A Centralized Local Algorithm for the Sparse Spanning Graph Problem

Christoph Lenzen

Max Planck Institute for Informatics, Saarbrücken, Germany
clenzen@mpi-inf.mpg.de

Reut Levi[1]

Weizmann Institute of Science, Rehovot, Israel
reut.levi@weizmann.ac.il
🄳 https://orcid.org/0000-0003-3167-1766

— Abstract —

Constructing a sparse *spanning subgraph* is a fundamental primitive in graph theory. In this paper, we study this problem in the Centralized Local model, where the goal is to decide whether an edge is part of the spanning subgraph by examining only a small part of the input; yet, answers must be globally consistent and independent of prior queries.

Unfortunately, maximally sparse spanning subgraphs, i.e., spanning trees, cannot be constructed efficiently in this model. Therefore, we settle for a spanning subgraph containing at most $(1 + \epsilon)n$ edges (where n is the number of vertices and ϵ is a given approximation/sparsity parameter). We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$,[2] where Δ is the maximum degree of the input graph. Our algorithm is the first to do so on arbitrary bounded degree graphs. Moreover, we achieve the additional property that our algorithm outputs a spanning subgraph of bounded stretch i.e., distances are approximately preserved. With high probability, for each deleted edge there is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops in the output that connects its endpoints.

1 Introduction

When operating on very large graphs, it is often impractical or infeasible to (i) hold the entire graph in the local memory of a processing unit, (ii) run linear-time (or even slower) algorithms, or even (iii) have only a single processing unit perform computations sequentially. These constraints inspired the Centralized Local model [17], which essentially views the input as being stored in a (likely distributed) database that provides query access to external processing units. To minimize the coordination overhead of such a system, it is furthermore required that there is no shared memory or communication between the querying processes, except for shared randomness provided alongside the access to the input. The idea is now to

---

run sublinear-time algorithms that extract useful global properties of the graph and/or to examine the input graph locally upon demand by applications.

Studying graphs in this model, lead to the emergence of algorithms that provide query access to a variety of graph-theoretical structures like, e.g., independent or dominating sets. In such a case, it is crucial that locally evaluating whether a node participates in such a set is *consistent* with the same evaluation for other nodes. This is a non-trivial task, as local decisions can only be coordinated implicitly via the structure of the input (which is to be examined as little as possible) and the shared randomness. Nonetheless, this budding field brought forth a number of elegant algorithms solving, e.g., maximal independent set, hypergraph coloring, k-CNF, approximate maximum matching and approximate minimum vertex cover for bipartite graphs [1, 4, 5, 11, 12, 13, 17].

In this work, we consider another very basic graph structure: sparse spanning subgraphs. Here, the task is to select a subset of the edges of the (connected) input graph so that the output is still connected, but has only few edges. By "few" we mean that, for some input parameter $\varepsilon > 0$, the number of selected edges is at most $(1 + \varepsilon)n$, where n denotes the number of nodes. One may see this as a relaxed version of the problem of outputting a spanning tree of the graph, which is a too rigid requirement when looking for fast algorithms: on a cycle, a single edge has to be deleted, but this necessitates to first verify that the input graph is not, in fact, a line.

▶ **Definition 1** ([9])**.** An algorithm $\mathcal{A}$ is a *Local Sparse Spanning Graph (LSSG) algorithm* if, given $n, \Delta \geq 1$, a parameter $\epsilon \geq 0$, and query access to the incidence list representation of a connected graph $G = (V, E)$ over n vertices and of degree at most Δ, it provides oracle access to a subgraph $G' = (V, E')$ of G such that:

1. G' is connected.
2. $|E'| \leq (1 + \epsilon) \cdot n$ with high probability (w.h.p.),[3] where E' is determined by G and the internal randomness of $\mathcal{A}$.

By "providing oracle access to G'" we mean that on input $\{u, v\} \in E$, $\mathcal{A}$ returns whether $\{u, v\} \in E'$, and for any sequence of edges, $\mathcal{A}$ answers consistently with respect to the same G'.

We are interested in LSSG algorithms that, for each given edge, perform as few queries as possible to G. Observe that Item 2 implies that the answers of an LLSG algorithm to queries cannot depend on previously asked queries.

We note that relaxing from requiring a tree as output makes it possible to ask for additional guarantees that, in general, cannot be met by a spanning tree. Instead of merely preserving connectivity, it becomes possible to maintain *distances* up to small factors (for unweighted graphs). Such subgraphs are known as (sparse, multiplicative) *spanners* [14, 15]. In fact, choosing $\varepsilon \in o(1)$ then yields *ultra-sparse* spanners that are $o(n)$ edges away from being trees.

1.1 Our Contribution

We give the first non-trivial LSSG algorithm in the Centralized Local model that runs on arbitrary graphs. We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$ per edge, w.h.p. Moreover, we guarantee that for each edge that is not selected into the spanner, w.h.p. there

---

[3] That is, with probability at least $1 - 1/n^c$ for an arbitrary constant $c > 0$ that is chosen upfront.

is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops consisting of edges that are selected into the spanner; this is referred to as a *stretch* of $O(\log n \cdot (\Delta + \log n)/\epsilon)$.

For simplicity, assume for the moment that Δ and ε are constants. Our algorithm combines the following key ideas.

