Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux 1, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report RR-1388-06

Fast Deterministic Distributed Algorithms for Sparse Spanners

by Bilel Derbel and Cyril Gavoille

February, 2006

Fast Deterministic Distributed Algorithms for Sparse Spanners

Bilel Derbel* Cyril Gavoille*†

Abstract

This paper concerns the efficient construction of sparse and low stretch spanners for unweighted arbitrary graphs with n nodes. All previous deterministic distributed algorithms, for constant stretch spanner of $o(n^2)$ edges, have a running time $\Omega(n^\epsilon)$ for some constant $\epsilon > 0$ depending on the stretch. Our deterministic distributed algorithms construct constant stretch spanners of $o(n^2)$ edges in $o(n^\epsilon)$ time for any constant $\epsilon > 0$.

More precisely, in the Linial's free model, we construct in $n^{O(1/\sqrt{\log n})}$ time, for every graph, a 5-spanner of $O(n^{3/2})$ edges. The result is extended to $O(k^{2.322})$-spanners with $O(n^{1+1/k})$ edges for every parameter $k \geqslant 1$. If the minimum degree of the graph is $\Omega(\sqrt{n})$, then, in the same time complexity, a 9-spanner with $O(n)$ edges can be constructed.

Keywords: distributed algorithms, graph spanners, time complexity, Linial's free model, deterministic and randomized algorithms

1 Introduction

This paper deals with deterministic distributed construction of sparse and low stretch graph spanners. Intuitively, spanners can be thought of as a generalization of the concept of a spanning tree. We look for a spanning subgraph such that the distance between any two nodes in the subgraph is bounded by some constant times the distance in the whole graph. More formally, H is a *k-spanner* of a graph G if H is a spanning subgraph of G, and if $d_G(u,v) \leqslant k \cdot d_H(u,v)$ for all nodes u,v of G, where $d_X(u,v)$ denotes the distance from u to v in the graph X. The smallest k for which H is a k-spanner is called the *stretch* of H, and the *size* of H is its number of edges. The quality of a spanner refers to the trade-off between the stretch and the size of the spanner.

*Laboratoire Bordelais de Recherche en Informatique, Université Bordeaux 1. {*derbel,gavoille*}*@labri.fr*.

†Supported by the project "PairAPair" of the ACI Masses de Données.

The distributed model of computation we will be concerned with is the Linial's free model [Lin87], also known as $\mathcal{LOCAL}$ model in [Pel00]. In this model, communication is completely synchronous and reliable. At every time unit, each node may send or receive a message of unlimited size to or from all its neighbors, and can locally compute any function. The model also assumes that each node is equipped with a unique identifier. Much as PRAM algorithms in parallel computing give a good indication of parallelism, the free model gives a good indication of the locality and distributed time.

From a theoretical point of view, we are interested in the *locality nature* of constructing graph spanners, i.e., what spanners can we compute assuming only some local knowledge? The locality of a distributed problem is often expressed in term of the time needed to resolve it. In fact, in the distributed setting, the best a node can do in $O(t)$ time units is to collect its t neighborhood. For instance, $\Theta(\log^* n)$ time are necessary and sufficient to compute a maximal independent set for trees, bounded degree graphs, or bounded growth graphs with n nodes [CV86, GPS88, Lin92, KMNW05]. Results are known for other fundamental problems such as non-uniform coloring [Awe87, PS96], minimum spanning tree [Elk04a, Elk04b, LPSP01, LPSPP05, PR00], small dominating set [KP98, PV04], and maximal matching [KMW04, Lin92].

Graph spanners are in the basis of various applications in distributed systems. For instance, Peleg and Ullman [PU89a] establishes the relationship between the quality of spanners, and the time and message complexity of network synchronizers (see also [Awe85, MS00]). Spanners are also implicitly used for the design of low stretch routing schemes with compact tables [Cow01, EGP03, PU89b, RTZ02, TZ01], and appear in many parallel and distributed algorithms for computing approximate shortest paths and for the design of compact data-structures, a.k.a. distance oracles [BS04, GPPR04, RTZ05, TZ05, Coh98].

1.1 Related Works

Sparse and low stretch spanners can be constructed from (d,c)-decomposition of Awerbuch and Peleg [AP90], that is a partition of the graph into clusters of diameter at most d such that the graph obtained by contracting each cluster can be properly c-colored. There are several deterministic algorithms for constructing (d,c)-decompositions [ABCP93, ABCP96, ABCP98, PS96]. The resulting distributed algorithms provide $O(k)$-spanners of size $O(n^{1+1/k})$, for any integral parameter $k \geqslant 1$. However, these algorithms run in $\Omega(n^{1/k+\epsilon})$ time, where $\epsilon = \Omega(1/\sqrt{\log n}\,)$, and provide a stretch $s \geqslant 4k-3$.

Better stretch-size trade-offs exist but with an increasing time complexity. Recently, a deterministic distributed algorithm has been proposed for constructing a $(2k-1)$-spanner of size $O(n^{1+1/k})$ in $O(n^{1-1/k})$ time [DMZ06]. In particular, 3-spanners of size $O(n^{3/2})$ can be deterministically constructed in $O(\sqrt{n}\,)$ time.

Elkin et al. [Elk01, EZ04, EP04] develop a distributed algorithm for spanners such that the distance between two nodes in the spanner is at most $1+\epsilon$ times the distance in the original graph plus β. The size is $O(\beta n^{1+\delta})$ whereas the time is $O(n^{\delta})$, where $\beta = \beta(\delta, \epsilon)$

is independent of n but grows super-polynomially in δ^{-1} and ϵ^{-1}.

Randomized algorithms achieving better performances exist. Baswana et al. [BS03, BKMP05] gave a randomized algorithm which computes an optimal $(2k-1)$-spanner with expected size $O(n^{1+1/k})$ in $O(k)$ time. The latter stretch-size trade-off is optimal since, according to an Erdös Conjecture verified for $k = 1, 2, 3, 5$ [Wen91], there are graphs with $\Omega(n^{1+1/k})$ edges and girth $2k+2$ (the length of the smallest induced cycle), thus for which every s-spanner requires $\Omega(n^{1+1/k})$ edges if $s < 2k+1$. However, as mentioned in [ABCP96], a randomized solution might not be acceptable in some cases, especially for distributed computing applications. In the case of graph spanners, deterministic algorithms that *guarantee* a high quality spanner is more than a theoretical interest. Indeed, one cannot just run a randomized distributed algorithm several times to guarantee a good decomposition, since it is impossible to efficiently check the global quality of the spanner in the distributed model.

1.2 Results

We consider unweighted connected graphs with n nodes. All previous deterministic distributed algorithm for $O(1)$-spanner of size $o(n^2)$ have a running time $\Omega(n^\delta)$ for some constant $\delta > 0$ depending on the stretch. In this paper we construct constant stretch spanner of size $o(n^2)$ in $o(n^\epsilon)$ time for any constant $\epsilon > 0$.

More precisely, in the free model we construct in $n^{O(1/\sqrt{\log n})}$ time for every graph a 5-spanner of $O(n^{3/2})$ edges. The result is extended to larger stretch spanner of size $O(n^{1+1/k})$ for every $k \geqslant 1$. More precisely, for k power of two, the stretch is at most $k^{\log_2 5} < k^{2.322}$. For other values of k, we obtain stretches $s = s(k)$ which surprisingly depend on the positions of the first two leading 1's in the binary written of k (cf. the table below for the first values).

k	1	2	3	4	5
$s(k)$	1	5	9	25	33

We also show that if the minimum degree of the graph is $\Omega(\sqrt{n})$, then, in the same time complexity, a 9-spanner with $O(n)$ edges can be constructed.

