ is determined by Corollary 8 to ensure that the secondary centers are distributed almost uniformly in V .

In the second stage, the algorithm constructs spanning trees of the Voronoi cells, which are defined w.r.t. the secondary centers (Steps 7-8). Since the secondary centers are distributed almost uniformly in V , with high-probability, all vertices $v \in V$ will have a secondary center in $\Gamma_h(v)$ where $h \stackrel{\text{def}}{=} \sqrt{n}$. If this is not the case (i.e., no secondary center is found in $\Gamma_h(v)$), then all the edges incident to v are added to the spanner (Step 9). Note that removing Step 9 yields an LSSG algorithm that spans G w.h.p., while including it yields an algorithm that outputs a spanning subgraph with probability 1.

■ **Algorithm 1** Globally Computing a Sparse Spanning Subgraph.

Input: A graph $G = (V, E)$ with conductance at least ϕ and maximum degree bounded by d .

Output: $G' = (V, E')$ is a sparse spanning subgraph of G w.h.p.

- 1 $E' \leftarrow \emptyset$.
- 2 Select a *primary-center* $\mathcal{P} \in V$ arbitrarily.
- 3 Let $r \stackrel{\text{def}}{=} \Theta(\sqrt{n} \cdot \log n)$, $h \stackrel{\text{def}}{=} \sqrt{n}$, $s \stackrel{\text{def}}{=} \Theta(\log n)$, $\tau \stackrel{\text{def}}{=} \Theta\left(\frac{\log n}{\phi^2}\right)$.
- 4 Perform r s -wise independent lazy random walks $\mathcal{R} \stackrel{\text{def}}{=} \{\rho_1, \dots, \rho_r\}$ where $\rho_i = (v_1^{(i)} = \mathcal{P}, \dots, v_\tau^{(i)})$.
- 5 Let \mathcal{S} denote the set of *secondary centers* defined as follows $\mathcal{S} \stackrel{\text{def}}{=} \bigcup_{i \in [r]} \{v_\tau^{(i)}\}$. // The end-vertices of the random-walks are selected to be the set of secondary centers.
- 6 $E' \leftarrow E(\mathcal{R}) \cap E$. // where $E(\mathcal{R})$ denotes the edge set of the random walks in \mathcal{R} . Add all the edges traversed by the random walks (that are not self-loops) in \mathcal{R} to the set E' .
- 7 Every vertex $v \in V$ assigns itself to the closest secondary center in $\Gamma_h(v)$ denoted by $\sigma(v)$ (break ties by choosing the one with the smallest ID). If $\Gamma_h(v) \cap \mathcal{S} = \emptyset$ then set $\sigma(v) = \perp$.
- 8 For each $s \in \mathcal{S}$, let $\text{Vor}(s) \stackrel{\text{def}}{=} \{v \in V \mid \sigma(v) = s\}$. Let $\text{STree}(s)$ denote the BFS tree that spans $\text{Vor}(s)$. $E' \leftarrow E' \cup \text{STree}(s)$.
- 9 Add to E' all the edges incident to vertices, v , such that $\sigma(v) = \perp$.
- 10 **return** $G' = (V, E')$.

3.2 Correctness of the Global Algorithm for a Single Cluster

To prove the correctness of the global algorithm, we claim that w.h.p. when the algorithm stops, there is a path in G' from each vertex to a secondary center and that this path is short; We begin by proving that w.h.p., every vertex $v \in V$ has a secondary center in $\Gamma_h(v)$. We shall use the following concentration bound, which applies for s -wise independent random variables defined as follows.⁷

► **Definition 14** (s -wise independence). *A set of discrete random variables X_1, \dots, X_n are called s -wise independent if for any set $I \subseteq \{1, \dots, n\}$ with $|I| \leq s$ and any values x_i we have $\Pr[\bigwedge_{i \in I} (X_i = x_i)] = \prod_{i \in I} \Pr[X_i = x_i]$.*

Note that the random walks that the algorithm performs are also random variables. In particular, this is a set of s -wise independent random walks, i.e., any subset of size at most s of walks (out of the r walks that the algorithm performs) are mutually independent.

► **Theorem 15** (Theorem 5(III) in [28]). *If X is a sum of s -wise independent random variables, each of which is in the interval $[0, 1]$ with $\mu = \mathbb{E}(X)$, then For $\delta \leq 1$ and $s \leq \lfloor \delta^2 \mu e^{-1/3} \rfloor$, it holds that $\Pr[|X - \mu| \geq \delta \mu] \leq e^{-\lfloor s/2 \rfloor}$.*

▷ **Claim 16.** With high probability, for every $u \in V$, $\Gamma_h(u) \cap \mathcal{S} \neq \emptyset$.

Proof. Fix \mathcal{P} and let v be a vertex in V . Consider a random walk on $(G)_{\text{reg}}$ (or alternatively, a lazy random-walk on G) that starts at \mathcal{P} . By Corollary 13, when performing a random-walk from \mathcal{P} of length τ in $(G)_{\text{reg}}$, the probability that v is the end-vertex of the random walk is at

⁷ Having bounded independence reduces the number of random bits used by the local implementation of this algorithm, as shown in the next section.

least $\frac{1}{2n}$. Let \mathcal{E}_i^v denote the event that the i -th random walk ended at v , namely the event that $v^{(i)} = v$, and let X_i^v denote the indicator variable for this event. Thus, $\Pr[\mathcal{E}_i^v] = \mathbb{E}[X_i^v] \geq \frac{1}{2n}$. Let $H \subseteq V$ be a set of cardinality h and let \mathcal{E}_i^H denote the event that the i -th random walk ended at H . Let X_i^H denote the indicator variable for this event. Since the events $\{\mathcal{E}_i^v\}_{v \in V}$ are disjoint (i.e. mutually exclusive) we obtain that $\mathbb{E}[X_i^H] = \Pr[\mathcal{E}_i^H] = \sum_{v \in H} \Pr[\mathcal{E}_i^v] \geq \frac{h}{2n}$. By linearity of expectation, $\mathbb{E}\left[\sum_{i \in [r]} X_i^H\right] \geq \frac{hr}{2n} = \Theta(\log n)$, where the last equality follows since $h = \sqrt{n}$. Since we perform r s -wise independent random walks from \mathcal{P} , the random variables $\{X_i^H\}_{i \in [r]}$ are also s -wise independent. Set $Y \stackrel{\text{def}}{=} \sum_{i \in [r]} X_i^H$ and $\mu \stackrel{\text{def}}{=} \mathbb{E}[Y] = \Theta(\log n)$. By Theorem 15, $\Pr[|Y - \mu| \geq \mu/2] \leq e^{-\lfloor s/2 \rfloor}$ where $s \stackrel{\text{def}}{=} \lfloor \frac{1}{4} \left(\frac{hr}{2n}\right) e^{-1/3} \rfloor \leq \lfloor \frac{1}{4} \mu e^{-1/3} \rfloor$. If $|Y - \mu| < \mu/2$, then $Y > \mu/2 \geq 1$. Thus, the probability that none of the r random-walks ends in H is at most $e^{-\lfloor s/2 \rfloor} = 1/n^c$, where c is determined by the exact setting of r . Hence, by the union bound over all vertices, we obtain that with high probability $\Gamma_h(v) \cap \mathcal{S} \neq \emptyset$ for every $v \in V$. \triangleleft

The following claim (the proof of which appears in Appendix C) argues that BFS trees are of height logarithmic in the number of their vertices.

