Improved Deterministic Distributed Construction of Spanners[*]

Ofer Grossman[1] and Merav Parter[2]

1 **MIT, CSAIL, Cambridge, USA**
`ofer.grossman@gmail.com`

2 **Weizmann Institute of Science, Rehovot, Israel**
`merav.parter@weizmann.ac.il`

Abstract

Graph spanners are fundamental graph structures with a wide range of applications in distributed networks. We consider a standard synchronous message passing model where in each round $O(\log n)$ bits can be transmitted over every edge (the CONGEST model).

The state of the art of deterministic distributed spanner constructions suffers from large messages. The only exception is the work of Derbel et al. [9], which computes an optimal-sized $(2k-1)$-spanner but uses $O(n^{1-1/k})$ rounds.

In this paper, we significantly improve this bound. We present a deterministic distributed algorithm that given an unweighted n-vertex graph $G=(V,E)$ and a parameter $k>2$, constructs a $(2k-1)$-spanner with $O(k\cdot n^{1+1/k})$ edges within $O(2^k\cdot n^{1/2-1/k})$ rounds for every even k. For odd k, the number of rounds is $O(2^k\cdot n^{1/2-1/(2k)})$. For the weighted case, we provide the first deterministic construction of a 3-spanner with $O(n^{3/2})$ edges that uses $O(\log n)$-size messages and $\widetilde{O}(1)$ rounds. If the vertices have IDs in $[1,\Theta(n)]$, the spanner is computed in only 2 rounds!

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases spanners, clustering, deterministic algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.24

1 Introduction & Related Work

Graph spanners are fundamental graph structures that are used as a key building block in various communication applications, e.g., routing, synchronizers, broadcasting, distance oracles, and shortest path computations. For this reason, the distributed construction of sparse spanners has been studied extensively [2, 5, 6, 7, 8, 9, 15]. The standard setting is a synchronous message passing model where per round each node can send one message to each of its neighbors. Of special interest is the case where the message size is limited to $O(\log n)$ bits, a.k.a. the CONGEST model.

The common objective in distributed computation of spanners is to achieve the best-known existential size-stretch trade-off as fast as possible: It is folklore that for every graph $G=(V,E)$, there exists a $(2k-1)$-spanner $H\subseteq G$ with $O(n^{1+1/k})$ edges. Moreover, this size-stretch tradeoff is believed to be optimal, following the girth conjecture of Erdős.

Designing deterministic algorithms for local problems has been receiving a lot of attention since the foundation of the area in 1980's. Towards the end of this section, we elaborate more on the motivation for studying deterministic algorithms in the distributed setting.

[*] A full version of the paper is available at `https://arxiv.org/abs/1708.01011`.

State of the art for deterministic distributed constructions of spanners. Whereas there are efficient randomized constructions for spanners, as the reader will soon notice, the state of the art for distributed deterministic spanner constructions suffers from large message sizes: Derbel and Gavoille [5] construct constant stretch spanners with $o(n^2)$ edges and $O(n^\epsilon)$ rounds for any constant ϵ, using messages of size $O(n)$. Derbel, Gavoille and Peleg improved this result and presented in [6] a construction of an $O(k)$-spanner with $O(kn^{1+1/k})$ edges in $O(\log^{k-1} n)$ rounds. This was further improved in the seminal work of Derbel, Gavoille, Peleg, and Viennot [7], which provides a deterministic k-round algorithm for constructing $(2k-1)$-spanners with optimal size. However, again the algorithm uses messages of size $O(n)$. Using large messages is indeed inherent to all known efficient deterministic techniques, which are mostly based on network decomposition and graph partitioning. In the conventional approaches of network decomposition, the deterministic algorithms for spanners usually require a vertex to learn the graph topology induced by its $O(1)$-neighborhood. This cannot be done efficiently with small messages.

As Pettie [15] explicitly noted, *all* these constructions have the disadvantage of using large messages. Derbel et al. [8] also pointed out that constructing sparse spanners deterministically with small message sizes remains open.

The state of the art when using small messages. There are only two exceptions for this story. Barenboim et al. [1] showed a construction of $O(\log^{k-1} n)$ spanner with $O(n^{1+1/k})$ edges in $O(\log^{k-1} n)$ rounds. Hence, whereas the runtime is polylogarithmic, the stretch-size tradeoff of the output spanner is quite far from the optimal one.

We are then left with only *one* previous work that fits our setting, due to Derbel, Mosbah and Zemmari [9]. They provide a deterministic construction of an optimal-size $(2k-1)$-spanner but using $O(n^{1-1/k})$ rounds.

The state of the art in other distributed settings. Turning to *randomized* constructions, perhaps one of the most well known approaches to construct a spanner is given by Baswana and Sen [2], which we review soon. Recently, [3] showed that the Baswana-Sen algorithm can be derandomized in the congested clique model of communication in which every pair of nodes (even non-neighbors in the input graph) can exchange $O(\log n)$ bits per round. Note that this model is much stronger than the standard model in which only *neighboring* vertices can communicate. Indeed the algorithm of [3] requires a global evaluation of the random seed, thus implementing this algorithm in the standard CONGEST model requires $\Omega(\text{diam}(G) + n^{1-1/k})$ rounds where $\text{diam}(G)$ is the diameter of the graph. Hence, deterministic construction of spanners in the CONGEST model calls for new ideas!

Before we proceed with introducing our main contribution, we make a short pause to further motivate the study of deterministic distributed algorithms.

A note on deterministic distributed algorithms. Much effort has been invested in designing deterministic distributed algorithms for local problems. Examples include MIS (maximal independent set), vertex coloring, edge coloring and and matching. Until recently, a deterministic poly-log n round algorithm was known only for the maximal matching problem (see [13] and a recent improvement by [10]). In a recent breakthrough [11], a polylogarithmic solution was provided also for the $(2\Delta-1)$ edge coloring[1]. Aside from the general theoretical question,

[1] Where Δ is the maximum degree in the graph.

the distributed setting adds additional motivation for studying deterministic algorithms (as nicely noted in [10]). First, in the centralized setting, if the randomized algorithm ends with an error, we can just repeat. In the distributed setting, detecting a global failure requires communicating to a leader, which blows up the runtime by a factor of network diameter. Second, for problems as MIS, [4] showed that improving the randomized complexity requires an improvement in the deterministic complexity.

Wheres most results for deterministic local problems are for the LOCAL model, which allows unbounded messages; the size of messages that are sent throughout the computation is a second major attribute of distributed algorithms. It is therefore crucial to study the complexity of local problems under bandwidth restrictions. Surprisingly, most of the algorithms for local problems already use only small messages. The problem of spanners is distinguished from these problems, and in fact, spanners is the only setting we are aware of, in which all existing deterministic algorithms use *large* messages. Hence, the main challenge here is in the *combination* of **deterministic** algorithms with **congestion** constraints.

1.1 Our Contribution

Our main result is:

▶ **Theorem 1.** *For every n-vertex unweighted graph $G = (V, E)$ and even k, there exists a deterministic distributed algorithm that constructs a $(2k-1)$-spanner with $O(k \cdot n^{1+1/k})$ edges in $O(2^k \cdot n^{1/2-1/k})$ rounds using $O(\log n)$-size messages*[2].

A key element in our algorithm is the construction of sparser spanners for unbalanced bipartite graph. This construction might become useful in other spanner constructions.

▶ **Lemma 2** (Bipartite Spanners). *Let $G = (A \cup B, E)$ be an unweighted bipartite graph, with $|A| \leq |B|$. For even $k \geq 4$, one can construct (in the* CONGEST *model) a $(2k-1)$ spanner H with $|E(H)| = O(k|A|^{1+2/k} + |B|)$ edges within $O(|A|^{1-2/k})$ rounds*[3].

