Local Algorithms for Sparse Spanning Graphs*

Reut Levi[1], Dana Ron[2], and Ronitt Rubinfeld[3]

1 **School of Computer Science, Tel Aviv University.**
Tel Aviv 69978, Israel.
`reuti.levi@gmail.com`
2 **School of Electrical Engineering, Tel Aviv University.**
Tel Aviv 69978, Israel.
`danar@eng.tau.ac.il`
3 **CSAIL, MIT.**
Cambridge MA 02139, USA.
School of Electrical Engineering, Tel Aviv University.
Tel Aviv 69978, Israel.
`ronitt@csail.mit.edu`

Abstract

We initiate the study of the problem of designing sublinear-time (*local*) algorithms that, given an edge (u, v) in a connected graph $G = (V, E)$, decide whether (u, v) belongs to a sparse spanning graph $G' = (V, E')$ of G. Namely, G' should be connected and $|E'|$ should be upper bounded by $(1 + \epsilon)|V|$ for a given parameter $\epsilon > 0$. To this end the algorithms may query the incidence relation of the graph G, and we seek algorithms whose query complexity and running time (per given edge (u, v)) is as small as possible. Such an algorithm may be randomized but (for a fixed choice of its random coins) its decision on different edges in the graph should be consistent with the same spanning graph G' and independent of the order of queries.

We first show that for general (bounded-degree) graphs, the query complexity of any such algorithm must be $\Omega(\sqrt{|V|})$. This lower bound holds for graphs that have high expansion. We then turn to design and analyze algorithms both for graphs with high expansion (obtaining a result that roughly matches the lower bound) and for graphs that are (strongly) non-expanding (obtaining results in which the complexity does not depend on $|V|$). The complexity of the problem for graphs that do not fall into these two categories is left as an open question.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases local, spanning graph, sparse

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

When dealing with large graphs, it is often important to work with a sparse subgraph that maintains essential properties, such as connectivity, bounded diameter and other distance metric properties, of the original input graph. Can one provide fast random access to such a sparsified approximation of the original input graph? In this work, we consider the property of connectivity: Given a connected graph $G = (V, E)$, find a sparse subgraph of G' that spans G. This task can be accomplished by constructing a spanning tree in linear time. However, it can be crucial to *quickly* determine whether a particular edge e belongs to such

* This work was partially supported by the Israel Science Foundation grant number nos. 1147/09, 246/08, and 671/13 and by NSF grants CCF-1217423 and CCF-1065125.

a subgraph G', where by "quickly" we mean in time much faster than constructing all of G'. The hope is that by inspecting only some small local neighborhood of e, one can answer in such a way that maintains consistency with the same G' on queries to all edges. We focus on such algorithms, which are of use when we do not need to know the answer for every edge at any single point in time, or if there are several independent processes that want to determine the answer for edges of their choice, possibly in parallel.

If we insist that G' would have the *minimum* number of edges sufficient for spanning G, namely, that G' be a spanning tree, then it is easy to see that the task cannot be performed in general without inspecting almost all of G. Interestingly, this is in contrast to the seemingly related problem of estimating the weight of a minimum spanning tree in sublinear-time, which can be performed with complexity that does not depend on $n \overset{\text{def}}{=} |V|$ [11] (see further discussion in Subsection 1.3.5). To verify this observe that if G consists of a single path, then the algorithm must answer positively on all edges, while if G consists of a cycle, then the algorithm must answer negatively on one edge. However, the two cases cannot be distinguished without inspecting a linear number of edges. If on the other hand we allow the algorithm some more slackness, and rather than requiring that G' be a tree, require that it be relatively sparse, i.e., contains at most $(1+\epsilon)n$ edges, then the algorithm may answer positively on all cycle edges, so distinguishing between these two cases is no longer necessary.

We thus consider the above relaxed version of the problem and also allow the algorithm a small failure probability (for a precise formal definition, see Section 2). Our first finding (Theorem 1) is that even when allowing this relaxation, for general (bounded-degree) graphs, the algorithm must inspect $\Omega(\sqrt{n})$ edges in G in order to decide for a given e whether it belongs to the sparse spanning graph G' defined by the algorithm. We then turn to design several algorithms and analyze their performance for various families of graphs. The formal statements of our results can be found in Theorems 2, 3, 4, 5, and 6 as well as Corollaries 8 and 9. Here we provide a high-level description of our algorithms and the types of graphs they give meaningful results for.

1.1 Our results

1.1.1 Expanders

The first algorithm we provide, the Centers' Algorithm (which is discussed further in Subsection 1.2), gives meaningful results for graphs in which the neighborhoods of almost all the vertices in the graph expands in a similar rate. In particular, for graphs with high expansion we get query and running time complexity nearly $O(n^{1/2})$. Since our lower bound applies for graphs with high expansion we obtain that for these graphs, our algorithm is nearly optimal in terms of the complexity in n. More specifically, if the expansion of small sets (of size roughly $O(n^{1/2})$) is $\Omega(d)$, where d is the maximum degree in the graph, then the complexity of the algorithm is $n^{1/2+O(1/\log d)}$. In general, we obtain a sublinear complexity for graphs with expansion (of small sets) that is at least $d^{1/2+1/\log n}$.

1.1.2 Anti-expanders (hyperfinite graphs) and slowly expanding graphs

A graph is *ρ-hyperfinite* for a function $\rho : \mathbb{R}_+ \to \mathbb{R}_+$, if its vertices can be partitioned into subsets of size at most $\rho(\epsilon)$ that are connected and such that the number of edges between the subsets is at most ϵn. For the family of hyperfinite graphs we provide an algorithm, the Kruskal-based algorithm, which has success probability 1 and time and query complexity $O(d^{\rho(\epsilon)})$. In particular, the complexity of the algorithm does not depend on n. (where we assume that $\rho(\epsilon)$ is known).

1.1.2.1 Subfamilies of hyperfinite graphs.

For the subfamily of hyperfinite graphs known as graphs with subexponential-growth, we can estimate the diameter of the sets in the partition and hence replace $\rho(\epsilon)$ with such an estimate. This reduces the complexity of the algorithm when the diameter is significantly smaller than $\rho(\epsilon)$, and removes the assumption that $\rho(\epsilon)$ is known. For the subfamily of graphs with an excluded minor (e.g., planar graphs) we can obtain a quasi-polynomial dependence on d and $1/\epsilon$ by using a *partition oracle* for such graphs [21], and the same technique gives polynomial dependence on these parameter for bounded-treewidth graphs (applying [14]).