\triangleright **Claim 17.** Let $G = (V, E)$ be a d -bounded degree graph and let ϕ be a lower bound on $\phi(G)$. For any $v \in V$ and $x \leq |V|/2$, the ℓ -ball centered at v contains at least x vertices provided that $\ell \geq \frac{\log x}{\log(1+\phi)}$.

Claim 17 implies that the paths' length from every vertex to its secondary center is logarithmic in n , formalized as follows.

\blacktriangleright **Corollary 18.** Let $s \in \mathcal{S}$. Then the depth of $\text{STree}(s)$ is at most $\frac{\log h}{\log(1+\phi)}$.

We now prove the main theorem of this section, which bounds the stretch factor and size of the graph, $G' = (V, E')$ obtained by Algorithm 1. In particular, for constant ϕ it follows that $|E'| = n + o(n)$, and the stretch is $\Theta(\log n)$.

\blacktriangleright **Theorem 19.** Algorithm 1 computes a Sparse Spanning Graph of G , $G' = (V, E')$, such that:

- (1) The attained stretch is $\Theta\left(\frac{\log n}{\phi^2}\right)$, and
- (2) $|E'| = n + O\left(\frac{\sqrt{n} \log^2 n}{\phi^2}\right)$ with high probability.

Proof. We first prove Item 1 of the claim. Consider an edge $\{u, v\}$ which belongs to E but not to E' . We first note that it follows that both $\sigma(u) \neq \perp$ and $\sigma(v) \neq \perp$ because otherwise $\{u, v\}$ is added to E' in Step 9. If u and v belong to the same Voronoi cell then the distance between them in G' is at most $2 \frac{\log \sqrt{n}}{\log(1+\phi)}$ (since, by Corollary 18, the distance from every vertex to its secondary center is at most $\frac{\log \sqrt{n}}{\log(1+\phi)}$). If u and v do not belong to the same Voronoi cell, then the distance in G' between their respective centers, $\sigma(u)$ and $\sigma(v)$ is at most 2τ (because the distance in G' between every secondary center to \mathcal{P} is at most τ), where $\tau \stackrel{\text{def}}{=} \frac{\log(2n^{3/2})}{\log\left(\left(1 - \frac{\phi^2}{2}\right)^{-1}\right)}$. Thus, in this case the distance in G' between u and v is at most $2 \cdot \left(\tau + \frac{\log \sqrt{n}}{\log(1+\phi)}\right)$. Since for all $x \geq 0$ it holds that $x - x^2/2 \leq \ln(1+x)$, it follows that the attained stretch is $\Theta\left(\frac{\log n}{\phi^2}\right)$, as claimed.

Item 2 of the claim follows from the construction. More specifically, at Step 6, we add at most $r \cdot \tau$ edges to E' . In Step 8, we add at most $n - 1$ edges since the Voronoi cells are vertex-disjoint. Finally, by Claim 16, with high probability, in Step 9, we do not add any edges to E' . \blacktriangleleft

3.3 Details of the Implementation of the Local Algorithm for a Single Cluster

In this section, we describe how to implement Algorithm 1 locally. We also bound the query and time complexity of the local algorithm as well as the total number of bits it uses. We conclude this section with the proof of Theorem 4.

Performing lazy random-walks in G . Performing a lazy-random walk of length ℓ in G requires at most ℓ probes to G . In each step, we select an index uniformly at random from $i \in [2d]$ and perform a neighbor-query (v, i) to reveal the i th neighbor of v , where v denotes the ID of the current vertex.⁸ If the probe returns a vertex ID, then we move to that neighbor; otherwise, the walk stays at v . We note that performing a lazy-random walk in G is equivalent to performing a random walk in $(G)_{\text{reg}}$.

Bounding the Number of Random Bits. Following [23], we shall use the classical result from [31] for obtaining random bits with bounded independence in a local manner.

► **Definition 20.** For $N, M, s \in \mathbb{N}$ such that $s \leq N$, a family of functions $\mathcal{H} = \{h : [N] \rightarrow [M]\}$ is s -wise independent if for all distinct $x_1, \dots, x_s \in [N]$, the random variables $h(x_1), \dots, h(x_s)$ are independent and uniformly distributed in $[M]$ when h is chosen randomly from \mathcal{H} .

► **Lemma 21** (Corollary 3.34 in [31]). For every $\gamma, \beta, s \in \mathbb{N}$, there is a family of s -wise independent functions $\mathcal{H}_{\gamma, \delta} = \{h : \{0, 1\}^\gamma \rightarrow \{0, 1\}^\delta\}$ such that choosing a random function from $\mathcal{H}_{\gamma, \delta}$ takes $s \cdot \max\{\gamma, \delta\}$ random bits, and evaluating a function from $\mathcal{H}_{\gamma, \delta}$ takes time $\text{poly}(\gamma, \delta, s)$.

▷ **Claim 22.** Performing r s -wise independent random walks from \mathcal{P} in $(G)_{\text{reg}}$ can be implemented by using $O(\log n(\log n + \tau \log d))$ random bits and time-complexity $r \cdot \text{poly}(\tau, \log n)$.