Turning to weighted graphs, much less in known about the deterministic construction of spanners in the distributed setting. The existing deterministic constructions of optimal-sized $(2k-1)$-spanners (even in the LOCAL model) are restricted to *unweighted* graphs, already for $k = 2$. If the edge weights are bounded by some number W, there is a simple reduction[4] to the unweighted setting, at the cost of increasing the stretch by a factor of $(1+\epsilon)$ and the size of the spanner by a factor of $\log_{1+\epsilon} W$. Hence, already in the LOCAL model and $k = 2$, we only have a $(3+\epsilon)$ spanner with $\widetilde{O}(n^{3/2})$ edges. Whereas our general approach does not support the weighted case directly, our algorithm for 3-spanners does extend for weighted graphs. Hence, we give here the first deterministic construction with nearly tight tradeoff between the size, stretch and *runtime*.

▶ **Theorem 3** (3-Spanners for Weighted Graphs). *For every n-vertex weighted graph $G = (V, E)$, there exists a deterministic distributed algorithm that constructs a 3-spanner with $O(n^{3/2})$ edges in $O(\log n)$ rounds using $O(\log n)$-size messages. If vertices have IDs in the range of $[1, O(n)]$, it can be done in two rounds.*

[2] For odd k, we obtain a similar theorem, but with $O(2^k \cdot n^{1/2-1/(2k)})$ rounds.
[3] Hence, yielding an improved edge bound, for $|A| \leq n^{(k+1)/(k+2)}$.
[4] Apply the algorithm for unweighted graphs separately for every weight scale $((1+\epsilon)^i, (1+\epsilon)^{i+1}]$.

1.2 Our Approach and Key Ideas in a Nutshell

For the sake of discussion, let $k = O(1)$ throughout this section.

A brief description of the randomized construction by Baswana-Sen. A clustering $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ is a collection of vertex disjoint sets which we call clusters. Every cluster has some a special vertex which we call the *cluster center*. In the high level, the Baswana-Sen algorithm computes k levels of clustering $\mathcal{C}_0, \ldots, \mathcal{C}_{k-1}$ where each clustering $\mathcal{C}_i$ is obtained by sampling the cluster center of each cluster in $\mathcal{C}_{i-1}$ with probability $n^{-1/k}$. Each cluster $C \in \mathcal{C}_i$ has in the spanner H, a BFS tree of depth i rooted at the cluster center spanning[5] all the nodes of C. The vertices that are not incident to the sampled clusters become unclustered. For each unclustered vertex v, the algorithm adds one edge to each of the clusters incident to v in $\mathcal{C}_{i-1}$. This randomized construction is shown to yield a spanner with $O(kn^{1+1/k})$ edges in expectation and it can be implemented in $O(k^2)$ rounds[6]. Note that the only randomized step in Baswana-Sen is in picking the cluster centers of the i^{th} clustering. That is, given the $n^{1-(i-1)/k}$ cluster centers Z_{i-1} of the clusters in $\mathcal{C}_{i-1}$, it is required to pick $n^{-1/k}$ fraction of it, to be centers of the clusters in $\mathcal{C}_{i-1}$.

The brute-force deterministic solution in $O(n)$ rounds. A brute-force approach to pick the new cluster centers of $\mathcal{C}_i$ is to iterate over the clusters in $\mathcal{C}_{i-1}$ one by one, checking if they satisfy some expansion criteria. Informally, the expansion is measured by the number of vertices in the i^{th}-neighborhood of the cluster center (i.e., number of vertices that can be covered[7] by the cluster center in case it proceeds to the i^{th} level). If the expansion is large enough, the current cluster "expands" (i.e., covers vertices up to distance i), and joins the i^{th}-level of the clustering $\mathcal{C}_i$. Since in the first level there are $O(n)$ clusters (each vertex forms a singleton cluster), this approach gives an $O(n)$-round algorithm. With some adaptations, this approach can yield an improved $O(n^{1-1/k})$ round algorithm (as in [9]).

Our $O(n^{1/2-1/k})$-round deterministic solution. Inspired by the randomized construction of Baswana-Sen and the work of Derbel, Gavoille, and Peleg [6], we present a new approach for constructing spanners, based on two novel components which we discuss next.

1.2.1 Key Idea (I): Grouping Baswana-Sen Clusters into Superclusters

Our approach is based on adding an additional level of clustering on top of Baswana-Sen clustering. We introduce the novel notion of a supercluster – a subset of Baswana-Sen clusters that are *close* to each other in G. In every level $i \le k/2$, we group the $O(n^{1-i/k})$ clusters of $\mathcal{C}_i$ into $O(\sqrt{n})$ *superclusters*, each containing $O(n^{1/2-i/k})$ clusters which are also close to each other in G. Specifically, the superclusters have the following useful structure: cluster-centers of the same supercluster are connected in G by a *constant depth tree* (i.e., the weak diameter of the superclusters is $O(1)$), and the trees of different superclusters are *edge-disjoint.*

Unlike the brute-force $O(n)$-round algorithm mentioned above, our algorithm iterates over *superclusters* rather than clusters. We define the neighborhood of the supercluster to be all vertices that belong to – or have a neighbor in– one of the clusters of that supercluster. The expansion of the supercluster is simply the size of this neighborhood. The importance

[5] The vertices of the tree are precisely the vertices of the cluster.
[6] With some care, we believe the algorithm can also be implemented in $O(k)$ rounds
[7] We say the a vertex is covered by a cluster-center if it gets into its cluster.

of having this specific structure in each supercluster is that it allows the superclusters to compute their expansion in $O(1)$ rounds[8]. The faith of the superclusters (i.e., whether they continue on to the next level of clustering), in our algorithm, is determined by their expansion. If the expansion of a supercluster is sufficiently high, *all* the cluster centers of that supercluster join the next level i of the clustering. Otherwise, all these clusters are discarded from the clustering. As we will show in depth, the algorithm makes sure that at most $O(n^{1/2-1/k})$ superclusters pass this "expansion test" and the remaining superclusters with low-expansion are handled using our second key tool as explained next.

1.2.2 Key Idea (II): Better Spanners for Unbalanced Bipartite Graphs

In our spanner construction, each supercluster with low-expansion has additional useful properties: it has $|A| = O(\sqrt{n})$ vertices and only $|B| = O(n^{1/2+1/k})$ many "actual" neighbors[9]. We then apply Lemma 2 on these superclusters by computing the $(2k-1)$-spanner for each of these bipartite graphs obtained taking the vertices of the supercluster to be on one side of the bipartition, A, and their "actual" neighbors on the other side B. Since there are $O(n^{1/2})$ superclusters, this adds $O(n^{1/2+1/k} \cdot n^{1/2}) = O(n^{1+1/k})$ edges to the spanner.

Finally to provide a good stretch in the spanner for all the edges in G between vertices of the same supercluster[10], we simply recurse inside each supercluster– this can be done efficiently since the superclusters are vertex disjoint (as they contain sets of vertex disjoint clusters), and each of supercluster has $O(\sqrt{n})$ vertices.

Roadmap. The structure of the paper is as follows. We start by considering in Section 3 the simplified case of 3-spanners (hence $k = 2$) and present a deterministic construction with $O(\log n)$ rounds. Section 4 considers the general case of $k > 2$. In Sec. 4.1, we first describe an $O(n^{1-1/k})$-round algorithm that already contains some of the ideas of the final algorithm. Then, before presenting the algorithm, we describe the two key tools that it uses. For didactic reasons, in Sec. 4.2, we first describe the construction of sparser spanners for unbalanced bipartite graphs. Only later 4.3, we present the new notion of superclusters. Finally, in Sec. 4.4, we show how these tools can be used to construct $(2k-1)$-spanners for graphs of low diameter. The extension for general graphs is deferred to the full version [12].