1.1.2.2 Graphs with slow growth rate.

If we do not require that the algorithm work for every ϵ but rather for some fixed constant ϵ, then the Kruskal-based algorithm gives sublinear complexity under a weaker condition than that defining (ρ-)hyperfinite graphs (in which the desired partition should exist for every ϵ). Roughly speaking, the sizes of the neighborhoods of vertices should have bounded growth rate, where the rate may be exponential but the base of the exponent should be bounded (for details, see Theorem 6).

1.1.2.3 Graphs with an Excluded Minor - the Weighted Case.

In the full version of this paper [23] we provide a local minimum weight spanning graph algorithm, the Borůvka based algorithm, for *weighted* graphs with an excluded fixed minor. The minimum weight spanning graph problem is a generalization of the sparse spanning graph problem for the weighted case. The requirement is that the weight of the graph G' is upper bounded by $(1+\epsilon)$ times the minimum weight of a spanning tree. The time and query complexity of the algorithm are quasi-polynomial in $1/\epsilon$, d and W, where W is the maximum weight of an edge. We use ideas from [21], but the algorithm differs from the abovementioned partition-oracle based algorithm for the unweighted case.

1.2 Our algorithms

On a high-level, underlying each of our algorithms is the existence of a (global) partition of the graph vertices where edges within parts are dealt with differently than edges between parts, either explicitly by the algorithm, or in the analysis. The algorithms differ in the way the partitions are defined, where in particular, the number of parts in the partition may be relatively small or relatively large, and the subgraphs they induce may be connected or unconnected. The algorithms also differ in the way the spanning graph edges are chosen, and in particular whether only some of the edges between parts are selected or possibly all. While one of our algorithms works in a manner that is oblivious of the partition (and the partition is used only in the analysis), the other algorithms need to determine in a local manner whether the end points of the given edge belong to the same part or not, as a first step in deciding whether the edge belongs to the sparse spanning graph.

Centers' algorithm. This algorithm is based on the following idea. Suppose we can partition the graph vertices into $\sqrt{\epsilon n}$ disjoint parts where each part is connected. If we now take a spanning tree over each part and select one edge between every two parts that have an edge between them, then we obtain a connected subgraph with at most $(1+\epsilon)n$ edges. The partition is defined based on $\sqrt{\epsilon n}$ special *center* vertices, which are selected uniformly at random. Each vertex is assigned to the closest center (breaking ties according to a fixed order over the centers), unless it is further than some threshold k from all centers, in which case it is a singleton in the partition. This definition ensures the connectivity of each part.

Given an edge (x, y), the algorithm finds the centers to which x and y are assigned (or determines that they are singletons). If x and y are assigned to the same center, then the algorithm determines whether the edge between them belongs to the BFS-tree rooted at the center. If they belong to different centers, then the algorithm determines whether (x, y) is the single edge allowed to connect the parts corresponding to the two centers (according to a prespecified rule).[1] If one of them is a singleton, then (x, y) is taken to the spanning graph.

From the above description one can see that there is a certain tension between the complexity of the algorithm and the number of edges in the spanning graph G'. On one hand, the threshold k should be such that with high probability (over the choice of the centers) almost all vertices have a center at distance at most k from them. Thus we need a lower bound (of roughly $\sqrt{n/\epsilon}$) on the size of the distance-k neighborhood of (most) vertices. On the other hand, we also need an upper bound on the size of such neighborhoods so that we can efficiently determine which edges are selected between parts. Hence, this algorithm works for graphs in which the sizes of the aforementioned local neighborhoods do not vary by too much and its complexity (in terms of the dependence on n) is $\tilde{O}(\sqrt{n})$. In particular this property holds for good expander graphs. We note that the graphs used in our lower bound construction have this property, so for such graphs we get a roughly tight result.

Kruskal based algorithm. This algorithm is based on the well known algorithm of Kruskal [19] for finding a minimum weight spanning tree in a weighted graph. We use the order over the edges that is defined by the ids of their endpoints as (distinct) "weights". This ensures that there is a unique "minimum weight" spanning tree. Here the algorithm simply decides whether to include an edge in the spanning graph G' if it does not find evidence in the distance-k neighborhood of the edge that it is the highest ranking (maximum weight) edge on some cycle.

Borůvka based algorithm. This algorithm is based on the "Binary Borůvka" algorithm [36] for finding a minimum-weight spanning tree. Recall that Borůvka's algorithm begins by first going over each vertex in the graph and adding the lightest edge adjacent to that vertex. Then the algorithm continues joining the formed clusters in a similar manner until a tree spanning all vertices is completed. We aim to locally simulate the execution of Borůvka's algorithm to a point that on one hand all the clusters are relatively small and on the other hand the number of edges outside the clusters is small. The size of the clusters directly affect the complexity of the algorithm and thus our main challenge is in maintaining these clusters small. To this end we use two different techniques. The first technique is to control the growth of the clusters at each iteration by using a certain random orientation on the edges of the graph. This controls the size of the clusters to some extent. In order to deal with clusters that exceeded the required bound (since the local simulation is recursive even small deviations can have large impact on the complexity), after each iteration we separate large clusters into smaller ones (here we use the fact that the graph excludes a fixed minor in order to obtain a small separator to each cluster).

[1] In fact, it may be the case that for two parts that have edges between them, none of the edges are taken, thus making the argument that the subgraph G' is connected more subtle.

1.3 Related work

1.3.1 Local algorithms for other graph problems

The model of *local computation algorithms* as used in this work, was defined by Rubinfeld et al. [37] (see also [2]). Such algorithms for maximal independent set, hypergraph coloring, k-CNF and maximum matching are given in [37, 2, 26, 27]. This model generalizes other models that have been studied in various contexts, including locally decodable codes (e.g., [25]), local decompression [13], and local filters/reconstructors [1, 38, 9, 18, 17, 12]. Local computation algorithms that give approximate solutions for various optimization problems on graphs, including vertex cover, maximal matching, and other packing and covering problems, can also be derived from sublinear time algorithms for parameter estimation [33, 28, 31, 15, 41].

Campagna et al. [10] provide a local reconstructor for connectivity. Namely, given a graph which is almost connected, their reconstructor provides oracle access to the adjacency matrix of a connected graph which is close to the original graph. We emphasize that our model is different from theirs, in that they allow the addition of new edges to the graph, whereas our algorithms must provide spanning graphs whose edges are present in the original input graph.

1.3.2 Distributed algorithms

The name *local algorithms* is also used in the distributed context [29, 30, 24]. As observed by Parnas and Ron [33], local distributed algorithms can be used to obtain local computation algorithms as defined in this work, by simply emulating the distributed algorithm on a sufficiently large subgraph of the graph G. However, while the main complexity measure in the distributed setting is the number of rounds (where it is usually assumed that each message is of length $O(\log n)$), our main complexity measure is the number of queries performed on the graph G. By this standard reduction, the bound on the number of queries (and hence running time) depends on the size of the queried subgraph and may grow exponentially with the number of rounds. Therefore, this reduction gives meaningful results only when the number of rounds is significantly smaller than the diameter of the graph.

