Congested Clique Algorithms for Graph Spanners

Merav Parter
Weizmann IS, Rehovot, Israel
merav.parter@weizmann.ac.il

Eylon Yogev
Weizmann IS, Rehovot, Israel
eylon.yogev@weizmann.ac.il

Abstract

Graph spanners are sparse subgraphs that faithfully preserve the distances in the original graph up to small stretch. Spanner have been studied extensively as they have a wide range of applications ranging from distance oracles, labeling schemes and routing to solving linear systems and spectral sparsification. A k-spanner maintains pairwise distances up to multiplicative factor of k. It is a folklore that for every n-vertex graph G, one can construct a $(2k-1)$ spanner with $O(n^{1+1/k})$ edges. In a distributed setting, such spanners can be constructed in the standard CONGEST model using $O(k^2)$ rounds, when randomization is allowed.

In this work, we consider spanner constructions in the congested clique model, and show:

- a randomized construction of a $(2k-1)$-spanner with $\widetilde{O}(n^{1+1/k})$ edges in $O(\log k)$ rounds. The previous best algorithm runs in $O(k)$ rounds;
- a deterministic construction of a $(2k–1)$-spanner with $\widetilde{O}(n^{1+1/k})$ edges in $O(\log k+(\log\log n)^3)$ rounds. The previous best algorithm runs in $O(k\log n)$ rounds. This improvement is achieved by a new derandomization theorem for hitting sets which might be of independent interest;
- a deterministic construction of a $O(k)$-spanner with $O(k\cdot n^{1+1/k})$ edges in $O(\log k)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Graph Algorithms, Spanner, Congested Clique

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.40

Related Version A full version of the paper is available at [21], https://arxiv.org/abs/1805.05404.

Acknowledgements We are grateful to Mohsen Ghaffari for earlier discussions on congested-clique spanners via streaming ideas. We thank Roei Tell for pointing out [15].

1 Introduction & Related Work

Graph spanners introduced by Peleg and Schäffer [23] are fundamental graph structures, more precisely, subgraphs of an input graph G, that faithfully preserve the distances in G up to small multiplicative stretch. Spanners have a wide-range of distributed applications [22] for routing [27], broadcasting, synchronizers [24], and shortest-path computations [3].

The common objective in distributed computation of spanners is to achieve the best-known existential size-stretch trade-off within small number of *rounds*. It is a 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, by the girth conjecture of Erdős.

There are plentiful of distributed constructions of spanners for both the LOCAL and the CONGEST models of distributed computing [8, 2, 9, 10, 11, 25, 12, 16]. The standard setting is a synchronous message passing model where per round each node can send one

message to each of its neighbors. In the LOCAL model, the message size is unbounded, while in the CONGEST model it is limited to $O(\log n)$ bits. One of the most notable distributed randomized constructions of $(2k-1)$ spanners is by Baswana & Sen [2] which can be implemented in $O(k^2)$ rounds in the CONGEST model.

Currently, there is an interesting gap between deterministic and randomized constructions in the CONGEST model, or alternatively between the deterministic construction of spanners in the LOCAL vs. the CONGEST model. Whereas the deterministic round complexity of $(2k-1)$ spanners in the LOCAL model is $O(k)$ due to [10], the best deterministic algorithm in the CONGEST model takes $O(2^{\sqrt{\log n \cdot \log\log n}})$ rounds [13].

We consider the *congested clique* model, introduced by Lotker et al. [20]. In this model, in every round, each vertex can send $O(\log n)$ bits to each of the vertices in the graph. The congested clique model has been receiving a lot of attention recently due to its relevance to overlay networks and large scale distributed computation [17, 14, 4].

Deterministic local computation in the congested clique model. Censor et al. [7] initiated the study of *deterministic* local algorithms in the congested clique model by means of derandomization of randomized LOCAL algorithms. The approach of [7] can be summarized as follows. The randomized complexity of the classical local problems is $\mathsf{polylog}(n)$ rounds (in both LOCAL and CONGEST models). For these randomized algorithms, it is usually sufficient that the random choices made by vertices are sampled from distributions with *bounded independence*. Hence, any round of a randomized algorithm can be simulated by giving all nodes a shared random seed of $\mathsf{polylog}(n)$ bits.

To completely derandomize such a round, nodes should compute (deterministically) a seed which is at least as "good"[1] as a random seed would be. This is achieved by estimating their "local progress" when simulating the random choices using that seed. Combining the techniques of conditional expectation, pessimistic estimators and bounded independence, leads to a simple "voting"-like algorithm in which the bits of the seed are computed *bit-by-bit*. The power of the congested clique is hence in providing some global leader that collects all votes in 1 round and broadcasts the winning bit value. This approach led to deterministic MIS in $O(\log\Delta \log n)$ rounds and deterministic $(2k-1)$ spanners with $\widetilde{O}(n^{1+1/k})$ edges in $O(k \log n)$ rounds, which also works for weighted graphs. Barenboim and Khazanov [1] presented deterministic local algorithms as a function of the graph's *arboricity*.

Deterministic spanners via derandomization of hitting sets. As observed by [26, 5, 13], the derandomization of the Baswana-Sen algorithm boils down into a derandomization of p-dominating sets or *hitting-sets*. It is a well known fact that given a collection of m sets $\mathcal{S}$, each containing at least Δ elements coming from a universe of size n, one can construct a hitting set Z of size $O((n \log m)/\Delta)$. A randomized construction of such a set is immediate by picking each element into Z with probability p and applying Chernoff. A centralized deterministic construction is also well known by the greedy approach (e.g., Lemma 2.7 of [5]).