The previous algorithms have simple randomized versions with improved performances. In particular, we can compute a 5-spanner of size $O(n \log^2 n)$ in $O(\log n)$ time if the minimum degree is $\Omega(\sqrt{n})$.

1.3 Outline of the paper

The main idea to break the $O(n^\delta)$ time barrier is to abandon the optimality on the stretch-size trade-off. We show that constant stretch spanners can be constructed on the basis of a maximal independent set, i.e., a set of pairwise non-adjacent nodes, maximal for inclusion.

This can be deterministically computed in $n^{O(1/\sqrt{\log n})}$ time [ABCP96, PS96]. Therefore, the time complexity to construct our spanners is improved by a factor of $n^{1/k}$.

The generic algorithm is described in Section 2 and analyzed in Section 3, where a distributed implementation is presented.

We mainly reduce the problem to the computation of an *independent ρ-dominating set*, that is a set X of pairwise non-adjacent nodes such that every node of the graph is at distance at most ρ from X. Using the terminology of [Pel00], a ρ-dominating set if nothing else than a (ρ, s)-ruling set for some $s > 1$. Actually, in order to optimize the stretch, the main algorithm combines two strategies in a way depending on the binary written of k.

In Section 4, we present the main results about 5- and 9-spanners. Observing that for $\rho = 1$ an independent ρ-dominating set is a maximal independent set, we conclude that our generic algorithm can be implemented to run in $n^{O(1/\sqrt{\log n})}$ time for $\rho = 1$. Several optimizations are then proposed including randomization and graphs of large minimum degree.

2 A Generic Algorithm

2.1 Definitions

Let us consider an unweighted connected graph $G = (V, E)$. Given an integer $t \geqslant 1$, the t-th power of G, denoted by G^t, is the graph obtained from G by adding an edge between any two nodes at distance at most t in G. For a set of nodes H, $G[H]$ denotes the subgraph of G induced by H. For $X, Y \subseteq V$, let $d_G(X, Y) = \min\{d_G(x, y) \mid x \in X \text{ and } y \in Y\}$.

We associate with each $v \in V$ a *region*, denoted by $R(v)$, that is a set of nodes containing v and inducing a connected subgraph of G. Given $C \subseteq V$, G_C denotes the graph whose node set is C, and there is an edge between u and v in C if $d_G(R(v), R(u)) = 1$. We denote by $R^+(v) = \{u \in V \mid d_G(u, R(v)) \leqslant 1\}$ and by $R^+_C(v) = \{u \in C \mid d_G(R(u), R(v)) \leqslant 1\}$.

The *eccentricity* of a node v in G is defined as $\max_{u \in V}\{d_G(u, v)\}$. For a node $v \in X$, we denote by $\mathrm{BFS}(v, X)$ a Breadth First Search spanning tree in X rooted at v. We define $\mathrm{IDS}(G, \rho)$ as any independent ρ-dominating set of G. Finally, we define the integer $\ell(x)$ as follows:

$$\ell(x) = \begin{cases} -1 & \text{if } x \leqslant 0, \\ \lfloor \log_2 x \rfloor & \text{otherwise.} \end{cases}$$

In the reminder of the paper we assume the free model of computation as defined in the introduction. We define the time complexity of a distributed algorithm to be the worst-case number of time units from the beginning of the algorithm to its termination.

2.2 Description of the algorithm

The main idea of the algorithm is to find an efficient clustering of dense regions in the graph. A high level description of the algorithm, named SPANNER, is given in Fig. 1. Intuitively, i_0 represents the relative position of the first two leading 1's in the binary written of k.

SPANNER

Input: a graph $G = (V, E)$ with $n = |V|$, and integers $\rho, k \geqslant 1$
Output: a spanner S

1. $i_0 := \ell(k) - \ell(k - 2^{\ell(k)})$; $C := V$; $r = 0$; $\forall v \in V$, $R(v) := \{v\}$, and $c(v) := v$
2. **for** $i := 1$ **to** $\ell(k) + 1$ **do: if** $i = i_0$ **then** STRATEGY 1 **else** STRATEGY 2

Figure 1: The algorithm SPANNER.

The algorithm works in many phases, where new regions are formed at each phase. There are two types: the light regions (L) and the heavy regions (H). At a given phase, some of the heavy regions are selected and enlarged by including nodes from other neighboring regions. One important observation is that each new enlarged region is connected and the new constructed regions are mutually disjoint.

At each phase of the algorithm, one of the two strategies depicted in Fig. 2 and Fig. 3 applies. The main idea behind the two strategies is the same: choose some well selected dense regions and merge them with the other ones in order to form new larger regions. The main difference is that the density of a region is computed in a different way. The stretch of the output spanner depends on the way the radius of the regions increases and on the total number of phases of the algorithm, depending on the volume of the regions. And, radius and volume increasing vary differently.

On one hand, in STRATEGY 1, a region is dense if its neighborhood is $n^{1/k}$ times greater than its size. Applying only STRATEGY 1 allows to obtain small stretch for small values of k. However, asymptotically, the stretch is exponential in k. On the other hand, in STRATEGY 2, a region is dense if the number of its neighboring regions is $n^{1/k}$ times greater than its size which provides an exponential growth of the size of a region. Applying only STRATEGY 2 allows to obtain asymptotically stretches polynomial in k.

The main idea of algorithm SPANNER is to switch from one strategy to an other at each phase in order to obtain the smallest possible stretch. A full analysis shows that, by alternating STRATEGY 1 and STRATEGY 2, the best stretch can be obtained by applying STRATEGY 1 only once at a well chosen phase i_0. Typically, $i_0 = p - q$ if $k = 2^p + 2^q$ with $p > q$.

We associate with each region $R(v)$ an active node, called *center*, and the set of centers forms C. Initially, each node is the center of the region formed by itself. Each phase $i \in \{1, \ldots, \ell(k)+1\}$ can be decomposed in seven parts we briefly sketch.

1. $L := \left\{v \in C,\ |R^+(v)| \leqslant n^{1/k} \cdot |R(v)|\right\}$ and $H := C \setminus L$;
2. $\forall (u,v) \in L \times V$ such that $\exists$ edge e between $R(u)$ and z, $S := S \cup \{e\}$
3. $X := \mathrm{IDS}(G^{2(r+1)}[H], \rho)$
4. $\forall z \in V$, if $d_G(z, X) \leqslant (2\rho+1)r + 2\rho$, then set $c(z)$ to be its closest node of X, breaking ties with identities.
5. $\forall v \in X$, $R(v) := \{z \in V \mid c(z) = v\}$
6. $\forall v \in X$, $S := S \cup \mathrm{BFS}(v, R(v))$
7. $C := X$ and $r := (2\rho+1)r + 2\rho$

Figure 2: Strategy 1.

1. $L := \left\{v \in C,\ |R_C^+(v)| \leqslant n^{1/k} \cdot |R(v)|\right\}$ and $H := C \setminus L$
2. $\forall (u,v) \in L \times C$ such that $\exists$ edge e between $R(u)$ and $R(v)$, $S := S \cup \{e\}$
3. $X := \mathrm{IDS}((G_C)^2[H], \rho)$
4. $\forall u \in C$, if $d_{G_C}(u, X) \leqslant 2\rho$, then set $c(u)$ to be its closest node of X in G_C, breaking ties with identities.
5. $\forall v \in X$, $R(v) := \{R(u) \mid u \in C \text{ and } c(u) = v\}$
6. $\forall v \in X$, $S := S \cup \mathrm{BFS}(v, R(v))$
7. $C := X$ and $r := (4\rho+1)r + 2\rho$

Figure 3: Strategy 2.