The problem of computing a minimum weight spanning tree in this model is a central one. Kutten and Peleg [20] provided an algorithm that works in $O(\sqrt{n}\log^* n + D)$ rounds, where D denotes the diameter of the graph. Their result is nearly optimal in terms of the complexity in n, as shown by Peleg and Rubinovich [34] who provided a lower bound of $\Omega(\sqrt{n}/\log n)$ rounds (when the length of the messages must be bounded).

Another problem studied in the distributed setting that is related to the one studied in this paper, is finding a sparse spanner. The requirement for spanners is much stronger since the distortion of the distance should be as minimal as possible. Thus, to achieve this property, it is usually the case that the number of edges of the spanner is super-linear in n. Pettie [35] was the first to provide a distributed algorithm for finding a low distortion spanner with $O(n)$ edges without requiring messages of unbounded length or $O(D)$ rounds. The number of rounds of his algorithm is $\log^{1+o(1)} n$. Hence, the standard reduction of [33] yields a local algorithm with a trivial linear bound on the query complexity.

1.3.3 Parallel algorithms

The problems of computing a spanning tree and a minimum weight spanning tree were studied extensively in the parallel computing model (see, e.g., [6], and the references therein). However, these parallel algorithms have time complexity which is at least logarithmic in n

and therefore do not yield an efficient algorithm in the local computation model. See [37, 2] for further discussion on the relationship between the ability to construct local computation algorithms and the parallel complexity of a problem.

1.3.4 Local cluster algorithms

Local algorithms for graph theoretic problems have also been given for PageRank computations on the web graph [16, 7, 39, 4, 3]. Local graph partitioning algorithms have been presented in [40, 4, 5, 42, 32], which find subsets of vertices whose internal connections are significantly richer than their external connections in time that depends on the size of the cluster that they output. Even when the size of the cluster is guaranteed to be small, it is not obvious how to use these algorithms in the local computation setting where the cluster decompositions must be consistent among queries to all vertices.

1.3.5 Other related sublinear-time approximation algorithms for graphs

The problem of estimating the weight of a minimum weight spanning tree in sublinear time was considered by Chazelle, Rubinfeld and Trevisan [11]. They describe an algorithm whose running time depends on the approximation parameter, the average degree and the range of the weights, but does not directly depend on the number of nodes. A question that has been open since that time, even before local computation algorithms were formally defined, is whether it is possible to quickly determine which edges are in the minimum spanning tree. Our lower bound for spanning trees applies to this question.

2 Preliminaries

The graphs we consider have a known degree bound d, and we assume we have query access to their incidence-lists representation. Namely, for any vertex v and index $1 \leq i \leq d$ it is possible to obtain the i^{th} neighbor of v by performing a query to the graph (if v has less than i neighbors, then a special symbol is returned).[2] If the graph is edge-weighted, then the weight of the edge is returned as well. The number of vertices in the graph is n and we assume that each vertex v has an id, $id(v)$, where there is a full order over the ids.

Let $G = (V, E)$ be a graph. 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 k, let $\Gamma_k(v, G)$ denote the set of vertices at distance at most k from v and let $C_k(v, G)$ denote the subgraph of G induced by $\Gamma_k(v, G)$. Let $n_k(G) \overset{\text{def}}{=} \max_{v \in V} |\Gamma_k(v, G)|$. When the graph G is clear from the context, we shall use the shorthand $d(u, v)$, $\Gamma_k(v)$ and $C_k(v)$ for $d_G(u, v)$, $\Gamma_k(v, G)$ and $C_k(v, G)$, respectively.

▶ Definition 1 (Local Algorithms for sparse spanning graphs). An algorithm $\mathcal{A}$ is a *local sparse spanning graph algorithm* if, given parameters $n \geq 1$, $\epsilon \geq 0$ and $0 \leq \delta < 1$ and given query access to the incidence-lists representation of a graph $G = (V, E)$, the algorithm $\mathcal{A}$ provides query access to a subgraph of G, $G' = (V, E')$ such that the following hold:

1. G' is connected with probability 1.
2. $|E'| \leq (1 + \epsilon) \cdot n$ with probability at least $1 - \delta$, where the probability is taken over the internal coin flips of $\mathcal{A}$.

[2] Graphs are allowed to have self-loops and multiple edges, but for our problem we may assume that there are no self-loops and multiple-edges (since the answer on a self-loop can always be negative, and the same is true for all but at most one among a set of parallel edges).

3. E' is determined by G and the internal randomness of the oracle.
Namely, on input $(u, v) \in E$, $\mathcal{A}$ returns whether $(u, v) \in E'$ and for any sequence of queries, $\mathcal{A}$ answers consistently with the same G'.

An algorithm $\mathcal{A}$ is a *local sparse spanning graph algorithm with respect to a class of graphs* $\mathcal{C}$ if the above conditions hold, provided that the input graph G belongs to $\mathcal{C}$.

We are interested in local algorithms that have small query complexity, namely, that perform few queries to the graph (for each edge they are queried on) and whose running time (per queried edge) is small as well. As for the question of randomness and the implied space complexity of the algorithms, we assume we have a source of (unbounded) public randomness. Under this assumption, our algorithms do not keep a state and a global space is not required. However, if unbounded public randomness is not available, then we note that for our algorithms this is not an issue: One of our algorithms (see Section 5) is actually deterministic, and for the others, the total number of random bits that is actually required (over all possible queries) is upper bounded by the running time of the algorithm, up to a multiplicative factor of $O(\log n)$. In what follows we sometimes describe a global algorithm first, i.e., an algorithm that reads the entire graph and decides the subgraph G'. After that we describe how to locally emulate the global algorithm. Namely on query $e \in E$, we emulate the global algorithm decision on e while performing only a sublinear number of queries.

3 A Lower Bound for General Bounded-Degree Graphs

▶ Theorem 1. Any local sparse spanning graph algorithm has query complexity $\Omega(\sqrt{n})$. This result holds for graphs with a constant degree bound d and for constant $0 \le \epsilon \le 2d/3$ and $0 \le \delta < 1/3$.

