Deterministic Distributed Construction of Linear Stretch Spanners in Polylogarithmic Time

Bilel Derbel[1], Cyril Gavoille[2,⋆], and David Peleg[3,⋆⋆]

[1] Laboratoire d'Informatique Fondamentale (LIF),
Université de Provence Aix-Marseille 1, France
derbel@cmi.univ-mrs.fr
[2] Laboratoire Bordelais de Recherche en Informatique (LaBRI),
Université de Bordeaux, France
gavoille@labri.fr
[3] Department of Computer Science and Applied Mathematics,
The Weizmann Institute, Rehovot, Israel
david.peleg@weizmann.ac.il

Abstract. The paper presents a deterministic distributed algorithm that given an n node unweighted graph constructs an $O(n^{3/2})$ edge 3-spanner for it in $O(\log n)$ time. This algorithm is then extended into a deterministic algorithm for computing an $O(k\, n^{1+1/k})$ edge $O(k)$-spanner in $2^{O(k)} \log^{k-1} n$ time for every integer parameter $k \geqslant 1$. This establishes that the problem of the deterministic construction of a linear (in k) stretch spanner with few edges can be solved in the distributed setting in polylogarithmic time.

The paper also investigates the distributed construction of sparse spanners with almost pure additive stretch $(1+\epsilon, \beta)$, i.e., such that the distance in the spanner is at most $1+\epsilon$ times the original distance plus β. It is shown, for every $\epsilon > 0$, that in $O(\epsilon^{-1} \log n)$ time one can deterministically construct a spanner with $O(n^{3/2})$ edges that is both a 3-spanner and a $(1+\epsilon, 8\log n)$-spanner. Furthermore, it is shown that in $n^{O(1/\sqrt{\log n})} + O(1/\epsilon)$ time one can deterministically construct a spanner with $O(n^{3/2})$ edges which is both a 3-spanner and a $(1+\epsilon, 4)$-spanner. This algorithm can be transformed into a Las Vegas randomized algorithm with guarantees on the stretch and time, running in $O(\epsilon^{-1} + \log n)$ expected time.

Keywords: distributed algorithms, graph spanners, time complexity.

1 Introduction

Background: The purpose of this paper is to study the *locality properties* of *graph spanners*, and particularly, of efficient *deterministic distributed* construction methods for spanners. Graph spanners are a fundamental graph structure

⋆ Supported by the ANR-project "GRAAL", and the équipe-projet INRIA "CÉPAGE".

⋆⋆ Supported in part by grants from the Israel Science Foundation and the Minerva Foundation.

A. Pelc (Ed.): DISC 2007, LNCS 4731, pp. 179–192, 2007.

which can be thought of intuitively as a generalization of the concept of spanning trees. We say that H is an (α, β)-*spanner* of a graph G if H is a spanning subgraph of G and $d_H(u, v) \leqslant \alpha \cdot d_G(u, v) + \beta$ for all nodes u, v of G, where $d_X(u, v)$ denotes the distance from u to v in the graph X. A pair (α, β) for which H is an (α, β)-spanner is called *stretch* of H, and the *size* of H is the number of its edges. An $(\alpha, 0)$-spanner is also referred to as an α-spanner. The quality of a spanner is measured by the trade-off between its stretch and size.

The locality level of constructing a graph spanner can be measured by the time needed to construct such a spanner. In the distributed setting, the best a node can do in t time units is to collect information from its t neighborhood. Hence the time complexity of a distributed algorithm for a given problem can be related to the amount of information needed to solve the problem. Many fundamental problems such as maximal independent set (MIS), coloring, and sparse covers and decompositions, have been studied from the locality point of view in the past. In general, such problems appear to have time efficient distributed algorithms in the randomized setting or for some restricted families of graphs. By "efficient" we mean algorithms breaking the polylogarithmic time barrier. However, no deterministic distributed algorithms having a polylogarithmic running time for every graph are known for any of these problems. The main difficulty in solving such problems is to break the symmetry in a distributed and efficient way when making decisions. While randomization helps to achieve the goal of symmetry breaking, trying to do it by a deterministic method leads to nontrivial combinatorial and algorithmic problems, essentially due to the local nature of distributed computations. Deterministic construction of graph spanners is also a typical problem where breaking the symmetry appears as the major problem for finding fast algorithms. In this paper we overcome this difficulty, showing that near-optimal high quality graph spanners can be constructed in polylogarithmic time. Our algorithm is based on breaking the symmetry using independent dominating sets and on a new sequential construction of spanners exploiting some particular stretch-size properties for bipartite graphs.

Constructing spanners efficiently is also of interest from a practical point of view, since such structures are often used in many applications. In fact, graph spanners are in the basis of various applications in distributed systems (cf. [23]). For instance, the relationship between the quality of spanners and the time and message complexity of network synchronizers is established in [24] (see also [1,21]). Spanners are also implicitly used for the design of low stretch routing schemes with compact tables [11,16,25,27,29], 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 [8,20,28,30,10].