Proof. Performing a single random-walk in $(G)_{\text{reg}}$ of length τ from \mathcal{P} requires $O(\tau \log d)$ random bits (since we are performing τ steps and in each step, we are selecting an edge u.a.r. out of $2d$ edges). Let $\gamma \stackrel{\text{def}}{=} \Theta(\log r)$ denote the number of bits that are required to indicate the index of the random walk in $[r]$ and let $\delta \stackrel{\text{def}}{=} \Theta(\tau \log d)$ denote the number of random bits that are required for performing a single walk. To perform r s -wise independent random walks, it suffices to choose a random function from a family of s -wise independent functions, $\mathcal{H}_{\gamma, \delta}$ which takes as parameter the index of the random walk and returns δ bits. By Lemma 21 this can be done by using $s \cdot \max\{\gamma, \delta\} = O(\log n(\log n + \tau \log d))$ random bits. Furthermore, the time complexity of retrieving the bits for performing a single random walk is $\text{poly}(\gamma, \delta, s) = \text{poly}(\tau, \log n)$. The claim follows. ◁

Locally Computing a Sparse Spanning Subgraph. On query $\{u, v\}$, the local algorithm first performs the random walks as listed in Algorithm 1, Step 4. If $\{u, v\} \in E(\mathcal{R})$, then the algorithm returns YES. Otherwise, it finds $\sigma(u)$ and $\sigma(v)$. If either $\sigma(u) = \perp$ or $\sigma(v) = \perp$ then it returns YES. Otherwise, if $\sigma(u) = \sigma(v)$, then the algorithm returns YES iff $\{u, v\} \in \text{STree}(\sigma(v))$. Otherwise, if $\sigma(u) \neq \sigma(v)$, the algorithm returns NO. Finding $\sigma(v)$

⁸ A common assumption in LCA in general and hence also in LSSG's is that the algorithm knows n . Similarly, as in [17], for bounded-degree graphs, the bound on the maximum vertex degree d of the graph instance at hand is also known to the LSSG algorithm.

can be done by making $O(hd^2)$ queries. To see this, observe that the number of vertices we explore by performing a BFS, layer by layer, until we first see at least h vertices is at most $(h-1)d$. The subgraph induced on these vertices contains at most hd^2 edges, thus the query and time complexity of this step is indeed $O(hd^2)$ ⁹. Checking if $\{u, v\} \in \text{STree}(\sigma(v))$ can be implemented at the same cost. To see this, observe that when we find $\sigma(v)$ as described above, then we also reveal all the shortest paths between v and $\sigma(v)$ in G . Since we can decide which one of these paths is the path between v and $\sigma(v)$ in $\text{STree}(\sigma(v))$, we can decide if $\{u, v\}$ belongs to this path at the same cost of finding $\sigma(v)$ and $\sigma(u)$ ¹⁰. This concludes the description of the local implementation of Algorithm 1. We are now ready to prove Theorem 4.

Proof of Theorem 4. The correctness of the algorithm, that is, Item (1), follows from Theorem 19. As described above (in the analysis of the local implementation), the algorithm makes $r \cdot \tau + hd^2 = O\left(\sqrt{n} \cdot \left(\frac{\log^2 n}{\phi^2} + d^2\right)\right)$ graph queries. By Claim 22 the local implementation of Algorithm 1 uses $O(\log n(\log n + \tau \log d)) = O\left(\frac{\log d \cdot \log^2 n}{\phi^2}\right)$ random bits, which concludes the proof of the theorem. ◀

4 LSSG for Clusterable Graphs

In this section, we prove Theorem 5. First, we describe our LSSG algorithm that works under the promise that the input graph is a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph where each cluster is of size at least βn .

4.1 The Global Algorithm for Clusterable graphs

The listing of our global algorithm appears as Algorithm 2. We next describe how it proceeds, step by step. All the parameters we will mention are defined in Step 2 of the algorithm. Initially, the set of edges of the spanner, E' , is empty (Step 1). The algorithm begins with selecting p primary-centers uniformly at random from V (Step 3). It then performs r s -wise independent lazy-random walks from each primary center (Step 4). It picks the endpoints of these random-walks to be the set of *secondary-centers* (Step 5). Additionally, all the edges traversed by these random walks are added to E' (Step 4). After that, the vertices of the graph are partitioned as follows. Each one of the vertices, v , joins the cell of the secondary center, which is closest to v in $\Gamma_h(v)$, breaking ties by ID (Step 6). If $\Gamma_h(v)$ does not include a secondary center, then all the edges that are incident to v are added to E' (Step 9). In Step 10 the algorithm picks t random pairs from $V \times [d]$. For each such pair, (v, i) , such that v has an i -th neighbor, the edge $\{u, v\}$ is added to E' where u is the i -th neighbor of v (Step 10). We define an artificial-cluster to be a maximal set of vertices that agree on their primary-center (see formal definition in Step 7). For every pair of artificial-clusters such that E' does not include an edge from their cut, all the edges in the corresponding cut are added to E' (Step 11). This concludes the description of the algorithm.

4.2 Correctness of the Global Algorithm for Clusterable Graphs

To prove the correctness of the algorithm, we need to show that the obtained subgraph spans the graph with a logarithmic stretch and that it is sparse. Proving that the obtained subgraph spans the graph is relatively straightforward and follows by the design of the

⁹ This procedure appears in [17] as Algorithm **Find-Center**.

¹⁰ See more details in algorithm **Get BFS outgoing-edges endpoints** in [17].

■ **Algorithm 2** Globally Computing a Sparse Spanning Subgraph of a Clusterable Graph.

-
- Input:** A connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph $G = (V, E)$ where each cluster is of size at least $\beta|V|$.
- Output:** $G' = (V, E')$ is a sparse spanning subgraph of G w.h.p.
- 1 $E' \leftarrow \emptyset$.
 - 2 Let $p \stackrel{\text{def}}{=} \Theta(\beta^{-1} \log n)$, $r \stackrel{\text{def}}{=} \Theta(\frac{\log n}{\tau \phi_{\text{out}}})$, $s \stackrel{\text{def}}{=} \Theta(\log n)$, $\tau \stackrel{\text{def}}{=} \Theta(\frac{\log n}{\phi_{\text{in}}^2})$, and $h \stackrel{\text{def}}{=} \Theta(k\tau\phi_{\text{out}}n)$.
 - 3 Select p primary-centers $\mathcal{P} \stackrel{\text{def}}{=} \{\psi_1, \dots, \psi_p\}$ u.a.r. in V .
 - 4 From each primary-center, $\psi \in \mathcal{P}$, perform r s -wise independent lazy random-walks. Add all the edges in E traversed by these random walks to E' .
 - 5 Let \mathcal{S} denote the set of endpoints of these random-walks. We refer to \mathcal{S} as the set of secondary centers. We say that the primary-center of $v \in \mathcal{S}$ is ψ if the random-walk that ended in v started at ψ (break ties by choosing the one with the smallest ID).
 - 6 Every vertex $v \in V$ assigns itself to the closest secondary center in $\Gamma_h(v)$ denoted by $\sigma(v)$ (break ties by choosing the one with the smallest ID). If $\Gamma_h(v) \cap \mathcal{S} = \emptyset$ then set $\sigma(v) = \perp$.
 - 7 The primary-center of v is set to be the primary-center of $\sigma(v)$. The *artificial-cluster* of v is defined to be the set of all vertices whose primary-center equals the primary-center of v .
 - 8 For each $s \in \mathcal{S}$, let $\text{Vor}(s) \stackrel{\text{def}}{=} \{v \in V \mid \sigma(v) = s\}$. Let $\text{STree}(s)$ be a spanning tree of $\text{Vor}(s)$. $E' \leftarrow E' \cup \text{STree}(s)$.
 - 9 Add to E' all the edges that are incident to vertices, v , such that $\sigma(v) = \perp$.
 - 10 Select $t \stackrel{\text{def}}{=} \Theta(\varepsilon^{-1}k^2d \log n)$ pairs u.a.r. from $V \times [d]$. Let \mathcal{T} denote the set of edges that correspond to these pairs (an edge $\{u, v\}$ corresponds to the pair (v, i) if u is the i -th neighbor of v). $E' \leftarrow E' \cup \mathcal{T}$.
 - 11 Add to E' all the edges between artificial-clusters not connected by an edge in \mathcal{T} .
 - 12 **return** $G' = (V, E')$.
-