The full proof can be found in the full version of this paper. Here we give the high-level idea. Let V be a set of vertices and let v_0 and v_1 be a pair of distinct vertices in V. In order to prove the lower bound we construct two families of random d-regular graphs over V, $\mathcal{F}^{+}_{(v_0,v_1)}$ and $\mathcal{F}^{-}_{(v_0,v_1)}$. $\mathcal{F}^{+}_{(v_0,v_1)}$ is the family of d-regular graphs, $G = (V, E)$, for which $(v_0, v_1) \in E$. $\mathcal{F}^{-}_{(v_0,v_1)}$ is the family of d-regular graphs for which $(v_0, v_1) \in E$ and the removal of (v_0, v_1) leaves the graph with two connected components, each of size [3] [4] $n/2$. We prove that given (v_0, v_1), any algorithm that performs at most $\sqrt{n}/c$ queries for some sufficiently large constant $c > 1$ cannot distinguish the case in which the graph is drawn uniformly at random from $\mathcal{F}^{+}_{(v_0,v_1)}$ from the case in which the graph is drawn uniformly at random from $\mathcal{F}^{-}_{(v_0,v_1)}$. Essentially, if the number of queries is at most $\sqrt{n}/c$, then with high constant probability, each new query to the graph returns a new random vertex in both families. By "new vertex" we mean a vertex that neither appeared in the query history nor in the answers history. Since the algorithm must answer consistently with a connected graph G', for every graph in the support of $\mathcal{F}^{-}_{(v_0,v_1)}$ it must answer with probability 1 positively on the query (v_0, v_1). But since the distributions on query-answer histories in both cases are very close statistically, this can be shown to imply that there exist graphs for which the algorithm answers positively on a large fraction of the edges.

[3] Although a graph that is drawn uniformly from $\mathcal{F}^{+}_{(v_0,v_1)}$ (or $\mathcal{F}^{-}_{(v_0,v_1)}$) might be disconnected, this event happens with negligible probability [8]. Hence, the proof of the lower bound remains valid even if we consider $\mathcal{F}^{+}_{(v_0,v_1)} \cap \mathcal{C}$ and $\mathcal{F}^{-}_{(v_0,v_1)} \cap \mathcal{C}$ where $\mathcal{C}$ is the family of connected graphs.

[4] Assume the without loss of generality that $n \cdot d$ is even and that $(n \cdot d)/2$ is odd.

4 Graphs with High Expansion

In this section we describe an algorithm that gives meaningful results for graphs in which, roughly speaking, the local neighborhood of almost all vertices expands in a similar rate. In particular this includes graphs with high expansion. In fact we only require that the graph expands quickly for small sets: A graph G is an (s, α)*-vertex expander* if for all sets S of size at most s, $N(S)$ is of size at least $\alpha|S|$, where $N(S)$ denotes the set of vertices adjacent to vertices in S that are not in S. Define $h_s(G)$ to be the maximum α such that G is an (s, α)-vertex expander. We prove the following theorem in the full version of this paper.

▶ Theorem 2. Given a graph $G = (V, E)$ with degree bound d, there is a local sparse spanning graph algorithm with query complexity and running time $(d \cdot s)^{\log_{h_s(G)} d}$ where $s = s(n, \epsilon, \delta) \overset{\text{def}}{=} \sqrt{2n/\epsilon} \cdot \log(n/\delta)$.

By Theorem 2, for bounded degree graphs with high expansion we get query and running time complexity nearly $O(n^{1/2})$. In particular, if $h_s(G) = \Omega(d)$ for $s = s(n, \epsilon, \delta)$ then the complexity is $n^{1/2+O(1/\log d)}$. In fact, even for $h_s(g) \geq d^{1/2+1/\log n}$ the complexity is $o(n)$. Recall that in the construction of our lower bound of $\Omega(n^{1/2})$ we construct a pair of families of d-regular random graphs. In both families, the expansion (of small sets) is $\Omega(d)$, implying that for these families the gap between our lower bound and upper bound is at most $n^{O(1/\log d)}$.

Our algorithm, the local Centers' Algorithm (which appears as Algorithm 1), is based on a global algorithm which is presented in Subsection 4.1. The local Centers' Algorithm appears in Subsection 4.2 and it is analyzed in the proof of Theorem 3. Theorem 2 follows easily from Theorem 3.

4.1 The Global Algorithm

For a given parameter k the global algorithm first defines a global partition of (part or all of) the graph vertices in the following randomized manner.

1. Select $\ell = \sqrt{\epsilon n/2}$ *centers* uniformly and independently at random from V, and denote them $v_1, \ldots, v_\ell$. Initially, all vertices are *unassigned*.
2. For $i = 0, \ldots, k$, for $j = 1, \ldots, \ell$:
   Let L_j^i denote the vertices in the i^{th} level of the BFS tree of v_j (where $L_j^0 = \{v_j\}$). Assign to v_j all vertices in L_j^i that were not yet assigned to any other $v_{j'}$.

Let $S(v_j)$ denote the set of vertices that are assigned to the center v_j. By the above construction, the subgraph induced by $S(v_j)$ is connected.

The subgraph $G' = (V, E')$ is defined as follows.

1. For each center v, let $E'(v)$ denote the edges of a BFS-tree that spans the subgraph induced by $S(v)$ (where the BFS-tree is determined by the order over the ids of the vertices in $S(v)$). For each center v, put in E' all edges in $E'(v)$.
2. For each vertex w that does not belong to any $S(v)$ for a center v, put in E' all edges incident to w.
3. For each pair of centers u and v, let $P(u, v)$ be the shortest path between u and v that has minimum lexicographic order among all shortest paths (as determined by the ids of the vertices on the path). If all vertices on this path belong either to $S(u)$ or to $S(v)$, then add to E' the single edge $(x, y) \in P(u, v)$ such that $x \in S(u)$ and $y \in S(v)$, where we denote this edge by $e(u, v)$.

In what follows we shall prove that G' is connected and that for k that is sufficiently large, G' is sparse with high probability as well. We begin by proving the latter claim. To this end we define a parameter which determines the minimum distance needed for most vertices to see roughly $\sqrt{n}$ vertices. More formally, define $k^C_{\epsilon,\delta}(G)$ to be the minimum distance k ensuring that all but an $\epsilon/(2d)$-fraction of the vertices have at least $s(n,\epsilon,\delta)$ vertices in their k-neighborhood. That is,

$$k^C_{\epsilon,\delta}(G) \stackrel{\text{def}}{=} \min_k \{|\{v : \Gamma_k(v) \geq s(n,\epsilon,\delta)\}| \geq (1-\epsilon/(2d))\,|V|\} \ . \tag{1}$$

We next establish that for $k \geq k^C_{\epsilon,\delta}(G)$ it holds that $|E'| \leq (1+\epsilon)n$ with probability at least $1-\delta$, over the random choice of centers. Since for $j = 1,\ldots,\ell$ the sets $E'(v_j)$ are disjoint, we have that $\left|\bigcup_{j=1}^{\ell} E'(v_j)\right| < n$. Since there is at most one edge $e(u,v)$ added to E' for each pair of centers u, v and the number of centers is $\ell = \sqrt{\epsilon n/2}$, the total number of these edges in E' is at most $\epsilon n/2$. Finally, Let $T \subseteq V$ denote the subset of the vertices, v, such that $|\Gamma_k(v)| \geq s(n,\epsilon,\delta)$. Since the centers are selected uniformly, independently at random, for each $w \in T$ the probability that no vertex in $\Gamma_k(w)$ is selected to be a center is at most $(1-\log(n/\delta)/\sqrt{\epsilon n/2})^{\sqrt{\epsilon n/2}} < \delta/n$. By taking a union bound over all vertices in T, with probability at least $1-\delta$, every $w \in T$ is assigned to some center v. Since the number of vertices in $V \setminus T$ is at most $\epsilon n/(2d)$ and each contributes at most d edges to E', we get the desired upper bound on $|E'|$.