- We classify edges as expanding if there are sufficiently many (roughly $n^{1/3}$) nodes within $O(\log n)$ hops of their endpoints. For non-expanding edges, we can efficiently simulate a standard distributed spanner algorithm at small query complexity, as solutions of running time $O(\log n)$ are known (e.g. [3]).

- On the node set induced by expanding edges, we can construct a partition into Voronoi cells with respect to roughly $n^{2/3}$ randomly sampled centers. The Voronoi cells are spanned by trees of depth $O(\log n)$, as expanding nodes have their closest center within $O(\log n)$ hops w.h.p. Finding the closest center has query complexity $\tilde{O}(n^{1/3})$.

- We refine the partition into Voronoi cells further into *clusters* of $\tilde{O}(n^{1/3})$ nodes. We simply let a node be a singleton cluster if its subtree in the spanning tree of its cell contains more than $\tilde{O}(n^{1/3})$ nodes. This construction has query complexity $\tilde{O}(n^{2/3})$ for constructing a complete cluster, yet ensures that there are $\tilde{O}(n^{2/3})$ clusters in total due to the low depth of the trees that span the Voronoi cells; moreover, each cluster is completely contained in some Voronoi cell.

- It remains to select few edges to interconnect the Voronoi cells. This is the main challenge, for which the above properties of the partition are crucial. To keep the number of selected edges small in expectation, we mark a subset of expected size $\tilde{\Theta}(n^{1/3})$ of the clusters by marking each Voronoi cell (and thereby its constituent clusters) with probability $n^{-1/3}$. We then try to ensure that (i) clusters select an edge to each adjacent marked Voronoi cell and (ii) for each marked Voronoi cell adjacent to an adjacent cluster, they select one edge connecting to *some* cluster adjacent to this cell.

- The main issue with the previous step is that we cannot afford to construct each adjacent cluster, preventing us from guaranteeing (ii). We circumvent this obstacle by identifying for adjacent clusters in which cell they are and keeping an edge for the purpose of (ii) if it satisfies a certain minimality requirement with respect to the *rank* of the cell used for symmetry breaking purposes. This way, we avoid construction of adjacent clusters, instead needing to determine the rank of their Voronoi cells only. This way, we maintain query complexity $\tilde{O}(n^{2/3})$.

- However, this now entails an inductive argument for ensuring connectivity, which also affects stretch. By choosing Voronoi cell ranks uniformly at random, we ensure that the length of such an inductive chain is bounded by $O(\log n)$ w.h.p. Together with the depth of the Voronoi cell trees of $O(\log n)$ and the stretch of the spanner algorithm for non-expanding edges (also $O(\log n)$), this yields the total bound of $O(\log^2 n)$ on the stretch of our scheme.

Finally, we note that we can place the above routine in a wrapper verifying that, w.h.p., the number of globally selected edges does not significantly exceed the expectation. If this is not the case, the wrapper starts the process all over. Since in each attempt the success probability is constant (and the verifier succeeds w.h.p.), we get within $O(\log n)$ attempts that the bound on the number of selected edges is satisfied w.h.p. and the routine terminates.

Relation to Property Testing. As observed in [10], testing cycle-freeness with one sided-error in the bounded-degree model can be reduced to the LSSG problem. From this reduction it follows that we obtain a tester for cycle-freeness that works in $\tilde{O}(n^{2/3})$ time. Czumaj et al. [2] studied the problem of C_k-minor freeness with one sided-error. For cycle-freeness

(C_3-minor freeness) their complexity is $\tilde{O}(\sqrt{n})$, therefore their complexity is better than ours. However, we would like to point out that an LSSG algorithm gives a stronger guarantee than a one-sided error tester for cycle-freeness. An LSSG algorithm can be used to find for all but at most $(1 + \epsilon)|V|$ edges e, a cycle that e belongs to, i.e., a witness for the violation by e can be provided. In contrast, a one-sided error tester merely guarantees to find a single cycle in instances that are ϵ-far from being cycle-free.

Perhaps more importantly, our approach proves useful when testing for other minors. Recently, Fichtenberger et al. [6] built on our work to construct one-sided error tester for outerplanarity and other properties that are characterized by a set of forbidden minors.

1.2   Related work

The problem of finding a sparse spanning subgraph in the Centralized Local model was first studied in [9], where the authors show a lower bound of $\Omega(\sqrt{n})$ queries for constant ϵ and Δ (see also survey by Rubinfeld [18]). They also present an upper bound with nearly tight query complexity for graphs that have very good expansion properties. However, for general (bounded degree) graphs their algorithm might query the entire graph for completing a single call to the oracle. They also provide an efficient algorithm for minor-free graphs that was later improved in [8]. The algorithm presented in [8] achieves a query complexity that is polynomial in Δ and $1/\epsilon$ and is independent of n. The stretch factor of this algorithm is also independent of n and depends only on Δ, $1/\epsilon$, and the size of the excluded minor.

A characterization of the query complexity of the problem was presented in [7]. Specifically, [7] provide an upper bound (which builds on an algorithm in [9]) that has a query complexity that is independent of n (however, super-exponential in $1/\epsilon$) for families of graphs which 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.

In the (distributed) Local model, Ram and Vicari [16] study the same problem and provide an algorithm that runs in $\min\{D, O(\log n)\}$ rounds, where D is diameter of the input graph. Their algorithm achieves the sparsity property by breaking all short cycles.