In Step 1, we compute the two sets H and L corresponding respectively to heavy and light regions. In Step 2, a light region is connected with some neighboring nodes. This step is crucial in the stretch bound analysis. If Strategy 1 is applied, then each light region is connected with each neighboring node in V, i.e., $\forall u \in L, R^+(u)$ is spanned. If Strategy 2 is applied, then each light region is connected with every neighboring region. Note that at the beginning of a given phase, every region is spanned by a BFS tree constructed in Step 6 of the previous phase.

The nodes H are then processed at the aim of constructing new regions with a set of new centers. The key point of our construction is to efficiently merge *all* the regions defined by the set H into *more dense*, *connected* and *disjoint* regions. In order to guarantee that the algorithm terminates quickly, the dense regions must grow enough. More precisely, if a dense region $R(v)$ is enlarged it must contain at least its neighborhood $R^+(v)$ when STRATEGY 1 is applied or its neighborhood in the graph G_C if STRATEGY 2 is applied. It is clear that two regions at distance one or two (in G or in G_C depending on the strategy 1 or 2) cannot grow simultaneously without overlapping. Thus, a difficulty is to elect in an efficient way the centers of regions that are allowed to grow in parallel.

In Step 3, we compute an independent ρ-dominating set X in the graph $G^{2(r+1)}[H]$ if STRATEGY 1 is applied (resp. $(G_C)^2[H]$ for STRATEGY 2), where r is a radius that grows at each phase. The set X defines the set of nodes allowed to grow in parallel.

In order to guarantee that nodes in non selected regions in Step 3 (the set $H \setminus X$) will be spanned by the output spanner, we must merge them with nodes in the selected regions. Thus, in Step 4, we define a coloring strategy allowing a correct merge process. In fact, in order to ensure that the new regions are disjoint, we let nodes choose their new region in a consistent manner, i.e., a node chooses to be in the region of the closest node in X breaking ties using identities. If STRATEGY 1 is applied then each node chooses by itself its new dominator, i.e., its new region. However, once a node u chooses its new dominator node v, and in order to ensure that the new formed regions are connected, we include all the nodes in the shortest path between u and v, even those in non dense region. If STRATEGY 2 is applied then, the center of each region chooses a new region and merge its hole region with the new chosen region.

In Step 5, the new regions are formed according to the coloring step. Note that as soon as the new region are formed, they are spanned in Step 6.

Finally, in Step 7, the set C and the variable r are updated for the next phase.

3 Analysis of the Algorithm

For every phase i, we denote by H_i (resp. X_i and L_i) the set H (resp. X and L) computed during phase i, i.e., after Steps 1 and 3 of phase i. Similarly, we denote by $c_i(z)$ the color of z assigned during phase i, i.e., after Step 4 of phase i. We denote by C_i the set C at the beginning of phase i, and r_i denotes the value of r at the beginning of phase i. For a node $v \in C_i$, we denote by $R_i(v)$ the region of v at the beginning of phase i.

In the following we need the four important properties.

Lemma 3.1 *At the beginning of phase i, every $v \in C$ is of eccentricity at most r in $G[R(v)]$.*

Proof. We prove the lemma by induction. The lemma is clearly true for $i = 1$. Let us consider a node $v \in C_i$ and a node $z \in R_i(v)$ at a given phase $i > 1$. From Step 7 of the algorithm, C_i corresponds to the set X_{i-1}. Thus, the region $R_i(v)$ was computed in Step 5 of phase $i - 1$.

Case 1: Strategy 1 was used at phase $i - 1$. Hence, using Step 5, for every node $z \in R_i(v)$, z was colored v at phase $i - 1$. Thus, $\forall z \in R_i(v), c_{i-1}(z) = v$. Thus, from the coloring step, there exists a shortest path $P = (z = z_1, z_2, z_3, \dots, z_l, v)$ connecting z and v in G, with $l \leqslant 2(r_{i-1} + 1)\rho + r_{i-1}$. Let us consider $z_j \in P$ with $1 \leqslant j \leqslant l$. Let us first note that $d_G(z_j, v) \leqslant 2(r_{i-1} + 1)\rho + r_{i-1}$ and $v \in X_{i-1}$. Hence, z_j has also chosen a color at phase $i - 1$. Using the fact that the coloring is consistent, $c_{i-1}(z_j) = v$. Thus, using Step 5, $P \subseteq R_i(v)$. Hence, $d_{G[R_i(v)]}(z, v) \leqslant 2(r_{i-1} + 1)\rho + r_{i-1} = r_i$.

Case 2: Strategy 2 was used at phase $i-1$. Hence, using Step 5, there exists $u \in C_{i-1}$ such that $c_{i-1}(u) = v$ and $z \in R_{i-1}(u)$. Using Step 4, there exists a path P in $G_{C_{i-1}}$ such that $P = (u = u_1, u_2, ..., u_l = v)$ with $l \leqslant 2\rho$. From now, it is convenient for the proof of the lemma to view the path P as a directed path where u_1 is the leftmost node and u_l is the rightmost one. From the definition of the graph $G_{C_{i-1}}$, for every pair of successive nodes u_j and u_{j+1}, there exists a pair of neighboring nodes z_j and z_{j+1} such that $z_j \in R_{i-1}(u_j)$, $z_{j+1} \in R_{i-1}(u_{j+1})$ and $(z_j, z_{j+1}) \in E$. Thus, $d_{R_{i-1}(u_j)\cup R_{i-1}(u_{j+1})}(u_j, u_{j+1}) \leqslant d_{R_{i-1}(u_j)}(u_j, z_j) + 1 + d_{R_{i-1}(u_{j+1})}(u_{j+1}, z_{j+1})$. Thus, using the induction hypothesis, $d_{R_{i-1}(u_j)\cup R_{i-1}(u_{j+1})}(u_j, u_{j+1}) \leqslant 2r_{i-1} + 1$. Therefore, $d_{\cup_{1\leqslant j\leqslant l} R_{i-1}(u_j)}(u_1, u_l) \leqslant 2\rho \cdot (2r_{i-1} + 1) = 4\rho \cdot r_{i-1} + 2\rho$. Because we have $z \in R_{i-1}(u = u_1)$, then $d_{\cup_{1\leqslant j\leqslant l} R_{i-1}(u_j)}(z, u_l) \leqslant 4\rho \cdot r_{i-1} + 2\rho + r_{i-1} = (4\rho + 1)r_{i-1} + 2\rho$.

Remark that $\forall 1 \leqslant j \leqslant l$, $d_{G_{C_{i-1}}}(u_j, v) \leqslant 2\rho$. Hence, u_j has also chosen a color in Step 4 of phase $i - 1$. Using the fact that the coloring is consistent, $c_{i-1}(u_j) = v$. Thus, $\forall 1 \leqslant j \leqslant l$, $R_{i-1}(u_j) \subset R_i(v)$. Therefore, $d_{R(v)}(z, v) \leqslant (4\rho + 1)r_{i-1} + 2\rho$. By Step 7, we have $r_i = (4\rho + 1)r_{i-1} + 2\rho$ which completes the proof. □

Lemma 3.2 *At the beginning of phase i, for every two nodes $u \neq v \in C$, $R(u) \cap R(v) = \varnothing$.*

Proof. We prove the lemma by induction. The lemma is clearly true for $i = 1$. Let us consider a phase $i > 1$. Let us consider two nodes $u, v \in C_i$ and $u \neq v$. From Step 7 of the algorithm, the set C_i corresponds to the set X_{i-1}. Hence, $u, v \in X_{i-1}$.