2 Preliminaries, Notation and Model

Notations and Definitions. We consider an undirected unweighted n-vertex graph $G = (V, E)$ where V represents the set of processors and E is the set of links between them. Let $\text{diam}(H)$ be the diameter of the subgraph $H \subseteq G$. We denote the diameter of G by $D = \text{diam}(G)$. For $u, v \in V(G)$ and a subgraph H, let $\texttt{dist}(u, v, H)$ denote the $u - v$ distance in the subgraph $H \subseteq G$. When $H = G$, we omit it and write $\texttt{dist}(u, v)$. Let $\Gamma(u) = \{v \mid (u, v) \in E\}$ be the set of u's neighbors in G and $\Gamma^+(u) = \Gamma(u) \cup \{u\}$. For a subset of the vertices $V' \in V$, let $\Gamma(V', G) = \bigcup_{u \in V'} \Gamma(u)$ and $\Gamma^+(V', G) = \Gamma(V', G) \cup V'$. Let $\Gamma_i(v) = \{u \mid \texttt{dist}(u, v) \leq i\}$, for a subset V', $\Gamma_i(V')$ is defined accordingly.

For a subgraph $H \subseteq G$, let $E(v, H) = \{(u, v) \in E(H)\}$ be the set of edges incident to v in the subgraph H and let $\deg(v, H) = |E(v, H)|$ denote the degree of node v in G. For a set

[8] using the collection of edge disjoint $O(1)$-depth trees that connect their cluster centers.

[9] This term is informal, the actual neighbors are the vertices that are no longer clustered by the current clustering.

[10] Vertices whose clusters belong to the same supercluster.

of vertices C, let $G(C)$ be the induced graph of G on C. We say the a tuple $(a, b) > (c, d)$ if $a > c$ or $a = c$ but $b > d$.

Spanners and Clustering. A subgraph $H \subseteq G$ is a $(2k-1)$-*spanner* if $\mathtt{dist}(u, v, H) \leq (2k-1)\mathtt{dist}(u, v, G)$ for every $u, v \in V$. Given a graph G, a subgraph H, and an edge $e = (u, v)$ in G, we define the *stretch* of e in H to be the length of the shortest path from u to v in H. If no such path exists, we say that the stretch is infinite. We say that an edge $e = (u, v)$ is *taken care of* in H if $\mathtt{dist}(u, v, H) \leq (2k-1)$.

A *cluster* is a connected set C of vertices of the original graph. Often, a cluster will have one of its vertices $s \in C$ be the *cluster center*. The ID of the cluster is the ID of its center. Two clusters C_1 and C_2 are *neighbors* if $\Gamma(C) \cap C' \neq \emptyset$. For a subset of vertices $S \subseteq V$, the diameter of the subset is simply the diameter of the induced graph G on S.

Ruling Sets. An (α, β)-*ruling set* with respect to G and $V' \subseteq V$ is a subset $U \subseteq V'$ satisfying the following: (I) All pairs $u, v \in U$ satisfy $\mathtt{dist}(u, v) \geq \alpha$, (II) For all $v \in V'$, there exists a $u \in U$ such that $\mathtt{dist}(u, v) \leq \beta$.

The Communication Model. We use a standard message passing model, the CONGEST model [14], where the execution proceeds in synchronous rounds and in each round, each node can send a message of size $O(\log n)$ to each of its neighbors. In this model, local computation is done for free at each node and the primary complexity measure is the number of communication rounds. Each node holds a processor with a unique and arbitrary ID of $O(\log n)$ bits. Throughout, we assume that the nodes know a constant approximation on the number of nodes n, same holds also for the randomized algorithm of Baswana-Sen[11].

3 3-Spanners in $\widetilde{O}(1)$ Rounds

The key building block of the algorithm is the construction of a linear sized 3-spanner for a $\sqrt{n} \times n$ bipartite graph. A similar idea already appeared in [6], but using $O(n)$-bit messages.

The core construction: 3-spanners for unbalanced bipartite graphs

▶ **Lemma 4.** *Let $G = (A, B, E)$ be a (possibly weighted) bipartite graph where $|A| = O(\sqrt{n})$, $|B| = O(n)$, and each vertex knows whether it is in A or in B. Then one can construct (in the* CONGEST *model) a 3-spanner H with $O(n)$ edges within* two *rounds.*

Algorithm Bipartite3Spanner first forms $|A|$ vertex-disjoint star clusters (clusters of radius 1), each centered at a vertex of A. To do that, every vertex $v_b \in B$ picks one of its neighbors $v_a \in \Gamma(v_b) \cap A$ to be its cluster center and sends the ID of its chosen neighbor v_a to all of its neighbors. We write $c(v_b) = v_a$ to denote that the cluster center of v_b is v_a.

All edges $(v_b, c(v_b))$ are added to the spanner H. At this point, the graph contains $O(\sqrt{n})$ clusters centered at the vertices in A. We say that two stars S_1 and S_2 are *neighbors* if the *center* of S_1 has a neighbor in S_2, or vice-versa. Note that because the graph is bipartite, this is the only possible connection between clusters. Then, for each vertex u_a in A, and each neighboring star-cluster u'_a, the vertex u_a adds to the spanner H one edge to one of its neighbors in the cluster of u'_a. For a complete description of the algorithm see Alg. 1.

[11] In Baswana-Sen, each center samples itself with probability $n^{-1/k}$, which requires knowing n.

Algorithm 1 Bipartite3Spanner($G = (A \cup B, E)$) for $|A| = O(\sqrt{n})$ and $|B| = O(n)$.

```
1: $H \leftarrow \emptyset$
2: Each vertex $v_b \in B$ selects an arbitrary neighboring vertex $v_a \in A$, assigns $c(v_b) = v_a$ and send $c(v_b)$
   to all its neighbors. It adds the edge $(v_b, c(v_b))$ to $H$.
3: Each vertex $u_a \in A$ does the following (in parallel):
4: for each ID $v_a$ received do
5:     Pick a single neighbor $v_b$ satisfying $c(v_b) = v_a$. Add the edge $(u_a, v_b)$ to $H$.
```

To adapt the algorithm for the weighted case, we simply let each $v_b \in B$ pick its closest neighbor in A. In addition, each vertex u_a connects to its closest neighbor in each star of u'_a. It is easy to see that the algorithm takes 2 rounds. In the full version [12], we show:

▶ **Lemma 5.** *The output H of Alg.* Bipartite3Spanner *is a 3-spanner with $O(n)$ edges.*

Constructing 3-spanners for general graphs in $O(\log n)$ rounds. Let $V_h = \{v \in V \mid \deg(v, G) \geq \sqrt{n}\}$ be the set of *high* degree vertices in G and let $V_\ell = V \setminus V_h$ be the remaining *low*-degree vertices. First, the algorithm adds to the spanner H, all the edges of the low-degree vertices V_ℓ. Then, it proceeds by partitioning (in a way that will be described later) the *high-degree* vertices V_h into $t = O(\sqrt{n})$ balanced sets $V_1, \ldots, V_t$. This partition gives rise to t bipartite $\sqrt{n} \times n$ graphs B_i obtained by taking V_i to be on one side of the partition and $V \setminus V_i$ on the other side. We describe the partitioning procedure in Lemma 6. We can then apply Algorithm Bipartite3Spanner to construct 3-spanners for all these subgraphs in parallel. Finally, we simply add to H, all the internal edges $V_i \times V_i$ for every i, again adding total of $t \cdot O(n)$ edges.