It remains to establish that G' is connected. To this end it suffices to prove that there is a path in G' between every pair of centers u and v. This suffices because for each vertex w that is assigned to some center v, there is a path between w and v (in the BFS-tree of v), and for each vertex w that is not assigned to any center, all edges incident to w belong to E'. The proof proceeds by induction on $d(u,v)$ and the sum of the ids of u and v as follows. For the base case consider a pair of centers u and v for which $d(u,v) = 1$. In this case, the shortest path $P(u,v)$ consists of a single edge (u,v) where $u \in S(u)$ and $v \in S(v)$, implying that $(u,v) \in E'$. For the induction step, consider a pair of centers u and v for which $d(u,v) > 1$, and assume by induction that the claim holds for every pair of centers (u',v') such that either $d(u',v') < d(u,v)$ or $d(u',v') = d(u,v)$ and $id(u') + id(v') < id(u) + id(v)$. Similarly to base case, if the set of vertices in $P(u,v)$ is contained entirely in $S(u) \cup S(v)$, then u and v are connected by construction. Namely, $P(u,v) = (u, x_1, \ldots, x_t, y_s, \ldots, y_1, v)$ where $x_1,\ldots,x_t \in S(u)$ and $y_1,\ldots,y_s \in v$. The edge (x_t, y_s) was added to E' and there are paths in the BFS-trees of u and v between u and x_t and between v and y_s, respectively. Otherwise, we consider two cases.

1. There exists a vertex x in $P(u,v)$, and a center (different from u and v), y, such that $x \in S(y)$. Note that this must be the case when $d(u,v) \leq 2k+1$. This implies that either $d(x,y) < d(x,v)$ or that $d(x,y) = d(x,v)$ and $id(y) < id(v)$. Hence, either

$$d(u,y) \leq d(u,x) + d(x,y) < d(u,x) + d(x,v) = d(u,v)$$

   or $d(u,y) = d(u,v)$ and $id(u) + id(y) < id(u) + id(v)$. In either case we can apply the induction hypothesis to obtain that u and y are connected. A symmetric argument gives us that v and y are connected.
2. Otherwise, all the vertices on the path $P(u,v)$ that do not belong to $S(u) \cup S(v)$ are vertices that are not assigned to any center. Since E' contains all edges incident to such vertices, u and v and connected in this case as well.

4.2 The Local Algorithm

▶ Theorem 3. *Algorithm 1, when run with $k \geq k^C_{\epsilon,\delta}(G)$, is a local sparse spanning graph algorithm. The query complexity and running time of the algorithm are $O(d \cdot n_k(G))$.*

Algorithm 1 (Centers' Algorithm)

For a random choice of $\ell = \sqrt{\epsilon n/2}$ centers, $v_1, \ldots, v_\ell$ in V (which is fixed for all queries), and for a given parameter k, on query (x, y):

```
1. Perform a BFS to depth $k$ in $G$ from $x$ and from $y$.
2. If either $\Gamma_k(x) \cap \{v_1, \ldots, v_\ell\} = \emptyset$ or $\Gamma_k(y) \cap \{v_1, \ldots, v_\ell\} = \emptyset$, then return YES.
3. Otherwise, let $u$ be the center closest to $x$ and let $v$ be the center closest to $y$ (if there is
   more than one such center, break ties according to the order $v_1, \ldots, v_\ell$).
4. If $u = v$ then do the following:
   - If $d(x, u) = d(y, u)$, then return NO.
   - If $d(y, u) = d(x, u) + 1$, then consider all neighbors of $y$, $w$, on a shortest path between
     $y$ and $u$. If there exists such neighbor $w$ for which $id(w) < id(x)$, then return NO,
     otherwise, return YES.
5. If $u \neq v$, then perform a BFS of depth $k$ from both of the centers, $u$ and $v$. Find the
   shortest path between $u$ and $v$ that has the smallest lexicographical order, and denote it
   by $P(u, v)$. Return YES if both $x \in P(u, v)$ and $y \in P(u, v)$. Otherwise, return NO.
```

Notice that Algorithm 1 is given a parameter k that determines the depth of the BFS that the algorithm performs. By Theorem 3 is suffices to require that $k \geq k^C_{\epsilon,\delta}(G)$ in order to ensure that the spanning graph obtained by the algorithm is sparse. In the full version of this paper we describe how to compute k efficiently and prove Theorem 2.

Proof of Theorem 3: We prove the theorem by showing that Algorithm 1 is a local emulation of the global algorithm that appears in Subsection 4.1. Given x and y, by performing a BFS to depth k from each of the two vertices, Algorithm 1 either finds the centers u and v that x and y are (respectively) assigned to (by the global algorithm, for the same selection of centers), or for at least one of them it finds no center in the distance k neighborhood. In the latter case, the edge (x, y) belongs to E', and Algorithm 1 returns a positive answer, as required. In the former case, there are two subcases.

1. If x and y are assigned to the same center, that is, $u = v$, then Algorithm 1 checks whether the edge (x, y) is an edge in the BFS-tree of u (i.e., $(x, y) \in E'(u)$). If x and y are on the same level of the tree (i.e., are at the same distance from u), then Algorithm 1 returns a negative answer, as required. If y is one level further than x, then Algorithm 1 checks whether y has another neighbor w that is also assigned to u, is on the same level as x and has a smaller id than x. Namely, a neighbor of y that is on a shortest path between y and u and has a smaller id than x. If this is the case, then the edge (x, y) does not belong to the tree (but rather the edge (w, y)) so that the algorithm returns a negative answer. If no such neighbor of y exists, then the algorithm returns a positive answer (as required).
2. If x and y are assigned to different centers, that is, $u \neq v$, then Algorithm 1 determines whether $(x, y) = e(u, v)$ exactly as defined in the global algorithm: The algorithm finds $P(u, v)$ and returns a positive answer if and only if (x, y) belongs to $P(u, v)$. Notice that from the fact that $x \in S(u)$ and $y \in S(v)$ and the fact that (x, y) belongs to $P(u, v)$ it follows that all the vertices on $P(u, v)$ belong to either $S(u)$ or $S(v)$. This is implied from the fact that for every center u and a vertex which is assigned to u, w, it holds that every vertex on a shortest path between u and w is also assigned to u.