In our setting we are interested in deterministic constructions of hitting sets in the congested clique model. In this setting, each vertex v knows a subset S_v of size at least Δ, that consists of vertices in the $O(k)$-neighborhood of v, and it is required to compute a small set Z that hits (i.e., intersects) all subsets. Censor et al. [7] showed that the above mentioned randomized construction of hitting sets still holds with $g = O(\log n)$-wise independence,

[1] The random seed is usually shown to provide a large progress in expectation. The deterministically computed seed should provide a progress at least as large as the expected progress of a random seed.

Table 1

	Stretch	#Rounds	Type
Adaptation of Baswana & Sen [2]	$2k-1$	$O(k)$	Randomized
This Work	$2k-1$	$O(\log k)$	
Censor-Hillel et al. [7]	$2k-1$	$O(k \log n)$	Deterministic
This Work	$2k-1$	$O(\log k + (\log\log n)^3)$	
This Work	$O(k)$	$O(\log k)$	

and presented an $O(g)$-round algorithm that computes a hitting set deterministically by finding a good seed of $O(g \log n)$ bits. Applying this hitting-set algorithm for computing the k levels of Baswana-Sen's clustering yields a deterministic algorithm for $(2k-1)$ spanners with $O(k \log n)$ rounds.

Our Results and Approach in a Nutshell

We provide improved randomized and deterministic constructions of graph spanners in the congested clique model. Our randomized solution is based on an $O(\log k)$-round algorithm that computes the $O(\sqrt{n})$ nearest vertices in radius $k/2$ for every vertex v[2]. This induces a partitioning of the graph into sparse and dense regions. The sparse region is solved "locally" and the dense region simulates only two phases of Baswana-Sen, leading to a total round complexity of $O(\log k)$. We show the following for n-vertex unweighted graphs.

▶ **Theorem 1.** *There exists a randomized algorithm in the congested clique model that constructs a $(2k-1)$-spanner with $\widetilde{O}(k \cdot n^{1+1/k})$ edges within $O(\log k)$ rounds w.h.p.*

Our deterministic algorithms are based on constructions of hitting-sets with short seeds. Using the pseudorandom generator of Gopalan et al. [15], we construct a hitting set with seed length $O(\log n \cdot (\log\log n)^3)$ which yields the following for n-vertex unweighted graphs.

▶ **Theorem 2.** *There exists a* deterministic *algorithm in the congested clique model that constructs a $(2k-1)$-spanner with $\widetilde{O}(k \cdot n^{1+1/k})$ edges within $O(\log k + (\log\log n)^3)$ rounds.*

In addition, we also show that if one settles for stretch of $O(k)$, then a hitting-set seed of $O(\log n)$ bits is sufficient for this purpose, yielding the following construction:

▶ **Theorem 3.** *There exists a* deterministic *algorithm in the congested clique model that constructs a $O(k)$-spanner with $O(k \cdot n^{1+1/k})$ edges within $O(\log k)$ rounds.*

A summary of our results are given in the Table 1. All results in the table are with respect to spanners with $\widetilde{O}(n^{1+1/k})$ edges for an unweighted n-vertex graph G. All these bounds are for the congested clique model[3].

In what follows we provide some technical background and then present the high level ideas of these construction.

[2] To be more precise, the algorithm computes the $O(n^{1/2-1/k})$ nearest vertices at distance at most $k/2-1$.

[3] Baswana-Sen [2] does not mention the congested clique model, but the best randomized solution in the congested clique is given by simulating [2].

A brief exposition of Baswana-Sen [2]. The algorithm is based on constructing k levels of clustering $\mathcal{C}_0, \ldots, \mathcal{C}_{k-1}$, where a clustering $\mathcal{C}_i = \{C_{i,1}, \ldots, \}$ consists of vertex disjoint subsets which we call clusters. Every cluster $C \in \mathcal{C}_i$ has a special node that we call *cluster center*. For each $C \in \mathcal{C}_i$, the spanner contains a depth-i tree rooted at its center and spanning all cluster vertices. Starting with the trivial clustering $\mathcal{C}_0 = \{\{v\}, v \in V\}$, in each phase i, the algorithm is given a clustering $\mathcal{C}_i$ and it computes a clustering $\mathcal{C}_{i+1}$ by sampling the cluster center of each cluster in $\mathcal{C}_{i-1}$ with probability $n^{-1/k}$. Vertices that are adjacent to the sampled clusters join them and the remaining vertices become unclustered. For the latter, the algorithm adds some of their edges to the spanner. This construction yields a $(2k-1)$ spanner with $O(kn^{1+1/k})$ edges in expectation.

It is easy to see that this algorithm can be simulated in the congested clique model using $O(k)$ rounds. As observed in [26, 16], the only randomized step in Baswana-Sen is picking the cluster centers of the $(i+1)^{th}$ clustering. That is, given the $n^{1-i/k}$ cluster centers of $\mathcal{C}_i$, it is required to compute a subsample of $n^{1-(i+1)/k}$ clusters without having to add too many edges to the spanner (due to unclustered vertices). This is exactly the hitting-set problem where the neighboring clusters of each vertex are the *sets* that should be covered, and the *universe* is the set of centers in $\mathcal{C}_i$ (ideas along these lines also appear in [26, 13]).

Our Approach. In the following, we provide the high level description of our construction while omitting many careful details and technicalities. We note that some of these technicalities stems from the fact that we insist on achieving the (nearly) optimal spanners, as commonly done in this area. Settling for an $O(k)$-spanner with $\widetilde{O}(kn^{1+1/k})$ edges could considerably simplify the algorithm and its analysis. The high-level idea is simple and it is based on dividing the graph G into sparse edges and dense edges, constructing a spanner for each of these subgraphs using two different techniques. This is based on the following intuition inspired by the Baswana-Sen algorithm.