Related Work: We consider unweighted connected graphs with n nodes. Sparse and low stretch spanners can be constructed from the sparse partitions and covers of [6] or the (d, c)-decompositions of [5], which give 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 [2,3,4,22]. 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 at least $4k-3$. Better stretch-size trade-offs exist but with an increasing time complexity. More 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 [13]. The latter stretch-size trade-off is optimal since, according to an Erdös Conjecture verified for $k = 1, 2, 3, 5$ [32], 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 (α, β)-spanner requires $\Omega(n^{1+1/k})$ edges if $\alpha + \beta < 2k+1$.

More time efficient algorithms were given in [12] at the price of slightly increasing the stretch. The algorithm runs in $n^{O(1/\sqrt{\log n})}$ times and provides (α_k, β_k)-spanners with $O(\log k \cdot n^{1+1/k})$ edges where α_k and β_k depend on the positions of the two leading 1's in the binary representation of k and are essentially in order of $k^{\log_2 5}$. In particular, for $k = 2$, the stretch is $(3, 2)$.

Randomized distributed algorithms achieving better performances exist. There is a straightforward (Las Vegas[1]) randomized implementation of the algorithms of [12] that provides $O(k^{\log_2 5})$-spanners of $O(\log k \cdot n^{1+1/k})$ edges in $O(\log n)$ expected time. An algorithm for sparsifying a graph was used in [15] at the bottleneck of constructing small connected dominating sets. This (Monte-Carlo[2]) algorithm constructs, with high probability, a $O(\log n)$-spanner with $O(n)$ edges in $O(\log^3 n)$ time. A (Monte Carlo) algorithm that computes a $(2k-1)$-spanner with expected size $O(k\, n^{1+1/k})$ in $O(k^2)$ time was given in [9].

However, as mentioned in [3], a randomized solution (in particular those coming from Monte Carlo algorithms) 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 are of more than a theoretical interest. Indeed, one cannot just run a randomized distributed algorithm several times to guarantee a good decomposition, since checking the global quality of the spanner in the distributed model is time consuming.

Sequential and distributed algorithms for constructing $(1+\epsilon, \beta)$-spanners were developed in [17,19,18]. The resulting spanner size is $O(\beta n^{1+\delta})$ and the construction time is $O(n^{\delta})$, where $\beta = \beta(\delta, \epsilon)$ is independent of n but grows super-polynomially in δ^{-1} and ϵ^{-1}. Recently, a sequential algorithm based on a randomized sampling technique was given in [31], providing a spanner with $O(k\, n^{1+1/k})$ edges such that the distance d between any two nodes in the original graph is bounded by $d + o(d)$ in the spanner. Pure additive spanners, i.e., spanners whose stretch is on the form $(1, \beta)$, are known only for $k = 2$ and $k = 3$. Sequential algorithms that construct a $(1, 2)$-spanner with $O(n^{3/2})$ edges and a $(1, 6)$-spanner with $O(n^{4/3})$ edges were given respectively in [18] and in [7]. See [26] for further discussions.

Main results: In this paper, we construct in $O(\log n)$ time and for every graph a 3-spanner with $O(n^{3/2})$ edges. This result is generalized to construct in $2^{O(k)}$

[1] The bounds on the stretch and the size are always guaranteed.

[2] There are no deterministic guarantees for the size and the stretch.

$\log^{k-1} n$ time and for every graph a $(4k-3)$-spanner with $O(k\, n^{1+1/k})$ edges for every $k \geqslant 1$. Our construction improves all previous deterministic constructions of low stretch spanners with few edges.

Our algorithms are based on two main ideas. The first idea enables us to achieve the polylogarithmic time complexity. It is based on clustering the graph using any known time-efficient algorithm for constructing an *independent ρ-dominating set*, namely, a set X of pairwise non-adjacent nodes such that every node of the graph is at distance at most ρ from X. The second idea enables us achieve the linear stretch bound with the desired size. It is based on spanning independently in parallel *(i)* the intra-cluster edges using any known sequential algorithm, and *(ii)* the inter-cluster edges in the border of each cluster using a new size-constrained spanner for bipartite graphs. Known algorithms for constructing independent ρ-dominating sets are more time consuming when ρ is small (typically when ρ is a constant). The key point of our fast construction is to use our spanner algorithm for bipartite graphs in order to keep the stretch low and to choose ρ to be any parameter possibly depending on n. In particular, we show that the parameter ρ does not affect the stretch but only the time construction. The construction time of low stretch spanner is then dominated by the construction of an independent ρ-dominating set. Since the fastest known deterministic algorithm for constructing an independent ρ-dominating set is obtained for $\rho = O(\log n)$ and has running time $O(\log n)$, we are able to construct the desired low stretch spanner in polylogarithmic time.

A generic scheme called Generic_Spanner that utilizes these ideas is first described and analyzed in Section 2. Our generic scheme assumes the existence of a sequential algorithm Z_Spanner$_k$ for bipartite graphs that constructs a spanner with some desired constraints on the size and the stretch. For the case $k = 2$, such an algorithm is described and analyzed in detail in Section 3, yielding our first main result: the deterministic construction of an optimal 3-spanner with $O(n^{3/2})$ edges in $O(\log n)$ time (Theorem 1). This result is generalized for any k in Section 4, giving Algorithm Z_Spanner$_k$ which yields our second main result: the deterministic construction of a $(4k-3)$-spanner with $O(k\, n^{1+1/k})$ edges in $2^{O(k)} \log^{k-1} n$ time (Theorem 2).