2   Preliminaries

The graphs we consider are undirected and have a known degree bound Δ, and we assume we have query access to their incidence list representation. Namely, for any vertex v and index $1 \le i \le \Delta$, it is possible to obtain the ith neighbor of v by performing a query to the graph (if v has less than i neighbors, then a special symbol is returned). Without loss of generality, we assume that graphs are simple, i.e., contain neither loops nor parallel edges.[4] The number of vertices in the graph is n and we assume that each vertex v has a unique id, which for simplicity we also denote by v. There is a total order on the ids, i.e., given any two distinct ids u and v, we can decide whether $u < v$ or $v < u$.

Let $G = (V, E)$ be a graph, where $V = [n]$. We denote the distance between two vertices u and v in G by $d_G(u, v)$. For vertex $v \in V$ and an integer r, let $\Gamma_r(v, G)$ denote the set of vertices at distance at most r from v. When the graph G is clear from the context, we shall use the shorthands $d(u, v)$ and $\Gamma_r(v)$ for $d_G(u, v)$ and $\Gamma_r(v, G)$, respectively.

---

[4] The answer on a self-loop can always be negative, and we can default to rejecting all but the first edge between two nodes.
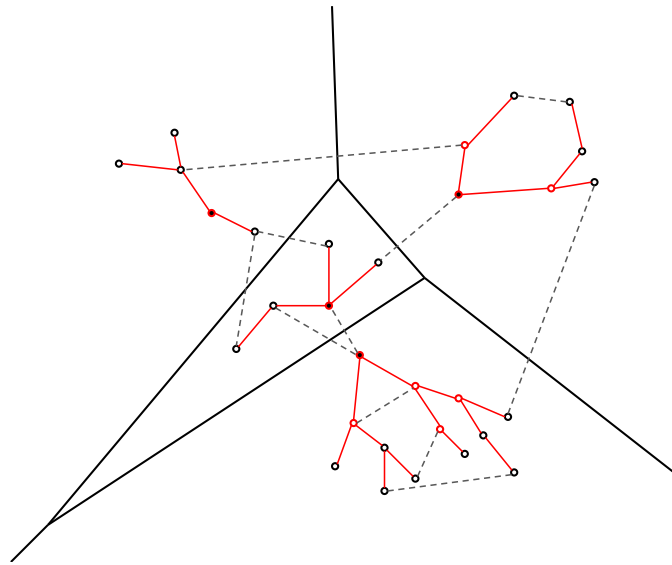
Figure 1 Partition of a graph into cells and clusters, where $k = 6$. Black lines are the borders of Voronoi cells, whose centers have black fillings. Red edges belong to the BFS trees spanning the clusters, while dashed gray lines are non-tree edges. Red circles indicate singleton clusters (if the node has a red child) or the roots of subtrees that form a cluster (if the children are black).

The total order on the vertices induces a total order r on the edges of the graph in the following straightforward manner: $r(\{u, v\}) < r(\{u', v'\})$ if and only if $\min\{u, v\} < \min\{u', v'\}$ or $\min\{u, v\} = \min\{u', v'\}$ and $\max\{u, v\} < \max\{u', v'\}$. 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.

Whenever referring to one of the above orders, we may refer to the *rank* of an element in the respective order. This is simply the index of the respective element when listing all elements ascendingly with respect to the order.

For a graph $G = (V, E)$ and a pair of disjoint subsets of vertices $A \subset V$ and $B \subset V$ let $E_G(A, B) \stackrel{\text{def}}{=} \{(u, v) \in E \mid u \in A \land v \in B\}$. When it is clear from the context, we omit the subscript. We say that a pair of subsets of vertices A and B are *adjacent* if $E_G(A, B) \neq \emptyset$.

3   An Algorithm that Works under a Promise

We begin by describing an LSSG algorithm which works under the following promise on the input graph $G = (V, E)$. Sample ℓ uniformly at random from $[b \log n / \log(1 + \epsilon), b \log n / \log(1 + \epsilon) + \Delta/\epsilon]$, and let $k \stackrel{\text{def}}{=} cn^{1/3} \ln n \cdot \ell \Delta/\epsilon$, where c and b are sufficiently large constants. For every $v \in V$, let $i_v \stackrel{\text{def}}{=} \min_r\{|\Gamma_r(v)| \geq k\}$. We are promised that $\max_{v \in V}\{i_v\} \leq \ell$. In words, we assume that the ℓ-hop neighborhood of every vertex in G contains at least k vertices. First, we fix a simple partition of V.

3.1   The Underlying Partition

Centers.   Pick a set $S \subset |V|$ of $r \stackrel{\text{def}}{=} \Theta(\epsilon n^{2/3}/\ln n)$ vertices at random. 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 $\text{Vor}(v)$, 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.

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 k 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 at least k vertices, then the cluster of v is the singleton $\{v\}$. Otherwise, v has a unique ancestor u (including v) for which $T(u)$ contains less than k vertices and $T(p(u))$ contains at least k vertices. The cluster of v is the set of vertices in $T(u)$. For a cluster C, let $c(C)$ denote the center of the vertices in C (all the vertices in the same cluster have the same center as they all belong to the same Voronoi cell). Let $\text{Vor}(C)$ denote the Voronoi cell of the vertices in C.

This describes a partition of V into Voronoi cells, and a refinement of this partition into clusters. See Figure 1 for an illustration.