Case 1: Strategy 1 was used at phase $i - 1$. Suppose that there exists $z \in R_i(u) \cap R_i(v)$. From Step 5 of phase $i - 1$, we have: $c_{i-1}(z) = u$ and $c_{i-1}(z) = v$. Thus, $u = v$ which is a contradiction.

Case 2: Strategy 2 was used at phase $i - 1$. Suppose that there exists $z \in R_i(u) \cap R_i(v)$. From Step 5 of phase $i - 1$, there exist $w_1, w_2 \in C_{i-1}$ such that $c_{i-1}(w_1) = u$ and $c_{i-1}(w_2) = v$ and $z \in R_{i-1}(w_1) \cap R_{i-1}(w_2)$. Using the induction hypothesis, $w_1 = w_2$. Thus, $c_{i-1}(w_1) = c_{i-1}(w_2) = u = v$ which is a contradiction.

Therefore for every two nodes $u, v \in C_i$ such that $u \neq v$, $R_i(u) \cap R_i(v) = \varnothing$, which completes the proof. □

Lemma 3.3 *At the beginning of phase $i \neq i_0$, if $|R_i(v)| \geqslant \mathcal{V}_i$ for every $v \in C_i$, then $|R_{i+1}(v)| \geqslant n^{1/k} \cdot \mathcal{V}_i^2$ for every $v \in C_{i+1}$.*

Proof. First, because $i \neq i_0$, the STRATEGY 2 is applied at phase i. Note also that the set C_{i+1} corresponds to the set X_i ($\subseteq H_i \subseteq C_i$) computed at phase i. Let us consider a node v in X_i. Consider a node $u \in R^+_{C_i}(v)$. Clearly, $u \in C_i \setminus X_i$, otherwise the independence of set X_i is violated. Suppose that there exists a node v' at distance 1 from u in the graph G_{C_i}. Thus, v' is at distance 1 from v in $G^2_{C_i}$. Thus, $v' \in C_{i-1} \setminus X_{i-1}$, otherwise the independence of set X_i is violated. Therefore, v is the closest node in X_i to u. Thus, $c_i(u) = v$ and hence, $R_i(u) \subseteq R_{i+1}(v)$. Therefore, using Lemma 3.2,

$$|R_{i+1}(v)| > |R^+_{C_i}(v)| \cdot \min\{R_i(u) \mid u \in C_i \text{ and } c_i(u) = v\} \ .$$

Thus from Step 1 of phase i, we have

$$\begin{array}{rcl} |R_{i+1}(v)| & > & n^{1/k} \cdot |R_i(v)| \cdot \mathcal{V}_i \\ & > & n^{1/k} \cdot \mathcal{V}_i^2 \end{array}$$

□

Lemma 3.4 *For every node $u \in V$, there exists a phase i and a node $v \in V$ such that:*

- *at the beginning of phase i, $v \in C$ and $u \in R(v)$; and*
- *v is in the set L computed in Step 1 of phase i.*

Proof. Let us denote by $i_1 = \ell(k)$ and $i_2 = \ell(k - 2^{\ell(k)})$, i.e., $i_0 = i_1 - i_2$. Let $\mathcal{V}_i$ the minimum size of any region corresponding to a node in C_i.

First, let us show that

Claim 3.1 *Every node in $C_{\ell(k)+1}$ are in $L_{\ell(k)+1}$, i.e. $C_{\ell(k)+1} = L_{\ell(k)+1}$*

Proof. *Case 1:* $i_2 = -1$. Hence, $i_0 = \ell(k) + 1$ and $k = 2^{\ell(k)}$. By induction and using Lemma 3.3, at the beginning of phase i_0, the size of the region of any node in C_{i_0} is at least $\mathcal{V}_{i_0} \geqslant n^{(2^{i_0-1}-1)/k} = n^{(k-1)/k}$ (because $\mathcal{V}_1 = 1$). Hence, for every $v \in C_{i_0}$,

$$n^{1/k} \cdot |R_{i_0}(v)| \geqslant n^{1/k} \cdot n^{(k-1)/k} = n \geqslant |R^+_{i_0}(v)|$$

Because, STRATEGY 1 is applied at phase i_0, after Step 1, $C_{\ell(k)+1} = L_{\ell(k)+1}$.

Case 2: $i_2 \geqslant 0$. We have $\mathcal{V}_{i_0} \geqslant n^{(2^{i_0-1}-1)/k}$. At phase i_0, we apply STRATEGY 1. Thus, using the same arguments than in Lemma 3.3, the new enlarged regions constructed at phase i_0 contain all their neighborhood (in G) otherwise the independence of set X_{i_0} is violated. Thus, $\mathcal{V}_{i_0+1} \geqslant n^{1/k} \cdot \mathcal{V}_{i_0} \geqslant n^{2^{i_0-1}/k}$.

Subcase 2.1: Suppose $i_2 = 0$. Then, $i_0 = i_1 = \ell(k)$ and $k = 2^{\ell(k)} + 1$. Thus,

$$n^{1/k} \cdot \mathcal{V}^2_{\ell(k)+1} \geqslant n^{(1+2^{\ell(k)})/k} = n$$

Subcase 2.2: Suppose $i_2 \neq 0$. Then, applying Lemma 3.3 to phase i going from $i_0 + 1$ to $\ell(k) + 1$ (excluding $\ell(k) + 1$), we have

$$\begin{aligned}\mathcal{V}_{\ell(k)+1} \geqslant n^{1/k} \cdot \mathcal{V}^2_{\ell(k)} \;&\geqslant\; n^{1/k} \cdot \left(n^{1/k} \cdot \mathcal{V}^2_{\ell(k)-1}\right)^2 \;\geqslant\; \ldots \;\geqslant\; n^{(1+2^1+\cdots+2^{j-1})/k} \cdot \mathcal{V}^{2^j}_{\ell(k)+1-j} \\ \geqslant n^{\sum_{j=0}^{(\ell(k)+1)-(i_0+1)-1} 2^j/k} \cdot \mathcal{V}^{2^{(\ell(k)+1)-(i_0+1)}}_{i_0+1} \;&\geqslant\; n^{(2^{\ell(k)-i_0}-1+2^{\ell(k)-1})/k}\end{aligned}$$

Thus,

$$n^{1/k} \cdot \mathcal{V}^2_{\ell(k)+1} \;\geqslant\; n^{(1+2(2^{\ell(k)-i_0}-1+2^{\ell(k)-1}))/k} \;\geqslant\; n^{(2^{\ell(k)-i_0+1}-1+2^{\ell(k)})/k}$$

Note that $\ell(k) - i_0 + 1 = i_1 - i_0 + 1 = i_2 + 1$. Thus,

$$2^{\ell(k)-i_0+1} - 1 + 2^{\ell(k)} = 2^{i_2+1} + 2^{i_1} - 1 \;\geqslant\; k$$

Thus, we get $n^{1/k} \cdot \mathcal{V}^2_{\ell(k)+1} \geqslant n$.