We also investigate the construction of almost pure additive spanners for $k = 2$. In Section 5, we construct an almost pure additive $(1+\epsilon, \beta)$-spanner with $O(n^{3/2})$ edges for any $\epsilon > 0$. This is obtained by simply adding breadth-first searching (BFS) trees up to some fixed parameter around the dense clusters constructed by Generic_Spanner for $k = 2$. This allows us to reduce the stretch while preserving the size bound and increasing the time complexity by only a small factor. Several bounds on the stretch and the time complexity are then obtained by using either an independent $O(\log n)$-dominating set algorithm or a MIS algorithm. More precisely, we refine our 3-spanner construction to obtain two fast distributed algorithms. The first one runs in $O(\epsilon^{-1} \log n)$ time and provides additive stretch $\beta = 8 \log n$. The second algorithm runs in $n^{O(1/\sqrt{\log n})} + O(\epsilon^{-1})$ time and provides additive stretch $\beta = 4$. The latter algorithm can also

be implemented in $O(\epsilon^{-1} + \log n)$ expected time (using $O(\log n)$ expected time algorithm for MIS) with deterministic stretch and size.

Model and definitions: We assume the classical $\mathcal{LOCAL}$ distributed model of computation (cf. [23], Chapter 2). More precisely, the network is modeled by a connected graph G, whose nodes represent the autonomous computation entities of the network and whose edges represent direct communication links. For simplicity, we assume that communication is synchronous, i.e., there exists a common global clock that generates pulses. At each pulse, nodes can send and receive messages of unlimited size. We assume that a message which is sent at a given pulse arrives before the beginning of the next pulse. The local computations done by a node are assumed to take negligible time. Define the time complexity of a distributed algorithm to be the worst-case number of pulses from the beginning of the algorithm execution to its termination.

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 $d_G(X,Y) = \min\{d_G(x,y) : x \in X \text{ and } y \in Y\}$.

We associate with each $v \in V$ a *region*, denoted by $R(v)$, which is a set of nodes containing v and inducing a connected subgraph of G. We denote by $R^+(v) = \{u \in V : d_G(u, R(v)) \leqslant 1\}$. Given a set $U \subseteq V$, we denote by $\Gamma_G(U)$ the neighborhood of the set U, i.e., $\Gamma_G(U) = \{u \in V : d_G(u,U) = 1\}$. Note that $\Gamma_G(R(v)) = R^+(v) \setminus R(v)$.

Given a region $R(v)$, the *surrounding graph of* $R(v)$ is the graph B_v induced by the edges $\{x,y\} \in E(G)$ such that $x \in R(v)$ and $y \in \Gamma_G(R(v))$. Informally speaking, B_v is the collection of the outgoing edges of $R(v)$, namely, the edges having one of their end-points in $R(v)$ and the other one outside $R(v)$. One can easily see that B_v is a collection of connected bipartite subgraphs of G lying at the frontier of $R(v)$.

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 tree rooted at v and spanning X. We denote by $\mathrm{IDS}(G,\rho)$ (respectively, $\mathrm{MIS}(G)$) any independent ρ-dominating set (resp., maximal independent set) of G. One can check that a set is an $\mathrm{MIS}(G)$ if and only if it is an $\mathrm{IDS}(G,1)$. We denote by $\mathrm{IDS}(n,\rho)$ the time complexity needed to construct an independent ρ-dominating set on a graph of n nodes. We note that $\mathrm{MIS}(n) = IDS(n,1)$.

Due to lack of space, the proofs of our lemmas and theorems are omitted and will appear in the full version of the paper.

2 A General Scheme

In this section we give a high level description of Algorithm Generic_Spanner (see Fig 1). It uses two sub-routines:

- Seq_Spanner$_k$: can be any algorithm that given an n-node graph and a parameter $k > 0$ constructs a $s(k)$-spanner with $O(k\, n^{1+1/k})$ edges.

- Z_Spanner$_k$: can be any algorithm that given a bipartite graph $B = (W \cup V, E)$ and a parameter $k > 0$ constructs a $z(k)$-spanner with $O(|V \cup W| + |W| \cdot |V \cup W|^{1/k})$ edges.

These two algorithms are executed by only some nodes *locally* and in a *non-interfering manner*. Thus, we can use any two possibly sequential algorithm without affecting the distributed time complexity of the overall algorithm.

Many algorithms are known for the first of the latter tasks, namely, providing spanners with $O(k\, n^{1+1/k})$ edges and stretch $s(k) = 2k - 1$ for any graph. In contrast, no trivial constructions are known for the second task, of providing spanners with both a low stretch $z(k)$ and the constrained size $O(|V \cup W| + |W| \cdot |V \cup W|^{1/k})$ for bipartite graphs. Solving this latter task will be the aim of sections 3 and 4. In the rest of this section, we simply assume the existence of such a construction and focus on the properties of the general scheme defined by Generic_Spanner.