3.2   The Edge Set

Our spanner, $H = (V, E')$, 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 (see Section 2 for more details). Clearly, these edges also span the clusters. Next, we describe which edges we add to E' in order to connect the clusters.
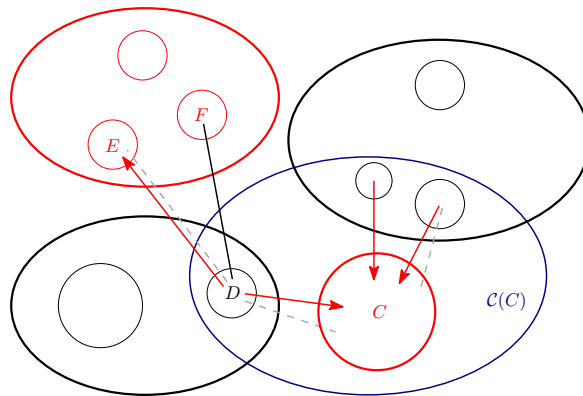
Marked Clusters and Clusters-of-Clusters

Each center in S is *marked* independently with probability $p \overset{\text{def}}{=} 1/n^{1/3}$. 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. A cluster B is *participating* in $\mathcal{C}(C)$ if $B \in \mathcal{C}(C)$ and the edge of minimum rank in $E(B, \text{Vor}(C))$ also belongs to $E(B, C)$. Thus, if B is adjacent to $\text{Vor}(C)$ and $\text{Vor}(C)$ is marked, then there is a unique cluster $D \subseteq \text{Vor}(C)$ such that B participates in $\mathcal{C}(D)$. See Figure 2 for a visualization.

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* $\text{Vor}(\partial A) \overset{\text{def}}{=} \{\text{Vor}(v) \mid \exists u \in A \text{ such that } \{u, v\} \in E\} \setminus \{\text{Vor}(A)\}$, i.e., the set of Voronoi cells that are adjacent to A. This definition explicitly excludes $\text{Vor}(A)$, as there is no need to connect A to its own Voronoi cell.

Figure 2 Illustration of marked clusters and clusters of clusters. Thick red and black ovals are marked and unmarked cells, respectively. Thin circles are clusters, where cluster C comprises its entire cell. Thick edges are the ones of minimum rank between their incident clusters, while the dotted edges do not meet this criterion. The arrows of red edges indicate participation in the respective adjacent marked cluster; note that D does not participate in $\mathcal{C}(F)$, as for each adjacent marked cell Vor it exclusively participates in the cluster-of-clusters connected to it by the edge of minimum rank in $E(D, \text{Vor})$. $\mathcal{C}(C)$ is marked in blue; all its constituent clusters also participate in $\mathcal{C}(C)$, as $\text{Vor}(C) = C$.

We next describe how we connect the clusters. The high-level idea is to make sure that every marked cluster and the clusters that participate in the respective cluster-of-clusters remains connected. This implies that the cluster-of-clusters remain connected as well, as every Voronoi cell is connected. For clusters which are not adjacent to any marked 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 participating in any cluster-of-clusters (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 Vor. Denote by C the (unique) cluster in cell Vor for which B participates in $\mathcal{C}(C)$. We connect A with B if the following conditions hold:
   - $\text{Vor}(B)$ has minimum rank amongst $\text{Vor}(\partial A) \cap \text{Vor}(\partial C)$
   - the minimum ranked edge in $E(A, \text{Vor}(B))$ is also in $E(A, B)$

Figure 3 showcases the third rule. Roughly speaking, the idea is that we want to connect cluster A with $\mathcal{C}(C)$ to preserve connectivity, however, at the same time, we want to make sure that we do not add too many edges to our spanning subgraph. Therefore, we connect A and B only if we do not see an evidence that A and $\mathcal{C}(C)$ remain connected through a different path.

3.3 Sparsity

▶ **Lemma 1.** *The number of clusters, denoted by s, is at most $|S| + n\ell(\Delta + 1)/k$.*

Proof. We first observe that, due to the promise on G, it follows that for every $v \in V$, $d(v, c(v)) \leq \ell$ w.h.p. Recall the terminology from Subsection 3.1. Consider v for which $|T(v)| \geq k$ and therefore its cluster is the singleton $\{v\}$. We say that a vertex u is *special* if $|T(u)| \geq k$ and for every child of u in $T(u)$, t, it holds that $|T(t)| < k$. By an inductive argument, it follows that v is an ancestor of a special vertex. Since for every pair of special vertices u and w, $T(u)$ and $T(w)$ are vertex disjoint, we obtain that there are at most n/k
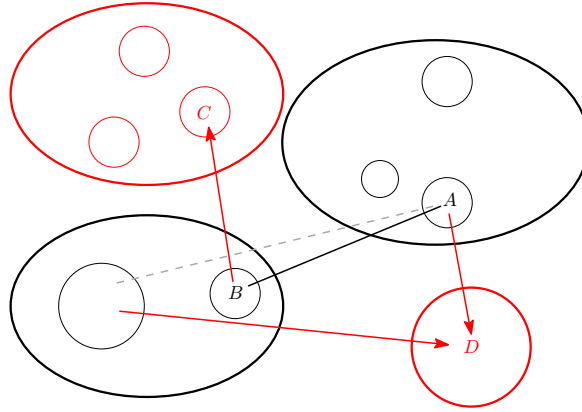
■ **Figure 3** Illustration of the third edge selection rule. In this example, the thick black edge has minimum rank in $E(A, \mathrm{Vor}(B))$ and B has minimum rank in $\partial A \cap \partial C$. However, the rank of $\mathrm{Vor}(D)$ is smaller than that of $\mathrm{Vor}(B)$, and hence the dashed edge is not selected. Here, A will select the direct edge connecting it to D, due to the first rule.

special vertices. Since for every special vertex, there are at most ℓ ancestors, the total number of vertices v with $|T(v)| \geq k$ is bounded by $n\ell/k$.