In Baswana-Sen, the vertices that are clustered in level-i of the clustering are vertices whose i-neighborhood is sufficiently *dense*, i.e., contains at least $n^{i/k}$ vertices. We then divide the vertices into *dense* vertices V_{dense} and *sparse* vertices V_{sparse}, where V_{dense} consists of vertices that have $\Omega(\sqrt{n})$ vertices in their $k/2$-ball, and V_{sparse} consists of the remaining vertices. This induces a partitioning of G edges into $E_{sparse} = (V_{sparse} \times V) \cap E(G)$ and E_{dense} that contains the remaining G-edges, i.e., edges whose both endpoints are dense.

Collecting Topology of Closed Neighborhood. One of the key-building blocks of our construction is an $O(\log k)$-round algorithm that computes for each vertex u the subgraph $G_{k/2}(u)$ induced on its closest $O(\sqrt{n})$ vertices within distance at most $k/2$ in G. Hence the algorithm computes the entire $k/2$-neighborhoods for the sparse vertices. For the sake of the following discussion, assume that the maximum degree in G is $O(\sqrt{n})$. Our algorithm handles the general case as well. Intuitively, collecting the $k/2$-neighborhood can be done in $O(\log k)$ rounds if the graph is sufficiently *sparse* by employing the graph exponentiation idea of [19]. In this approach, in each phase the radius of the collected neighborhood is doubled. Employing this technique in our setting gives rise to several issues. First, the input graph G is not entirely sparse but rather consists of interleaving sparse and dense regions, i.e., the $k/2$-neighborhood of a sparse vertex might contain dense vertices. For that purpose, in phase i of our algorithm, each vertex (either sparse or dense) should obtain a subset of its closest $O(\sqrt{n})$ vertices in its 2^i neighborhood. Limiting the amount collected information is important for being able to route this information via Lenzen's algorithm [18] in $O(1)$ rounds in each phase.

Another technicality concerns the fact that the relation "u is in the $\sqrt{n}$ nearest vertices to v" is not necessarily symmetric. This entitles a problem where a given vertex u is "close"[4] to many vertices w, and u is not close to any of these vertices. In case where these w vertices need to receive the information from u regarding its closest neighbors (i.e., where some their close vertices are close to u), u ends up sending too many messages in a single phase. To overcome this, we carefully set the growth of the radius of the collected neighborhood in the graph exponentiation algorithm. We let only vertices that are close to each other exchange their topology information and show that this is sufficient for computing the $G_{k/2}(u)$ subgraphs. This procedure is the basis for our constructions as explained next.

Handling the Sparse Region. The idea is to let every sparse vertex u locally simulate a LOCAL spanner algorithm on its subgraph $G_{k/2}(u)$. For that purpose, we show that the deterministic spanner algorithm of [10] which takes k rounds in general, in fact requires only $k/2$ rounds when running by a sparse vertex u. At the end of these $k/2$ rounds, for each spanner edge (u, v), at least one of the endpoints know that this edge is in the spanner. This implies that the subgraph $G_{k/2}(u)$ contains all the information needed for u to locally simulate the spanner algorithm. This seemingly harmless approach has a subtle defect. Letting only the sparse vertices locally simulate a spanner algorithm might lead to a case where a certain edge (u, v) is not added by a sparse vertex due to a decision made by a dense vertex w in the local simulation u in $G_{k/2}(u)$. Since w is a dense vertex it did not run the algorithm locally and hence is not aware of adding these edges[5]. To overcome this, the sparse vertices notify the dense vertices about their edges added in their local simulations. We show how to do it in $O(1)$ rounds.

Handling the Dense Region. In the following, we settle for stretch of $(2k + 1)$ for ease of description. By applying the topology collecting procedure, every dense vertex v computes the set $N_{k/2}(v)$ consisting of its closest $\Theta(\sqrt{n})$ vertices within distance $k/2$. The main benefit in computing these $N_{k/2}(v)$ sets, is that it allows the dense vertices to "skip" over the first $k/2 - 1$ phases of Baswana-Sen, ready to apply the $(k/2)$ phase.

As described earlier, picking the centers of the clusters can be done by computing a hitting set for the set $\mathcal{S} = \{N_{k/2}(v) \mid v \in V_{dense}\}$. It is easy to construct a random subset $Z \subseteq V$ of cardinality $O(n^{1/2})$ that hits all these sets and to cluster all the dense vertices around this Z set. This creates clusters of strong diameter k (in the spanner) that cover all the dense vertices. The final step connects each pair of adjacent clusters by adding to the spanner a single edge between each such pair, this adds $|Z|^2 = O(n)$ edges to the spanner.

Hitting Sets with Short Seed. The description above used a randomized solution to the following hitting set problem: given n subsets of vertices $S_1, \ldots, S_n$, each $|S_i| \geq \Delta$, find a small set Z that intersects all S_i sets. A simple randomized solution is to choose each node v to be in Z with probability $p = O(\log n/\Delta)$. The standard approach for derandomization is by using distributions with limited independence. Indeed, for the randomized solution to hold, it is sufficient to sample the elements from a $\log n$-wise distribution. However, sampling an element with probability $p = O(\log n/\Delta)$ requires roughly $\log n$ random bits, leading to a total seed length of $(\log^2 n)$, which is too large for our purposes.

