A Local Algorithm for Constructing Spanners in Minor-Free Graphs*

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

1 **MPI for informatics, Saarbrücken, Germany**
`rlevi@mpi-inf.mpg.de`
2 **School of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel**
`danar@eng.tau.ac.il`
3 **CSAIL, MIT, Cambridge, MA, USA, and**
Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
`ronitt@csail.mit.edu`

Abstract

Constructing a spanning tree of a graph is one of the most basic tasks in graph theory. We consider this problem in the setting of local algorithms: one wants to quickly determine whether a given edge e is in a specific spanning tree, without computing the whole spanning tree, but rather by inspecting the local neighborhood of e. The challenge is to maintain consistency. That is, to answer queries about different edges according to the *same* spanning tree. Since it is known that this problem cannot be solved without essentially viewing all the graph, we consider the relaxed version of finding a spanning subgraph with $(1+\epsilon)n$ edges instead of $n-1$ edges (where n is the number of vertices and ϵ is a given approximation/sparsity parameter).

It is known that this relaxed problem requires inspecting $\Omega(\sqrt{n})$ edges in general graphs (for any constant ϵ), which motivates the study of natural restricted families of graphs. One such family is the family of graphs with an excluded minor (which in particular includes planar graphs). For this family there is an algorithm that achieves constant success probability, and inspects $(d/\epsilon)^{\mathrm{poly}(h)\log(1/\epsilon)}$ edges (for each edge it is queried on), where d is the maximum degree in the graph and h is the size of the excluded minor. The distances between pairs of vertices in the spanning subgraph G' are at most a factor of $\mathrm{poly}(d, 1/\epsilon, h)$ larger than in G.

In this work, we show that for an input graph that is H-minor free for any H of size h, this task can be performed by inspecting only $\mathrm{poly}(d, 1/\epsilon, h)$ edges in G. The distances between pairs of vertices in the spanning subgraph G' are at most a factor of $\tilde{O}(h\log(d)/\epsilon)$ larger than in G. Furthermore, the error probability of the new algorithm is significantly improved to $\Theta(1/n)$. This algorithm can also be easily adapted to yield an efficient algorithm for the distributed (message passing) setting.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases spanners, sparse subgraphs, local algorithms, excluded-minor

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.38

1 Introduction

Given graph $G = (V, E)$, a basic task is to find a sparse spanning subgraph G', where G' may be required to be a tree, or may be required to approximately preserve distances between

* This work was partially supported by the Israel Science Foundation grant No. 671/13 and grant No. 1536/14 and the NSF grant CCF-1420692.

vertices (i.e., have small *stretch* [14, 13]). Suppose we are interested in determining whether a given edge e belongs to the spanning subgraph $G' = (V, E')$. We can of course run an algorithm that constructs G' and check whether $e \in E'$, but can we do better? In particular, is it possible to answer such queries with a number of operations that is *sublinear* in $n = |V|$ or possibly even independent of n?

This local version of the problem was first studied in [8]. Observe that the main challenge is to answer queries about different edges consistently with the same sparse spanning graph G', while being allowed to inspect only a small (local) part of the graph for each queried edge. Moreover, while the underlying spanning graph may depend on the internal randomness of the algorithm that answers the queries, it should not depend on the (order of the) queries themselves. The following simple but important observation appears in [8]. If one insists that G have a minimum number of edges, namely, that G' be a tree, then it is easy to see that there are graphs G that contain edges e for which determining whether $e \in G'$ requires inspecting a linear number of edges in G. To see 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.

Given the above observation, the question posed in [8] is whether a relaxed version of this task can be solved by inspecting a sublinear number of edges, where the relaxation is that the spanning graph G' may contain $(1 + \epsilon) \cdot n$ edges, for a given approximation/sparsity parameter ϵ. As shown in [8], in general, even after this relaxation, local algorithms for sparse subgraphs require the inspection of $\Omega(\sqrt{n})$ edges for each queried edge e and for a constant ϵ. This lower bound holds for graphs with strong expansion properties, and there is an almost matching upper bound for such graphs. In fact, even for graphs with relatively weak expansion properties, there is no local algorithm that inspects a number of edges that is independent of n [9]. However, for graphs that are sufficiently non-expanding there are local algorithms whose complexity is independent of n (depending only on the maximum degree d and the approximation parameter ϵ) [8, 9].

A family of non-expanding graphs that is of wide interest, is the family of *minor-free* graphs, which can be parameterized by the size, h, of the excluded minor. In particular, this family includes planar graphs. It was shown in [8] that, based on [7], it is possible to obtain a local sparse spanning tree algorithm for graphs that are free of a minor of size h, where the complexity of the algorithm is $(d/\epsilon)^{\mathrm{poly}(h)\log(1/\epsilon)}$ and which has high constant success probability.

1.1 Our Result and Techniques

In this work we significantly improve on the aforementioned result for the class of minor-free graphs, by designing a local sparse spanning graph algorithm for minor-free graphs whose complexity is polynomial in h, d and $1/\epsilon$ and which has success probability $1 - 1/\Omega(n)$. We note that though graphs with excluded minors have bounded average degree, the *maximum* degree of such graphs remains unbounded, and our local algorithms have complexity that depends on this maximum degree. The spanning graph G' maintains the minimum distances between pairs of vertices in G up to a factor of $\tilde{O}(h \log(d)/\epsilon)$. Namely, it has *stretch* [14, 13] $\tilde{O}(h \log(d)/\epsilon)$ (the stretch obtained in [8] is somewhat higher, and in particular polynomial in d).

Similarly to some of the other local algorithms for sparse spanning graphs presented in [8, 9], our algorithm is based on defining an underlying global partition (which is randomized). The global partition determines the sparse spanning graph, and the local algorithm decides

whether a given edge belongs to the spanning graph by constructing parts of the partition. The differences between the algorithms are in the way the partition is defined, the way the spanning graph is determined by the partition (more specifically, the choice of edges between parts), and the local partial construction of the partition. Interestingly, our partition and its construction bear more similarities to the underlying partition defined by the local sparse spanning graph algorithm for highly expanding graphs, than the partition used by the previous algorithm for minor-free graphs [8]. We discuss the similarities and differences further in Subsection 1.1.8. In what follows we provide a high level description of the partition, the sparse spanning graph it defines, and the corresponding local algorithm. We also explain how we can directly obtain a distributed algorithm, and give a bound on its round complexity.