Observe that any cluster either (i) is a singleton $\{v\}$ with $|T(v)| \geq k$, (ii) contains a node v such that $|T(p(v))| \geq k$, or (iii) is an entire Voronoi cell. We just bounded the number of clusters of type (i) by $n\ell/k$, and immediately get a bound on the number of type (ii) clusters of $n\ell\Delta/k$. The number of type (iii) clusters is bounded by the number of Voronoi cells $|S|$, showing the desired bound on s.   ◀

▶ **Lemma 2.** $\mathrm{Exp}(|E'|) \leq (1 + O(\epsilon))|V|$.

Proof. The number of edges which are taken due to Condition 1 is at most s times the number of marked clusters. In expectation, there are $s \cdot p$ marked clusters, yielding at most $s^2 p$ edges in expectation. Since $s = O(\epsilon n^{2/3}/\ln n)$ and $p = 1/n^{1/3}$ we obtain that $s^2 p = O(\epsilon n/\ln n)$.

Let A be a cluster. The number of edges which are adjacent to A and are taken due to Condition 3 is bounded by the total number of clusters-of-clusters. The number of clusters-of-clusters is exactly the number of marked clusters. Thus, the total number of edges which are taken due to Condition 3 is bounded by $s^2 p$.

Observe that the probability that cluster A is not adjacent to a marked cell is $(1 - p)^{|\mathrm{Vor}(\partial A)|} \leq e^{-p|\mathrm{Vor}(\partial A)|}$. Hence, if $|\mathrm{Vor}(\partial A)| \geq 3p^{-1}\ln n$, A is adjacent to a marked cell w.h.p. Using a union bound over all clusters, it follows that w.h.p. each cluster A without an adjacent marked cell satisfies that $|\mathrm{Vor}(\partial A)| \leq 3p^{-1}\ln n$ with probability at least $1 - 1/n^2$; the probability at most $1/n^2$ event that this bound is violated cannot contribute more than $|E|/n^2 < 1$ to the expectation. Therefore, the total number of edges which are taken due to Condition 2 is bounded by $(3s\ln n)/p + 1 = O(\epsilon n)$.

Since the number of edges we add due to the BFS trees of the Voronoi cells is at most $|V| - 1$, we conclude that $\mathrm{Exp}(|E|') \leq (1 + O(\epsilon))|V|$, as desired.   ◀

3.4   Connectivity and Stretch

▶ **Lemma 3.** *H is connected.*

Proof. Recall that H contains a spanning tree on every Voronoi cell, hence it suffices to show that we can connect any pair of Voronoi cells by a path between some of their vertices. Moreover, the facts that G is connected and the Voronoi cells are a partition of V imply that it is sufficient to prove this for any pair of adjacent Voronoi cells. Accordingly, let Vor and Vor_1 be two cells such that $E(\text{Vor}, \text{Vor}_1) \neq \emptyset$.

Consider clusters $A \subseteq \text{Vor}$ and $B \subseteq \text{Vor}_1$ such that the edge e of minimum rank in $E(\text{Vor}, \text{Vor}_1)$ is in $E(A, B)$. If B is not adjacent to a marked cell, then Condition 2 implies that e is selected into H. Thus, we may assume that B is adjacent to a marked cell Vor'. Accordingly, there exists a marked cluster $C \subseteq \text{Vor}'$ such that B is participating in $\mathcal{C}(C)$.

If the rank of Vor_1 is minimum in $\text{Vor}(\partial C) \cap \text{Vor}(\partial A)$, then e is selected into H by Condition 3 and we are done. Otherwise, observe that Vor_1 is connected to Vor', as the edge of minimum rank in $E(B, C)$ is selected into H by Condition 1. Therefore, it suffices to show that Vor gets connected to Vor'. Let Vor_2 be the cell of minimum rank among $\text{Vor}(\partial C) \cap \text{Vor}(\partial A)$. Let $D \subseteq \text{Vor}_2$ be the cluster satisfying that the edge e' of minimum rank in $E(A, \text{Vor}_2)$ is in $E(A, D)$. Note that Vor_2 is connected to Vor' (which we saw to be connected to Vor_1), as there is some cluster $D' \subseteq \text{Vor}(D)$ that is adjacent to C and selects the edge of minimum rank in $E(D', C)$ by Condition 1.