[4] By *close* we mean being among the $\sqrt{n}$ nearest vertices.

[5] If we "add" one more round and simulate $k/2 + 1$ rounds, then there is no such problem as both endpoints of a spanner edge know that the edge is in the spanner. However, we could only collect the information up to radius $k/2$.

Our key observation is that for any set S_i the event that $S_i \cap Z \neq \emptyset$ can be expressed by a *read-once DNF formula*. Thus, in order to get a short seed it suffices to have a pseudoranom generator (PRG) that can "fool" read-once DNFs. A PRG is a function that gets a short random seed and expands it to a long one which is indistinguishable from a random seed of the same length for such a formula. Luckily, such PRGs with seed length of $O(\log n \cdot (\log\log n)^3)$ exist due to Gopalan et al. [15], leading to deterministic hitting-set algorithm with $O((\log\log n)^3)$ rounds.

Graph Notations. For a vertex $v \in V(G)$, a subgraph G' and an integer $\ell \in \{1, \ldots, n\}$, let $\Gamma_\ell(v, G') = \{u \mid \text{dist}(u, v, G') \leq \ell\}$. When $\ell = 1$, we omit it and simply write $\Gamma(v, G')$, also when the subgraph G' is clear from the context, we omit it and write $\Gamma_\ell(v)$. For a subset $V' \subseteq V$, let $G[V']$ be the induced subgraph of G on V'. Given a disjoint subset of vertices C, C', let $E(C, C', G) = \{(u, v) \in E(G) \mid u \in C \text{ and } v \in C\}$. we say that C and C' are *adjacent* if $E(C, C', G) \neq \emptyset$. Also, for $v \in V$, $E(v, C, G) = \{(u, v) \in E(G) \mid u \in C\}$. A vertex u is *incident* to a subset C, if $E(v, C, G) \neq \emptyset$.

Road-Map. Section 2 presents algorithm NearestNeighbors to collect the topology of nearby vertices. At the end of this section, using this collected topology, the graph is partitioned into sparse and dense subgraphs. Section 3 describes the spanner construction for the sparse regime. Section 4 considers the dense regime and is organized as follows. First, Section 4.1 describes a deterministic construction spanner given an hitting-set algorithm as a black box. Then, Section 5 fills in this missing piece and shows deterministic constructions of small hitting-sets via derandomization. Finally, Section 5.3 provides an alternative deterministic construction, with improved runtime but larger stretch.

2 Collecting Topology of Nearby Neighborhood

For simplicity of presentation, assume that k is even, for k odd, we replace the term $(k/2-1)$ with $\lfloor k/2 \rfloor$. In addition, we assume $k \geq 6$. Note that randomized constructions with $O(k)$ rounds are known and hence one benefits from an $O(\log k)$ algorithm for a non-constant k. In the full version, we show the improved deterministic constructions for $k \in \{2, 3, 4, 5\}$.

2.1 Computing Nearest Vertices in the $(k/2-1)$ Neighborhoods

In this subsection, we present an algorithm that computes the $n^{1/2-1/k}$ nearest vertices with distance $k/2-1$ for every vertex v. This provides the basis for the subsequent procedures presented later on. Unfortunately, computing the nearest vertices of each vertex might require many rounds when $\Delta = \omega(\sqrt{n})$. In particular, using Lenzen's routing[6][18], in the congested clique model, the vertices can learn their 2-neighborhoods in $O(1)$ rounds, when the maximum degree is bounded by $O(\sqrt{n})$. Consider a vertex v that is incident to a heavy vertex u (of degree at least $\Omega(\sqrt{n})$). Clearly v has $\Omega(n^{1/2-1/k})$ vertices at distance 2, but it is not clear how v can learn their identities. Although, v is capable of receiving $O(n^{1/2-1/k})$ messages, the heavy neighbor u might need to send $n^{1/2-1/k}$ messages to each of its neighbors, thus $\Omega(n^{3/2-1/k})$ messages in total. To avoid this, we compute the $n^{1/2-1/k}$ nearest vertices in a *lighter* subgraph G_{light} of G with maximum degree $\sqrt{n}$. The neighbors of heavy vertices might not learn their 2-neighborhood and would be handled slightly differently in Section 4.

[6] Lenzen's routing can be viewed as a $O(1)$-round algorithm applied when each vertex v is a target and a sender of $O(n)$ messages.

▶ **Definition 4.** A vertex v is *heavy* if $\deg(v, G) \geq \sqrt{n}$, the set of heavy vertices is denoted by V_{heavy}. Let $G_{light} = G[V \setminus V_{heavy}]$.

▶ **Definition 5.** For each vertex $u \in V(G_{light})$ define $N_{k/2-1}(u)$ to be the set of $y(u) = \min\{n^{1/2-1/k}, |\Gamma_{k/2-1}(u, G_{light})|\}$ closest vertices at distance at most $(k/2-1)$ from u (breaking ties based on IDs) in G_{light}. Define $T_{k/2-1}(u)$ to be the truncated BFS tree rooted at u consisting of the u-v shortest path in G_{light}, for every $v \in N_{k/2-1}(u)$.