algorithm. Both the proof of the low stretch and sparsity of the obtained spanner appears in Theorem 28, where for the sparsity proof, we need to show that the number of edges that we add in Steps 4, and 8-11 is not too much. The proof that the obtained spanner is of a low stretch is the main technical challenge of this section. To this end, it is sufficient to show that w.h.p. for every $v \in V$, $\Gamma_h(v)$ includes a secondary-center (see Claim 26). To this end, we next analyze the distribution of the endpoints of the lazy-random walks that the algorithm performs.

For any subset $C \subseteq V$, we slightly abuse notation and denote by $(C)_{\text{reg}}$ the $2d$ -regular graph $(G[C])_{\text{reg}}$, where $G[C]$ is the subgraph induced on C in G .

Throughout this section, $G = (V, E)$ is a $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph, $C \subseteq V$ is a $(\phi_{\text{in}}, \phi_{\text{out}})$ -cluster of G and S is the set of self-loops that are added to $(C)_{\text{reg}}$ due to the edges in the cut between C and $V \setminus C$. Namely, for each $v \in C$, S contains $e(\{v\}, V \setminus C)$ self-loops of v .

For every $u, v \in C$, let $p^{\ell, v}(u)$ denote the probability that an ℓ -length random-walk that starts at v in $(C)_{\text{reg}}$ ends at u . Thus $\sum_{u \in C} p^{\ell, v}(u) = 1$. We let $p_{\text{bad}}^{\ell, v}(u)$ denote the probability that an ℓ -length random-walk in $(C)_{\text{reg}}$ that starts at v ends in u and traverses an edge of S . Let $p_{\text{good}}^{\ell, v}(u) \stackrel{\text{def}}{=} p^{\ell, v}(u) - p_{\text{bad}}^{\ell, v}(u)$.

▷ **Claim 23.** For at least half of the vertices $v \in C$, it holds that $\sum_{u \in C} p_{\text{bad}}^{\ell, v}(u) \leq \ell\phi_{\text{out}}$. We call such a vertex, v , useful.

Proof. We prove that if we perform an ℓ -length random-walk, ρ , in $(C)_{\text{reg}}$ from a vertex v selected u.a.r. from C , then the probability that ρ traverses an edge from S is at most $\ell\phi_{\text{out}}/2$. In other words, we will show that $\mathbb{E}[\sum_{u \in C} p_{\text{bad}}^{\ell, v}(u)] \leq \ell\phi_{\text{out}}/2$, where the expectation is taken over v which is selected u.a.r. from C . The claim will then follow by Markov's inequality.

Since $(C)_{\text{reg}}$ is $2d$ -regular and v is selected u.a.r. from C , all the vertices in ρ are distributed uniformly at random from C as well (since $P\vec{u} = \vec{u}$ for every doubly-stochastic matrix P). The probability of traversing an edge from S when we select a vertex u.a.r. from C and then take a single step from this vertex is $\frac{|S|}{2d|C|}$. This is simply because each one of the edges in S is traversed with probability $\frac{1}{|C|} \cdot \frac{1}{2d}$ (recall that the edges in S correspond to self-loops). Thus, by union bound, the probability of traversing an edge from S in one of the ℓ steps of the random walk is at most $\frac{\ell|S|}{2d|C|}$. Since $\frac{|S|}{d|C|} \leq \phi_{\text{out}}$ we obtain that this probability is at most $\ell\phi_{\text{out}}/2$, as desired. \triangleleft

The following claim relates random-walks in $(G)_{\text{reg}}$ to random-walks in $(C)_{\text{reg}}$.

\triangleright **Claim 24.** If we perform an ℓ -length random-walk in $(G)_{\text{reg}}$ from $v \in C$ then the probability that this random walk ends at $u \in C$ is at least $p_{\text{good}}^{\ell,v}(u)$.

Proof. Recall that $p_{\text{good}}^{\ell,v}(u)$ is the probability that an ℓ -length random-walk in $(C)_{\text{reg}}$ that starts at v ends in u and does not traverse an edge of S . The edge set of $(C)_{\text{reg}}$ includes parallel self-loops. If we identify each edge by its endpoints and the labels of its ports, then we get that from each vertex $v \in C$, there are exactly $(2d)^\ell$ distinct paths of length ℓ in $(C)_{\text{reg}}$. Let P^v denote the set of these paths. When we perform a random walk in $(C)_{\text{reg}}$ from v , each path in P^v occurs with probability $1/(2d)^\ell$. Moreover, for every $u \in C$, let P_u^v denote the set of paths in P^v that end at u and do not traverse an edge from S . Each path in P_u^v contributes $1/(2d)^\ell$ to $p_{\text{good}}^{\ell,v}(u)$ and each path in $P^v \setminus P_u^v$ contributes 0. Assume, w.l.o.g., that $(C)_{\text{reg}}$ is obtained from G by first converting G to $(G)_{\text{reg}}$ and then replacing the edges in the cut $E(C, V \setminus C)$ with self-loops. This way, every self-loop in $(C)_{\text{reg}}$ which does not belong to S also appears in $(G)_{\text{reg}}$ (when taking into account the port numbers). Thus, if we perform a random-walk in $(G)_{\text{reg}}$ from v , each path from P_u^v occurs with probability $1/(2d)^\ell$ as well. The claim follows. \triangleleft

\triangleright **Claim 25.** Let v be a vertex which is useful w.r.t. C . For all $u \in C$, except for at most $4\tau\phi_{\text{out}}|C|$ vertices, it holds that a τ -length random-walk in $(G)_{\text{reg}}$ from v ends at u with probability at least $(4n)^{-1}$.

Proof. Let v be a vertex, which is useful w.r.t. C , and let $\ell \in \mathbb{N}$. By Claim 23, $\sum_{u \in C} p_{\text{bad}}^{\ell,v}(u) \leq \ell\phi_{\text{out}}$. Therefore $\mathbb{E}[p_{\text{bad}}^{\ell,v}(u)] \leq \frac{\ell\phi_{\text{out}}}{|C|}$, where the expectation is taken over u selected u.a.r. from C . Thus, by Markov's inequality for at most γ -fraction of the vertices $u \in C$ it holds that $p_{\text{bad}}^{\ell,v}(u) > \frac{\ell\phi_{\text{out}}}{\gamma|C|}$.