Overall, we see that it is sufficient to show that Vor gets connected to Vor_2, where Vor_2 has smaller rank than Vor_1. We now repeat the above reasoning inductively. In step i, we either succeed in establishing connectivity between Vor and Vor_i, or we determine a cell Vor_{i+1} that has smaller rank than Vor_i and is connected to Vor_i. As any sequence of Voronoi cells of descending ranks must be finite, the induction halts after finitely many steps. Because the induction invariant is that Vor_{i+1} is connected to Vor_i, this establishes connectivity between Vor and Vor_1, completing the proof. ◀

▶ **Lemma 4.** *Denote by G_{Vor} the graph obtained from G by contracting Voronoi cells and by H_{Vor} its subgraph obtained when doing the same in H. If the cells' ranks are uniformly random, w.h.p. H_{Vor} is a spanner of G_{Vor} of stretch $O(\log n)$.*

Proof. Recall the proof of Lemma 3. We established connectivity by an inductive argument, where each step increased the number of traversed Voronoi cells by two. Hence, it suffices to show that the induction halts after $O(\log n)$ steps w.h.p.

To see this, observe first that G_{Vor} is independent of the ranks assigned to Voronoi cells and pick any pair of adjacent cells Vor, Vor_1, i.e., neighbors in G_{Vor}. We perform the induction again, assigning ranks from high to low only as needed in each step, according to the following process. In each step, we query the rank of some cells, and given an answer of rank r, the ranks of all cells of rank at least r are revealed as well. In step i, we begin by querying the rank of Vor_i. Consider the cluster $D_i \subseteq \text{Vor}_i$ adjacent to A satisfying that the edge with minimum rank in $E(\text{Vor}_i, A)$ is also in $E(D_i, A)$. We can assume without loss of generality that D_i is adjacent to a marked cluster F_i and that it is participating in $\mathcal{C}(F_i)$ (as otherwise D_i connects to A directly and we can terminate the process). If the ranks of all the cells adjacent to both F_i and A were already revealed, then the process terminates. Otherwise, we query the rank of all such cells whose rank is still unrevealed. We set the cell of the queried cluster that has minimum rank to be Vor_{i+1} and we continue to the next step.

We claim that, in each step i, either the process terminates, or the rank of Vor_{i+1} is at most half of the rank of Vor_i with probability at least $1/2$. To verify this, observe that in the beginning of step i, any cell center whose rank was not revealed so far has rank which

is uniformly distributed in $[r_i - 1]$, where r_i is the rank of Vor_i.[5] With probability at least $1/2$, such a rank is at most $r_i/2$. If Vor_i has no adjacent cells whose ranks have not been revealed yet, the process terminates. Hence, regardless of whether the process terminates or not, the claim holds.

By Chernoff's bound, we conclude that the process terminates within $O(\log n)$ steps w.h.p., as r_1 is bounded by the number of Voronoi cells, which itself is trivially bounded by n. By the union bound over all pairs of cells Vor and Vor_1, we get the desired guarantee. ◀

▶ **Corollary 1.** *W.h.p., H is a spanner of G of stretch $O(\log n \cdot (\Delta + \log n)/\epsilon)$.*

Proof. Due to the promise on G, w.h.p. the spanning trees on Voronoi cells have depth $\ell \in O((\Delta + \log n)/\epsilon)$. Hence, for any edge within a Voronoi cell, the claim holds w.h.p. Moreover, for an edge connecting different Voronoi cells, by Lemma 4, w.h.p. there is a path of length $O(\log n)$ in H_{Vor} connecting the respective cells. Navigating with at most $2\ell \in O((\Delta + \log n)/\epsilon)$ hops in each traversed cell, we obtain a suitable path of length $O(\log n \cdot (\Delta + \log n)/\epsilon)$ in H. ◀

4 The Algorithm for General Graphs

We use a combination of the algorithm in Section 3 with the algorithm by Elkin and Neiman for ultra-sparse spanners [3]. We call a vertex v *remote*, with respect to a set of centers, if the ℓ-hop neighborhood of v does not include a center. Fix S, let R denote the set of remote vertices with respect to S, and abbreviate $\bar{R} \overset{\text{def}}{=} V \setminus R$.

First Step.   Run the algorithm from Section 3 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 Elkin and Neiman [3] on the subgraph induced by R, i.e., $\{u, v\} \in E$ with $u, v \in R$ is added to E' if and only if the algorithm outputs the edge.[6] Their algorithm proceeds as follows. Given an integer h, each vertex v draws r_v according to the exponential distribution with parameter $\beta = \ln(n/\delta)/h$, where δ is a parameter that controls the success probability of the algorithm. Each vertex v receives r_u from every vertex u within distance h, and stores $m_u(v) = r_u - d(u, v)$ and a neighbor on a shortest path between v and u, denoted by $n_u(v)$. The edges that are added to the spanner are $C(v) = \{\{v, n_u(v)\} \mid m_u(v) \geq \max_{w \in V}\{m_w(v) - 1\}\}$, for every $v \in R$.

▶ **Corollary 2.** *M[Claim 2.3 in [3]] With probability at least $1 - \delta$, $r_v < h$ for all $v \in V$.*

M We choose $\delta = 1/n^{b-1}$, where $b \geq 2$ is a constant (see begining of Section 3), and $h = \ell$. The following lemma implies that the total number of edges that we add to H in the second step is at most $|R| \cdot (n^b)^{1/\ell} \leq |R|(1 + \epsilon)$ in expectation.

▶ **Lemma 5** (Proof of Lemma 2.2 in [3]). *For every $v \in R$, $\mathrm{Exp}[C(v)] \leq (n/\delta)^{1/h}$.*

---

[5] In step 1, we first query Vor_1 and then observe that this statement holds.
[6] The algorithm is described for connected graphs; we simply apply it to each connected component of R.

Third Step. Add to E' all edges $e \in E(R, \bar{R})$.