▶ **Lemma 6.** *There exists a deterministic algorithm* NearestNeighbors *that within $O(\log k)$ rounds, computes the truncated BFS tree $T_{k/2-1}(u)$ for each vertex $u \in V(G_{light})$. That is, after running Alg.* NearestNeighbors*, each $u \in V(G_{light})$ knows the entire tree $T_{k/2-1}(u)$.*

Algorithm NearestNeighbors. For every integer $j \geq 0$, we say that a vertex u is j-sparse if $|\Gamma_j(u, G_{light})| \leq n^{1/2-1/k}$, otherwise we say it is j-dense. The algorithm starts by having each non-heavy vertex compute $\Gamma_2(u, G_{light})$ in $O(1)$ rounds using Lenzen's algorithm. This is the only place where it is important that we work on G_{light} rather than on G. Next, in each phase $i \geq 1$, vertex u collects information on vertices in its $\gamma(i+1)$-ball in G_{light}, where:

$$\gamma(1) = 2, \text{ and } \gamma(i+1) = \min\{2\gamma(i) - 1, k/2\}, \text{ for every } i \in \{1, \ldots, \lceil \log(k/2) \rceil\}.$$

At phase $i \in \{1, \ldots, \lceil \log(k/2) \rceil\}$ the algorithm maintains the invariant that a vertex u holds a partial BFS tree $\widehat{T}_i(u)$ in G_{light} consisting of the vertices $\widehat{N}_i(u) := V(\widehat{T}_i(u))$, such that:

(I1) For an $\gamma(i)$-sparse vertex u, $\widehat{N}_i(u) = \Gamma_{\gamma(i)}(u)$.

(I2) For an $\gamma(i)$-dense vertex u, $\widehat{N}_i(u)$ consists of the closest $n^{1/2-1/k}$ vertices to u in G_{light}.

Note that in order to maintain the invariant in phase $(i+1)$, it is only required that in phase i, the $\gamma(i)$-sparse vertices would collect the relevant information, as for the $\gamma(i)$-dense vertices, it already holds that $\widehat{N}_{i+1}(u) = \widehat{N}_i(u)$. In phase i, each vertex v (regardless of being sparse or dense) sends its partial BFS tree $\widehat{T}_i(v)$ to each vertex u only if (1) $u \in \widehat{N}_i(v)$ and (2) $v \in \widehat{N}_i(u)$. This condition can be easily checked in a single round, as every vertex u can send a message to all the vertices in its set $\widehat{N}_i(u)$. Let $\widehat{N}'_{i+1}(u) = \bigcup_{v \in \widehat{N}_i(u) \mid u \in \widehat{N}_i(v)} \widehat{N}_i(v)$ be the subset of all received $\widehat{N}_i$ sets at vertex u. It then uses the distances to $\widehat{N}_i(u)$, and the received distances to the vertices in the $\widehat{N}_i$ sets, to compute the shortest-path distance to each $w \in \widehat{N}_i(v)$. As a result it computes the partial tree $\widehat{T}_{i+1}(u)$. The subset $\widehat{N}_{i+1}(u) \subseteq \widehat{N}'_{i+1}(u)$ consists of the (at most $n^{1/2-1/k}$) vertices within distance $\gamma(i+1)$ from u. This completes the description of phase i. We next analyze the algorithm and show that each phase can be implemented in $O(1)$ rounds and that the invariant on the $\widehat{T}_i(u)$ trees is maintained.

Analysis. We first show that phase i can be implemented in $O(1)$ rounds. Note that by definition, $|\widehat{N}_i(u)| \leq \sqrt{n}$ for every u, and every $i \geq 1$. Hence, by the condition of phase i, each vertex sends $O(n)$ messages and receives $O(n)$ messages, which can be done in $O(1)$ rounds, using Lenzen's routing algorithm [18].

We show that the invariant holds, by induction on i. Since all vertices first collected their second neighborhood, the invariant holds[7] for $i = 1$. Assume it holds up to the beginning of phase i, and we now show that it holds in the beginning of phase $i+1$. If u is $\gamma(i)$-dense, then u should not collect any further information in phase i and the assertion holds trivially.

[7] This is the reason why we consider only G_{light}, as otherwise $\gamma(1) = 0$ and we would not have any progress.

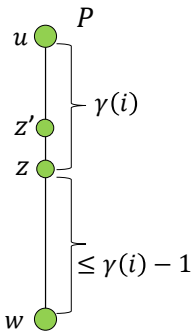

Figure 1 Shown is a path P between u and w where z is the first dense vertex on the $\gamma(i)$-length prefix of P. If $u \notin \widehat{N}_i(z)$ then $u, w \in \widehat{N}_i(z')$.

Consider an $\gamma(i)$-sparse vertex u and let $N_{\gamma(i+1)}(u)$ be the target set of the $n^{1/2-1/k}$ closest vertices at distance $\gamma(i+1)$ from u. We will fix $w \in N_{\gamma(i+1)}(u)$, and show that $w \in \widehat{N}_{i+1}(u)$ and in addition, u has computed the shortest path to w in G_{light}. Let P be u-w shortest path in G_{light}. If all vertices z on the $\gamma(i)$-length prefix of P are $\gamma(i)$-sparse, then the claim holds as $z \in \widehat{N}_i(u)$, $u \in \widehat{N}_i(z)$, and $w \in \widehat{N}_i(z')$ where z' in the last vertex on the $\gamma(i)$-length prefix of P. Hence, by the induction assumption for the $\widehat{N}_i$ sets, u can compute in phase i its shortest-path to w.