2.1 Description of the Generic Scheme

Algorithm Generic_Spanner (Fig. 1) is based on clustering the dense regions of the graph and spanning the edges of these regions efficiently. The algorithm works in at most k iterations, each of six steps. Step 1 computes the set L of light nodes, that is, the nodes whose corresponding regions have a sparse neighborhood. Step 2 considers (in parallel) each light region $R(v)$ and its surrounding graph B_v. Fig. 2 gives an idea of how a region $R(v)$ and the graph B_v may look like.Each connected component of B_v is a bipartite subgraph having one set of nodes on the border of $R(v)$ and the other set outside $R(v)$. The intra-region edges are

Input: a graph $G = (V, E)$ with $n = |V|$, and integers $\rho, k \geqslant 1$.
Output: a spanner S of G with stretch $\max\{z(k), s(k)\}$ and size $O(k\, n^{1+1/k})$.

Set $U := V$; $r = 0$; $S := \varnothing$; $\forall v \in V$, $R(v) := \{v\}$ and $c(v) := v$;
For $i := 1$ **to** k **do:**
Span light regions
1. $L := \{v \in U : |R^+(v)| \leqslant n^{i/k}\}$;
2. **For all** $v \in L$ **do (in parallel):**
(a) Let B_v be the surrounding graph $R(v)$;
(b) $S := S \cup \text{Seq\_Spanner}_k(G[R(v)]) \cup \text{Z\_Spanner}_k(B_v)$;
Form new dense regions
3. $X := \text{IDS}(G^{2(r+1)}[U \setminus L], \rho)$;
4. $\forall z \in U$, if $d_G(z, X) \leqslant (2\rho+1)r+2\rho$, then set $c(z)$ to be the closest node of X, breaking ties with identities;
5. $\forall v \in X$, $R(v) := \{z \in V : c(z) = v\}$;
6. $U := X$ and $r := (2\rho + 1)r + 2\rho$;

Fig. 1. Algorithm Generic_Spanner

using Algorithm SEQ_SPANNER$_k$ whereas the inter-region edges (those, of the surrounding graph B_v) are spanned using Algorithm Z_SPANNER$_k$. This step can be performed efficiently by collecting a copy of $R^+(v)$ in v that computes the result and then broadcasts this information.

Algorithm Z_SPANNER$_k$ spans inter-region edges using paths zigzagging from the border to the outside of a region, thus avoiding to use long paths going from the border to the center v. The intra-region edges are spanned using any distributed or sequential algorithm in order to guarantee the best possible stretch-size trade-offs inside a region. The radius of constructed regions (which depends only on parameter ρ) will affect only the time complexity but not the stretch of the obtained spanner. This observation will enable us to use a fast IDS algorithm without constraining ρ to be too small (typically, by taking ρ order of $\log n$).

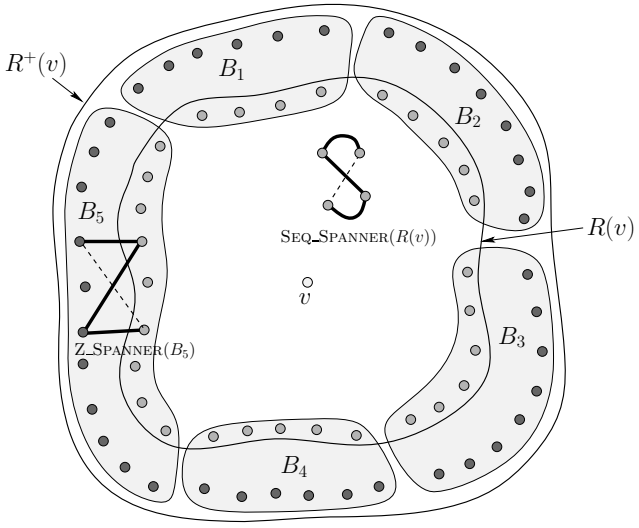

Fig. 2. A region $R(v)$ and its surrounding graph $B_v = \bigcup_{j\in[1,5]} B_j$.

After Step 2, only the neighborhood of sparse regions are spanned. In the other steps, the remaining dense regions are processed the in order to merge them together. The goal here is to grow the dense regions until they become sparse. Thus, we would be able to span them without adding too many edges.

In fact, in Steps 3, 4, and 5, we construct new dense regions centered around some well chosen dense nodes. First, we construct an independent ρ-dominating set X of the graph $G^{2(r+1)}[U \setminus L]$ where r is an upper bound of the radius of any region and U is the set of remaining dense nodes. Then, using a classical consistent coloring mechanism (cf. [23], Chapter 22, Lemma 22.1.2), all dense regions are merged into new regions having the nodes of the IDS as their centers (note that a light region might get merged with a dense one). The merging process guarantees that the new regions are disjoint and connected, and their

radius grows up by at most a multiplicative factor $O(\rho)$. In addition, one should remark that by considering the $2(r+1)$ power of G, it is guaranteed that the neighborhood of the regions induced by the set X are disjoint. Thus, each new formed region contains at least its neighborhood. This observation is essential to obtain the desired size for our spanner.