1.1.1 A partition-based algorithm and the construction of G'

Our local algorithm is based on defining an underlying global partition of the vertices into many small connected parts, where the partition satisfies certain properties defined in the next paragraph. Given such a partition, the edge set E' of G' is simply defined by taking a spanning tree in each part, and a single edge between every pair of parts that have at least one edge between them. The minor-freeness of the graph, together with a bound on the number of parts, ensure that $|E'| \leq (1+\epsilon)n$.

1.1.2 Properties of the partition

We define a partition that has each of the four following properties (which for simplicity are not precisely quantified in this introductory text): (1) the number of parts in the partition is not too large; (2) the size of each part is not too large; (3) each part induces a connected subgraph; (4) for every vertex v it is possibly to efficiently find the subset of vertices in the part that v belongs to.

1.1.3 An initial centers-based partition

Initially, the partition is defined by a selection of *centers*, where the centers are selected randomly (though not completely independently). Each vertex is initially assigned to the closest selected center. For an appropriate setting of the probability that a vertex is selected to be a center, with high probability, this initial partition has the first property. Namely, the number of parts in the partition is not too large. By the definition of the partition, each part is connected, so that the partition has the third property as well. However, the partition is not expected to have the second property, that of each part having small size. Even for a simple graph such as the line, we expect to see parts of size logarithmic in n. In addition, the same example shows that the fourth property may not hold, i.e. there may be many vertices for which it takes superconstant time to find the part that the vertex belongs to. To deal with these two problems, we give a procedure for refining the partition in two phases, as explained next.

1.1.4 Refining the partition, phase 1 (and a structural lemma)

A first refinement is obtained by the separation of vertices that in order to reach their assigned center in a Breadth First Search (BFS), must observe a number of vertices that is above a certain predetermined threshold, k. Each of these *remote* vertices becomes a singleton subset in the partition. We prove that with probability $1 - 1/\Omega(n)$, the number of

these remote vertices is not too large, so that the first property (concerning the number of parts) is maintained with high probability.

The probabilistic analysis builds on a structural lemma, which may be of independent interest. The lemma upper bounds, for any given vertex v, the number of vertices u such that v can be reached from u by performing a BFS until at most k vertices are observed. This number in turn upper bounds the size of each part in the partition after the aforementioned refinement. While this upper bound is not polynomial in k and d, it suffices for the purposes of our probabilistic (variance) analysis (and we also show that there is no polynomial upper bound).

1.1.5 Refining the partition, phase 2

In addition to the first property, the third property (connectivity of parts) is also maintained in the resulting refined partition, and the refinement partially addresses the fourth property, as remote vertices can quickly determine that they are far from all centers. In addition, the new parts of the partition will be of size 1 and thus will not violate the second property. However, after this refinement, there might be some large parts remaining so that the second and fourth properties are not necessarily satisfied. We further partition large parts into smaller (connected) parts by a certain decomposition based on the BFS-tree of the large part.

1.1.6 The local algorithm

Given an edge $\{u, v\} \in E$, the main task of the local algorithm is to find the two parts to which the vertices belong in the final partition. Once these parts are found, the algorithm can easily decide whether the edge between u and v should belong to E' or not. In order to find the part that u (similarly, v) belongs to, the local algorithm does the following. First it performs a BFS until it finds a center, or it sees at least k vertices (none of which is a center). In the latter case, u is a singleton (remote) vertex. In the former case, the algorithm has found the center that u is assigned to in the initial partition, which we denote by $\sigma(u)$. The algorithm next performs a BFS starting from $\sigma(u)$. For each vertex that it encounters, it checks whether this vertex is assigned to u (in the initial partition). If the number of vertices that are found to be assigned to $\sigma(u)$ is above a certain threshold, then the decomposition of the part assigned to $\sigma(u)$ needs to be emulated locally. We show that this can be done recursively in an efficient manner.

1.1.7 A distributed algorithm

Our algorithm easily lends itself to an implementation in the distributed (message passing) setting. In this setting a processor resided on each vertex in the graph. Computation proceeds in rounds, where in each round every vertex sends messages to its neighbors. In the distributed implementation, initially each vertex decided, independently at random (and with the appropriate probability), whether it is a center (of the initial partition). In the first round, each vertex sends its id (name) to each of its neighbors, as well as an indication whether it is a center. In the following rounds, each center send all its neighbors the information it has gathered about its local neighborhood (including the identity of the centers). It follows from our analysis that after $\tilde{O}(h \log(d)/\epsilon))$ rounds, each processor can determine if its incident edges belong to the sparse spanning graph G'.

1.1.8 A discussion of the algorithm for highly expanding graphs in [8]

As noted previously, there are similarities between our algorithm and the algorithm described in [8] for highly expanding graphs (for which the lower bound of $\Omega(\sqrt{n})$ holds). We refer to the latter algorithm as Centers-LRR, and provide a rough description of it next. The Centers-LRR algorithm is based on the selection of a set of random centers, which induces a partition of all the vertices, where each vertex is assigned to the center it is closest to. The number of centers is $\sqrt{\epsilon n}$, so that if we take the edges of a BFS tree for each part, and at most one edge between each pair of parts, we get at most $(1+\epsilon)n$ edges.

Based on the assumption regarding the expansion properties of the graph, with high probability over the random choice of the centers, for an appropriate threshold k, the distance between every vertex and its center is at most k, and the size of the distance-k neighborhood of every vertex is $\tilde{O}(\sqrt{n/\epsilon})$. This implies that each vertex can find its center by performing these many queries (for a constant degree). Hence, for an edge between two vertices that are assigned to the same center, we can easily determine whether it is a BFS edge (which belongs to the sparse spanning subgraph).