Algorithm Improved3Spanner

- **(S0) Handling Low-Degree Vertices:** Add to H all edges in $(V_\ell \times V) \cap E(G)$.
- **(SI) Balanced Partitioning of High-Degree Vertices V_h:** Partition the high-degree vertices of V into $\Theta(\sqrt{n})$ sets $V_1, \ldots, V_t$ each with $O(\sqrt{n})$ vertices.
- **(SII) Taking care of edges $V_i \times (V \setminus V_i)$, for every $i \in \{1, \ldots, t\}$:**
  - Define $B_i = (V_i, V \setminus V_i)$ for every $i \in \{1, \ldots, t\}$.
  - Construct a 3-spanner $H_i \subseteq B_i$ by applying Algorithm Bipartite3Spanner on each of the B_i graphs in parallel, for every i.
- **(SIII) Taking care of edges $V_i \times V_i$, $i \in \{1, \ldots, t\}$:** Add to H all edges in $V_i \times V_i$ for every $i \in \{1, \ldots, t\}$.

Note that eventhough the bipartite graphs B_i are not vertex disjoint, each edge belongs to at most two such graphs, and hence we can construct the 3-spanners for all B_i in parallel. It is also easy to see that the final spanner has $O(n^{3/2})$ edges.

The only missing piece at that point concerns the computation of partitioning V_h.

Balanced partitioning of V_h in $O(\log n)$ rounds. The partition procedure starts by computing $(4, O(\log n))$-ruling set $R \subseteq V$ for the high-degree vertices V_h. We will use the following lemma that uses standard technique for constructing $(t, t \log n)$-ruling sets.

▶ **Lemma 6.** *Given a graph $G = (V, E)$ and a subset $V_h \in V$ of the vertices, one can compute in $O(\log n)$ rounds in the* CONGEST *model, a $(4, O(\log n))$-ruling set $U \subseteq V_h$ with respect to G and the high-degree vertices V_h.*

We now view each of the vertices $r \in R$ as a center of a cluster of diameter $O(\log n)$: let each high-degree vertex join the cluster of the vertex closest to it in R, breaking ties based on IDs. Since every vertex in V_h is at distance $O(\log n)$ from R, all the vertices V_h will be clustered within $O(\log n)$ rounds. Each vertex r in R can then partition the vertices of its cluster into subsets of size $\lfloor\sqrt{n}\rfloor$, and an additional leftover subset of size at most $\sqrt{n}$ (this can be done using balanced partitioning lemma, Lemma 9). We now claim that this partition is balanced. Clearly, all sets are of size $O(\sqrt{n})$, so we just show that there $O(\sqrt{n})$ subsets. Since every $r \in R$ is high-degree and since every two vertices in R are at distance at least 4, we have that $|R| = O(\sqrt{n})$. For each $r \in R$, there is at most one subset of size less than $\lfloor\sqrt{n}\rfloor$. Therefore, there are $O(\sqrt{n})$ subsets of size less than $\lfloor\sqrt{n}\rfloor$. All other subsets are of size $\lfloor\sqrt{n}\rfloor$. However, there can be at most $O(\sqrt{n})$ disjoint subsets of size $\lfloor\sqrt{n}\rfloor$, hence there are $O(\sqrt{n})$ subsets in total, as desired.

We conclude by showing:

▶ **Lemma 7** (3-Spanner Given Partition). *Given a (possibly weighted) n-vertex graph $G = (V, E)$ with a vertex-partition $V_1, V_2, \ldots, V_t$ such that $|V_i| = O(\sqrt{n})$ and $t = O(\sqrt{n})$, one can construct a 3-spanner H of size $O(n^{3/2})$ in 2 rounds in the* CONGEST *model.*

Finally, if the vertices IDs are bounded, two rounds are sufficient to construct the spanner.

▶ **Theorem 8** (Small IDs). *Given a graph $G = (V, E)$ where the IDs of the vertices have $\log(n) + O(1)$ bits, one can construct a 3-spanner H of G with $|H| = O(n^{3/2})$ edges in two rounds in the* CONGEST *model.*

4 $(2k - 1)$ Spanners

The structure of Baswana-Sen clustering. At the heart of the algorithm is a construction of $(k-1)$-levels of clustering $\mathcal{C}_0, \ldots, \mathcal{C}_{k-1}$. The initial clustering $\mathcal{C}_0 = \{\{v\}, v \in V\}$ simply contains n singleton clusters. For every i, each cluster $C \in \mathcal{C}_i$ has a cluster center z and we denote by Z_i the collection of cluster centers. We define $V_i = \bigcup_{z \in Z_i} \Gamma_i(z)$. A vertex v is *i-clustered* if $v \in V_i$, otherwise it is i-unclustered. Hence V_i is the set of clustered vertices appearing in the clusters of $\mathcal{C}_i$. The algorithm consists of $k-1$ steps where at the end of step $i \in \{1, \ldots, k-1\}$, we have an i^{th}-level clustering $\mathcal{C}_i = \{C_1, \ldots, C_\ell\}$ and a partial spanner H_i that satisfies the following: (P1) The clustering $\mathcal{C}_i$ contains $\ell = O(n^{1-i/k})$ clusters. (P2) For each cluster $C_j \in \mathcal{C}_i$ with a center z_j, the subgraph H_i contains a BFS tree $T_i(C)$ of depth at most i that spans all the vertices of C (i.e., the vertices of $T_i(C)$ are precisely C) and (P3) For every $u \in V_{i-1} \setminus V_i$, and every $v \in \Gamma(u)$, $\mathtt{dist}(u, v, H_i) \leq 2k - 1$.

High-Level Description of Phase i in Baswana-Sen Algorithm

- **(SI) Selecting $O(n^{1-i/k})$ cluster centers $Z_i \subseteq Z_{i-1}$.** In the randomized algorithm, this is done by sampling each center in Z_{i-1} independently with probability $n^{-1/k}$. The i-clustered vertices are $V_i = \Gamma_i^+(Z_i)$.
- **(SII) Taking care of unclustered vertices $V_{i-1} \setminus V_i$.** That is, taking care of the vertices that stopped being clustered at that point.
- **(SIII) Forming the clusters of $\mathcal{C}_i$ around Z_i.** This is done by letting each $u \in V_i$ join the cluster of its *closest* center in Z_i breaking tie based on ID's. The latter can be implemented in $O(i)$ rounds of constructing BFS trees of depth i from all centers Z_i while breaking ties appropriately.

At the final phase of Baswana-Sen, there are $O(n^{1/k})$ clusters in $\mathcal{C}_{k-1}$ and at that point, each vertex $v \in V$ adds one edge to each of its neighboring clusters in $\mathcal{C}_{k-1}$.

Note that the only step that uses randomness in this algorithm is sub-step (SI), and the other two sub-steps (SII-SIII) and the final phase are completely deterministic. Our challenge is to implement sub-step (SI) deterministically in a way that in sub-step (SII) we do not add too many edges to the spanner. The algorithms presented from now on, will simulate the i^{th} phase of Baswana-Sen only without using randomness. Sub-step (SIII) and the final phase will be implemented exactly as in Baswana-Sen.