In Step 6, the set of dense regions is updated for the next iteration. On iteration k the sparsity condition of Step 1 is always true, hence all the regions are light, which guarantees that all nodes are spanned.

2.2 Analysis of the Algorithm

For every phase i, denote by L_i (resp. X_i) the set L (resp. X) computed during phase i, i.e., after Steps 1 and 3 of phase i. Similarly, denote by $c_i(z)$ the color of z assigned during phase i, i.e., after Step 4 of phase i. Denote by U_i the set U at the beginning of phase i, and let r_i denote the value of r at the beginning of phase i. Observe that $U_i = X_{i-1}$ for every $i > 1$. For a node $v \in U_i$, denote by $R_i(v)$ the region of v at the beginning of phase i. The following lemma is easily proved by induction relying on the description of the algorithm.

Lemma 1. *For every phase $i > 0$, and for every $v \in U_i$, $|R_i(v)| \geqslant n^{(i-1)/k}$, and if $v \in X_i$, then $R_i^+(v) \subseteq R_{i+1}(v)$.*

Inspired by the proofs of lemmas 1 to 4 in [12], one can prove the following:

Lemma 2. *in Algorithm* Generic_Spanner*, the following holds:*

- *For every phase i and for every $v \in U_i$, r_i is the eccentricity of v in $G[R_i(v)]$.*
- *For every phase i and for all nodes $u \neq v \in U_i$, $R_i(u) \cap R_i(v) = \varnothing$.*
- *For every node $u \in V$, there exists a phase i and a node v such that $u \in R_i(v)$ and $v \in L_i \cap U_i$.*

The following two main lemmas are used in our construction.

Lemma 3. *The output S of Algorithm* Generic_Spanner *is a $\max\{z(k), s(k)\}$-spanner of G with at most $O(k\, n^{1+1/k})$ edges.*

Lemma 4. *Algorithm* Generic_Spanner *can be implemented in the distributed $\mathcal{LOCAL}$ model in $O((2\rho+1)^{k-2} \cdot \mathrm{IDS}(n,\rho) + (2\rho+1)^{k-1})$ time.*

3 3-Spanner Construction

In order to apply Lemma 3, it is necessary to provide an algorithm that given a surrounding graph B_v of some region $R(v)$ constructs a spanner with the desired constraints on the stretch and the size. Since any surrounding graph B_v is a collection of connected bipartite components, it is clear that the desired stretch and size spanner can be obtained by providing an algorithm with these

properties for bipartite graphs and then applying that algorithm in parallel for each connected component.

In this section we describe in detail a Z_Spanner algorithm providing the desired properties for a bipartite graph. More precisely, we prove the following more powerful result.

Lemma 5. *Every connected bipartite graph $B = (W \cup V, E)$ has a spanner S with at most $O(|V|+|W|\cdot\sqrt{|V|})$ edges satisfying the following stretch properties:*

$$\forall v, w \in V \cup W, \quad d_S(v,w) \leqslant \begin{cases} 2 \cdot d_B(v,w) + 1 & \text{if } d_B(v,w) \text{ is odd} \\ 2 \cdot d_B(v,w) + 2 & \text{otherwise.} \end{cases}$$

Let us remark that for $k = 2$ in Generic_Spanner, Lemma 5 leads to the construction of a 3-spanner. In [23], it is shown how to construct a $(2\log n, 3)$-ruling set in $O(\log n)$ deterministic time. A (ρ, s)-ruling set with $s > 1$ is in particular an independent ρ-dominating set. Thus, one can construct an independent $2\log n$-dominating set deterministically in $O(\log n)$ time. Hence, for $k = 2$ and $\rho = 2\log n$ in Generic_Spanner, we obtain:

Theorem 1. *There exists a distributed algorithm that given an n-node graph constructs a 3-spanner with $O(n^{3/2})$ edges in $O(\log n)$ deterministic time.*

In the rest of this section, we give a Z_Spanner$_2$ algorithm providing the properties claimed in Lemma 5.

3.1 Description of Z_Spanner$_2$

Algorithm Z_Spanner$_2$ with input a bipartite graph B is based on a sequential greedy technique (see Fig. 3):

Input: a connected bipartite graph $B = (W \cup V, E)$.
Output: a 3-spanner S of B with $O(|V| + |W| \cdot \sqrt{|V|})$ edges.