On the other hand, the decision regarding edges between vertices that are assigned to different centers is more subtle. Since at most one edge between every pair of parts is allowed, it seems that it is necessary to determine, for a given center, the identity of all vertices that are assigned to it. Though the number of such vertices is not too large, it is not clear how to perform this task in a local and efficient manner. The Centers-LRR algorithm overcomes this obstacle by defining a rule for the selection of at most one edge between every pair of parts, which can be implemented locally and efficiently. This rule is such that for some pairs of parts that have edges between them, it might be that no edge is selected. Nonetheless, it is proved in [8] that the final resulting subgraph is connected.

If we compare the Centers-LRR algorithm to the algorithm presented in this work we see that while there is a clear similarity in terms of the underlying partition (which is only initial in the current work), there are many differences in the difficulties that need to be overcome and the local implementation.

1.2 Related Work

As mentioned earlier, the works most closely related to the current work are [8, 9]. In addition to the results in these works that were already described, in [8] there is a local sparse spanning graph algorithm for the family of ρ-hyperfinite graphs[1] whose time complexity and probe complexity are $O(d^{\rho(\epsilon)})$, assuming that ρ is known. Minor-free graphs are a subclass of hyperfinite graphs (for an appropriate choice of ρ that depends on the size of the excluded minor). In [9] the authors show that in every f-non-expanding graph[2] G where $f(t) = \Omega((\log t)^{-1}(\log\log t)^{-2})$ one can remove ϵn edges from G so that each connected component of the remaining graph is of size at most $2^{2^{O(1/\epsilon)}}$. This enables them to provide a local sparse spanning graph algorithm for this family of graphs with probe complexity $d^{2^{2^{O(1/\epsilon)}}}$ and stretch $2^{2^{O(1/\epsilon)}}$. Both algorithms, in [8] and in [9], sparsify the graph by simulating a localized version of Kruskal's algorithm.

[1] 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.

[2] A graph is *f-non-expanding* if every t-vertex subgraph H satisfies $\phi_H \leq f(t)$ where ϕ_G is the (edge) *expansion* of G, that is, $\phi_G = \min_S |\partial_G(S)| / |S|$ where the minimum is taken over all $S \subseteq V(G)$ of size $1 \leq |S| \leq |V(G)|/2$.

By using the partition oracle in [7] one can obtain a local sparse spanning graph algorithm with quasi-polynomial complexity in d and ϵ^{-1}. The partition is obtained by contracting edges iteratively in a controlled manner. The advantage of their partition is that its edge cut is guaranteed to be small (with high constant probability). However, as mentioned before, their complexity is much higher.

Graph partitioning for graphs with an excluded minor has been studied extensively (see e.g., [3, 6]), and was found useful in constructing spanners and distance oracles (see e.g., [5]). However, compared to previous results, the main benefit of our partitioning technique is that the partition is designed to be constructed locally in an efficient manner.

Ram and Vicari [15] studied the problem of constructing sparse spanning graphs in the distributed model and provided an algorithm that runs in $\min\{D(G), O(\log n)\}$ number of rounds where $D(G)$ denotes the diameter of G.

The model of *local computation algorithms* as considered in this work, was defined by Rubinfeld et al. [16] (see also Alon et al. [2]). Other local algorithms, for maximal independent set, hypergraph coloring, k-CNF and maximum matching include those given in [16, 2, 11, 12, 4].

2 Preliminaries

In this section we provide the precise definition of the algorithmic problem addressed in this paper, as well as several other useful definitions and notations.

We consider undirected, simple graphs over n vertices, where the degree of each vertex is bounded by d. We assume for simplicity that the set of vertices, V, is simply $[n] = \{1, \ldots, n\}$, so that there is a total order over the (identifiers of the) vertices. For each vertex v, there is some arbitrary, but fixed order over its neighbours. The input graph $G = (V, E)$ is given via an *oracle access to its incidence-list representation*. Namely, the algorithm is supplied with n and d, and has access to an oracle that for any pair (v, i) such that $v \in [n]$ and $i \in [d]$, the oracle either returns the i^{th} neighbour of v (according to the aforementioned order over neighbours) or an indication that v has less than i neighbours. We refer to each such access to the oracle as a *probe* to the graph. We now turn to formally define the algorithmic problem we consider in this paper.

▶ **Definition 1.** An algorithm $\mathcal{A}$ is an (ϵ, q, δ)-*local sparse spanning graph algorithm* if, given $n, d \geq 1$ and oracle access to the incidence-lists representation of a connected graph $G = (V, E)$ over n vertices and degree at most d, it provides query access to a subgraph $G' = (V, E')$ of G such that:

(i) G' is connected.
(ii) $|E'| < (1 + \epsilon) \cdot n$ with probability at least $1 - \delta$ (over the internal randomness of $\mathcal{A}$).
(iii) E' is determined by G and the internal randomness of $\mathcal{A}$.
(iv) $\mathcal{A}$ makes at most q probes to G.

By "providing query 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 queries, $\mathcal{A}$ answers consistently with the same G'.

An algorithm $\mathcal{A}$ is an (ϵ, q, δ)-*local sparse spanning graph algorithm for a family 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 for each edge they are queried on, perform as few probes as possible to G. Ideally, we would like the number of probes to be independent of n and polynomial in $1/\epsilon$, d, and possibly some parameters of the family $\mathcal{C}$. We are also interested in bounding the total amount of space used by the local algorithm, and its running

time (in the word-RAM model). Note that Item 3 implies that the answers of the algorithm to queries cannot depend on previously asked queries.

We denote by $\mathrm{dist}_G(u,v)$ (and sometimes by $\mathrm{dist}(u,v)$ when G is clear from the context) the distance between two vertices u and v in G. We let $N(v)$ denote the set of neighbors of v and for $\ell \geq 0$ let $\Gamma_\ell(v) \stackrel{\text{def}}{=} \{u \in V : \mathrm{dist}(u,v) \leq \ell)\}$.