4.1 Take (I): $O(n^{1-1/k})$-Round Algorithm NaiveSpanner

It is easy to see that 0^{th}-level clustering containing n singleton clusters satisfies properties (P1-P3). To simulate the i^{th} phase of Baswana-Sen algorithm, we employ $O(i \cdot n^{1-i/k})$ deterministic rounds: Initially, we unmark all the vertices and over time, some of the vertices will get marked (i.e., indicating that they are i-clustered). The procedure consists of $O(n^{1-i/k})$ steps where at each step, we look at the remaining set Z'_{i-1} of cluster centers in Z_{i-1} that have not yet been added to Z_i. Let U be the current set of unmarked vertices and let $\mathcal{C}'_{i-1} \subseteq \mathcal{C}_{i-1}$ be the corresponding clusters of Z'_{i-1}. For each cluster $C \in \mathcal{C}_{i-1}$, define its unmarked neighborhood by $\Gamma^U(C) = \bigcup_{u \in C} \Gamma(u) \cap U$ and its current unmarked-degree by $\deg^U(C) = |\Gamma^U(C)|$. We say that cluster C is a *local-maxima* in its unmarked neighborhood if it has the maximum tuple (lexicographically) $(\deg^U(C), ID(C))$ among all other clusters C' that have mutual unmarked neighbors (i.e., $\Gamma^U(C) \cap \Gamma^U(C') \neq \emptyset$).

Phase i of Algorithm NaiveSpanner

(SI) Defining the centers Z_i.
Set $Z'_{i-1} \leftarrow Z_{i-1}$, $U = V$ and for $O(n^{1-i/k})$ steps do the following:
- Every center $z \in Z'_{i-1}$ of cluster C computes $\deg^U(C)$.
- Every center $z \in Z'_{i-1}$ whose cluster C has the maximum tuple $(\deg^U(C), ID(C))$ in its unmarked neighborhood, $\deg^U(C)$, joins Z_i only if $\deg^U(C) \geq n^{i/k}$.
- Remove from Z'_i the centers $z \in C$ that join Z_i and mark $\Gamma^U(C)$.

(SII) Taking care of unclustered vertices.
- Let $\mathcal{C}'_{i-1}$ be the clusters whose centers did *not* join Z_i.
- For every unmarked vertex u, add one edge per neighboring cluster in $\mathcal{C}'_{i-1}$.

(SIII) Forming the $\mathcal{C}_i$ clusters centered at Z_i. As in Baswana-Sen.

Sketch of the Analysis. The key part to notice is that by picking the local-maxima clusters, we have that for any two cluster-centers $z_1 \in C_1, z_2 \in C_2$ that join Z_i, their unmarked neighborhoods $\Gamma^U(C_1), \Gamma^U(C_2)$ are vertex disjoint, hence Z_i contains $O(n^{1-i/k})$ centers; in addition, after $O(n^{1-i/k})$ steps, the clusters of all remaining centers have $O(n^{i/k})$ unmarked neighbors. Hence, at step (SII), total of $O(n^{1-(i-1)/k}) \cdot O(n^{i/k}) = O(n^{1+1/k})$ edges are added to the spanner. Turning to runtime, we claim that each of the $O(n^{1-i/k})$ steps can be implemented in $O(i)$ rounds. Since each cluster $C \in \mathcal{C}_{i-1}$ is connected i G by a depth-i tree, and since trees of different clusters are vertex-disjoint, computing the unmarked degree $\deg^U(C)$ of each cluster C can be done in $O(i)$ rounds; To avoid the double of counting of unmarked vertices that have many neighbors at the same cluster, each unmarked vertex respond to only one its neighbors in each cluster. Similarly, also selecting the local maxima clusters can be done in $O(i)$ rounds. We note that the time complexity of the algorithm is $O(n^{1-1/k})$, as opposed to $O(n)$, since after $O(n^{1-1/k})$ iterations of finding clusters of locally maximal unmarked degree, all remaining clusters will have low unmarked degree, and can be dealt with in parallel in $O(1)$ rounds, by adding an edge to every unmarked neighbor. Towards speeding up this algorithm, we now introduce our key technical tools.

A remark regarding step (SII). Let $V'_i = V_{i-1} \setminus V_i$ be the set of newly unclustered vertices. In Baswana-Sen algorithm, step (SII) takes care of all the edges in $V'_i \times V$. That is, the edges added to the spanner H at that stage provide that $\texttt{dist}(u,v,H) \leq 2i-1$ for every $(u,v) \in (V'_i \in V) \cap E$. Most of the algorithms we present in this paper, have a weaker but sufficient guarantee when implementing step (SII). In particular, we only add edges between the remaining unmarked vertices and the remaining clusters whose centers did not join Z_i. We now show why it is sufficient. Consider an edge $(u,v) \in E$. Let i_u be the largest level of the clustering such that u is i_u-clustered, define the same for v. Without loss of generality, assume that $i_v \leq i_u$.

Case (1): $i_u = k-1$: Let C be the cluster of u in $\mathcal{C}_{k-1}$. Since in the last step, v adds one edge to $\Gamma(u) \cap C$, the claim holds.

Case (2): $i_u \leq k-2$: Consider phase (i_u+1) where the clustering $\mathcal{C}_{i_u+1}$ is constructed given $\mathcal{C}_{i_u}$.

By definition, in step (SII) of phase (i_u+1) we have that the vertex v is unmarked and the vertex u belongs to a remaining cluster $C \in \mathcal{C}_{i_u}$. Since every unmarked vertex adds one edge to each remaining cluster, we have that v added one edge to $C \cap \Gamma(v)$. The claim follows.

4.2 Key Tool (I): Sparser Spanner for Unbalanced Bipartite Graphs

In this section, we consider Lemma 2. Similarly to the construction of 3-spanners in Section 3, a key ingredient in our algorithm is the construction of *sparser* spanners for unbalanced $A \times B$ bipartite graphs for $|A| \leq |B|$. The algorithm of [6] constructs a $(2k-1)$ spanners for these bipartite graphs with $O(|A||B|^{2/k})$ edges in the LOCAL model, using large messages. Our algorithm is slower than that of [6], but has the benefit of obtaining a sparser $(2k-1)$-spanner with only $O(k|A|^{1+2/k} + |B|)$ edges and while using $O(\log n)$-bit messages.

The high-level strategy of Alg. SparserBipartiteSpanner is to first compute $|A|$ star clusters (clusters of radius 1) by letting each vertex of B join an arbitrary neighbor in A. Hence, after one step of clustering, we have $|A|$ clusters rather than $O(n^{1-1/k})$ clusters is in Baswana-Sen. We then consider *star graph* G_S obtained contracting each star into a vertex, and essentially apply Alg. NaiveSpanner on the star-graph G_S to construct a $(k-1)$-spanner $H_S \subseteq G_S$ with $O(|A|^{1+2/k})$ edges within $O(k|A|^{1-2/k})$ rounds. To get a $(2k-1)$ spanner $H \subseteq G$ from H_S, for every star-edge $(S_i, S_j) \in H_S$, add a single edge in $(S_1 \times S_2) \cap E$ to H. Finally, adding the star edges to the spanner, gives a total of $O(|A|^{1+2/k} + |B|)$ edges. Simulating Alg. NaiveSpanner on the star-graph in the CONGEST model requires some effort. The description of Alg. SparserBipartiteSpanner and its analysis is in the full version [12].

4.3 Key Tool (II): Superclustering – Grouping Baswana-Sen Clusters

Why Superclusters? In this section, we describe the main tool that allows us to speed up Alg. NaiveSpanner by a factor of $\sqrt{n}$. The idea is to group the $n^{1-i/k}$ clusters in the i^{th}-clustering $\mathcal{C}_i$ into $\sqrt{n}$ superclusters, each containing $O(n^{1/2-i/k})$ clusters. Then, instead of iterating over clusters one by one (as in Alg. NaiveSpanner), we iterate over the superclusters. Each time, either *all* the cluster centers of a given supercluster join the next level of clustering, or none of them join. As will be shown later, in order to construct the i^{th}-clustering $\mathcal{C}_i$, it will be sufficient for our algorithm to consider $n^{1/2-1/k}$ superclusters (and not all $\sqrt{n}$ superclusters), hence yielding the round complexity of $O(n^{1/2-1/k})$ (for fixed k). For a supercluster to compute the number of its (unmarked) neighbors, all cluster centers in a given supercluster should be able to communicate efficiently. For that purpose, we make sure that the cluster centers in each supercluster are connected by an $O(2^k)$-depth tree[12], and that the trees of different superclusters are edge-disjoint. These trees will be used for communication purposes, and will allow us to aggregate information to leaders of all superclusters in parallel.