Hence, in both subcases, we have

$$\frac{n}{\mathcal{V}_{\ell(k)+1}} \leqslant n^{1/k} \cdot \mathcal{V}_{\ell(k)+1}$$

Now, let us take a node $v \in C_{\ell(k)+1}$. Because STRATEGY 2 is applied at phase $\ell(k)+1$, v is in $L_{\ell(k)+1}$ iff $|R^+_{C_{\ell(k)+1}}(v)| \leqslant n^{1/k} \cdot |R_{\ell(k)+1}(v)|$. Let us show that, in fact v is in $L_{\ell(k)+1}$. There is at least $\mathcal{V}_{\ell(k)+1}$ nodes in a region, thus using Lemma 3.2, we have

$$|R^+_{C_{\ell(k)+1}}(v)| \;\leqslant\; \frac{n}{\mathcal{V}_{\ell(k)+1}} \;\leqslant\; n^{1/k} \cdot \mathcal{V}_{\ell(k)+1} \;\leqslant\; n^{1/k} \cdot |R_{\ell(k)+1}(v)|$$

Thus, All nodes of $C_{\ell(k)+1}$ are in $L_{\ell(k)+1}$.

Therefore, in both cases, $C_{\ell k+1} = L_{\ell(k)+1}$. □

We are now ready to prove Lemma 3.4.

Let us consider a node $u \in V$. At the beginning of the algorithm, the node u is also in C. If $R(u)$ does not satisfy the condition of Step 1 of the algorithm then $u \in L_1$. Hence, the lemma is true. Otherwise, u participates in Step 3 and u is at distance at most 2ρ from a node in X_1. Thus, u joins the region of some other node $u_1 \in X_1 \subset C_2$ (possibly equal to u) and $u \in R_2(u_1)$. Let us call u_1 the new dominator of u. At the next phase $i = 2$, if u_1 is in L_2 then the lemma holds. Otherwise, u_1 participates in Step 3. There are two cases:

Case 1: Suppose that STRATEGY 1 is applied at phase 2. Then, u_1 is at distance at most $2(r_2+1)\rho$ from a node in set X_2 in the G. Hence, u is at distance at most $2(r_2+1)\rho+r_2$

from a node in set X_2 in the graph G. Hence u will be colored in phase 2 and it will join the region of a new dominator $u_2 \in X_2 \subset C_3$.

Case 2: Suppose that STRATEGY 2 is applied at phase 2. Then, u_1 is at distance at most 2ρ from a node in set X_2 in the graph G_{C_2}. Hence, the region of u_1 joins the region of a new dominator $u_2 \in X_2 \subset C_3$. Because u is in the region of u_1, u will join the region of the new dominator u_2.

In both case, u will be in the region of a new dominator $u_2 \in C_3$. It is not difficult to show by induction that, if the $(i-1)$-th dominator of u is still not in L_i at phase i, then u will join the region of a new dominator u_i.

In the worst-case, the $\ell(k)$-th dominator of u will be in $L_{\ell(k)+1}$ in phase $\ell(k)+1$. Thus, there must exist some node v such that u is in the region of v at the beginning of some phase i, i.e., v is the $(i-1)$-th dominator of u, and v is in the set L_i computed in Step 1 of phase i. □

3.1 Stretch and size analysis

Lemma 3.5 *For any integer $k, \rho \geqslant 1$, the stretch s of the output spanner S of algorithm* SPANNER *verifies*

$$s \leqslant \begin{cases} (4\rho+1)^{\ell(k)} & \text{if } k = 2^{\ell(k)}, \\ 2(2\rho+1)(4\rho+1)^{\ell(k)-1} + 4\rho(4\rho+1)^{\ell(k-2^{\ell(k)})} - 1 & \text{otherwise.} \end{cases}$$

Proof. As a consequence of Lemma 3.4 and Step 6 of the algorithm, every node $u \in V$ is spanned by the output S of the algorithm, i.e., S is a spanner of G. From the initialization step of the algorithm, we have $r_1 = 0$. Let us denote by $i_1 = \ell(k)$ and $i_2 = \ell(k - 2^{\ell(k)})$, i.e., $i_0 = i_1 - i_2$. For every $1 < i \leqslant i_0$, we have $r_i = (4\rho+1)r_{i-1} + 2\rho$. Thus, $r_i = \frac{1}{2} \cdot ((4\rho+1)^{i-1} - 1)$ for $1 \leqslant i \leqslant i_0$.

Let us consider an edge $(z, z') \in E$. Using Lemma 3.4, there exists a phase j (resp. j') and a node v (resp. v') such that $v \in C_j$ (resp. $v' \in C_{j'}$), $z \in R_j(v)$ (resp. $z' \in R_{j'}(v')$) and $v \in L_j$ (resp. $v' \in L_{j'}$). We take v (resp. v') to be the first dominator of z, i.e., node in C whose region contains z, (resp. z') that fall into set L. In fact, one can see that node z (or z') can be in a sparse region at some phase and switch to a dense region at the next phase, because either its sparse region has been merged with a neighboring dense one (if STRATEGY 2 is applied), or it is in the neighborhood of a dense region, or it is on a shortest path leading to a dense region (if STRATEGY 1 is applied). W.l.o.g., suppose that $j \leqslant j'$.

Case 1: $i_2 = -1$. Hence, $i_0 = \ell(k) + 1$ and $k = 2^{\ell(k)}$. By induction and using Lemma 3.3, at the beginning of phase i_0, the size of the region of any node in C_{i_0} is at least $n^{(2^{i_0-1}-1)/k} = n^{(k-1)/k}$. Note that we apply STRATEGY 1 at phase i_0. Thus, every node in C_{i_0} will be in L.

Subcase 1.1: Suppose that $j \leqslant j' < i_0$. Thus, using Step 6, a BFS tree spanning $R_j(v)$ is added to the output spanner at phase $j-1$. In addition, using the same arguments than in the proof of Lemma 3.4, one can easily show that there exists a node $v'' \in C_j$ such that $z' \in R_j(v'')$. Hence, a BFS tree spanning $R_j(v'')$ is added to the output spanner at phase $j-1$. Using Step 2, there exists an edge $e \in S$ connecting $R_j(v)$ and $R_j(v'')$. Thus, $d_S(z,z') \leqslant 4r_j + 1 = 2(4\rho+1)^{j-1} - 1$.

Subcase 1.2: Suppose that $j = j' = i_0$. Hence, Strategy 1 is applied at phase j. Thus, a BFS tree spanning $R_j(v)$ is added the output spanner at phase $j-1$. Using Step 2, $R_j^+(v)$ is also spanned. Thus, because $z' \in R_j^+(v)$, $d_S(z,z') \leqslant 2r_j + 1 = 2r_{i_0} + 1 \leqslant (4\rho+1)^{i_0-1}$.

Finally, because $\rho > 0$, in both subcases, the stretch is bounded by $(4\rho+1)^{\ell(k)}$.

Case 2: $i_2 \geqslant 0$. Hence, at the beginning of phase i_0+1, the radius of a region is at most $(2\rho+1)r_{i_0} + 2\rho$. Thus,

$$r_{i_0+1} = \frac{1}{2}(2\rho+1)\cdot\big((4\rho+1)^{i_0-1} - 1\big) + 2\rho \tag{1}$$

Suppose $i_2 \neq 0$. For every $i_0 + 1 < i \leqslant \ell(k)+1$, we have $r_i = (4\rho+1)r_{i-1} + 2\rho$. Thus, by induction, for every $i_0+1 < i \leqslant \ell(k)+1$,

$$r_i = (4\rho+1)^{i-i_0-1}\cdot r_{i_0+1} + \frac{1}{2}((4\rho+1)^{i-i_0-1} - 1)$$

In particular,

$$r_{\ell(k)+1} = (4\rho+1)^{\ell(k)-i_0}\cdot r_{i_0+1} + \frac{1}{2}((4\rho+1)^{\ell(k)-i_0} - 1)$$

Thus,

$$r_{\ell(k)+1} = (4\rho+1)^{i_2}\cdot r_{i_0+1} + \frac{1}{2}((4\rho+1)^{i_2} - 1) \tag{2}$$

Now suppose that $i_2 = 0$. Hence, $i_0 = \ell(k)$ and it is easy to see that Eq. 2 is still true.