By replacing ℓ with τ and γ with $4\tau\phi_{\text{out}}$ we obtain that for at least $(1 - 4\tau\phi_{\text{out}})$ -fraction of the vertices $u \in C$ it holds that $p_{\text{good}}^{\tau,v}(u) \geq p^{\tau,v}(u) - \frac{1}{4|C|}$. Since C is a $(\phi_{\text{in}}, \phi_{\text{out}})$ -cluster in G it follows from Corollary 13 that $p^{\tau,v}(u) \geq 1/(2|C|)$ for every u . Thus it holds that $p_{\text{good}}^{\tau,v}(u) \geq 1/(4|C|) \geq 1/(4n)$. Hence, the claim follows from Claim 24. \triangleleft

\triangleright **Claim 26.** Let $G = (V, E)$ be a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph and let C_1, \dots, C_f be a partition of G into $f \leq k$ $(\phi_{\text{in}}, \phi_{\text{out}})$ -clusters. If for every $i \in [f]$, $C_i \cap \mathcal{P}$ contains a useful vertex w.r.t. C_i , then w.h.p. for every $v \in V$, $\Gamma_h(v) \cap \mathcal{S} \neq \emptyset$.

Proof. Let $G = (V, E)$ be a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph and let C_1, \dots, C_f be a partition of G into $f \leq k$ $(\phi_{\text{in}}, \phi_{\text{out}})$ -clusters. Assume for every $i \in [f]$, $C_i \cap \mathcal{P}$ contains a useful vertex, u_i , w.r.t. C_i . Let H be a set of vertices of cardinality at least h and let $\gamma = 4\tau\phi_{\text{out}}$ where h and τ are as defined in Step 2 of Algorithm 2. By Claim 25, for every $i \in [f]$ and for all vertices $u \in C_i$ except for at most $\gamma|C_i|$ it holds that an τ -length random-walk in $(G)_{\text{reg}}$ from u_i ends at u with probability at least $(4n)^{-1}$. Let us call these vertices *good* w.r.t. C_i .

Let C_j be the dominant cluster in H . Namely, the cluster which maximizes the intersection with H . By an averaging argument $C_j \cap H \geq h/k \geq 5\tau\phi_{\text{out}}n$, where the last inequality holds for an appropriate setting of h . Since the number of vertices in C_j which are not good (w.r.t. C_j) is at most $\gamma|C_j| \leq 4\tau\phi_{\text{out}}n$, we obtain that the number of good vertices in C_j is at least $\tau\phi_{\text{out}}n$. From this point, the rest of the proof is similar to the proof of Claim 16. For $v \in V$, let \mathcal{E}_i^v denote the event that the i -th random walk from u_j ended at v and let X_i^v denote the indicator variable for this event. Thus, for every good vertex, v , w.r.t. C_j holds that $\Pr[\mathcal{E}_i^v] = \mathbb{E}[X_i^v] \geq \frac{1}{4n}$. Let \mathcal{E}_i^H denote the event that the i -th random walk from u_j ended at H . Let X_i^H denote the indicator variable for this event. Since the events $\{\mathcal{E}_i^v\}_{v \in V}$ are disjoint (i.e. mutually exclusive) we obtain that $\mathbb{E}[X_i^H] = \Pr[\mathcal{E}_i^H] = \sum_{v \in H} \Pr[\mathcal{E}_i^v] \geq \frac{\tau\phi_{\text{out}}n}{4n} = \frac{\tau\phi_{\text{out}}}{4}$. By linearity of expectation, $\mathbb{E}\left[\sum_{i \in [r]} X_i^H\right] \geq r \cdot \frac{\tau\phi_{\text{out}}}{4} = \Theta(\log n)$. Since we perform r s -wise independent random walks from u_j , the random variables $\{X_i^H\}_{i \in [r]}$ are also s -wise independent. Set $Y \stackrel{\text{def}}{=} \sum_{i \in [r]} X_i^H$ and $\mu \stackrel{\text{def}}{=} \mathbb{E}[Y] = \Theta(\log n)$. By Theorem 15, $\Pr[|Y - \mu| \geq \mu/2] \leq e^{-\lfloor s/2 \rfloor}$ where $s \stackrel{\text{def}}{=} \lfloor \frac{1}{4} \left(\frac{r\tau\phi_{\text{out}}}{4} \right) e^{-1/3} \rfloor \leq \lfloor \frac{1}{4} \mu e^{-1/3} \rfloor$. If $|Y - \mu| < \mu/2$, then $Y > \mu/2 \geq 1$. Thus, the probability that none of the r random-walks ends in H is at most $e^{-\lfloor s/2 \rfloor} = 1/n^c$, where c is determined by the exact setting of r . Thus, by union bound over all vertices, we obtain that with high probability $\Gamma_h(v) \cap \mathcal{S} \neq \emptyset$ for every $v \in V$. \triangleleft

We say that a pair of artificial-clusters are *heavy* if the number of edges in their edge-cut is at least $\varepsilon n/(2k^2)$.

\triangleright **Claim 27.** With high probability, \mathcal{T} contains an edge from the cut of every pair of heavy artificial-clusters.

Proof. Let C_1 and C_2 be a heavy pair of artificial-clusters. By definition, the number of edges in their cut is at least $\varepsilon n/(2k^2)$. Thus the probability that a pair chosen u.a.r. from $V \times [d]$ hits this cut is at least $\frac{\varepsilon n}{2k^2} \cdot \frac{1}{dn} = \frac{\varepsilon}{2k^2d}$. Thus, the claim follows by union bound over all pairs of artificial-clusters and the setting of t . \triangleleft

We are now ready to prove the correctness of the global algorithm, as stated in the next theorem.

\blacktriangleright **Theorem 28.** *Under the promise that the input graph, $G = (V, E)$, is a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph, with clusters of size at least βn , Algorithm 2 computes a sparse spanning subgraph of G , $G' = (V, E')$, such that:*

1. *The attained stretch is $\Theta\left(\frac{\log n}{\phi_{\text{in}}^2}\right)$, and*
2. *$|E'| \leq n(1 + \varepsilon)$ with high probability.*

Proof. Let $G = (V, E)$ be a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph and let C_1, \dots, C_f be a partition of G into $f \leq k$ $(\phi_{\text{in}}, \phi_{\text{out}})$ -clusters of size at least βn . We begin by proving Item 1 of the claim. Let \mathcal{E}_1 denote the event that for every $i \in [f]$, $C_i \cap \mathcal{S}$ contains a useful vertex w.r.t. C_i . Fix $i \in [f]$. By definition (see Claim 23), at least half of the vertices in C_i are useful. Since $|C_i| \geq \beta n$, the probability that v which selected u.a.r. from V is a useful vertex of C_i is at least $\beta/2$. Therefore w.p. at least $1 - 1/n^c$, \mathcal{P} contains a useful vertex w.r.t. C_i where c is determined by the exact setting of p . Thus, by union bound over all $i \in [f]$, w.h.p. \mathcal{E}_1 occurs.