Defining the Superclusters. Let $\mathcal{C}_i$ be a collection of $O(n^{1-i/k})$ i-clusters. A *supercluster* $SC_{i,j} = \{C_{j_1}, \ldots, C_{j_\ell}\}$ is a collection of clusters from $\mathcal{C}_i$. A *Superclustering* $\mathcal{SC}_i = \{SC_{i,1}, \ldots, SC_{i,p}\}$ is a covering partition of all clusters from $\mathcal{C}_i$. That is, $\bigcup_{j=1}^{p} SC_{i,j} = \mathcal{C}_i$, and the superclusters are cluster-disjoint (every cluster in $\mathcal{C}_i$ belongs to exactly one supercluster). To select the cluster centers of level i, the algorithm constructs in each phase $i \in \{1, \ldots, k/2\}$ a superclustering $\mathcal{SC}_i$ which satisfies some helpful properties. We call a superclustering satisfying these properties a *nice superclustering*. Before defining the properties of a nice supercluster, we introduce some notation. For a supercluster $SC_{i,j} = \{C_{j_1}, \ldots, C_{j_p}\}$, let $V(SC_{i,j}) = \bigcup_{C \in SC_{i,j}} C$ be the set of all vertices in its clusters and $N_V(SC_{i,j}) = |V(SC_{i,j})|$ be the number of vertices in the supercluster $SC_{i,j}$. Also, let $N_C(SC_{i,j})$ denote the number of clusters that the supercluster $SC_{i,j}$ contains. A supercluster $SC_{i,j}$ with only one cluster (i.e., $N_C(SC_{i,j}) = 1$) is called a *singleton*. In addition, a singleton supercluster is called a *small-singleton* if $N_V(SC_{i,j}) \leq \sqrt{n}$ (otherwise, if $N_V(SC_{i,j}) > \sqrt{n}$, it is a *large-singleton*). Our $(2k-1)$-spanner construction is based upon the construction of superclusters with some *nice* useful properties, as defined next.

Nice Superclustering. A superclustering $\mathcal{SC}_i = \{SC_{i,1}, \ldots, SC_{i,\ell}\}$ is *nice* if it contains $\ell = O(\sqrt{n})$ superclusters, and each of these superclusters $SC_{i,j} \in \mathcal{SC}_i$ satisfies the following:

(N0) Singleton: If $N_V(SC_{i,j}) = \Omega(\sqrt{n})$, then $N_C(SC_{i,j}) = 1$.

Every non-singleton supercluster $SC_{i,j}$ (i.e., every supercluster containing at least two clusters) satisfies:

(N1) Cluster Balance: $N_C(SC_{i,j}) = O(n^{1/2-i/k})$, and

(N2) Vertex Balance: $N_V(SC_{i,j}) = O(\sqrt{n})$.

(N3) Connectivity: In the graph G, each $SC_{i,j} \in \mathcal{SC}_i$ has a tree $T(SC_{i,j})$ of depth[13] $O(2^k)$. In addition, the trees $T(SC_{i,1}), \ldots, T(SC_{i,\ell})$ are edge-disjoint.

Intuitive discussion of these properties. Property (N0) implies that if a supercluster has many vertices (more than $\sqrt{n}$), then it is a singleton supercluster. Property (N1) implies that non-singleton superclusters with at least two clusters are balanced with respect to the number of *clusters* from $\mathcal{C}_i$ that they contain. Since there are $O(n^{1-i/k})$ clusters in the i^{th} clustering, dividing it "fairly" between $\sqrt{n}$ superclusters yields this bound. Property (N2) also implies a balance among non-singleton superclusters, but this time with respect to the number of vertices. Finally, Property (N3) provides the existence of a $O(2^k)$-depth tree that connects the cluster centers of that supercluster. This "weird" looking depth of $O(2^k)$ shows up when computing the 0^{th}-level superclustering for general graphs (for graphs of constant diameter a much simpler construction exists). In particular, it shows up in Step (SI) of Alg. ConsZeroSuperclustering [12]. Finally, (N4) requires these trees to be edge-disjoint to allow communication within different superclusters, *in parallel* without congestion. As will

[12] This bound arises in the algorithm for graphs with general diameter, in the full version, and will be discussed later on.

[13] When the diameter of the original graph G is $O(1)$, the diameter of $T(SC_{i,j}) = O(1)$. The term $O(2^k)$ appears when dealing with graphs of large diameter, as described in the full version.

be shown in the next subsection, to satisfy Properties (N1) and (N2), the construction of the i^{th}-level of superclustering requires to partition both the vertices and the clusters into *balanced* the $\sqrt{n}$ superclusters. The key tool to achieve it is the following:

The Balanced Partitioning Lemma. The input to the partitioning lemma is a vertex-weighted tree T, where every vertex v in T has a non-negative weight $w(v)$ and in addition, we are given a bound B on the allowed total weight of each tree. The goal is to partition the tree into edge-disjoint subtrees, such that, all but one of the subtrees have a weight in $[B, 2B]$. The lemma achieves this but with some subtle specification. It partitions the vertices of the tree T into p disjoint sets: $\widehat{V}(T_0), \widehat{V}(T_1), \ldots, \widehat{V}(T_p)$. The total weight of each set $\widehat{V}(T_i)$, except for at most one, $\widehat{V}(T_0)$, is bounded by $[B, 2B]$. Hence, the partition respects the weight bound. Next, each set $\widehat{V}(T_i)$ is connected by a subtree $T_i \subseteq T$. The important feature of these trees T_i is that they might contain an additional vertex $v \in V(T) \setminus \widehat{V}(T_i)$. This additional vertex v, if exists, is the root of T_i and it is essential to connect the vertices in $\widehat{V}(T_i)$. Intuitively, this additional vertex helps us to communicate between the vertices of $\widehat{V}(T_i)$. Even though the trees T_i are not vertex disjoint, they are shown to be edge disjoint, which is sufficient for our applications.

▶ **Lemma 9** (Balanced Partitioning Lemma). *In $O(\text{diam}(T))$ rounds, one can construct subtrees $T_0, T_1, \ldots, T_p \subseteq T$, with roots $r(T_0), r(T_1), \ldots r(T_p)$ and corresponding* disjoint *vertex sets $\widehat{V}(T_0), \widehat{V}(T_1), \ldots, \widehat{V}(T_p)$ such that*

(D1) *The $\widehat{V}(T_i)$ sets are vertex disjoint and $\bigcup_{i=1}^{p} \widehat{V}(T_i) = V(T)$.*

(D2) *$W(T_i) \in [B, 2B]$ for every $i \geq 1$, and $W(T_0) \leq 2B$ where $W(T_i) = \sum_{u \in \widehat{V}(T_i)} w(u)$.*

(D3) *$V(T_i) = \widehat{V}(T_i) \cup r(T_i)$.*

(D4) *All $T_0, \ldots, T_p$ are edge-disjoint and with diameter at most* $\text{diam}(T)$*.*

Intuitively, the important vertex set of the tree T_i is the set of vertices $\widehat{V}(T_i)$ and hence the weight of the tree in Property (D2) is defined by summing over all these vertices (instead of summing over all vertices in the tree). Property (D3) implies that the tree T_i might contain, in addition to $\widehat{V}(T_i)$, also an additional vertex – its root – that allows the connectivity of the set $\widehat{V}(T_i)$ in T_i. The full proof of Lemma 9 appears in [12]. In the common application of this lemma, the tree T is a tree that connects the cluster-centers of a given supercluster, these cluster-centers are given a weight (e.g., the size of their cluster) and the remaining vertices in T are given a zero weight. The bound corresponds to the maximum allowed number of clusters (or vertices) in the supercluster (as in Section 4.3, (N2,N3)).