Finally, the bound on the query complexity and running time of Algorithm 1 follows directly by inspection of the algorithm. ◀

5 Hyperfinite Graphs

In this section we provide an algorithm that is designed for the family of hyperfinite graphs. Roughly speaking, hyperfinite graphs are non-expanding graphs. Formally, a graph $G = (V, E)$ is (ϵ, k)-*hyperfinite* if it is possible to remove at most $\epsilon|V|$ edges of the graph so that the remaining graph has connected components of size at most k. We refer to these edges as the *separating edges* of G. A graph G is ρ-*hyperfinite* for $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ if for every $\epsilon \in (0, 1]$, G is $(\epsilon, \rho(\epsilon))$-hyperfinite. The family of hyperfinite graphs includes many subfamilies of graphs such as graphs with an excluded-minor (e.g. planar graphs), graphs that have subexponential growth and graphs with bounded treewidth. The complexity of our algorithm does not depend on the size of the graph as stated in the next theorem.

▶ Theorem 4. Algorithm 2, when run with $k = \rho(\epsilon)$, is a local sparse spanning graph algorithm for the family of ρ-hyperfinite graphs with a degree bounded by d. The query complexity and running time of Algorithm 2 are $O(d^{\rho(\epsilon)+1})$, and its success probability is 1.

We note that we could also obtain a local sparse spanning graph algorithm for hyperfinite graphs by using the partition oracle of [15] (see the reduction described in Section 6) but the complexity would be higher ($O(d^{d^{\rho(\epsilon)}})$).

We present Algorithm 2 in Subsection 5.1. In Subsection 5.2 we give an improved analysis for the subfamily of graphs that have subexponential growth.

5.1 The Algorithm

Recall that Kruskal's algorithm for finding a minimum-weight spanning tree in a weighted connected graph works as follows. First it sorts the edges of the graph from minimum to maximum weight (breaking ties arbitrarily). Let this order by $e_1, \ldots e_m$. It then goes over the edges in this order, and adds e_i to the spanning tree if and only if it does not close a cycle with the previously selected edges. It is well known (and easy to verify), that if the weights of the edges are distinct, then there is a single minimum weight spanning tree in the graph. For an unweighted graph G, consider the order defined over its edges by the order of the ids of the vertices. Namely, we define a ranking r of the edges as follows: $r(u, v) < r(u', v')$ if and only if $\min\{id(u), id(v)\} < \min\{id(u'), id(v')\}$ or $\min\{id(u), id(v)\} = \min\{id(u'), id(v')\}$ and $\max\{id(u), id(v)\} < \max\{id(u'), id(v')\}$. If we run Kruskal's algorithm using the rank r as the weight function (where there is a single ordering of the edges), then we obtain a spanning tree of G.

While the local algorithm we give in this section (Algorithm 2) is based on the aforementioned global algorithm, it does not exactly emulate it, but rather emulates a certain *relaxed* version of it. In particular, it will answer YES for every edge selected by the global algorithm (ensuring connectivity), but may answer YES also on edges not selected by the global algorithm.

Algorithm 2 (Kruskal-based Algorithm)

The algorithm is provided with an integer parameter k, which is fixed for all queries. On query (x, y):

1. Perform a BFS to depth k from x, thus obtaining the subgraph $C_k(x)$ induced by $\Gamma_k(x)$ in G.
2. If (x, y) is the edge with largest rank on some cycle in $C_k(x)$, then answer NO, otherwise, answer YES.

Proof of Theorem 4: By the description of Algorithm 2 it directly follows that its answers are consistent with a connected subgraph G'. We next show that the algorithm returns YES on at most $(1+\epsilon)n$ edges. Let $k = \rho(\epsilon)$. For a vertex u, let $\widetilde{C}(u) = (\widetilde{V}(u), \widetilde{E}(u))$ denote the component of u after the removal of the separating edges (as defined at the start of the subsection). We next prove that G' does not contain a cycle on the subgraph induced on $\widetilde{V}(u)$. In our proof we use properties of $\widetilde{C}(u)$, however, we note that the algorithm does not compute $\widetilde{C}(u)$. By definition, $|\widetilde{V}(u)| \leq k$, thus the diameter of $\widetilde{C}(u)$ is at most $k-1$. This implies that $C_k(u)$ contains $\widetilde{C}(u)$ for every $u \in G$. Let σ be a cycle in $\widetilde{C}(u)$ and let $e = (w, v)$ be the edge in σ with the largest rank. Since $\widetilde{C}(u) = \widetilde{C}(v) = \widetilde{C}(w)$ it follows that on query (w, v) the algorithm returns NO. We conclude that for every $u \in V$ the algorithm returns YES only on at most $|\widetilde{V}(u)| - 1$ among the edges in $\widetilde{E}(u)$. Since the number of edges that do not belong to any component $\widetilde{C}(u)$, that is, the number of separating edges in an $(\epsilon, k = \rho(\epsilon))$-hyperfinite graph is at most $\epsilon|V|$ we have that the total number of edges for which the algorithm returns YES is at most $(1+\epsilon)|V|$. ◀

5.2 Graphs with Subexponential-Growth

In this subsection we analyze Algorithm 2 when executed on graphs with subexponential-growth and for an appropriate k. We first show that graphs with subexponential-growth are $(\epsilon, \rho(\epsilon))$-hyperfinite. In order to obtain an improved analysis of the complexity of Algorithm 2 for graphs with subexponential-growth, we bound not only the size of each component but also the diameter of each component.

A monotone function $f : \mathbb{N} \to \mathbb{N}$ has *subexponential growth* if for any $\beta > 0$, there exists $r_f(\beta) > 0$ such that $f(r) \leq \exp(\beta \cdot r)$ for all $r \geq r_f(\beta)$. A graph G has *growth bounded by f* if for every $k \geq 1$, $n_k(G) \leq f(k)$.

▶ Theorem 5. Given a graph $G = (V, E)$ with degree bounded by d that has growth bounded by $f : \mathbb{N} \to \mathbb{N}$ where f has subexponential growth, there is a local sparse spanning graph algorithm with query complexity and running time $O(d \cdot n_{r_f(\beta)}(G)) = O(d \cdot \exp(\beta \cdot r_f(\beta)))$ for $\beta = \frac{\epsilon}{2d}$.