Let \mathcal{E}_2 denote the event that for every $v \in V$, $\Gamma_h(v) \cap \mathcal{S} \neq \emptyset$. Conditioned on \mathcal{E}_1 , by Claim 26 w.h.p. \mathcal{E}_2 occurs.

Consider an edge $\{u, v\}$ which belongs to E but not to E' . We first note that it follows that both $\sigma(u) \neq \perp$ and $\sigma(v) \neq \perp$ because otherwise $\{u, v\}$ is added to E' in Step 9. From Claim 17 it follows that for every $v \in V$ such that $\sigma(v) \neq \perp$ the distance from v to $\sigma(v)$ is at

most $\frac{\log h}{\log(1+\phi_{\text{in}})}$. To see this, observe that the ℓ -ball centered at v in $G[C]$ is contained ℓ -ball centered at v in G where C denotes the cluster of v according to the partition mentioned above (in other words the neighborhood of v expands in G at least as fast as it does in $G[C]$). Thus, if u and v belong to the same Voronoi cell, then the distance between them in G' is at most $2\frac{\log h}{\log(1+\phi_{\text{in}})}$. If u and v belong to the same artificial-cluster then the distance in G' between their respective centers, $\sigma(u)$ and $\sigma(v)$ is at most 2τ (because the distance in G' between a secondary center to its primary center is at most τ). Thus, in this case the distance in G' between u and v is at most $2 \cdot \left(\tau + \frac{\log h}{\log(1+\phi_{\text{in}})}\right)$. Finally, if u and v belong to different artificial-clusters then by Steps 10 and 11 there exists $\{u', v'\} \in E'$ such that u' and u are in the same artificial-cluster and likewise for v and v' . Thus, by the above the distance in G' between u and v is at most $2 \cdot \left(2 \cdot \left(\tau + \frac{\log h}{\log(1+\phi_{\text{in}})}\right)\right) + 1$, where $\tau \stackrel{\text{def}}{=} \frac{\log(2n^{3/2})}{\log\left(\left(1 - \frac{\phi_{\text{in}}^2}{2}\right)^{-1}\right)}$.

Since for all $x \geq 0$ it holds that $x - x^2/2 \leq \ln(1+x)$ and since w.l.o.g. $h \leq n$, it follows that the attained stretch is $\Theta\left(\frac{\log n}{\phi_{\text{in}}^2}\right)$, as claimed.

Item 2 of the claim follows from the construction. More specifically, at Step 4 we add at most $p \cdot r \cdot \tau = \Theta(\phi_{\text{out}}^{-1} \beta^{-1} \log^2 n)$ edges to E' . This is $o(n)$ since we may assume that our query complexity is $\tilde{O}(n^{2/3}) = o(n)$ (otherwise we may run the algorithm by Lenzen and Levi [12]). In Step 8, we add at most $n - 1$ edges due to the fact that the Voronoi cells are vertex-disjoint. Conditioned on \mathcal{E}_2 , which occurs w.h.p., in Step 9 we do not add any edges to E' . In Step 10 we add at most t edges to E' . Let \mathcal{E}_3 denote the event that \mathcal{T} contains an edge from the edge-cut of every pair of artificial-clusters. By Claim 27 w.h.p. \mathcal{E}_3 occurs. Finally, Conditioned on \mathcal{E}_3 , in Step 11 we add at most $k^2 \cdot \varepsilon \cdot n / (2k^2) = \varepsilon n / 2$ edges to E' . The claim follows. \blacktriangleleft

4.3 Details of the Local Algorithm

In this section, we describe how to implement Algorithm 2 locally. The local implementation of all the steps of the algorithm is similar to the local implementation of the analogous steps of Algorithm 1 (with different parameters). The only new ingredient in the local implementation is the implementation of Step 11. Next, we describe how this step can be implemented locally. On query $\{u, v\}$, if u and v belong to the same artificial-cluster, then we proceed as before. Namely, we return YES if and only if u and v belong to the same Voronoi cell and $\{u, v\}$ belongs to the tree that spans the cell. If u and v belong to different artificial-clusters then we consider three cases. The first case we consider is when $\{u, v\} \in \mathcal{T}$, the sample of edges selected in Step 10. In this case, we would like to return YES on the query $\{u, v\}$. The second case is when $\{u, v\} \notin \mathcal{T}$ but there exists $\{u', v'\} \in \mathcal{T}$ such that u and u' are in the same artificial-cluster and likewise for v and v' . In this case, we would like to return NO. Otherwise, we would like to return YES. Therefore, we can decide if to return YES or NO if we find for each edge in \mathcal{T} the identities of the artificial-clusters (i.e., the IDs of the primary centers of the corresponding artificial-clusters) of its endpoints. This is accomplished as follows. For each one of the pairs, (v, i) , selected in Step 10, we perform a neighbor query to obtain the i -th neighbor of v . If such a neighbor exists then let us denote it by u . We then find $\sigma(v)$ and $\sigma(u)$ by making $O(hd^2)$ queries. This also reveals to which artificial-cluster v belongs and likewise for u . This allows us to obtain the list of pairs of artificial-clusters that already have an edge from their cut in \mathcal{T} . Thus, on query $\{u, v\}$ where u and v belong to different artificial-clusters which are not on that list, the algorithm will return YES. Overall, implementing this check requires $O(thd^2)$ queries and $\tilde{O}(thd^2)$ time. We conclude that the query complexity of the algorithm is dominated by the implementation of Step 4 which requires $O(pr\tau) = O(\log^2 n / (\beta\phi_{\text{out}}))$ queries and Step 11 which requires $O(thd^2) = O(\phi_{\text{in}}^{-1} \varepsilon^{-1} k^3 d^3 \phi_{\text{out}} \cdot n \log^2 n)$ queries.

In this next claim, we bound the number of random bits that our algorithm uses.

▷ **Claim 29.** Performing r s -wise independent random walks from each $\psi \in \mathcal{P}$ in $(G)_{\text{reg}}$ can be implemented by using $O(\log n(\log n + \tau \log d))$ random bits and time-complexity $r \cdot \text{poly}(\tau, \log n)$.