Set $V_1 := V$; $W_1 := W$; $B_1 := B$; $S := \varnothing$; $i := 1$;
While $W_i \neq \varnothing$ **do:**
1. Let $w_i \in W_i$ with the highest degree in B_i, breaking ties arbitrary.
2. $N_i := \Gamma_{B_i}(\{w_i\})$.
3. $S := S \cup B_i[N_i]$.
4. **For every** $j < i$ **such that** $w_i \in \Gamma_B(N_j)$ **do:**
   (a) Let e_{j,w_i} be an edge in E connecting N_j to w_i.
   (b) $S := S \cup \{e_{j,w_i}\}$.
5. Construct the new graph B_{i+1}:
   (a) $V_{i+1} := V_i \setminus N_i$ and $W_{i+1} := W_i \setminus \{w_i\}$.
   (b) $B_{i+1} := B_i[V_{i+1} \cup W_{i+1}]$.
6. $i := i + 1$;

Fig. 3. Algorithm Z_Spanner$_2$

In each iteration $i \in \{1, \ldots, |W|\}$, a new graph $B_i = (W_i \cup V_i, E_i)$ is considered on the basis of the graph B_{i-1} corresponding to the previous iteration ($B_1 := B$). Some edges of B are added to the spanner S as follows: Select a node $w_i \in W_i$ with the highest degree in B_i, and add to S its neighborhood N_i in B_i (with its incident edges). Then, if w_i is connected in the original graph B to N_j with $j < i$, a set computed at some previous step, an edge e_{i,w_j} connecting w_i to N_j is added to S. (See Fig. 4).

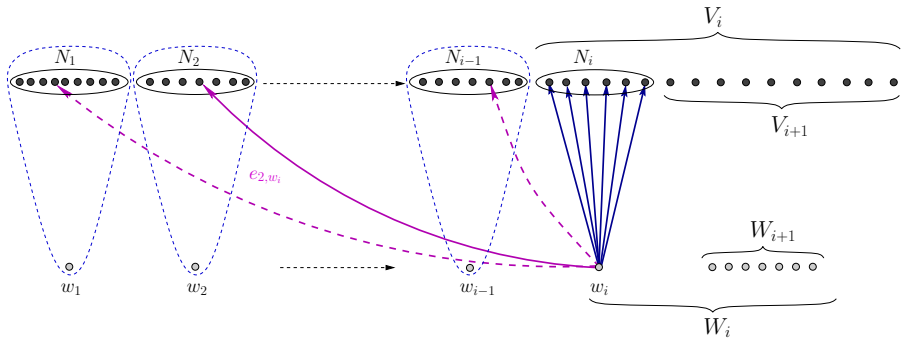

Fig. 4. The i-th iteration Algorithm Z_Spanner$_2$

3.2 Analysis of Z_Spanner$_2$

Lemma 6. *Let u, v, w be three nodes of a bipartite graph $B = (W \cup V, E)$ such that $u \in V$, $\{u, v\} \in E$ and $\{u, w\} \in E$. The output spanner S of Algorithm* Z_Spanner$_2(B)$ *satisfies $d_S(u, v) \leqslant 3$ and $d_S(v, w) \leqslant 4$.*

Lemma 7. *For any bipartite graph $B = (W \cup V, E)$, the output spanner of* Z_Spanner$_2(B)$ *has $O(|V| + |W| \cdot \sqrt{|V|})$ edges.*

Using the previous lemmas, we are able to prove the stretch and size bounds as stated in Lemma 5.

4 $O(k)$-Spanner Construction

Algorithm Z_Spanner$_2$ can be extended for $k > 2$. The extended algorithm, Z_Spanner$_k$, is given in Fig. 5. Given a bipartite graph $B = (W \cup V, E)$, we carefully construct a partial partition of B containing clusters of small radius. More precisely, at each iteration, a cluster is grown in a layered fashion around some node $w \in W$ until the cluster becomes sparse. Once a cluster is constructed, the neighborhood of the cluster is spanned by a BFS tree and the cluster is removed from the graph B. The algorithm terminates when all the nodes of W are clustered. We remark that at the end of our algorithm, some nodes in V may remain uncovered by the clustering, however each node in W belongs to a cluster.

The originality of our algorithm comparing with classical sparse partition (or covers) algorithms is to compute the sparsity of a layer depending on the range of

the layer. Let C be a cluster being constructed in the while loop of Z_Spanner$_k$. Suppose that C contains i successive layers $\mathcal{L}_0, \ldots, \mathcal{L}_{i-1}$. Then, consider the new layer $\mathcal{L}_i$ to be processed. Since B is bipartite, then either $\mathcal{L}_i \subseteq V$ (if i is odd) or $\mathcal{L}_i \subseteq W$ (if i is even). In the first case, we add $\mathcal{L}_i$ to the cluster C if it is dense enough comparing with layer $\mathcal{L}_{i-1}$. In the second case, we add $\mathcal{L}_i$ to the cluster C if it is dense enough comparing with layer $\mathcal{L}_{i-2}$. The main observation that will guarantee the desired size is that in the two cases layers $\mathcal{L}_{i-1}$ or $\mathcal{L}_{i-2}$ belong to W.