Subcase 2.1: Suppose that $j \neq i_0$. Thus, it easy to show that there exists a node $v'' \in C_j$ such that $z' \in R_j(v'')$. Using Step 6, $R_j(v'')$ and $R_j(v)$ were spanned by a BFS tree at phase $j-1$. In addition, because Strategy 2 is applied at phase j, an edge e connecting $R_j(v)$ and $R_j(v'')$ is added at phase j (Step 2). Thus, $d_S(z,z') \leqslant 4r_j + 1$.

Subcase 2.2: Suppose that $j = i_0$. Thus, because $R_j^+(v)$ is spanned, $d_S(z,z') \leqslant 2r_j + 1$.

Using the same arguments than in Lemma 3.4, at phase $\ell(k)+1$, all active nodes will be in the set $L_{\ell(k)+1}$. Thus the stretch is bounded by $4r_{\ell(k)+1} + 1$. Using Eq. (1) and (2), we have:

$$\begin{aligned} 4r_{\ell(k)+1} + 1 &= 4\left((4\rho+1)^{i_2}\cdot r_{i_0+1} + \tfrac{1}{2}((4\rho+1)^{i_2} - 1)\right) + 1 \\ &= 2(2\rho+1)(4\rho+1)^{i_1-1} + 4\rho(4\rho+1)^{i_2} - 1 \end{aligned}$$

□

Lemma 3.6 *For any integer $k, \rho \geqslant 1$, the size of the output spanner S of algorithm* Spanner *is $O(\log k \cdot n^{1+1/k})$.*

Proof. Let us fix a phase i of the algorithm. The output spanner S is updated in Steps 2 and 6 of each phase. Let us consider two consecutive phases i and $i-1$ and the edges added by Step 6 at phase $i-1$ and the edges added by Step 2 at phase i.

Case 1: Strategy 1 is applied at phase i. Thus, the number of edges is bounded by:

$$\begin{aligned}\sum_{v \in L_i} |\mathrm{BFS}(v, R_i(v))| + \sum_{v \in L_i} \sum_{z \in R^+(v)} 1 &< n + \sum_{v \in L_i} |R^+(v)| \\ &\leqslant n + \sum_{v \in L_i} n^{1/k} |R_i(v)| \\ &\leqslant n + n^{1+1/k}\end{aligned}$$

Case 2: Strategy 2 is applied at phase i. Thus, the number of edges added is bounded by:

$$\begin{aligned}\sum_{v \in L_i} |\mathrm{BFS}(v, R_i(v))| + \sum_{v \in L_i} |R^+_{C_i}(v)| &\leqslant n + n^{1/k} \sum_{v \in L_i} |R_i(v)| \\ &\leqslant n + n^{1+1/k}\end{aligned}$$

Since there are $O(\log k)$ phases in the algorithm, the lemma is true. □

3.2 Distributed implementation and time complexity

In the free model, distributed computation of some distributed procedure A on $G^t[H]$ can be easily simulated on G as follows, charging the overall time by a factor of t. Hereafter, we assume that each node $u \in G$ can determine if it belongs or not to H. Indeed, consider one communication step in A running on some node u of $G^t[H]$ followed by one local computation step. In G, an original message in A is sent from u with a counter initialized to $t-1$ as an extra field. Now, each node $v \in G$, upon the reception of a message with some counter in its header: 1) decrements the counter; 2) stores this message if $v \in H$; and 3) forwards the incoming message with the updated counter to all its neighbors in G if the updated counter is non-null (if many messages are received during a round, then they are concatenated before being sent). After t communication rounds in G, every node $u \in H$ starts the local computation step of A on the base of all received messages during the last t communication rounds.

Similarly, given $C \subseteq V$, the computation of some distributed procedure A on G_C can be simulated on G as follows, charging the overall time by a factor $O(r)$ where r is an upper bound of the eccentricity of a node $v \in C$ in $G[R(v)]$. At each time procedure A requires for a node v of G_C to send a message to a neighbor, v broadcasts the message in $G[R(v)]$ (which is connected). The nodes at the frontier of $R(v)$, i.e., nodes having neighbors in different regions, broadcast also the message out their region. Symmetrically, upon the reception of messages from different regions, messages are concatenated and a convergecast is performed to v. The time overhead for one step of A is at most $2r+1$.

Relying on the above discussions, running procedure A on $G^{2(r+1)}[H]$ or on $(G_C)^2[H]$ can be simulated on G within a factor of $O(r)$ on the time complexity.

Lemma 3.7 *For any integer $k, \rho \geqslant 1$,* Spanner *can be implemented with a deterministic distributed algorithm in $O(\log k \cdot \rho^{\log k} \cdot \tau)$ time, where τ is the time complexity to compute an independent ρ-dominating set in a graph of at most n nodes.*

Proof. Let us first remark that in the free model, we can make the nodes know their entire p-neighborhood in $O(p)$ time. Therefore, any task which requires only information about p-neighborhood can be solved within $O(p)$ time.

Let us consider a fixed phase $i \leqslant \ell(k)$. For every $v \in C$, Steps 1 and 6 of the algorithm can be implemented, using classical broadcast-convergecast, by traversing $R_i^+(v)$ a constant number of times which is $O(r_i)$ time consuming. Note that the BFS trees constructed in Step 6 can be maintained at each phase, by adding the new nodes being merged at each phase. The time needed to run procedure IDS on $G^{2(r_i+1)}[H]$ blows up by a factor of $2(r_i+1)$ as explained before. Thus, if Strategy 1 is applied, then Step 3 is $O(r_i \cdot \tau)$ time consuming. Similarly, the time complexity of procedure IDS on $G_C[H]$ blows up by a factor $2r_i$. Thus, if Strategy 2 is applied, Step 3 is $O(r_i \cdot \tau)$ time consuming.

Steps 4 and 5 can be implemented by letting each node exploring its $O(2(r_i+1)\rho + r_i)$-neighborhood which is $O(r_i)$ time consuming for fixed ρ. Finally, Step 7 is $O(1)$ time consuming. Summing up among all steps, every phase $i \leqslant \ell(k)$ is at most $O(r_i \cdot \tau)$ time consuming.

At phase $\ell(k)+1$, the set H is empty and thus phase $\ell(k)+1$ is $O(r_{\ell(k)+1})$ time consuming.

Using Lemma 3.1 (and the radius bounds given in the proof of Lemma 3.5) and summing up among all phases, the time complexity of the algorithm is $O(\ell(k) \cdot (4\rho+1)^{\ell(k)} \cdot \tau)$ which completes the proof. $\square$

4 Applications to Low Stretch Spanners

4.1 Constant stretch spanners with sub-quadratic size

Let MIS(n) denote the time complexity for computing, by a deterministic distributed algorithm, a maximal independent set (MIS) in a graph with at most n nodes. The fastest deterministic algorithm [PS96] shows that MIS$(n) \leqslant n^{O(1/\sqrt{\log n})}$. It is also known that MIS$(n) \geqslant \Omega(\sqrt{\log n / \log\log n})$ [KMW04].