We next consider the remaining case where not all the vertices on the $\gamma(i)$-length path are sparse. Let $z \in \widehat{N}_i(u)$ be the first $\gamma(i)$-dense vertex (closest to u) on the $\gamma(i)$-length prefix of P. Observe that $w \in \widehat{N}_i(z)$. Otherwise, $\widehat{N}_i(z)$ contains $n^{1/2-1/k}$ vertices that are closer to z than w, which implies that these vertices are also closer to u than w, and hence w should not be in $N_{\gamma(i+1)}(u)$ (as it is not among the closest $n^{1/2-1/k}$ vertices to u), leading to contradiction. Thus, if also $u \in \widehat{N}_i(z)$, then z sends to u in phase i its shortest-path to w. By the induction assumption for the $\widehat{N}_i(u), \widehat{N}_i(z)$ sets, we have that u has the entire shortest-path to w. It remains to consider the case where the first $\gamma(i)$-dense vertex on P, z, does not contain u in its $\widehat{N}_i(z)$ set, hence it did not send its information on w to u in phase i. Denote $x = \text{dist}(u, z, G_{light})$ and $y = \text{dist}(z, w, G_{light})$, thus $x + y = |P| \leq 2\gamma(i) - 1$. Since $w \in \widehat{N}_i(z)$ but $u \notin \widehat{N}_i(z)$, we have that $y \leq x$ and $2y \leq |P|$, which implies that $y \leq \gamma(i) - 1$. Let z' be the vertex preceding z on the P path, hence z' also appear on the $\gamma(i)$-length prefix of P and $z' \in N_i(u)$. By definition, z' is $\gamma(i)$-sparse and it also holds that $u \in \widehat{N}_i(z')$. Since $\text{dist}(z', w, G_{light}) = y + 1 \leq \gamma(i)$, it holds that $w \in \widehat{N}_i(z')$. Thus, u can compute the u-w shortest-path using the z'-w shortest-path it has received from z'. For an illustration, see Figure 1.

2.2 Dividing G into Sparse and Dense Regions

During the execution of NearestNeighbors every non-heavy vertex v computes the sets $N_{k/2-1}(v)$ and the corresponding tree $T_{k/2-1}(v)$. The vertices are next divided into dense vertices V_{dense} and sparse vertices V_{sparse}. Roughly speaking, the dense vertices are those that have at least $n^{1/2-1/k}$ vertices at distance at most $k/2-1$ in G. Since the subsets of nearest neighbors are computed in G_{light} rather than in G, this vertex division is more delicate.

▶ **Definition 7.** A vertex v is *dense* if either (1) it is heavy, (2) a neighbor of a heavy vertex or (3) $|\Gamma_{k/2-1}(v, G_{light})| > n^{1/2-1/k}$. Otherwise, a vertex is *sparse*. Let V_{dense}, V_{sparse} be the dense (resp., sparse) vertices in V.

▶ **Observation 8.** *For $k \geq 6$, for every dense vertex v it holds that $|\Gamma_{k/2-1}(v, G)| \geq n^{1/2-1/k}$.*

The edges of G are partitioned into:

$$E_{dense} = ((V_{dense} \times V_{dense}) \cap E(G)), \; E_{sparse} = (V_{sparse} \times V) \cap E(G)$$

Since all the neighbors of heavy vertices are dense, it also holds that $E_{sparse} = (V_{sparse} \times (V \setminus V_{heavy})) \cap E(G_{light})$.

Overview of the Spanner Constructions. The algorithm contains two subprocedures, the first takes care of the sparse edge-set by constructing a spanner $H_{sparse} \subseteq G_{sparse}$ and the second takes care of the dense edge-set by constructing $H_{dense} \subseteq G$. Specifically, these spanners will satisfy that for every $e = (u, v) \in G_i$, $\text{dist}(u, v, H_i) \leq 2k - 1$ for $i \in \{sparse, dense\}$. We note that the spanner $H_{dense} \subseteq G$ rather than being contained in G_{dense}. The reason is that the spanner H_{dense} might contain edges incident to sparse vertices as will be shown later. The computation of the spanner H_{sparse} for the sparse edges, E_{sparse}, is done by letting each sparse vertex locally simulating a local spanner algorithm. The computation of H_{dense} is based on applying two levels of clustering as in Baswana-Sen. The selection of the cluster centers will be made by applying an hitting-set algorithm.

3 Handling the Sparse Subgraph

In the section, we construct the spanner H_{sparse} that will provide a bounded stretch for the sparse edges. As we will see, the topology collected by applying Alg. NearestNeighbors allows every sparse vertex to *locally* simulate a deterministic spanner algorithm in its collected subgraph, and deciding which of its edges to add to the spanner based on this local view.

Recall that for every sparse vertex v it holds that $|\Gamma_{k/2-1}(v, G_{light})| \leq n^{1/2-1/k}$ where $G_{light} = G[V \setminus V_{heavy}]$ and that $E_{sparse} = (V_{sparse} \times V) \cap E(G)$. Let $G_{sparse}(u) = G_{sparse}[\Gamma_{k/2-1}(u, G)]$. By applying Alg. NearestNeighbors, and letting sparse vertices sends their edges to the sparse vertices in their $(k/2 - 1)$ neighborhoods in G_{light}, we have:

▶ **Claim 9.** *There exists a $O(\log k)$-round deterministic algorithm, that computes for each sparse vertex v its subgraph $G_{sparse}(v)$.*

Our algorithm is based on an adaptation of the local algorithm of [10], which is shown to satisfy the following in our context. The proof is in the full version [21].

▶ **Lemma 10.** *There exists a deterministic algorithm* LocalSpanner *that constructs a $(k - 3)$ spanner in the* LOCAL *model, such that every sparse vertex u decides about its spanner edges within $k/2 - 1$ rounds. In particular, u can simulate Alg.* LocalSpanner *locally on G_{sparse} and for every edge (u, z) not added to the spanner H_{sparse}, there is a path of length at most $(k - 3)$ in $G_{sparse}(u) \cap H_{sparse}$.*

A useful property of the algorithm[8] by Derbel et al. (Algorithm 1 in [10]) is that if a vertex v did not terminate after i rounds, then it must hold that $|\Gamma_i(v, G)| \geq n^{i/k}$. Thus in our context, every sparse vertex terminates after at most $k/2 - 1$ rounds[9]. We also show that for

[8] This algorithm works only for unweighted graphs and hence our deterministic algorithms are for unweighted graphs. Currently, there are no local deterministic algorithms for weighted graphs.