Input: a connected bipartite graph $B = (W \cup V, E)$ and an integer $k > 0$.
Output: a $(4k-3)$-spanner S of B with $O(|V \cup W| + |W| \cdot |V \cup W|^{1/k})$ edges.

```
S := ∅;
while W ≠ ∅ do
    pick a node v ∈ W;
    C := {v}, L_0 := {v} and i := 1;
    dense := True;
    while dense do
        L_i := Γ_B(C);
        j := i − 2 + (i mod 2);
        if |L_i| > |V ∪ W|^{1/k} · |L_j| then
            C := C ∪ L_i;
            i := i + 1;
        else
            dense := False;
    S := S ∪ BFS(v, C ∪ L_i);
    W := W \ C and V := V \ C;
```

Fig. 5. Algorithm Z_Spanner$_k$

By analyzing Algorithm Z_Spanner$_k$, we can show that:

Lemma 8. *Let $k \geqslant 1$. Every connected bipartite graph $B = (V \cup W, E)$, has a $(4k-3)$-spanner with $O(|V \cup W| + |W| \cdot |V \cup W|^{1/k})$ edges.*

Combining Lemma 3 and 4 for $\rho = 2 \log n$, we obtain:

Theorem 2. *There exists a distributed algorithm that given an n-node graph and an integer $k \geqslant 1$, constructs a $(4k-3)$-spanner for it with $O(k\, n^{1+1/k})$ edges in $2^{O(k)} \log^{k-1} n$ deterministic time.*

5 Improving the Stretch for $k = 2$

It is shown in [14,18] that every graph has a $(1,2)$-spanner with $O(n^{3/2})$ edges. Nevertheless, no fast distributed construction of such a spanner is known. In this section, we give fast distributed constructions that enable us to obtain 3-spanners of size $O(n^{3/2})$ which are also almost pure additive spanners. More

precisely, the multiplicative component on the stretch is $(1+\epsilon)$ and the additive component is independent of ϵ but depends on the time complexity.

Our construction works in two stages. In the first stage, we run Algorithm GENERIC_SPANNER with parameter $k = 2$ and we obtain a spanner S_1 and a set of dense nodes X. The set X here denotes the set of nodes computed by the first iteration of GENERIC_SPANNER, i.e., $X = X_1$. In the second stage, we add to S_1 a BFS tree up to a depth $2\rho + \beta$ rooted at each node $v \in X$ (β is a given parameter).

By setting $\rho = 2\log n$ and $\beta = \Theta(\epsilon^{-1}\log n)$ with $\epsilon > 0$, and using the $O(\log n)$ time deterministic algorithm for independent $(2\log n)$-dominating sets, one can prove:

Lemma 9. *There exists a distributed algorithm that given an n-node graph G and a parameter $\epsilon > 0$, constructs in $O(\epsilon^{-1}\log n)$ deterministic time a 3-spanner S with $O(n^{3/2})$ edges and satisfying the following stretch properties:*

$$\forall u, v \in V, \quad d_S(u,v) \leqslant \begin{cases} d_G(u,v) + 8\log n & \text{if } d_G(u,v) \leqslant 8\epsilon^{-1}\log n \\ (1+\epsilon)\, d_G(u,v) + 8\log n & \text{otherwise.} \end{cases}$$

Note that if the distance to be approximated is $d = \omega(\log n)$, then the distance in the spanner is at most $(1+\epsilon)\cdot d + o(d)$. Also, by choosing $\epsilon = o(1)$, Lemma 9 implies the construction in $\log^{1+o(1)} n$ time of a 3-spanner with $O(n^{3/2})$ edges which is also a $(1+o(1), 8\log n)$-spanner.

In order to obtain a better additive stretch, we use an MIS algorithm at the price of increasing the time complexity. In fact, it is also well-known that a MIS can be constructed by a deterministic (resp. randomized) algorithm in $n^{O(1/\sqrt{\log n})}$ (resp. $O(\log n)$ expected) time. Thus by taking $\rho = 1$ and $\beta = \Theta(\mathrm{MIS}(n))$, one can prove:

Lemma 10. *There exists a distributed algorithm that given an n-node graph G, constructs in $n^{O(1/\sqrt{\log n})}$ deterministic time a spanner S with $O(n^{3/2})$ edges and stretch (α, β) as given by Table 1.*

Table 1. Stretches (α, β) for distances d

$d_G(u,v)$	$d_S(u,v)$	(α,β)
1	3	$(2,1)$
2	6	$(2,2)$
3	7	$(2,1)$
4	8	$(2,0)$
5	9	$(1.8,0)$
$d > n^{O(1/\sqrt{\log n})}$	$(1+o(1))\ d$	$(1+o(1),0)$
$d \leqslant n^{O(1/\sqrt{\log n})}$	$d+4$	$(1,4)$

We observe that the spanner constructed in Lemma 10 has stretch at most $(2, 1)$ except for nodes at distance 2.

Combining previous lemmas we obtain:

Theorem 3. *There exists a distributed algorithm that given an n-node graph G and a parameter $\epsilon > 0$ constructs in $n^{O(1/\sqrt{\log n})} + O(\epsilon^{-1})$ (resp. $O(\epsilon^{-1} \log n)$) deterministic time a 3-spanner with $O(n^{3/2})$ edges which is also a $(1 + \epsilon, 4)$-spanner (resp. a $(1 + \epsilon, 8 \log n)$-spanner).*

6 Open Problems

While it is well-known that every graph has a $(1, 2)$-spanner with $O(n^{3/2})$ edges, we leave open the problem to find (distributively or not), for each $k > 2$, a $(1, f(k))$-spanner of size $O(n^{1+1/k})$ where f is a polynomial (or even an exponential) function.