4.4 Take (II): $(2k-1)$-Spanners in $O(2^k \cdot n^{1/2-1/k})$ Rounds

We first consider the construction for graphs with constant diameter. At the end of the section, we discuss the extension for general graphs with arbitrary diameter. Recall that for $i \leq k/2$, $\mathcal{C}_i$ is a clustering that contains $O(n^{1-i/k})$ vertex-disjoint clusters centered at the vertices Z_i. The set of i-clustered vertices V_i are in $\Gamma_i(Z_i)$.

The first part of the algorithm contains $k/2$ phases. In each phase $i \in \{1, \ldots, k/2\}$, we are given a $(i-1)^{th}$ nice superclustering $\mathcal{SC}_{i-1}$ (whose superclusters contain the clusters of $\mathcal{C}_{i-1}$) and the current spanner H. We then construct the i^{th} nice superclustering $\mathcal{SC}_i$ and add edges to H in order to take care of the newly unclustered vertices in $V_{i-1} \setminus V_i$. At the end of the first part, we have a $(k/2)^{th}$ superclustering $\mathcal{SC}_{k/2}$ with $O(\sqrt{n})$ clusters. At that point, the number of clusters is small enough, and so Alg. NaiveSpanner can be applied.

Constructing the 0^{th}-level superclustering $\mathcal{SC}_0$ in $O(\mathrm{diam}(G))$ rounds. To compute $\mathcal{SC}_0$, we apply the Partitioning Lemma 9 on a BFS tree T rooted at some arbitrary vertex (e.g., of maximum ID) using weights of $w(v) = 1$ for each $v \in V$ and bound $B = O(\sqrt{n})$. This partitions the vertices into $\Theta(\sqrt{n})$ subsets S_i, each of size $O(\sqrt{n})$. Each such subset $S_i = \{v_{i,0}, \ldots, v_{i,\ell}\}$ defines a supercluster $SC_{0,i} = \{\{v_{i_0}\}, \ldots, \{v_{i,\ell}\}\}$ containing the singleton clusters of S_i's vertices. By that, we get $O(\sqrt{n})$ superclusters $\mathcal{SC}_0 = \{SC_{0,1}, \ldots, SC_{0,\sqrt{n}}\}$. By the partitioning lemma, we also have a tree T_i for each $SC_{0,i}$, satisfying Prop. (N3).

The i^{th} phase of Algorithm ImprovedSpanner for $i \in \{1, \ldots, k/2\}$. At the beginning of the phase, we are given the $(i-1)^{th}$-clustering $\mathcal{C}_{i-1}$ grouped into the nice superclustering $\mathcal{SC}_{i-1}$. Our first goal is to use the superclustering $\mathcal{SC}_{i-1}$ to define the set of new $O(n^{1-i/k})$ cluster centers Z_i. The high-level idea here is to implement Alg. NaiveSpanner on each supercluster rather than on each cluster. Given a set U of unmarked vertices and a supercluster $SC \in \mathcal{SC}_{i-1}$, define its *unmarked neighborhood* and *unmarked degree* by

$$\Gamma^U(SC) = \bigcup_{v \in V(SC)} (\Gamma^+(v) \cap U) \quad \text{and} \quad \deg^U(SC) = |\Gamma^U(SC)| \ . \tag{1}$$

Similarly to before, we say that a supercluster SC is a *local-maxima* in its unmarked neighborhood, if for every other SC' such that $\Gamma^U(SC) \cap \Gamma^U(SC') \neq \emptyset$, it holds that

$$(\deg^U(SC), ID(SC)) > (\deg^U(SC'), ID(SC')).$$

We say that supercluster SC has *low-expansion* if $\deg^U(SC) \leq n^{1/2+1/k}$. Otherwise, it has *high-expansion*. Note that unlike the previous algorithms presented before, here the expansion threshold $n^{1/2+1/k}$ is independent[14] of the level i.

Step (S1) of phase i: Selecting the centers Z_i. Selecting the $O(n^{1-i/k})$ cluster centers of Z_i is done in $O(n^{1/2-1/k})$ iterations. We start by unmarking all vertices. At each iteration, we have a set U of remaining unmarked vertices and a subset of remaining superclusters $\mathcal{SC}'_{i-1}$ of superclusters whose cluster centers have not yet been added to Z_i. All superclusters $SC \in \mathcal{SC}'_{i-1}$ compute their unmarked degree $\deg^U(SC)$ in parallel. (This can be done in $O(i \cdot 2^k)$ rounds thanks to Prop. (N3) in Section 4.3).

▶ **Definition 10** (Successful Supercluster). A supercluster SC that is local-maxima in its unmarked neighborhood and has high-expansion, that is $\deg^U(SC) \geq n^{1/2+1/k}$, is called a *successful* supercluster.

It is easy to see that the leader (vertex v of maximum ID in $V(SC)$) of every supercluster SC can verify in $O(i \cdot 2^k)$ rounds whether it is a local-maxima in its unmarked neighborhood.

In the algorithm, each successful supercluster SC adds all its cluster centers to Z_i, and mark all the vertices in $\Gamma^U(SC)$. This continues for $O(n^{1/2-1/k})$ iterations.

As we will show in the analysis section, since the unmarked neighborhoods of successful superclusters are disjoint and large, there are at most $O(n^{1/2-1/k})$ such superclusters. In addition, by Prop. (N2), each supercluster has $O(n^{1/2-(i-1)/k})$ clusters, overall $|Z_i| = O(n^{1/2-(i-1)/k} \cdot n^{1/2-1/k}) = O(n^{1-i/k})$ as desired.

[14] The intuition is that in each level i, the superclusters have at most $\sqrt{n}$ vertices, and we say that it has high expansion if the size of its neighborhood size is factor $n^{1/k}$ larger.

Step (S2) of phase i: Taking care of unclustered vertices. After $O(n^{1/2-1/k})$ steps of computing successful superclusters, all remaining superclusters SC have low-expansion with respect to the remaining unmarked vertices U'. That is, $\deg^{U}(SC) \leq n^{1/2-1/k}$. First, we take care of the singleton superclusters.

(S2.1): Singleton supercluster SC with low-expansion. Each unmarked vertex $u \in U'$ add to H an edge to one of its neighbor in $\Gamma(u) \cap V(SC)$. Since there are $O(\sqrt{n})$ superclusters, each with $\deg^{U'}(SC) \leq n^{1/2+1/k}$, overall we add $O(n^{1+1/k})$ such edges.

(S2.2): Non-singleton superclusters SC with low-expansion. Here, the construction of sparser spanners for bipartite graphs comes into play (see Sec. 4.2). Recall that by Prop. (N2), $N_V(SC) = O(\sqrt{n})$ vertices. Let $\Gamma^{U',-}(SC) = \Gamma^{U'}(SC) \setminus V(SC)$ be the unmarked neighbors of SC excluding the vertices of the supercluster SC. Since SC has low-expansion, it also holds that $|\Gamma^{U',-}(SC)| = O(n^{1/2-1/k})$. For every such supercluster SC, we consider the bipartite graph $B(SC) = (V(SC), \Gamma^{U',-}(SC))$, and apply Alg. SparserBipartiteSpanner to compute for it a $(2k-1)$-spanner $H(SC) \subseteq B(SC)$ with $O(n^{1/2+1/k})$ edges (see Lemma 2). This is done for all the graphs $B(SC)$ in parallel.