[9] By definition we have that $|\Gamma_{k/2-1}(u, G_{light})| \leq n^{1/2-1/k}$. Moreover, since $G_{sparse} \subseteq G_{light}$ it also holds that $|\Gamma_{k/2-1}(u, G_{sparse})| \leq n^{1/2-1/k}$.

simulating these $(k/2-1)$ rounds of Alg. LocalSpanner by u, it is sufficient for u to know all the neighbors of its $(k/2-2)$ neighborhood in G_{sparse} and these edges are contained in $G_{sparse}(u)$. The analysis of Lemma 10 appears in the full version of the paper.

We next describe Alg. SpannerSparseRegion that computes H_{sparse}. Every vertex u computes $G_{sparse}(u)$ in $O(\log k)$ rounds and simulate Alg. LocalSpanner in that subgraph. Let $H_{sparse}(u)$ be the edges added to the spanner in the local simulation of Alg. LocalSpanner in $G_{sparse}(u)$. A sparse vertex u sends to each sparse vertex $v \in \Gamma_{k/2-1}(u, G_{sparse})$, the set of all v-edges in $H_{sparse}(u)$. Hence, each sparse vertex sends $O(n)$ messages (at most $\sqrt{n}$-edges to each of its at most $\sqrt{n}$ vertices in $\Gamma_{k/2-1}(v, G_{sparse})$). In a symmetric manner, every vertex receives $O(n)$ messages and this step can be done in $O(1)$ rounds using Lenzen's algorithm. The final spanner is given by $H_{sparse} = \bigcup_{u \in V_{sparse}} H_{sparse}(u)$. The stretch argument is immediate by the correctness of Alg. LocalSpanner and the fact that all the edges added to the spanner in the local simulations are indeed added to H_{sparse}. The size argument is also immediate since we only add edges that Alg. LocalSpanner would have added when running by the entire graph.

Algorithm SpannerSparseRegion (Code for a sparse vertex u)

1. Apply Alg. NearestNeighbors to compute $G_{sparse}(u)$ for each sparse vertex u.
2. Locally simulate Alg. LocalSpanner in $G_{sparse}(u)$ and let $H_{sparse}(u)$ be the edges added to the spanner in $G_{sparse}(u)$.
3. Send the edges of $H_{sparse}(u)$ to the corresponding sparse endpoints.
4. Add the received edges to the spanner H_{sparse}.

4 Handling the Dense Subgraph

In this section, we construct the spanner H_{dense} satisfying that $\text{dist}(u, v, H_{dense}) \leq 2k-1$ for every $(u, v) \in E_{dense}$. In this case, since the $(k/2-1)$ neighborhood of each dense vertex is large then there exists a small hitting that covers all these neighborhoods. The structure of our arguments is as follows. First, we describe a deterministic construction of H_{dense} using an hitting-set algorithm as a black box. This would immediately imply a randomized spanner construction in $O(\log k)$-rounds. Then in Section 5, we fill in this last missing piece and show deterministic constructions of hitting sets.

Constructing spanner for the dense subgraph via hitting sets. Our goal is to cluster all dense vertices into small number of low-depth clusters. This translates into the following *hitting-set* problem defined in [5, 28, 13]: Given a subset $V' \subseteq V$ and a set collection $\mathcal{S} = \{S(v) \mid v \in V'\}$ where each $|S(v)| \geq \Delta$ and $\bigcup_{v \in V'} S(v) \subseteq V''$, compute a subset $Z \subseteq V''$ of cardinality $O(|V''| \log n/\Delta)$ that intersects (i.e., *hits*) each subset $S \in \mathcal{S}$. A hitting-set of size $O(|V''| \log n/\Delta)$ is denoted as a *small* hitting-set.

We prove the next lemma by describing the construction of the spanner H_{dense} *given* an algorithm $\mathcal{A}$ that computes small hitting sets. In Section 5, we complement this lemma by describing several constructions of hitting sets. Let $G = (V, E)$ be an n-vertex graph, and let $\Delta \in [n]$ be a parameter. Let V' be a subset of nodes such that each node $u \in V'$ knows a set S_u where $|S_u| \geq \Delta$. Let $\mathcal{S} = \{S_u \subset V : u \in V'\}$ and suppose that V'' is such that $\bigcup S_u \subseteq V''$.

▶ **Lemma 11.** *Given an algorithm $\mathcal{A}$ for computing a small hitting-set in $r_{\mathcal{A}}$ rounds, there exists a* deterministic *algorithm* SpannerDenseRegion *for constructing the $(2k-1)$ spanner H_{dense} within $O(\log k + r_{\mathcal{A}})$ rounds.*

The next definition is useful in our context.

ℓ-depth Clustering. A *cluster* is a subset of vertices and a clustering $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ consists of vertex disjoint subsets. For a positive integer ℓ, a clustering $\mathcal{C}$ is a ℓ-depth clustering if for each cluster $C \in \mathcal{C}$, the graph G contains a tree of depth at most ℓ rooted at the cluster center of C and spanning all its vertices.