References

1. Awerbuch, B.: Complexity of network synchronization. J. ACM 32, 804–823 (1985)
2. Awerbuch, B., Berger, B., Cowen, L.J., Peleg, D.: Near-linear cost sequential and distributed constructions of sparse neighborhood coverss. In: 34th IEEE Symp. on Foundations of Computer Science, pp. 638–647. IEEE Computer Society Press, Los Alamitos (1993)
3. Awerbuch, B., Berger, B., Cowen, L.J., Peleg, D.: Fast distributed network decompositions and covers. J. Parallel and Distributed Computing 39, 105–114 (1996)
4. Awerbuch, B., Berger, B., Cowen, L.J., Peleg, D.: Near-linear time construction of sparse neighbourhood covers. SIAM J. on Computing 28, 263–277 (1998)
5. Awerbuch, B., Goldberg, A., Luby, M., Plotkin, S.: Network decomposition and locality in distributed computation. In: Proc. 30th IEEE Symp. on Foundations of Computer Science, pp. 364–369. IEEE Computer Society Press, Los Alamitos (1989)
6. Awerbuch, B., Peleg, D.: Sparse partitions. In: 31th IEEE Symp. on Foundations of Computer Science, pp. 503–513. IEEE Computer Society Press, Los Alamitos (1990)
7. Baswana, S., Kavitha, T., Mehlhorn, K., Pettie, S.: New constructions of (α, β)-spanners and purely additive spanners. In: 16th ACM-SIAM Symp. on Discrete Algorithms, pp. 672–681. ACM Press, New York (2005)
8. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In: 15th ACM-SIAM Symp. on Discrete Algorithms, pp. 271–280. ACM Press, New York (2004)
9. Baswana, S., Sen, S.: A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. Random Structures and Algorithms 30, 532–563 (2007)
10. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. SIAM J. on Computing 28, 210–236 (1998)
11. Cowen, L.J.: Compact routing with minimum stretch. J. Algorithms 38, 170–183 (2001)

12. Derbel, B., Gavoille, C.: Fast deterministic distributed algorithms for sparse spanners. In: Flocchini, P., Gąsieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 100–114. Springer, Heidelberg (2006)
13. Derbel, B., Mosbah, M., Zemmari, A.: Fast distributed graph partition and application. In: 20th IEEE Int. Parallel and Distributed Processing Symp., IEEE Computer Society Press, Los Alamitos (2006)
14. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. SIAM J. Computing 29, 1740–1759 (2000)
15. Dubhashi, D., Mai, A., Panconesi, A., Radhakrishnan, J., Srinivasan, A.: Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. J. of Computer and System Sciences 71, 467–479 (2005)
16. Eilam, T., Gavoille, C., Peleg, D.: Compact routing schemes with low stretch factor. J. Algorithms 46, 97–114 (2003)
17. Elkin, M.: Computing almost shortest paths. In: 20th ACM Symp. on Principles of Distributed Computing, pp. 53–62. ACM Press, New York (2001)
18. Elkin, M., Peleg, D.: $(1 + \epsilon, \beta)$-spanner constructions for general graphs. SIAM J. on Computing 33, 608–631 (2004)
19. Elkin, M., Zhang, J.: Efficient algorithms for constructing $(1 + \epsilon, \beta)$-spanners in the distributed and streaming models. In: 23rd ACM Symp. on Principles of Distributed Computing, pp. 160–168. ACM Press, New York (2004)
20. Gavoille, C., Peleg, D., Pérennès, S., Raz, R.: Distance labeling in graphs. J. Algorithms 53, 85–112 (2004)
21. Moran, S., Snir, S.: Simple and efficient network decomposition and synchronization. Theoretical Computer Science 243, 217–241 (2000)
22. Panconesi, A., Srinivasan, A.: On the complexity of distributed network decomposition. J. Algorithms 20, 356–374 (1996)
23. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Mathematics and Applications (2000)
24. Peleg, D., Ullman, J.D.: An optimal synchornizer for the hypercube. SIAM J. Computing 18, 740–747 (1989)
25. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. ACM 36, 510–530 (1989)
26. Pettie, S.: Low distortion spanners. In: 34^{th} International Colloquium on Automata, Languages and Programming. LNCS, vol. 4596. Springer, Heidelberg (2007)
27. Roditty, L., Thorup, M., Zwick, U.: Roundtrip spanners and roundtrip routing in directed graphs. In: 13th ACM-SIAM Symp. on Discrete Algorithms, pp. 844–851. ACM Press, New York (2002)
28. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 261–272. Springer, Heidelberg (2005)
29. Thorup, M., Zwick, U.: Compact routing schemes. In: 13th ACM Symp. on Parallel Algorithms and Architectures, pp. 1–10. ACM Press, New York (2001)
30. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM 52, 1–24 (2005)
31. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: 17th ACM-SIAM Symp. on Discrete Algorithm, pp. 802–809. ACM Press, New York (2006)
32. Wenger, R.: Extremal graphs with no C^4's, C^6's, or C^{10}'s. J. Combinatorial Theory, Series B 52, 113–116 (1991)