Proof. As claimed in the proof of Claim 22, performing a single random-walk in $(G)_{\text{reg}}$ of length τ from any vertex requires $\tau \log(2d)$ random bits. If we choose a random function from a family of s -wise independent functions, $\mathcal{H}_{\gamma, \beta} = \{h : \{0, 1\}^\gamma \rightarrow \{0, 1\}^\beta\}$, where $\gamma = \log(p \cdot r)$ is the number of bits that are required to indicate the index of the vertex in \mathcal{P} and the index of the walk in $[r]$ and $\beta = \tau \log(2d)$ is the number of bits that are required for performing the walk then we obtain that the total number of random bits that the algorithm uses for performing the walks is $s \cdot \max\{\gamma, \beta\} = O(\log n(\log n + \tau \log d))$. We obtain the desired result since performing a single random walk requires $\text{poly}(\gamma, \beta, s) = \text{poly}(\tau, \log n)$ time. ◀

We are now ready to prove Theorem 5.

► **Theorem 5.** *There is an LSSG algorithm that given query access to a connected $(k, \phi_{\text{in}}, \phi_{\text{out}})$ -clusterable graph $G = (V, E)$, where each cluster is of size at least $\beta \cdot n$, provides access to $G' = (V, E')$ such that the following holds. (1) The graph G' is a connected subgraph of G and with high probability $|E'| \leq n(1 + \varepsilon)$. Moreover, the stretch factor of G' is $\Theta\left(\frac{\log n}{\phi_{\text{in}}^2}\right)$. (2) The query complexity of the algorithm is $O\left(\log^2 n \cdot (\beta \phi_{\text{out}})^{-1} + n \log^2 n \cdot k^3 d^3 \phi_{\text{out}} (\varepsilon \phi_{\text{in}})^{-1}\right)$, and (3) the number of random bits it uses is $O\left(\frac{\log d \cdot \log^2 n}{\phi_{\text{in}}^2}\right)$, where d is a bound on the maximum degree of G .*

Proof of Theorem 5. The correctness of the algorithm, i.e., Item (1), follows from Theorem 19. The query complexity of the algorithm is analyzed in the description of the local implementation that appears above. Finally, by Claim 22 the local implementation of Algorithm 1 uses $O(\log n(\log n + \tau \log d)) = O\left(\frac{\log d \cdot \log^2 n}{\phi_{\text{in}}^2}\right)$ random bits, as claimed. This concludes the proof of the theorem. ◀

To obtain Corollary 6 from Theorem 5, we observe that we can take the upper bound on ϕ_{out} to be the maximum between ϕ_{out} and $1/\sqrt{n}$. More specifically, if we consider $k, d, \phi_{\text{in}}, \varepsilon$ and β to be constants then the query complexity is $\tilde{O}(1/\phi_{\text{out}} + n\phi_{\text{out}})$. Since $1/\phi_{\text{out}} > n\phi_{\text{out}}$ only when $\phi_{\text{out}} < 1/\sqrt{n}$ we only need to show that the query complexity in this case is $\tilde{O}(\sqrt{n})$. Indeed, since ϕ_{out} is only an upper bound on the outer-conductance, we may take $\phi_{\text{out}} = 1/\sqrt{n}$ in this case and obtain the desired complexity.

► **Corollary 6.** *There is an LSSG algorithm that given query access to a connected $(\Theta(1), \Theta(1), \phi_{\text{out}})$ -clusterable graph $G = (V, E)$, where each cluster is of size at least $\beta \cdot n$, provides access to $G' = (V, E')$ such that the following holds. (1) The graph G' is a connected subgraph of G and with high probability $|E'| \leq n(1 + \varepsilon)$. Moreover, the stretch factor of G' is $\Theta(\log n)$. (2) The query and time complexity of the algorithm is $\tilde{O}(\sqrt{n} + \phi_{\text{out}} n)$, and (3) the number of random bits it uses is $O(\log^2 n)$.*

References

- 1 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*, pages 1132–1139. SIAM, 2012.

- 2 Rubi Arviv, Lily Chung, Reut Levi, and Edward Pyne. Improved local computation algorithms for constructing spanners. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
- 3 Amartya Shankha Biswas, Ruidi Cao, Edward Pyne, and Ronitt Rubinfeld. Average-case local computation algorithms. *arXiv preprint arXiv:2403.00129*, 2024.
- 4 Greg Bodwin and Henry Fleischmann. Spanning adjacency oracles in sublinear time. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2024.
- 5 Jeff Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in analysis*, 625(195-199):110, 1970.
- 6 Artur Czumaj, Pan Peng, and Christian Sohler. Testing cluster structure of graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 723–732, 2015.
- 7 Guy Even, Moti Medina, and Dana Ron. Best of two local models: Centralized local and distributed local algorithms. *Information and Computation*, 262:69–89, 2018. doi: 10.1016/j.ic.2018.07.001.
- 8 Uriel Feige, Yishay Mansour, and Robert E Schapire. Learning and inference in the presence of corrupted inputs. *Journal of Machine Learning Research*, 40(2015), 2015.
- 9 Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1256–1266. SIAM, 2014.
- 10 Oded Goldreich. Basic facts about expander graphs. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 451–464, 2011.
- 11 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.
- 12 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)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 13 Reut Levi and Moti Medina. A (centralized) local guide. *Bulletin of the EATCS*, 122:60–92, 2017. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/495>.
- 14 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.
- 15 Reut Levi and Dana Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Transactions on Algorithms (TALG)*, 11(3):1–13, 2015.
- 16 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.
- 17 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. *Algorithmica*, 82(4):747–786, 2020.
- 18 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 4(77):971–994, 2016.
- 19 Reut Levi and Nadav Shoshan. Testing hamiltonicity (and other problems) in minor-free graphs. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPICs*, pages 61:1–61:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

- 20 Yishay Mansour, Aviad Rubinfeld, 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 39*, pages 653–664. Springer, 2012.
- 21 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 260–273. Springer, 2013.
- 22 Guy Moshkovitz and Asaf Shapira. Decomposing a graph into expanding subgraphs. *Random Struct. Algorithms*, 52(1):158–178, 2018. doi:10.1002/RSA.20727.
- 23 Merav Parter, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. Local computation algorithms for spanners. *Innovations in Theoretical Computer Science (ITCS)*, 2019.
- 24 Pan Peng. Robust clustering oracle and local reconstructor of cluster structure of graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2953–2972. SIAM, 2020.
- 25 Dana Ron Reut Levi and Ronitt Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. *Klaus Jansen, Claire Mathieu, José DP Rolim, and Chris Umans, editors, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 7–9, 2016.
- 26 Ronitt Rubinfeld. Can we locally compute sparse connected subgraphs? In *Computer Science—Theory and Applications: 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings 12*, pages 38–47. Springer, 2017.
- 27 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proceedings of The Second Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.
- 28 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.
- 29 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.
- 30 Daniel Spielman. Spectral and algebraic graph theory. *Yale lecture notes, draft of December*, 4:47, 2019.
- 31 Salil P Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