It is not difficult to check that a set X is an independent 1-dominating set if and only if X is a maximal independent set (cf. [Pel00, pp. 259, Ex. 4]). Thus, using the fast distributed MIS algorithm as a subroutine in algorithm SPANNER, we obtain:

Theorem 4.1 *There is a deterministic distributed algorithm that given a graph G with n nodes and any fixed integer $k = 2^p$ with $p \geqslant 0$, constructs a $(k^{\log_2 5})$-spanner for G with $O(n^{1+1/k})$ edges in $O(\mathrm{MIS}(n))$ time.*

Proof. Size and time are direct consequences of lemmas 3.3 and 3.7 fixing k and $\rho = 1$. Note that $\ell(k) = p = \log k$. Thus, using Lemma 3.5, the stretch of the output spanner is bounded by $5^{\log k} = k^{\log 5}$. □

Theorem 4.2 *There is a deterministic distributed algorithm that given a graph G with n nodes and any fixed integer $k = 2^p + 2^q - 1$ with $p \geqslant q > 0$, constructs a $(6 \cdot 5^{p-1} + 4 \cdot 5^{q-1} - 1)$-spanner for G with $O(n^{1+1/k})$ edges in $O(\mathrm{MIS}(n))$ time.*

Proof. Size and time are direct consequences of lemmas 3.3 and 3.7 fixing k and $\rho = 1$.

If $p = q$, then $k = 2^{p+1} - 1 = \sum_{j=0}^{p} 2^j$. Hence, $\ell(k) = p$ and $\ell(k - 2^{\ell(k)}) = \ell(2^{p+1} - 1 - 2^p) = \ell(2^p - 1) = p - 1$. Thus, using Lemma 3.5 (second case), the stretch of the output spanner is bounded by $6 \cdot 5^p + 4 \cdot 5^{p-1} - 1$.

If $p \neq q$, then $k = 2^p + \sum_{j=0}^{q-1} 2^j$. Hence, $\ell(k) = p$. In addition, $\ell(k - 2^{\ell(k)}) = \ell(2^p + 2^q - 1 - 2^p) = \ell(2^q - 1) = q - 1$. Thus, using Lemma 3.5 (second case), the stretch of the output spanner is bounded by $6 \cdot 5^p + 4 \cdot 5^{q-1} - 1$. □

Corollary 4.1 *For every integer k such that $k = 2^p + 2^q - 1$, where $p \geqslant q \geqslant 0$, there is a deterministic distributed algorithm that given a graph G with n nodes, constructs a $s[k]$-spanner for G with $O(n^{1+1/k})$ edges in $O(\mathrm{MIS}(n))$ time, where $s[k]$ is given by the Table 1 below.*

(p,q)	$(0,0)$	$(1,0)$	$(1,1)$	$(2,0)$	$(2,1)$	$(2,2)$	$(3,0)$	$(3,1)$	$(3,2)$	$(3,3)$
k	1	2	3	4	5	7	8	9	11	15
$s[k]$	1	5	9	25	33	49	125	153	169	249
i_0	1	1	1	3	2	1	4	3	2	1

Table 1: Stretch and Strategy examples for $k = 2^p + 2^q - 1$.

4.2 Graphs with large minimum degree

It is known that sparser spanners exist whenever the minimum degree increases (cf. the concluding remark of [BKMP05]). In this paragraph, we show that graphs with minimum degree large enough enjoy an $O(1)$-spanner with only $O(n)$ edges, moreover computable with a fast deterministic distributed algorithm.

Let us first note that if a graph G has a ρ-dominating set X, then G has a $(4\rho+1)$-spanner with at most $n + |X|^2/2$ edges. Assuming we are given such a dominating set, the spanner can be constructed distributively in $O(\rho)$ time by first clustering the nodes of the graph around the nodes in the dominating set, and then by connecting every two neighboring clusters. The two endpoints either belong to the same cluster, and thus the endpoints are at distance at most 2ρ in the spanner, or belong to two adjacent clusters. In that case the endpoints are at distance at most $2\rho + 1 + 2\rho$ in the spanner using the selected inter-cluster edge of the spanner.

Proposition 4.1 *For every parameter $\rho \geqslant 1$, there exists a deterministic distributed algorithm that given a graph G with n nodes and a ρ-dominating set X, constructs a $(4\rho+1)$-spanner for G with at most $n + |X|^2/2$ edges in $O(\rho)$ time.*

This proposition can be combined with the observation that if G has minimum degree $\delta \geqslant \sqrt{n \log n}$, then G has a 1-dominating set X of size $O(\sqrt{n \log n}\,)$. Indeed, this can be proved using the following greedy algorithm [Lov75]: one start with $X = \varnothing$ and with the set of all radius-1 balls, $\mathcal{B} = \{N[v] \mid v \in V\}$, where $N[v] = \{u \in V \mid d_G(u,v) \leqslant 1\}$. Then, while $\mathcal{B}$ is nonempty, one select a node $x \in V$ for X that belongs to the maximum number of balls in the current set $\mathcal{B}$. The set $\mathcal{B}$ is updated by removing all balls containing x. The constructed set X is a 1-dominating set and it can be shown that $|X| \leqslant n(1 + \ln n)/\min_{v \in V} |N[v]|$ which is at most $O(\sqrt{n \log n}\,)$ if $\delta \geqslant \sqrt{n \log n}$. Thus, the problem is to efficiently compute such 1-dominating set.

Unfortunately, no deterministic distributed implementation of the greedy algorithm faster than $O(|X|)$ is known. A small ρ-dominating set can be computed much more efficiently in $O(\rho \log^* n)$ time by the algorithm of [KP98]. Unfortunately, its guaranteed size for X is only of $O(n/\rho)$. Finally, no algorithm is known to run in $o(\sqrt{n \log n}\,)$ time for this problem.

However, using our algorithm, we obtain a 9-spanner with only $O(n)$ edges, moreover with a better time complexity.

Theorem 4.3 *There exists a deterministic distributed algorithm that given a graph G with n nodes and minimum degree $\delta \geqslant \sqrt{n}$, constructs a 9-spanner for G with at most $3n/2$ edges in $O(\mathrm{MIS}(n))$ time.*

Proof. The algorithm consists in two stages. First, we construct an MIS for G^2. Then, each node of the MIS constructs its region using the coloring technique of Spanner. The spanner is obtained by considering the edges spanning the regions and the edges connecting every two adjacent regions (cf. Proposition 4.1).

The number of nodes belonging to the MIS, and thus the number of regions, is at most $n/\delta \leqslant \sqrt{n}$. Therefore, the number of edges of the spanner is at most $n + \sqrt{n}^2/2 = 3n/2$. The stretch bound is obtained using the same arguments than Lemma 3.1. □

4.3 Randomized Distributed Implementation Issues

In [Lub86], Luby gives a simple and efficient randomized PRAM algorithm for computing an MIS in $O(\log n)$ expected time. Luby's algorithm can be turned to run in the free model, and we obtain a distributed algorithm for computing an independent 1-dominating set which terminates within $O(\log n)$ expected time. We remark that upon termination of the algorithm, the constructed 1-dominating set is always correct, the randomization is only on the running time, i.e., it is a *Las Vegas* algorithm.

The two randomized algorithms we present below guarantee the stretch and the size bounds for the constructed spanners, while the $O(k)$ time (*Monte Carlo*) randomized algorithms [BS03] do not give any guarantee on the spanner size. This is of course achieved at the price of increasing the stretch factor of the spanner.