Another parameter of interest is the stretch of G'. Given a connected graph $G = (V,E)$, a subgraph $G' = (V,E')$ is a t -spanner of G if for every $u, v \in V$, $\frac{\mathrm{dist}_{G'}(u,v)}{\mathrm{dist}_G(u,v)} \leq t$. In this case t is referred to as the *stretch factor* of G'.

The total order over the vertices induces a total order (ranking) r over 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 $\max\{u,v\} < \max\{u',v'\}$ (recall that $V = [n]$). 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.

Recall that a graph H is called a *minor* of a graph G if H is isomorphic to a graph that can be obtained by zero or more edge contractions on a subgraph of G. A graph G is *H-minor free* if H is not a minor of G. We will use the following theorem.

▶ **Theorem 2** ([10]). *Let $c(s)$ be the minimum number c such that every graph $G = (V,E)$ with $|E| \geq c \cdot |V|$ contracts to a complete graph K_s. Then $c(s) \leq 8s \log s$.*

We shall use the following result from previous work (see Section 6 in [1]).

▶ **Theorem 3.** *For every $1 \leq t \leq n$, there exists an explicit construction of a t-wise independent random variable $x = (x_1, \ldots, x_n) \in [q]^n$ for $q = \Theta(n)$ whose seed length is at most $O(t \log n)$ bits. Moreover, for any $1 \leq i \leq n$, x_i can be computed in $O(t)$ operations in the word-RAM model.*

3 The Algorithm

In this section we prove the following theorem.

▶ **Theorem 4.** *Algorithm 2 is an $(\epsilon, \mathrm{poly}(1/\epsilon, d, h), \delta)$-local sparse spanning graph algorithm for graphs that are H-minor free, where h is the size of H and $\delta = 1/\Omega(n)$. Furthermore, the stretch factor of G' is $\tilde{O}(h \cdot \log d/\epsilon)$. More precisely, the probe complexity of the algorithm is $\tilde{O}((h/\epsilon)^4 d^5)$. Its space complexity (length of the random seed) is $\tilde{O}((h/\epsilon) d \log n)$ bits, and its running time is $\tilde{O}((h/\epsilon)^5 d^5)$ (in the Word-RAM model).*

We begin by describing a global partition of the vertices. We later describe how to locally generate this partition and design our algorithm (and the sparse subgraph it defines), based on this partition.

3.1 The Partition $\mathcal{P}$

The partition described in this subsection, denoted by $\mathcal{P}$, is a result of a random process. We describe how the partition is obtained in three steps where in each step we refine the partition from the previous step. The partition is defined based on three parameters: $\gamma \in (0,1)$, an integer $k > 1$ and an integer $s > 1$, which is set subsequently (as polynomials of d, ϵ and h).

3.1.1 First Step

We begin with some notation. Given a subset of vertices $W \subseteq V$ and a vertex $v \in V$, we define the *center* of v with respect to W, denoted $\sigma(v)$ as the vertex in W that is closest to v, breaking ties using vertex ids. That is, $\sigma(v)$ is the vertex with the minimum identifier in the subset $\{y \in W : \mathrm{dist}(y, v) = \min_{w \in W} \mathrm{dist}(w, v)\}$.

For each $w \in W$ we define the *cell* of w with respect to W as $C(w) \stackrel{\mathrm{def}}{=} \{v \in V : \sigma(v) = w\}$. Namely, the set of vertices in $C(w)$ are the vertices which are closer to w more than any other vertex in W (where ties are broken according to the order of the vertices). Notice that these cells form a partition of V. Our initial partition is composed of these cells when picking W in the following way: each vertex $i \in [n]$ draws a γ-biased bit, denoted x_i, and $i \in W$ if an only if $x_i = 1$. The joint-distribution of $(x_1, \ldots, x_n)$ is t-wise independent where $t \stackrel{\mathrm{def}}{=} 2kd$. (The reason that the choice of W is determined by a t-wise independent distribution rather than an n-wise independent distribution is so as to bound the space used by the local emulation of the global algorithm.)

3.1.2 Second Step

In this step we identify a subset of *special* vertices, which we call the *remote vertices* and make these vertices singletons in the partition $\mathcal{P}$. The set of remote vertices, R, is defined with respect to W and an integer parameter k as described next. Let $\ell_k(v)$ be the minimum integer ℓ such that the BFS tree rooted at v of depth ℓ has size at least k. Let $B_k(v)$ be the set of vertices in the BFS tree rooted at v of depth $\ell_k(v)$. We define $R = \{v \in V : B_k(v) \cap W = \emptyset\}$, i.e., those vertices v for which $B_k(v)$ does not contain a center. Clearly, a vertex can identify efficiently if it is in R by probing at most kd vertices and checking whether they intersect W. In Subsection 3.3 we obtain a bound on the size of R.

3.1.3 Third Step

In this step we decompose cells that are still too big. We first argue that the cells are still connected (after the removal the vertices in R from all original cells). Thereafter we will use a procedure of tree decomposition in order to break the cells into smaller parts.

▶ **Lemma 5.** *For every $w \in W$, the subgraph induced by $C(w) \setminus R$ is connected. Furthermore, for every $v \in C(w) \setminus R$, the subgraph induced by $C(w) \setminus R$ contains all vertices that belong to the shortest paths between v and w.*

Proof. Fix $w \in W$, and consider any $v \in C(w) \setminus R$. We will prove that the subgraph induced by $C(w) \setminus R$ contains all vertices on the shortest paths between v and w and this will imply the connectivity as well. The proof is by induction on the distance to w. In the base case $v = w$. In this case $C(w) \setminus R$ clearly contains a path between v to itself because it contains v. Otherwise, we would like to show that for any $u \in N(v)$ for which $\mathrm{dist}(u, w) < \mathrm{dist}(v, w)$ it holds that $u \in C(w) \setminus R$. The proof will then follow by induction. let P be a shortest path between v and w and let $\{v, u\} \in E$ denote the first edge in P. We first observe that $\sigma(u) = w$ and thus $u \in C(w)$. Assume otherwise and conclude that there is a vertex in $w' \in W$ for which either $\mathrm{dist}(v, w') < \mathrm{dist}(v, w)$ or $\mathrm{dist}(v, w) = \mathrm{dist}(v, w')$ and $id(w') < id(w)$, in contradiction to the fact that $\sigma(v) = w$ (see the definition of $\sigma(\cdot)$ in the First Step). Since u is on a shortest path between v and w it follows that

$$\Gamma_{\mathrm{dist}(u,w)-1}(u) \subseteq \Gamma_{\mathrm{dist}(v,w)-1}(v) \ . \tag{1}$$