A Other Related Work

A.1 LCAs for Spanners

A desirable property for a spanning subgraph is that it also preserves, up to a predetermined multiplicative factor $\alpha \geq 1$ (a.k.a the stretch factor), the pairwise distances of the vertices in the original graph. Such a spanning subgraph is called an α -spanner. In [15, 25] $\text{poly}(h \log(d)/\varepsilon)$ -spanners for minor-free graphs are presented. For general (bounded degree) graphs, the algorithm of [12] outputs an $O(\log^2 n \cdot \text{poly}(d/\varepsilon))$ -spanner. Their result is later extended by Parter et al. [23], who presented an algorithm that constructs an $O(k^2)$ -spanner, independent of both n and d , but has $O(n^{1+1/k})$ edges. The query complexity of [23] is $O(n^{2/3}d^4)$, and is later improved by Arviv et al. [2] to $O(n^{2/3}d^2)$. A recent work by Biswas, Cao, Pyne, and Rubinfeld [3] presents several LCAs that receive random graphs (i.e., Erdős-Rényi or Preferential Attachment graph) as an input and gives access to a sparse spanner of that graph.

A.2 Graph Clustering

In the Property Testing model, Czumaj et al. [6] introduce an algorithm for testing the clusterability of a graph. In the Property Testing model, an algorithm accepts (with constant probability) any graph that is (k, ϕ) -clusterable and rejects graphs that are ε far from being (k, ϕ^*) -clusterable, i.e., εdn edges are required to be either added or removed from the input graph so that it becomes (k, ϕ^*) -clusterable, where $\phi^* = O(\frac{\phi^2 \varepsilon^2}{\log n})$. The query complexity of their algorithm is $\tilde{O}(\sqrt{n} \cdot \text{poly}(\phi, k, 1/\varepsilon))$ and with a success probability of at least $2/3$. Peng [24] uses similar ideas as [6] to construct a clustering oracle that given an input graph that is ε -close from being $(k, \phi, O(\frac{\varepsilon \phi}{k^3 \log n}))$ -clusterable gives w.h.p. query access to the adjacency list of a $(k, \frac{\phi}{2}, O(\frac{\sqrt{\varepsilon} \phi^{1.5}}{k^3 \log n}))$ -clusterable graph with preprocessing and query complexity of $O(\sqrt{n} \cdot \text{poly}(\frac{k \log n}{\phi \varepsilon}))$, and with at most $O(k \sqrt{\frac{\varepsilon}{\phi}} \cdot n)$ outliers (vertices that are not associated with any cluster).

A.3 LCAs for Other Graph Problems

The model of *local computation algorithms* (LCA) (sometimes also referred to as The Centralized-Local model (CentLocal)) as used in this work, was defined by Rubinfeld et al. [27] (see also [1] and survey in [13]). Such algorithms for maximal independent set, hypergraph coloring, k -CNF, approximated maximum matching and approximated minimum vertex cover for bipartite graphs are given in [27, 1, 20, 21, 7, 18, 8].

B Omitted Proofs of Section 2

Proof of Coro. 8. By Theorem 7, for any distribution vector \vec{p} it holds that $\|\hat{A}^\tau \vec{p} - \vec{u}\|_1 \leq \sqrt{n} \cdot \alpha^{\frac{\log(2n^{3/2})}{\log(1/\alpha)}} = \frac{1}{2n}$. If there exists an index i such that either $(\hat{A}^\tau \vec{p})_i < \frac{1}{2n}$ or $(\hat{A}^\tau \vec{p})_i > \frac{3}{2n}$ then $\left|(\hat{A}^\tau \vec{p})_i - \frac{1}{n}\right| > \frac{1}{2n}$, which implies that $\|\hat{A}^\tau \vec{p} - \vec{u}\|_1 > \frac{1}{2n}$, in contradiction to the above Equation. ◀

Proof of Claim 11. Since $(G)_{\text{reg}}$ is $2d$ -regular it holds that $\lambda_1 = 2d$. Let $\{\hat{\lambda}_i\}_i$ denote the eigenvalues of \hat{A} where $\hat{A} = \frac{1}{2d} A((G)_{\text{reg}})$. By Theorem 9, it follows that $\hat{\lambda}_1 = 1 > \hat{\lambda}_2, \dots, \hat{\lambda}_n \geq -1$. Since $\hat{A}_{i,j} \geq 0$ and $\hat{A}_{j,j} \geq \frac{1}{2}$ it holds that the matrix $M \stackrel{\text{def}}{=} 2\hat{A} - I$, where I is the identity matrix, is non-negative. Hence, M corresponds to a weighted connected graph. Moreover, the eigenvalues of M , $\{\mu_i\}_i$ satisfy $\mu_i = 2\hat{\lambda}_i - 1$.¹¹ In particular, $\mu_1 = 1$. Thus, Theorem 9 applied on M implies that $\mu_i \geq \mu_n \geq -1$ for all $1 \leq i \leq n$. Thus, $2\hat{\lambda}_i - 1 \geq -1$ and hence $\hat{\lambda}_i \geq 0$ for all $1 \leq i \leq n$. Since $\lambda_i = 2d\hat{\lambda}_i$, it follows that $|\lambda_2| > |\lambda_n|$. The claim follows. ◀

Proof of Claim 12. We first observe that $\phi(G) = \phi((G)_{\text{reg}})$. Since $(G)_{\text{reg}}$ is $2d$ -regular, by Cheeger's Inequality it holds that $\lambda_2 \leq 2d \cdot \left(1 - \frac{\phi^2(G)}{2}\right)$. Thus the claim follows from Claim 11. ◀

¹¹To see why $\mu_i = 2\hat{\lambda}_i - 1$, multiply the eigenvector that corresponds to $\hat{\lambda}_i$, ν_i , by M which gives $2\hat{\lambda}_i \nu_i - \nu_i$.

C Omitted Proofs of Section 3

Proof of Claim 17. Let v be a vertex in G . By Equation (1), for any subset $S \subseteq V$ of size at most $|V|/2$ it holds that $e(S, V \setminus S) \geq \phi \cdot d \cdot |S|$. In particular, if S is the set of vertices of the j -ball centered at some vertex v then the $j + 1$ -ball centered at v contains at least $|S| + \phi|S| = (1 + \phi)|S|$ vertices. Thus, after exploring ℓ layers of the BFS rooted at v at least one of the following holds: either we explored more than $|V|/2$ vertices, or we explored at least $(1 + \phi)^\ell$ vertices. Thus we explore at least x vertices for any ℓ such that $(1 + \phi)^\ell \geq x$. The claim follows. \triangleleft