Thus, we obtain the following randomized version of Theorem 4.1 and Theorem 4.2:

Theorem 4.4 *There is a (*Las Vegas*) randomized distributed algorithm that given a graph G with n nodes and any fixed integer $k = 2^p$ with $p \geqslant 0$, constructs a $(k^{\log_2 5})$-spanner for G with $O(n^{1+1/k})$ edges in $O(\log n)$ expected time.*

Theorem 4.5 *For every fixed integer $k \geqslant 3$, there is a (*Las Vegas*) randomized distributed algorithm that given a graph G with n nodes and any fixed integer $k = 2^p + 2^q - 1$ with $p \geqslant q > 0$, constructs a $(6 \cdot 5^{p-1} + 4 \cdot 5^{q-1} - 1)$-spanner for G with $O(n^{1+1/k})$ edges in $O(\log n)$ expected time.*

Recently, in [KMW06], Khun et al. show that every packing problem can be approximated by a constant factor with high probability in $O(\log n)$ time in the free model. Therefore, the (*Monte Carlo*) algorithm of [KMW06] implies a randomized constant approximation algorithm for the minimum 1-dominating set problem with $O(\log n)$ time. Thus, using Proposition 4.1, we obtain the following result (to be compared with Theorem 4.3 and [BS03]):

Theorem 4.6 *There exists a (*Monte Carlo*) randomized distributed algorithm that given a graph G with n nodes of minimum degree $\delta \geqslant \sqrt{n}$, constructs a 5-spanner for G in $O(\log n)$ time. The size is $O(n \log^2 n)$ edges with high probability. More generally, for a minimum degree δ graph, we obtain a 5-spanner with $O(n + (n \log n/\delta)^2)$ edges.*

Let us remark that, in Theorem 4.6, 5 is the best possible bound on the stretch if $\delta \geqslant w(n^{1/4} \log n)$. In fact, there exist graphs with minimum degree $c\sqrt{n}$ (for some constant $c > 0$) and girth 6 (the length of its smallest cycle). Thus, the deletion of any edge implies a stretch of at least 5 for its endpoints. Therefore, any spanner with size less than $\frac{1}{2}cn\sqrt{n}$ have stretch at least 5, and $O(n + (n \log n/\delta)^2) = o(n\sqrt{n}\,)$ if $\delta \geqslant w(n^{1/4} \log n)$.

5 Conclusion

In this paper we have considered deterministic distributed algorithm to construct low stretch and sparse spanners of unweighted arbitrary graphs. In particular we have shown that 5-spanner with $O(n^{3/2})$ edges can be constructed in $n^{O(1/\sqrt{\log n}\,)}$ time. Let us observe that $\log n < n^{1/\sqrt{\log n}}$ only for $n > 2^{4^2}$. In other words, deterministic distributed $n^{1/\sqrt{\log n}}$ time algorithms might be competitive[1] over randomized $\log n$ time algorithms for distributed system up to $n \leqslant 32656$ processors.

We left open the two following problems:

1. Reduce the stretch from 5 to optimal stretch 3, without increasing the size of the spanner and the running time. More generally, is it possible, for every $k \geqslant 1$, to compute with a deterministic distributed algorithm a $(2k-1)$-spanners of size $O(n^{1+1/k})$ in $O(\mathrm{MIS}(n))$ time?

2. Reduce the time complexity to $o(\mathrm{MIS}(n))$, possibly with some small stretch and size increasings. More precisely, is it possible to compute with a deterministic distributed algorithm a constant stretch spanner with $o(n^2)$ edges in $o(\mathrm{MIS}(n))$ time? Using our approach, it suffices to show that there is a constant ρ for which an independent ρ-dominating set can be computed in $o(\mathrm{MIS}(n))$ time for every graph.

[1] This obviously depends on the constants hidden in the O-notation.

References

[ABCP93] Baruch Awerbuch, Bonnie Berger, Lenore J. Cowen, and David Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood coverss. In 34^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 638–647. IEEE Computer Society Press, November 1993.

[ABCP96] Baruch Awerbuch, Bonnie Berger, Lenore J. Cowen, and David Peleg. Fast distributed network decompositions and covers. *Journal of Parallel and Distributed Computing*, 39:105–114, 1996.

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore J. Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, February 1998.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In 31^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE Computer Society Press, October 1990.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of the Association for Computing Machinery*, 32:804–823, 1985.

[Awe87] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In 19^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 230–240. ACM Press, May 1987.

[BKMP05] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. New constructions of (α, β)-spanners and purely additive spanners. In 16^{th} *Symposium on Discrete Algorithms (SODA)*, pages 672–681. ACM-SIAM, January 2005.

[BS03] Surender Baswana and Sandeep Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In 30^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of Lecture Notes in Computer Science, pages 384–396. Springer, July 2003.

[BS04] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In 15^{th} *Symposium on Discrete Algorithms (SODA)*, pages 271–280. ACM-SIAM, January 2004.

[Coh98] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998.

[Cow01] Lenore J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001.

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[DMZ06] Bilel Derbel, Mohamed Mosbah, and Akka Zemmari. Fast distributed graph partition and application. In 20^{th} *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society Press, April 2006.

[EGP03] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46:97–114, 2003.

[Elk01] Michael Elkin. Computing almost shortest paths. In 20^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 53–62. ACM Press, 2001.

[Elk04a] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In 15^{th} *Symposium on Discrete Algorithms (SODA)*, pages 359–368. ACM-SIAM, January 2004.

[Elk04b] Michael Elkin. Unconditional lower bounds on the time-approximation trade-offs for the distributed minimum spanning tree problems. In 36^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 331–340. ACM Press, May 2004.

[EP04] Michael Elkin and David Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[EZ04] Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models. In 23^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 160–168. ACM Press, July 2004.

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennès, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004.

[GPS88] Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[KMNW05] Fabian Kuhn, Thomas Moscibroda, Tim Nieberg, and Roger Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In 19^{th} *International Symposium on Distributed Computing (DISC)*, volume Lecture Notes in Computer Science. Springer, September 2005.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In 23^{rd} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 300–309. ACM Press, July 2004.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In 17^{th} *Symposium on Discrete Algorithms (SODA)*, pages 980–989. ACM-SIAM, January 2006.

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998.

[Lin87] Nathan Linial. Distributive graph algorithms - Global solutions from local data. In 28^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 331–335. IEEE Computer Society Press, October 1987.

[Lin92] Nathan Linial. Locality in distributed graphs algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[Lov75] Laszlo Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[LPSP01] Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. In 20^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 63–71. ACM Press, 2001.

[LPSPP05] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM Journal on Discrete Mathematics*, 35(1):120–131, 2005.

[Lub86] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, November 1986.

[MS00] Shlomo Moran and Sagi Snir. Simple and efficient network decomposition and synchronization. *Theoretical Computer Science*, 243(1-2):217–241, 2000.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[PR00] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing*, 30(5):1427–1442, 2000.

[PS96] Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

[PU89a] David Peleg and Jeffrey D. Ullman. An optimal synchornizer for the hypercube. *SIAM Journal on Computing*, 18(4):740–747, 1989.

[PU89b] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, July 1989.

[PV04] Lucia Draque Penso and C. Barbosa Valmir. A distributed algorithm to find k-dominating sets. *Discrete Applied Mathematics*, 141(1-3):243–253, May 2004.

[RTZ02] Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In 13^{th} *Symposium on Discrete Algorithms (SODA)*, pages 844–851. ACM-SIAM, January 2002.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In 32^{nd} *International Colloquium on Automata, Languages and Programming (ICALP)*, volume Lecture Notes in Computer Science, 2005.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In 13^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10. ACM Press, July 2001.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, January 2005.

[Wen91] Rephael Wenger. Extremal graphs with no C^4's, C^6's, or C^{10}'s. *Journal of Combinatorial Theory, Series B*, 52(1):113–116, 1991.