The following lemma implies that the expected number of edges which are added in the third step is at most ϵn.

▶ **Lemma 6.** $\mathrm{Exp}[|E(R, \bar{R})|] \leq \epsilon n$.

Proof. Observe that for an edge $\{u, v\} \in E$, there is at most one integer r such that $\Gamma_r(u) \cap S = \emptyset$ and $\Gamma_r(v) \cap S \neq \emptyset$ (or vice versa). If there is no such r or $\ell \neq r$, then the edge is not in $E(R, \bar{R})$. Over the random choice of ℓ, the probability of the event that the edge is included is at most $\Pr[\ell = r] \leq \epsilon/\Delta$. The lemma follows by linearity of expectation.   ◀

4.1 Stretch Factor

Consider any edge $e = \{u, v\} \in E \setminus E'$ we removed. If $u, v \in \bar{R}$, e was removed by the Algorithm from Section 3, which was applied to the subgraph induced by $\bar{R}$. Applying Corollary 1 to the connected component of e, we get that w.h.p. there is a path of length $O(\log n \cdot (\Delta + \log n)/\epsilon)$ from u to v in H. If $u, v \in R$, by Claim 2 and the choice of parameters, w.h.p. there is a path of length $O((\Delta + \log n)/\epsilon)$ from u to v in H. As $e \notin E(R, \bar{R})$ by the third step, we arrive at the following corollary.

▶ **Corollary 3.** *The above algorithm guarantees stretch* $O(\log n \cdot (\Delta + \log n)/\epsilon)$ *w.h.p. and satisfies that* $\mathrm{Exp}[|E'|] \in (1 + O(\epsilon))n$.

5 The Local Algorithm

In this section we prove our main theorem.

▶ **Theorem 2.** *Algorithm 1 is an LSSG algorithm. For any graph G over n vertices of maximum degree at most Δ and $\epsilon > 0$, its query complexity, space complexity (length of the random seed), and running time are* $\tilde{O}(n^{2/3}) \cdot \mathrm{poly}(1/\epsilon, \Delta)$.

Proof. The correctness of the algorithm follows from the previous sections. We shall prove that its complexity is as claimed. We analyze the complexity in terms of n. There are additional factors that depend polynomially in Δ and $1/\epsilon$. The following claims hold w.h.p., simultaneously, for all vertices.

Recall that a vertex u is remote if $\Gamma_\ell(u)$ does not contain a center. Due to the sampling probability for centers, a center is found after exploring $\tilde{O}(n^{1/3})$ vertices. Therefore, we can decide for any vertex u whether it is in R with query and time complexity $\tilde{O}(n^{1/3})$. Moreover, if $u \in \bar{R}$, without additional cost the respective subroutine can return $c(u)$, the center of u, and $d(u, c(u))$ (as we explore in a BFS fashion).

For Step 1, we need to determine $\Gamma_\ell(u) \cap R$. To do this, it suffices to explore $\Gamma_\ell(u)$, and for each vertex in it, to determine whether it is in R or not. Since $|\Gamma_\ell(u)| = \tilde{O}(n^{1/3})$ for every vertex $u \in R$, we obtain that the query and time complexity of this step is $\tilde{O}(n^{2/3})$, in total. Accordingly, Step 2 has query and time complexity $\tilde{O}(n^{1/3})$.

If $u, v \in \bar{R}$, the algorithm proceeds as in Section 3. We show first that we can reconstruct clusters efficiently. W.l.o.g., consider u. We determine $c(u)$ and $d(u, c(u))$. For each neighbor w of u, we determine whether it is in $\bar{R}$ (if not, the node is discarded), its center $c(w)$, and the distance $d(w, c(w))$. As $u \in \bar{R}$, and assuming that $u \neq c(u)$, it must have at least one neighbor w in distance $d(u, c(u)) - 1$ of $c(u)$; any such w satisfies that $c(w) = c(u)$, as otherwise $d(u, c(w)) = d(w, c(w))$ and $c(w) < c(u)$, a contradiction to the tie-breaking rule for centers. Among these candidates w, we know that the one with minimum rank is the

---

Algorithm 1 LSSG for general graphs.

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

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

1. If $u, v \in R$, compute the output of algorithm of Elkin and Neiman at u and v when running it on the connected component of u and v in the subgraph induced by R. Return **true** if $\{u, v\} \in C(u) \cup C(v)$ and **false** otherwise.

2. If $\{u, v\} \in E(R, \bar{R})$, return **true**.

3. Otherwise, $u, v \in \bar{R}$ and we proceed according to Section 3, where all nodes in R are ignored:

   a. If $\mathrm{Vor}(u) = \mathrm{Vor}(v)$, return **true** if $\{u, v\}$ is in the BFS tree of $\mathrm{Vor}(u)$ and **false** otherwise.

   b. Otherwise, let Q and W denote the clusters of u and v, respectively. Return **true** if at least one of the following conditions hold for $A = Q$ and $B = W$, or symmetrically, for $A = W$ and $B = Q$, and **false** otherwise.

      i. A is a marked cluster and $\{u, v\}$ has minimum rank amongst the edges in $E(A, B)$.

      ii. A is not participating in any cluster-of-clusters. 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 A is participating in $\mathcal{C}(C)$, and the following holds:
         - $\mathrm{Vor}(A)$ has minimum rank amongst $\mathrm{Vor}(\partial B) \cap \mathrm{Vor}(\partial C)$
         - $\{u, v\}$ has minimum rank amongst the edges in $E(B, \mathrm{Vor}(A))$.