From the fact that $v \notin R$ it follows that $|\Gamma_{\mathrm{dist}(v,w)-1}(v)| \leq k$ and hence from Equation (1) it follows that $|\Gamma_{\mathrm{dist}(u,w)-1}(u)| \leq k$ and so $u \notin R$ as well. We conclude that $u \in C(w) \setminus R$ and $\mathrm{dist}(u,w) = \mathrm{dist}(v,w) - 1$ as desired. ◀

We shall use the following notation. For each $w \in W$ let $\mathcal{T}(w)$ denote the BFS tree rooted at w of the subgraph induced by $C(w) \setminus R$ (recall that the BFS is performed by exploring the vertices according to the order of their identifiers (in $[n]$)). To obtain the final refinement of our partition, for each $w \in W$ such that $|\mathcal{T}(w)| > s$, we run Algorithm 1 on $\mathcal{T}(w)$, w and s.

Algorithm 1 (Recursive Tree decomposition)

Input: A tree $\mathcal{T}$, the root of the tree v and an integer s.
Output: A decomposition of $\mathcal{T}$ into subtrees, where each subtree is assigned a (sub-)center.

1. Initialize the set of vertices of the current part $Q := \emptyset$.
2. Perform a BFS starting from v and stop at level $\ell \stackrel{\mathrm{def}}{=} \ell_s(v)$ (see the definition of $\ell_s(\cdot)$ in the Second Step). Add to Q all the vertices explored in the BFS.
3. Let S denote the set of all the children of the vertices in the ℓ^{th} level of the BFS (namely, all the vertices in level $\ell + 1$).
4. For each vertex $u \in S$:
   a. If the subtree rooted at u, $\mathcal{T}_u$, has size at least s, then disconnect this subtree from $\mathcal{T}$ and continue to decompose by recursing on input $\mathcal{T}_u$, u and s.
   b. Otherwise, add the vertices of $\mathcal{T}_u$ to Q.
5. Set v to be the *sub-center* of all the vertices in Q.

3.2 The Edge Set

Given the partition $\mathcal{P}$ defined in the previous subsection (Subsection 3.1), we define the edge set of our sparse spanning graph E' in the following simple manner. In each part of $\mathcal{P}$ which is not a singleton, we take a spanning tree. Specifically, we take the BFS-tree rooted at the sub-center of that part (see Algorithm 1, Step 5). For every pair of parts of $X, Y \in \mathcal{P}$, if the set $E(X,Y) \stackrel{\mathrm{def}}{=} \{\{x,y\} \in E : x \in X \text{ and } y \in Y\}$ is not empty, then we add to E' the edge $e \in E(X,Y)$ with minimal ranking (where the ranking over edges is as defined in Section 2). Clearly G' is connected and spans G. We would like to bound the size of E'. To this end we will use the following claim.

▶ **Claim 6.** *The number of parts in $\mathcal{P}$ is bounded as follows: $|\mathcal{P}| \leq |W| + |R| + \frac{n}{s}$.*

Proof. Consider the three steps of refinement of the partition. Clearly, after the first step the size of the partition is exactly $|W|$. After the second step, the size of the partition is exactly $|W| + |R|$. Finally, since in the last step we break only parts whose size is greater than s into smaller parts that are of size at least s, we obtain that the number of new parts that are introduced in this step is at most n/s. The claim follows. ◀

The next lemma establishes the connection between the size of $\mathcal{P}$ and the sparsity of G'.

▶ **Lemma 7.** *For an input graph G which is a H-minor free for a graph H over h vertices,*

$$|E'| < n + |\mathcal{P}| \cdot c(h) \ ,$$

where $c(h)$ is as defined in Theorem 2.

Proof. Since for each $X \in \mathcal{P}$ the subgraph induced by X is connected, we can contract each part in $\mathcal{P}$ and obtain an H-minor free graph. The number of vertices in this graph is $|\mathcal{P}|$. If we replace multi-edges with single edges, then by Theorem 2 we obtain that the number of edges in this graph is at most $|\mathcal{P}| \cdot c(h)$. Finally, since the total number of edges in the union of spanning trees of each part it $n - |\mathcal{P}| < n$, we obtain the desired result. ◀

3.3 Bounding the Number of Remote Vertices

In this subsection we prove the following lemma.

▶ **Lemma 8.** *If $k = \Omega((\log^2(1/\gamma) + \log d)/\gamma)$, then with probability at least $1 - \frac{1}{\Omega(n)}$ it holds that $|R| \leq \gamma n$.*

In order to establish Lemma 8 we start defining for every $v \in V$,

$$Y_k(v) \stackrel{\text{def}}{=} \{u \in V : v \in B_k(u)\} . \tag{2}$$

Informally, $Y_k(v)$ is the set of vertices that encounter v while performing a BFS which stops after the first level in which the total number of explored vertices is at least k. We first establish the following simple claim.

▶ **Claim 9.** *For every vertex $u \in Y_k(v)$ and for every vertex w that is on a shortest path between u and v, we have that $w \in Y_k(v)$.*

Proof. Let $\ell = \text{dist}(u, v)$ and $\ell' = \text{dist}(w, v)$, so that $d(u, w) = \ell - \ell' \geq 1$. Assume, contrary to the claim, that $w \notin Y_k(v)$. This implies that $|\Gamma_{\ell'-1}(w)| \geq k$. But since $\Gamma_{\ell'-1}(w) \subseteq \Gamma_{\ell-1}(u)$, we get that $|\Gamma_{\ell-1}(u)| \geq k$, contrary to the premise of the claim that $u \in Y_k(v)$. ◀

We now turn to upper bound the size of $Y_k(v)$.