Recall that Algorithm 2 is provided with an integer parameter, k, which determines the depth of the BFS that is performed by the algorithm. In case the graph is $(\epsilon, \rho(\epsilon))$-hyperfinite we showed that setting $k = \rho(\epsilon)$ is sufficient. For a general graph G, we next define another parameter which is also sufficient for bounding the required depth of the BFS, as we show in Theorem 6. Thereafter, we shall prove that for graphs with subexponential-growth this parameter is small and can be computed efficiently.

Define $k^K_{\alpha,\beta}(G)$ to be the minimum distance k ensuring that all but an α-fraction of the vertices have at most $\exp(\beta k/2)$ vertices in their k-neighborhood (k is allowed to be larger than the diameter of G so that $k^K_{\alpha,\beta}(G)$ is well defined). Formally,

$$k^K_{\alpha,\beta}(G) = \min_k \{|\{v : |\Gamma_k(v)| \leq \exp(\beta k/2)\}| \geq (1-\alpha)\,|V|\} \ . \tag{2}$$

▶ Theorem 6. Algorithm 2, when run with $k \geq k^K_{\alpha,\beta}(G)$, where $\alpha + \beta = \epsilon/d$, is a local sparse spanning graph algorithm. The query complexity and running time of the algorithm are $O(dn_k(G)) = O(d^{k+1})$.

Theorem 5 follows directly from Theorem 6 and Theorem 6 follows directly from the proof of Theorem 4 and the following lemma.

▶ Lemma 1. Every graph $G = (V, E)$ is $(\epsilon, (1+\beta)^k)$-hyperfinite for $k = k^K_{\alpha,\beta}(G)$ and $\alpha + \beta = \epsilon/d$. Moreover, it is possible to remove at most $\epsilon|V|$ edges of the graph so that the remaining graph has connected components with diameter at most $2k$.

Proof: Let $S \subseteq V$ denote the set of vertices, v, for which $|\Gamma_k(v)| > \exp(\beta k/2)$. We start by removing all the edges adjacent to vertices in S. Overall, we remove at most $d\alpha|V|$ edges. For each vertex $v \in V - S$ it holds that $|\Gamma_k(v)| \leq \exp(\beta k/2)$. From the fact that $\exp(x) < 1 + 2x$ for every $x < 1$ we obtain that $|\Gamma_k(v)| < (1+\beta)^k$. Therefore, there exists $k' < k$ such that $|\Gamma_{k'+1}(v)| < |\Gamma_{k'}(v)|(1+\beta)$. Thus, $C_{k'}(v)$ can be disconnected from G by removing at most $d\beta|\Gamma_{k'}(v)|$ edges. Since it holds that $|\Gamma_k(v)| < (1+\beta)^k$ for every subgraph of G and every $v \in V - S$, we can continue to iteratively disconnect connected components of diameter at most $2k$ from the resulting graph. Hence, we obtain that by removing at most $d(\alpha+\beta)|V|$ edges, the remaining graph has connected components with diameter at most $2k$, as desired. ■

5.3 The Parameter k

Recall that Algorithm 2 is given a parameter k that determines the depth of the BFS that the algorithm performs. By Theorem 6 it is sufficient to require that $k \geq k^K_{\epsilon/(2d),\epsilon/(2d)}(G)$ in order to ensure that the resulting graph is sparse. For the case that k is not given in advance, we can compute k such that with probability greater than $1-\delta$ it holds that

$$k^K_{\epsilon/(2d),\epsilon/(2d)}(G) \leq k \leq k^K_{\epsilon/(4d),\epsilon/(2d)}(G) \ , \tag{3}$$

as follows. Sample $\Theta(1/\epsilon^2 \log(1/\delta))$ vertices. Start with $k = 1$ and iteratively increase k until for at least $\left(1 - \frac{3\epsilon}{8d}\right)$-fraction of the vertices, v, in the sample it holds that $|\Gamma_k(v)| \leq \exp(\epsilon k/(4d))$. By Chernoff's inequality we obtain that with probability greater than $1-\delta$ Equation (3) holds.

6 Partition Oracle Based Algorithm

In this section we describe a simple reduction from local algorithm for sparse spanning graph to partition oracle. We begin with a few definitions concerning partition oracles.

▶ Definition 2. For $\epsilon \in (0,1]$, $k \geq 1$ and a graph $G = (V,E)$, we say that a partition $\mathcal{P} = (V_1,\ldots,V_t)$ of V is an (ϵ,k)-*partition* (with respect to G), if the following conditions hold:

1. For every $1 \leq i \leq t$ it holds that $|V_i| \leq k$;
2. For every $1 \leq i \leq t$ the subgraph induced by V_i in G is connected;
3. The total number of edges whose endpoints are in different parts of the partition is at most $\epsilon|V|$ (that is, $|\{(v_i,v_j) \in E : \ v_i \in V_j, v_j \in V_j, i \neq j\}| \leq \epsilon|V|$).

Let $G = (V,E)$ be a graph and let $\mathcal{P}$ be a partition of V. We denote by $g_{\mathcal{P}}$ the function from $v \in V$ to 2^V (the set of all subsets of V), that on input $v \in V$, returns the subset $V_\ell \in \mathcal{P}$ such that $v \in V_\ell$.

▶ Definition 3 ([15]). An oracle $\mathcal{O}$ is a *partition oracle* if, given query access to the incidence-lists representation of a graph $G = (V,E)$, the oracle $\mathcal{O}$ provides query access to a partition $\mathcal{P} = (V_1,\ldots,V_t)$ of V, where $\mathcal{P}$ is determined by G and the internal randomness of the oracle. Namely, on input $v \in V$, the oracle returns $g_{\mathcal{P}}(v)$ and for any sequence of queries, $\mathcal{O}$ answers consistently with the same $\mathcal{P}$. An oracle $\mathcal{O}$ is an (ϵ,k)-*partition oracle with respect to a class of graphs* $\mathcal{C}$ if the partition $\mathcal{P}$ it answers according to has the following properties.

1. For every $V_\ell \in \mathcal{P}$, $|V_\ell| \leq k$ and the subgraph induced by V_ℓ in G is connected.
2. If G belongs to $\mathcal{C}$, then $|\{(u,v) \in E : g_{\mathcal{P}}(v) \neq g_{\mathcal{P}}(u)\}| \leq \epsilon|V|$ with high constant probability, where the probability is taken over the internal coin flips of $\mathcal{O}$.