Note that the graphs $B(SC)$ are not necessarily vertex disjoint since an unmarked vertex can appear in several such graphs. The key observation that allows the parallel computation of all these spanners, is that every edge (u,v) can belong to at most *two* bipartite graphs, say, $B(SC)$ and $B(SC')$, where SC, SC' is the supercluster of u, v respectively[15]. Overall, since there are $O(\sqrt{n})$ superclusters, this adds $O(n^{1/2} \cdot n^{1/2+1/k}) = O(n^{1+1/k})$ edges.

Finally, it remains to take care of all edges between vertices belonging to the *same* supercluster. Note that in Alg. NaiveSpanner, there was no need for such a step since all vertices belonging to the same cluster are connected in H by an i-depth BFS tree rooted at the cluster center. However, in our setting, vertices that belong to *different* clusters of the *same* superclusters might still have large stretch (as cluster centers of the same supercluster might be at distance $O(2^k)$ in G). At that point, we use the fact that all superclusters are vertex disjoint and each contains $O(\sqrt{n})$ vertices. We then recursively apply the algorithm ImprovedSpanner on each of these superclusters in parallel. That is, we apply ImprovedSpanner on the induced subgraph on $V(SC)$ for every such supercluster SC.

Note that since in each phase we unmark all the vertices, unclustered vertices can become clustered again and in particular, edges between newly unclustered vertices and clustered vertices will be taken care of later on. This completes the description of the second step.

Steps (SIII) and (SIV): Defining i^{th}-clustering and the i^{th}-superclustering. The clusters $\mathcal{C}_i$ centered at the cluster centers Z_i computed at step (SI) are computed exactly as in Baswana-Sen Algorithm. The depth i-trees of these clusters are added to the spanner. The main challenge here is to *re-group* the new $O(n^{1-i/k})$ clusters into $O(\sqrt{n})$ superclusters, in a way that satisfies all the properties of the nice superclustering mentioned in Section 4.3.

Our starting point is as follows: we have a collection of $O(n^{1/2-1/k})$ successful superclusters $SC \in \mathcal{SC}_{i-1}$ whose cluster centers joined Z_i. Since $\mathcal{SC}_{i-1}$ is nice, by Prop. (N3), each such supercluster $\mathcal{SC}$ has a tree $T(\mathcal{SC})$ of depth $O(2^k)$ that spans all its cluster centers.

First, we let each cluster $C \in \mathcal{C}_i$ with $\Omega(\sqrt{n})$ vertices, to define its own singleton superclusters. Since clusters are vertex disjoint, there are $O(\sqrt{n})$ such superclusters. It now remains to re-group the remaining clusters of $\mathcal{C}_i$ into $O(\sqrt{n})$ superclusters.

[15] Recall that the superclusters share no vertex in common.

For each successful supercluster SC, we now consider only its centers of clusters with $O(\sqrt{n})$ vertices. First, we consider Property (N1) and use Lemma 9 with the tree $T(SC)$, weights $w(z) = 1$ for every cluster-center z of SC (only those that have $O(\sqrt{n})$ vertices in their cluster) and bound $B = O(n^{1/2-i/k})$. All other vertices v' in $T(SC)$ have $w(v') = 0$ (in particular, the centers z of clusters in SC which have been turned into singleton superclusters, we set $w(z) = 0$). By Prop. (N2) for $\mathcal{SC}_{i-1}$, we know that $SC \in \mathcal{SC}_{i-1}$ has $O(n^{1/2-(i-1)/k})$ cluster centers. Hence, the partition procedure will partition each of these superclusters into $O(n^{1/k})$ superclusters $SC_1, \ldots, SC_\ell$. In addition, by Lemma 9(D5), all these resulting superclusters SC_j are equipped with edge-disjoint trees $T(SC_j)$ of diameter $O(2^k)$. Since there are $O(n^{1/2-1/k})$ successful superclusters, overall after this partition there are $O(n^{1/2-1/k}) \cdot O(n^{1/k})$ superclusters.

We then turn to property (N3), and farther partition the superclusters to obtain a balance partition of the *vertices* into superclusters. For that purpose, for each supercluster SC' (obtained from the step above), we again apply the Partitioning Lemma on $T(SC')$. This time we use $B = \sqrt{n}$ and the weight $w(z)$ of each cluster center z in SC' is the number of vertices in its cluster C, that is $w(z) = |C|$ (for clusters v' which have turned into singleton superclusters, or any other non-center vertex in $T(SC')$, we set $w(v') = 0$). Since the vertices of superclusters are disjoint, this step increase the number of superclusters only by an additive $O(\sqrt{n})$ term. Hence, overall the number of superclusters is kept bounded by $O(\sqrt{n})$. This completes the description of the i^{th} phase of Alg. ImprovedSpanner.

The terminating step $k/2$. At the $(k/2)^{th}$ step we have $O(\sqrt{n})$ superclusters, each containing $O(1)$ clusters, hence overall we have $O(\sqrt{n})$ clusters. Now we can afford using Algorithm NaiveSpanner (described near the beginning of Section 4), which iterates over the *clusters* one by one. This completes the description of the algorithm for graph with $\mathrm{diam}(G) = O(1)$. The analysis of stretch, size and round complexity is in the full version [12].

Extension for general graphs of diameter $\mathrm{diam}(G)$. The only step that requires adaptation is that of constructing the 0^{th}-level superclustering $\mathcal{SC}_0$.

▶ **Lemma 11.** *[12] One can construct in $O(2^k \cdot n^{1/2-1/k})$ rounds, nice superclustering $\mathcal{SC}_0 = \{SC_{0,1}, \ldots, SC_{0,p}\}$ along with a subgraph H' with $O(kn^{1+1/k})$ edges such that for every vertex u not participating in the clusters of these superclusters $SC_{0,j} \in \mathcal{SC}_0$, it holds that $\mathtt{dist}(u, v, H') \leq 2k - 1$ for every $v \in \Gamma(u)$.*

References

1 Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. *TCS*, 2016.

2 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

3 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *CoRR*, abs/1608.01689, 2016.

4 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the local model. In *FOCS*, 2016.

5 Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. *Theoretical Computer Science*, 2008.

6 Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In *DISC*, pages 179–192. Springer, 2007.

7 Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *PODC*, pages 273–282, 2008.

8 Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. Local computation of nearly additive spanners. In *DISC*, 2009.

9 Bilel Derbel, Mohamed Mosbah, and Akka Zemmari. Sublinear fully distributed partition with applications. *Theory of Computing Systems*, 47(2):368–404, 2010.

10 Manuela Fischer and Mohsen Ghaffari. Deterministic distributed matching: Simpler, faster, better. *arXiv preprint arXiv:1703.00900*, 2017.

11 Manuela Fischer, Mohsen Ghaffari, and Khun Fabian. Deterministic distributed edge-coloring via hypergraph maximal matching. *arXiv preprint arXiv:1704.02767*, 2017.

12 Ofer Grossman and Merav Pater. Improved deterministic distributed construction of spanners. *arXiv preprint arXiv:1708.01011*, 2017.

13 Michal Hanckowiak, Michal Karonski, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. *SIDMA*, 15(1):41–57, 2001.

14 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. SIAM, 2000.

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