▶ **Lemma 10.** *For every graph $G = (V, E)$ with degree bounded by d, and for every $v \in V$,*

$$|Y_k(v)| \leq d^3 \cdot k^{\log k + 1} .$$

Proof. Fix a vertex $v \in V$. For every $0 \leq j \leq k$, define $Y_k^j(v) \stackrel{\text{def}}{=} \{u \in Y_k(v) : \text{dist}(v, u) = j\}$. Observe that $Y_k(v) = \bigcup_{j=0}^{k} Y_k^j(v)$. Therefore, if we bound the size of $Y_k^j(v)$, for every $0 \leq j \leq k$, we will get a bound on the size of $Y_k(v)$. Consider first any $3 \leq j < k$ and any vertex $u \in Y_k^j(v)$. Recall that $\ell_k(u)$ is the the minimum integer ℓ such that the BFS tree rooted at v of depth ℓ has size at least k. Since $j \leq \ell_k(u)$, it follows that $|\Gamma_{j-1}(u)| < k$. Now consider a shortest path between u and v and let w be the vertex on this path for which $\text{dist}(u, w) = \lfloor (j-1)/2 \rfloor$. Denote $q \stackrel{\text{def}}{=} \text{dist}(w, v)$. By Claim 9, $w \in Y_k(v)$, and by the definition of q, $w \in Y_k^q(v)$. Therefore,

$$|\Gamma_{q-1}(w)| \leq k . \tag{3}$$

From the fact that w is on the shortest path between u and v it also follows that

$$\begin{aligned} q &= \text{dist}(v, u) - \text{dist}(u, w) = j - \lfloor (j-1)/2 \rfloor \\ &= \lceil (j-1)/2 \rceil + 1 \\ &\geq \lfloor (j-1)/2 \rfloor + 1 = \text{dist}(u, w) + 1 . \end{aligned} \tag{4}$$

Therefore $q - 1 \geq \text{dist}(u, w)$ and so $u \in \Gamma_{q-1}(w)$. It follows that

$$Y_k^j(v) \subseteq \bigcup_{w \in Y_k^q(v)} \Gamma_{q-1}(w) . \tag{5}$$

From Equations (3) and (5) we get that $|Y_k^j(v)| \leq k \cdot |Y_k^q(v)|$. For every $j \leq 3$ we have the trivial bound that $|Y_k^j(v)| \leq d^3$. By combining with Equation (4) we get that $|Y_k^j(v)| \leq d^3 \cdot k^{\log j}$. Since $Y_k(v) = \bigcup_{j=0}^{k} Y_k^j(v)$ we obtain the desired bound. ◀

While the bound on $|Y_k(v)|$ in Lemma 10 may not be tight, it suffices for our purposes. One might conjecture that it is possible to prove a polynomial bound (in k and d). We show that this is not the case (see Lemma 11 in the appendix).

We now use the bound in Lemma 10 in order to bound the number of remote vertices.

Proof of Lemma 8. Let χ_W denote the characteristic vector of the set W. For a subset $S \subseteq V$, let $\chi_W(S)$ denote the projection of χ_W onto S. That is, $\chi_W(S)$ is a vector of length $|S|$ indicating for each $x \in S$ whether $\chi_W(x) = 1$.

For each vertex $v \in V$ define a random variable Z_v indicating whether it is a remote vertex with respect to W. Recall that v is remote if and only if $B_k(v) \cap W = \emptyset$. Recalling that W is selected according to a t-wise independent distribution where $t = 2kd$ and that $k \leq |B_k(v)| < k \cdot d$, we get that $\Pr[Z_v = 1] \leq (1-\gamma)^k$. We also define $S_v \stackrel{\text{def}}{=} \{u \in V : B_k(u) \cap B_k(v) \neq \emptyset\}$. Fix $v \in V$ and observe that the value of Z_v is determined by $\chi_W(B_k(v))$. Furthermore, since for every $v \in V$ and $u \in V \setminus S_v$, $\chi_W(B_k(v))$ and $\chi_W(B_k(u))$ are independent it follows that Z_u and Z_v are independent as well. Hence, in this case $\mathrm{Cov}[Z_v, Z_u] = 0$, and we obtain the following upper bound on the variance of the number of remote vertices.

$$\begin{aligned}\mathrm{Var}\left[\sum_{v\in V} Z_v\right] &= \sum_{(v,u)\in V} \mathrm{Cov}[Z_v, Z_u] = \sum_{v\in V}\sum_{u\in S_v} (\mathrm{Exp}[Z_v \cdot Z_u] - \mathrm{Exp}[Z_u]\cdot\mathrm{Exp}[Z_v]) \\ &\leq \sum_{v\in V}\sum_{u\in S_v} \mathrm{Exp}[Z_v \cdot Z_u | Z_v = 1]\cdot\Pr[Z_v = 1] \leq \sum_{v\in V} |S_v|\cdot(1-\gamma)^k \,. \quad (6)\end{aligned}$$

By the definition of $Y_k(\cdot)$ in Equation (2) it follows that $S_v \subseteq \bigcup_{u\in B_k(v)} Y_k(u)$. By Lemma 10, $\max_{v\in V}\{|Y_k(v)|\} \leq d^3 \cdot k^{\log k+1}$. Therefore

$$|S_v| \leq |B_k(v)| \cdot d^3 \cdot k^{\log k+1} \leq d^4 \cdot k^{\log k+2} \,. \quad (7)$$