---

parent of u in the BFS tree of $\mathrm{Vor}(u)$ rooted at $c(u)$, due to the tie-breaking rule for the BFS construction. We can use this subroutine to partially explore the BFS of $\mathrm{Vor}(u)$: given any node $w \in \bar{R}$, we can determine its parent, $p(w)$, and children in $\mathrm{Vor}(w)$ at query and time complexity $\tilde{O}(n^{1/3})$ as follows. The shortest path that has the smallest lexicographical order from $c(w)$ to w is the path connecting these vertices in the BFS tree. Therefore, the parent of w in this tree can be found by performing a BFS from w until the first level in which $c(w)$ is reached. To determine the children we simply run this subroutine for each neighbor, y, of w and find out whether $w = p(y)$ or not. In order to determine whether $T(w) < k$ or $T(w) \geq k$ (by partially or completely exploring $T(w)$), we need to run this subroutine $O(k)$ times, therefore this can be obtained with query complexity $\tilde{O}(n^{2/3})$. if $T(w) < k$, we determine $T(w)$ completely. We collect this information for u and its $\ell - 1$ ancestors, and determine the cluster Q of u. Finally, we repeat the procedure to reconstruct the cluster W of v, and determine for all nodes adjacent to either cluster whether they are in $\bar{R}$ and, if so, their centers. The query and time complexity of this operation is $\tilde{O}(n^{2/3})$ in total.

For the cases $A = Q$, $B = W$ and $A = W$, $B = Q$, respectively, with the above information we can determine

- whether $\mathrm{Vor}(u) = \mathrm{Vor}(v)$,
- if $\mathrm{Vor}(u) = \mathrm{Vor}(v)$, whether $p(u) = v$ or $p(v) = u$,
- if $\mathrm{Vor}(u) \neq \mathrm{Vor}(v)$,
  - whether A is marked (i.e., $c(A)$ has been marked) and whether $\{u, v\}$ has minimum rank in $E(A, B)$,
  - whether A is not adjacent to any marked cluster (i.e., none of the adjacent nodes' centers has been marked) and whether $\{u, v\}$ has minimum rank in $E(A, \mathrm{Vor}(B))$, and
  - whether there is a marked cluster C adjacent to A so that A participates in $\mathcal{C}(C)$,

$\text{Vor}(A)$ has minimum rank in $\text{Vor}(\partial B) \cap \text{Vor}(\partial C)$, and $\{u, v\}$ has minimum rank in $E(B, \text{Vor}(A))$. We note that since we have degree bounded by Δ, the number of vertices in A is $\tilde{O}(n^{1/3})$, and the probability that a cell is marked is $n^{-1/3}$, the number of cluster-of-clusters that A participates in is $\tilde{O}(\Delta)$ w.h.p.

In other words, we can perform all necessary checks to decide whether $\{u, v\} \in E'$ or not.

The algorithm in Section 3 requires $\tilde{O}(n^{2/3})$ random bits for the selection of centers and marked clusters. For the emulation of the algorithm of Elkin and Neiman it suffices that the random variables $\{r_u\}$ be $\tilde{O}(n^{1/3})$-wise independent, because the outcome of the algorithm for a vertex $v \in R$ depends only on the random variables of at most $\tilde{O}(n^{1/3})$ vertices (the vertices in $\Gamma_\ell(v)$). Thus, overall $\tilde{O}(n^{2/3})$ random bits (up to $\text{poly}(\Delta/\epsilon)$ factors) are sufficient. ◄

References

1   N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012.

2   Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Structures and Algorithms*, 45(2):139–184, 2014.

3   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, 2017.

4   G. Even, M. Medina, and D. Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 394–405, 2014. `doi:10.1007/978-3-662-44777-2_33`.

5   U. Feige, Y Mansour, and R. E. Schapire. Learning and inference in the presence of corrupted inputs. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 637–657, 2015.

6   Hendrik Fichtenberger, Reut Levi, Yadu Vasudev, and Maximilian Wötzel. On testing minor-freeness in bounded degree graphs with one-sided error. Unpublished manuscript, 2017.

7   R. Levi, G. Moshkovitz, D. Ron, R. Rubinfeld, and A. Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, 50:n/a–n/a, 2016. `doi:10.1002/rsa.20652`.

8   R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms*, 11(3):24:1–24:13, 2015.

9   R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. In *Proceedings of the Eighteenth International Workshop on Randomization and Computation (RANDOM)*, pages 826–842, 2014.

10 R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. *CoRR*, abs/1402.3609, 2014. URL: `http://arxiv.org/abs/1402.3609`.

11 R. Levi, R. Rubinfeld, and A. Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, pages 1–24, 2016.

12 Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages and Programming: Thirty-Ninth International Colloquium (ICALP)*, pages 653–664, 2012.

13   Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Proceedings of the Sixteenth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 260–273, 2013.

14   D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

15   D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:229–243, 1989.

16   L. S. Ram and E. Vicari. Distributed small connected spanning subgraph: Breaking the diameter bound. Technical report, ETH Zürich, 2011.

17   R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of The Second Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.

18   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*, pages 38–47, 2017. `doi: 10.1007/978-3-319-58747-9_6`.