By the above definition, if $G \in \mathcal{C}$, then with high constant probability the partition $\mathcal{P}$ is an (ϵ, k)-partition, while if $G \notin \mathcal{C}$ then it is only required that each part of the partition is connected and has size at most k.

▶ Theorem 7. If there exists an (ϵ, k)-partition oracle, $\mathcal{O}$, for the family of graphs $\mathcal{C}$ having query complexity $q(\epsilon, k, d, n)$ and running time $t(\epsilon, k, d, n)$, then there exists a local sparse spanning graph algorithm, $\mathcal{A}$, for the family of graphs $\mathcal{C}$, whose success probability is the same as that of $\mathcal{O}$. The running time of of $\mathcal{A}$ is bounded from above by $t(\epsilon, k, d, n) + O(kd)$ and the query complexity of $\mathcal{A}$ is $q(\epsilon, k, d, n) + O(kd)$.

Proof: On query (u, v) the algorithm $\mathcal{A}$ proceeds as follows:

1. Query $\mathcal{O}$ on u and v and get $g_{\mathcal{P}}(u)$ and $g_{\mathcal{P}}(v)$, respectively.
2. If $g_{\mathcal{P}}(u) \neq g_{\mathcal{P}}(v)$, return YES.
3. Otherwise, let w denote the vertex in $g_{\mathcal{P}}(u)$ such that $id(w)$ is minimal.
4. Perform a BFS on the subgraph induced on $g_{\mathcal{P}}(u)$, starting from w.
5. If (u, v) belongs to the edges of the above BFS then return YES, otherwise, return NO.

The fact that $\mathcal{A}$ returns YES on at most $(1+\epsilon)|V|$ edges follows from the fact that $\mathcal{P}$ is a partition can that $|\{(u, v) \in E : g_{\mathcal{P}}(v) \neq g_{\mathcal{P}}(u)\}| \leq \epsilon|V|$. The connectivity follows from the fact that the subgraph induced on V_i is connected for every $V_i \in \mathcal{P}$. The additional term of $O(kd)$ in the time and query complexity is due to the BFS performed on $g_{\mathcal{P}}(u)$. ■

The following corollaries follow from [14] and [21, 22], respectively.

▶ Corollary 8. There exists a local sparse spanning graph algorithm for the family of graphs with bounded treewidth. This algorithm has high constant success probability and its query complexity and running time are poly$(1/\epsilon, d)$.

▶ Corollary 9. There exists a local sparse spanning graph algorithm for the family of graphs with a fixed excluded minor. This algorithm has high constant success probability and its query complexity and running time are $(d/\epsilon)^{O(\log(1/\epsilon))}$.

References

1 N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.

2 N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of SODA*, pages 1132–1139, 2012.

3 R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S. Teng. Local computation of pagerank contributions. *Internet Mathematics*, 5(1–2):23–45, 2008.

4 R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proceedings of FOCS*, pages 475–486, 2006.

5 R. Andersen and Y. Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of STOC*, pages 235–244, 2009.

6 D. A. Bader and G. Cong. A fast, parallel spanning tree algorithm for symmetric multiprocessors (smps). *J. Parallel Distrib. Comput.*, 65(9):994–1006, 2005.

7 P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.

8 B. Bollobás. *Random Graphs*. Cambridge University Press, 2001.

9 Z. Brakerski. Local property restoring. Unpublished manuscript, 2008.

10 A. Campagna, A. Guo, and R. Rubinfeld. Local reconstructors and tolerant testers for connectivity and diameter. In *Proceedings of RANDOM*, pages 411–424, 2013.

11 B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SICOMP*, 34(6):1370–1379, 2005.

12 B. Chazelle and C. Seshadhri. Online geometric reconstruction. In *Proceedings of the Twenty-Second Annual ACM Symposium on Computation Geometry (SoCG)*, pages 386 – 394, 2006.

13 A. Dutta, R. Levi, D. Ron, and R. Rubinfeld. A simple online competitive adaptation of lempel-ziv compression with efficient random access support. In *Proceedings of the Data Compression Conference (DCC)*, pages 113–122, 2013.

14 A. Edelman, A. Hassidim, H. N. Nguyen, and K. Onak. An efficient partitioning oracle for bounded-treewidth graphs. In *Proceedings of RANDOM*, pages 530–541, 2011.

15 A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of FOCS*, pages 22–31, 2009.

16 G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 271–279, 2003.

17 M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *Proceedings of FOCS*, 2011.

18 S. Kale, Y. Peres, and C. Seshadhri. Noise tolerance of expanders and sublinear expander reconstruction. In *Proceedings of FOCS*, pages 719–728, 2008.

19 J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the AMS*, 7(1):48–50, 1956.

20 S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

21 R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. pages 709–720, 2013.

22 R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *CoRR*, abs/1302.3417, 2013.

23 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. *CoRR*, abs/1402.3609, 2014.

24 N. Linial. Locality in distributed graph algorithms. *SICOMP*, 21(1):193–201, 1992.

25 L. Trevisan M. Sudan and S. Vadhan. Pseudorandom generators without the XOR lemma. In *Proceedings of STOC*, pages 537–546, 1999.

26 Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Proceedings of ICALP*, pages 653–664, 2012.

27 Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Proceedings of APPROX*, pages 260–273, 2013.

28 S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. *TALG*, 5(2), 2009. Article number 22.

29 A. Mayer, S. Naor, and L. Stockmeyer. Local computations on static and dynamic graphs. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems (ISTCS)*, 1995.

30 M. Naor and L. Stockmeyer. What can be computed locally? *SICOMP*, 24(6):1259–1277, 1995.

31 H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proceedings of FOCS*, pages 327–336, 2008.

32 L. Orecchia and Z. A. Zhu. Flow-based algorithms for local graph clustering. *CoRR*, abs/1307.2855, 2013.

33 M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *TCS*, 381(1-3):183–196, 2007.

34 D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SICOMP*, 30(5):1427–1442, 2000.

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

36 S. Pettie and V. Ramachandran. Randomized minimum spanning tree algorithms using exponentially fewer random bits. *TALG*, 4(1), 2008.

37 R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of ICS*, pages 223–238, 2011.

38 M. E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SICOMP*, 39(7):2897–2926, 2010.

39 T. Sarlos, A. Benczur, K. Csalogany, D. Fogaras, and B. Racz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proceedings of the 15th International Conference on WorldWide Web*, pages 297–306, 2006.

40 D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of STOC*, pages 81–90, 2004.

41 Y. Yoshida, M. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of STOC*, pages 225–234, 2009.

42 Z. A. Zhu, S. Lattanzi, and V. Mirrokni. A local algorithm for finding well-connected clusters. In *Proceedings of the Thirtieth International Conference on Machine Learning*, 2013.