Hence, by Equations (6) and (7) we get $\mathrm{Var}\left[\sum_{v\in V} Z_v\right] \leq nd^4 \cdot k^{\log k+2}\cdot(1-\gamma)^k$. Since $(1-\gamma)^k \leq e^{-\gamma k}$ we obtain that $\mathrm{Var}\left[\sum_{v\in V} Z_v\right] \leq \gamma^2 n$ for $k = \Omega((\log^2(1/\gamma) + \log d)/\gamma)$. Since for every $v \in V$, $\Pr[Z_v = 1] \leq (1-\gamma)^k \leq \gamma$, we get that $\mathrm{Exp}\left[\sum_{v\in V} Z_v\right] \leq \gamma n/2$. By applying Chebyshev's inequality we get that

$$\Pr\left[\sum_{v\in V} Z_v \geq \mathrm{Exp}\left[\sum_{v\in V} Z_v\right] + \gamma n/2\right] \leq \frac{4\mathrm{Var}\left[\sum_{v\in V} Z_v\right]}{\gamma^2 n^2} \leq \frac{4}{n} \,.$$

Since $|R| = \sum_{v\in V} Z_v$ it follows that $|R| < \gamma n$ with probability at least $1 - (4/n)$, as desired. ◀

3.4 The Local Algorithm

In this subsection we provide Algorithm 2, which on input $e \in E$, locally decides whether $e \in E'$, as defined in Subsection 3.2, based on the (random, but not completely independent) choice of W. Given an edge $\{u, v\}$, the algorithm first finds, for each $y \in \{u, v\}$, the center, $\sigma(y)$, that y is assigned to by the initial partition, under the condition that $\sigma(y) \in B_k(y)$. This is done by calling Algorithm 3, which simply performs a BFS starting from y until it encounters a vertex in W, or it reaches level $\ell_k(y)$ without encountering such a vertex (in

which case y is a remote vertex). Algorithm 3 assumes access to χ_W, which is implemented using the random seed that defines the t-wise independent distribution, and hence determines W. If y is not found to be a remote vertex, then Algorithm 2 next determines to which sub-part of $C(\sigma(y)) \setminus R$ does y belong. This is done by emulating the tree decomposition of the BFS tree rooted at $\sigma(y)$ and induced by $C(\sigma(y)) \setminus R$. A central procedure that is employed in this process is Algorithm 4. This algorithm is given a vertex $v \in W$, and a vertex u in the BFS subtree rooted at v and induced by $C(v) \setminus R$. It returns the edges going from u to its children in this tree, by performing calls to Algorithm 3.

Algorithm 2 (Sparse Spanning Graph)

Input: An edge $\{u, v\} \in E$.
Output: YES if $\{u, v\} \in E'$ and NO otherwise.

1. For each $y \in \{u, v\}$ find the part that y belongs to as follows:
   - **a.** Use Algorithm 3 to obtain $\sigma(y)$.
   - **b.** If $\sigma(y)$ is 'null' then the part that y belongs to is the singleton set $\{y\}$.
   - **c.** Let $\mathcal{T}$ denote the BFS tree rooted at $\sigma(y)$ in the subgraph induced by $C(\sigma(y)) \setminus R$. By Lemma 5 every shortest path between $\sigma(y)$ and $v \in C(\sigma(y)) \setminus R$ is contained in the subgraph induced on $C(\sigma(y)) \setminus R$. Therefore the edges of $\mathcal{T}$ can be explored via Algorithm 4.
   - **d.** Reveal the part (the subset of vertices) that y belongs to in $\mathcal{P}$ (as defined in Subsection 3.1). Recall that the part of y is the subtree of $\mathcal{T}$ that contains y after running Algorithm 1 on input $\mathcal{T}$, $\sigma(y)$ and s. This part can be revealed locally as follows.
     - **i.** Reveal the path between $\sigma(y)$ and y in $\mathcal{T}$, denoted by $P(y)$. Since $P(y)$ is the shortest path between y and $\sigma(y)$ with the lexicographically smallest order it can revealed by performing a BFS from y until $\sigma(y)$ is encountered.
     - **ii.** Run Algorithm 1 on $\mathcal{T}$ while recursing in Step 4a only on the subtrees in which the root is contained in $P(y)$.
2. If u and v are in the same part, then return YES iff the edge $\{u, v\}$ belongs to the BFS tree of that part.
3. Otherwise, return YES iff the edge $\{u, v\}$ is the edge with minimum rank connecting the two parts.

Proof of Theorem 4. We set $\gamma = \epsilon/2c(h)$, $k = \Theta((\log^2(1/\gamma) + \log d)/\gamma)$, and $s = c(h)/4\epsilon$. We start by bounding the size of W (with high probability). By the definition of W we have that $\mathrm{Exp}[|W|] = \mathrm{Exp}\left[\sum_{i\in[n]} \chi_W(i)\right] = \gamma n$. Since for every $1 \leq i < j \leq n$, $\chi_W(i)$ and $\chi_W(j)$ are pairwise-independent, we obtain that

$$\mathrm{Var}\left[\sum_{i\in[n]} \chi_W(i)\right] = \sum_{i\in[n]} \mathrm{Var}\left[\chi_W(i)\right] = \sum_{i\in[n]} \left(\mathrm{Exp}\left[\chi^2_W(i)\right] - \mathrm{Exp}\left[\chi_W(i)\right]^2\right) = n\gamma(1-\gamma)\ .$$

Therefore, by Chebyshev's inequality $\Pr\left[|W| \geq 2\gamma n\right] \leq \frac{1-\gamma}{\gamma n}$. By Lemma 8 (and the setting of k), with probability $1 - 1/\Omega(n)$, $|R| \leq \gamma n$. By Claim 6, Lemma 7 and the settings of γ and s, we get that $|E'| \leq (1+\epsilon)n$ with probability $1 - 1/\Omega(n)$.

The claim about the stretch of G' follows from the fact that the diameter of every part of $\mathcal{P}$ is bounded by $2k$.

The number of vertices that Algorithm 3 inspects (for any vertex v it is called with) is at most kd. Since the degree of each vertex is bounded by d, its probe complexity is bounded by kd^2. Algorithm 3 makes at most kd accesses to χ_W, hence, by Theorem 3 its running time is bounded by $O(k^2d^2)$. The probe complexity of Algorithm 4 is upper bounded by the total probe complexity of at most d calls to Algorithm 3, plus d executions of a BFS until

at most kd vertices are encountered (Steps 2b and 2c, which for simplicity we account for separately from Algorithm 3). A similar bound holds for the running time. Hence, the total probe complexity and running time of Algorithm 4 are $O(kd^3)$ and $O(k^2d^3)$, respectively.

The size of any subtree returned by Algorithm 1 is upper bounded by s^2d^2. To verify this, recall that at the end of Step 2 of Algorithm 1, at most sd vertices were explored. Hence, the number of vertices that are incident to the explored vertices is at most sd^2. Thus, due to Step 4a, the total number of vertices in each part is at most s^2d^2. Since Step 4a can be implemented locally by calling Algorithm 4 at most s times, we obtain that the total probe complexity and running time of revealing each part is at most $O(s^2kd^5)$ and $O(s^2k^2d^5)$, respectively. The probe complexity and running time of Algorithm 2 are dominated by those of Step 1d. Observe that in Step 1d at most $|P(y)|$ parts are revealed. Since $|P(y)| \leq k$, we obtain that the overall probe complexity and running time of Algorithm 2 are bounded by $O(s^2k^2d^5)$ and $O(s^2k^3d^5)$, respectively. By the settings of k and s we obtain the final result. ◀

Algorithm 3 (Find Center)

Input: A vertex v and an integer k. Query access to χ_W.

Output: $\sigma(v)$ if $\sigma(v) \in B_k(v)$ or 'null' otherwise.

1. Perform a BFS from v until the first level that contains a vertex in W or until at least k vertices are reached. That is, defining $W_j \stackrel{\text{def}}{=} \Gamma_j(v) \cap W$, stop at level ℓ where $\ell \stackrel{\text{def}}{=} \min\{\ell_k(v), \min_j\{W_j \neq \emptyset\}\}$.
2. If $W_\ell = \emptyset$ then return **'null'** (v is remote).
3. Otherwise, return the vertex with the minimum id from W_ℓ.

Algorithm 4 (get BFS outgoing-edges endpoints)

Input: $v \in W$ and a vertex $u \in \mathcal{T}(v)$ where $\mathcal{T}(v)$ denotes the BFS tree induced by $C(v) \setminus R$ and rooted at v.

Output: The outgoing edges from u in $\mathcal{T}(v)$ (the orientation of the edges is from the root to the leaves).

1. Initialize $S := \{v\}$
2. For each $u' \in N(u)$, if the following three conditions hold, then add u' to S:
   a. Algorithm 3 on input u' returns v. Namely, $\sigma(u') = v$.
   b. u is on a shortest path between u' and v. Namely, $\text{dist}(u', v) = \text{dist}(u, v) + 1$.
   c. u is the vetex with the minimum id among all vertices in $\{w \in N(u') : \text{dist}(w, v) = \text{dist}(u', v) - 1\}$.
3. Return S.

Acknowledgement. We would like to thank Oded Goldreich for his helpful suggestions.

References

1 N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

2 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.

3 Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 293–299, 1990. doi:10.1145/100216.100254.

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 Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 135–146, 2011. doi:10.1007/978-3-642-22006-7_12.

6 Ken-ichi Kawarabayashi and Bruce A. Reed. A separator theorem in minor-closed classes. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 153–162, 2010. doi:10.1109/FOCS.2010.22.

7 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.

8 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.

9 Reut Levi, Guy Moshkovitz, Dana Ron, Ronitt Rubinfeld, and Asaf Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, pages n/a–n/a, 2016. doi:10.1002/rsa.20652.

10 W. Mader. Homomorphiesätze für graphen. *Mathematische Annalen*, 178:154–168, 1968.

11 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.

12 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.

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

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

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

16 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.

A A non-polynomial relation between k and $Y_k(v)$

Recall that $Y_k(v) \stackrel{\text{def}}{=} \{u \in V : v \in B_k(u)\}$. In the following lemma we show that $|Y_k(v)|$ can be super polynomial.

▶ **Lemma 11.** *There exists a graph $G = (V, E)$ with degree bounded by d and $v \in V$ such that $|Y_k(v)| = k^{\Omega(\log \log d)}$.*

Proof. The graph G is a tree, rooted at v, and defined as follows. For simplicity, we let the degree bound be $d + 1$ (so that a vertex may have d children, and hence degree $d + 1$). We partition the levels of the tree into consecutive subsets: $L_0 = \{1, \ldots, \ell_0\}$ (where the root is

at level 1), $L_1 = \{\ell_0 + 1, \ldots, \ell_1\}$, $\ldots$, $L_r = \{\ell_{r-1} + 1, \ldots, \ell_r\}$. For each subset L_i, and for each level j in the subset, all vertices in level j have the same number of children, which is $d_i \stackrel{\text{def}}{=} d^{2^{-i}}$. We set $r = \log\log d$, so that all vertices in levels belonging to L_r have two children. Finally we set $s_i = |L_i| = \log_{d_i} g^{1/(r+1)}$, where g determines the size of the tree, as well as the minimum k that ensures that all vertices in the tree belong to $Y_k(v)$.

By the construction of the tree, the number of vertices in it is of the order of $\prod_{i=0}^{r} d_i^{s_i} = g$. In order to upper-bound k (such that all vertices belong to $Y_k(v)$), consider any vertex u in some level $j \in L_i$, where $0 \leq i \leq r$. Since $d_i = d_{i-1}^{1/2}$, so that $s_t = 2s_{t-1}$ for each t, we get that $s_i \leq \sum_{i'<i} s_{i'}$. Therefore, $\text{dist}(u, v) \leq 2s_i$. It follows that the number of vertices in the subtree rooted at u that are at distance at most $\text{dist}(u, v)$ from u is upper bounded by $d_i^{s_i} \cdot d_{i+1}^{s_i} < g^{3/2(r+1)}$. Since this is true for every vertex in the tree, we get that $|\Gamma_{\text{dist}(u,v)}(u)| = O(g^{3/2(r+1)} \cdot s_r) = O(g^{3/2(r+1)} \cdot \log g)$, which gives us an upper bound on k, from which the lemma follows. ◀