4.1 Description of Algorithm SpannerDenseRegion

The algorithm is based on clustering the dense vertices in two levels of clustering, in a Baswana-Sen like manner. The first clustering $\mathcal{C}_1$ is an $(k/2-1)$-depth clustering covering all the dense vertices. The second clustering, $\mathcal{C}_2$ is an $(k/2)$-depth clustering that covers only a *subset* of the dense vertices. For k odd, let $\mathcal{C}_2$ be equal to $\mathcal{C}_1$.

Defining the first level of clustering. Recall that by running Algorithm NearestNeighbors, every non-heavy vertex $v \in G_{light}$ knows the set $N_{k/2-1}(v)$ containing its $n^{1/2-1/k}$ nearest neighbors in $\Gamma_{k/2-1}(v, G_{light})$. For every heavy vertex v, let $N_{k/2-1}(v) = \Gamma(v, G)$. Let V_{nh} be the set of all non-heavy vertices that are *neighbors* of *heavy* vertices. By definition, $V_{nh} \subseteq V_{dense}$. Note that for every dense vertex $v \in V_{dense} \setminus V_{nh}$, it holds that $|N_{k/2-1}(v)| \geq n^{1/2-1/k}$. The vertices u of V_{nh} are in G_{light} and hence have computed the set $N_{k/2-1}(u)$, however, there is in guarantee on the size of these sets.

To define the clustering of the dense vertices, Algorithm SpannerDenseRegion applies the hitting-set algorithm $\mathcal{A}$ on the subsets $\mathcal{S}_1 = \{N_{k/2-1}(v) \mid v \in V_{dense} \setminus V_{nh}\}$ and the universe V. Since every set in $\mathcal{S}_1$ has size at least $\Delta := n^{1/2-1/k}$, the output of algorithm $\mathcal{A}$ is a subset Z_1 of cardinality $O(n^{1/2+1/k})$ that hits all the sets in $\mathcal{S}_1$.

We will now construct the clusters in $\mathcal{C}_1$ with Z_1 as the cluster centers. To make sure that the clusters are vertex-disjoint and connected, we first compute the clustering in the subgraph G_{light}, and then cluster the remaining dense vertices that are not yet clustered. For every $v \in G_{light}$ (either dense or sparse), we say that v is *clustered* if $Z_1 \cap N_{k/2-1}(v) \neq \emptyset$. In particular, every dense vertex v for which $|\Gamma_{k/2-1}(v, G_{light})| \geq n^{1/2-1/k}$ is clustered (the neighbors of heavy vertices are either clustered or not). For every clustered vertex $v \in G_{light}$ (i.e., even sparse ones), let $c_1(v)$, denoted hereafter the *cluster center* of v, be the closest vertex to v in $Z_1 \cap N_{k/2-1}(v)$, breaking shortest-path ties based on IDs. Since v knows the entire tree $T_{k/2-1}(v)$, it knows the distance to all the vertices in $N_{k/2-1}(v)$ and in addition, it can compute its next-hop $p(v)$ on the v-$c_1(v)$ shortest path in G_{light}. Each clustered vertex $v \in G_{light}$, adds the edge $(v, p(v))$ to the spanner H_{dense}. It is easy to see that this defines a $(k/2-1)$-depth clustering in G_{light} that covers all dense vertices in G_{light}. In particular, each cluster C has in the spanner a tree of depth at most $(k/2-1)$ that spans all the vertices in C. Note that in order for the clusters C to be connected in H_{dense}, it was crucial that all vertices in G_{light} compute their cluster centers in $N_{k/2-1}(v)$, if such exists, and not only the dense vertices. We next turn to cluster the remaining dense vertices. For every heavy vertex v, let $c_1(v)$ be its closest vertex in $\Gamma(v, G) \cap Z_1$. It then adds the edge $(v, c_1(v))$ to the spanner H_{dense} and broadcasts its cluster center $c_1(v)$ to all its neighbors. Every neighbor u of a heavy vertex v that is not yet clustered, joins the cluster of $c_1(v)$ and adds the edge (u, v) to the spanner. Overall, the clusters of $\mathcal{C}_1$ centered at the subset Z_1 cover all the dense vertices. In addition, all the vertices in a cluster C are connected in H_{dense} by a tree of depth $k/2-1$. Formally, $\mathcal{C}_1 = \{C_1(s), \mid s \in Z_1\}$ where $C_1(s) = \{v \mid c_1(v) = s\}$.

Defining the second level of clustering. Every vertex v that is clustered in $\mathcal{C}_1$ broadcasts its cluster center $c_1(v)$ to all its neighbors. This allows every dense vertex v to compute the subset $N_{k/2}(v) = \{s \in Z_1 \mid E(v, C_1(s), G) \neq \emptyset\}$ consisting of the centers of its adjacent clusters in $\mathcal{C}_1$. Consider two cases depending on the cardinality of $N_{k/2}(v)$. Every vertex v with $|N_{k/2}(v)| \leq n^{1/k} \log n$, adds to the spanner H_{dense} an arbitrary edge in $E(v, C_1(s), G)$ for every $s \in N_{k/2}(v)$. It remains to handle the remaining vertices $V'_{dense} = \{v \in V_{dense} \mid |N_{k/2}(v)| > n^{1/k} \log n\}$. These vertices would be clustered in the second level of clustering $\mathcal{C}_2$. To compute the centers of the clusters in $\mathcal{C}_2$, the algorithm applies the hitting-set algorithm $\mathcal{A}$ on the collection of subsets $\mathcal{S}_2 = \{N_{k/2}(v) \mid v \in V'_{dense}\}$ with $\Delta = n^{1/k} \log n$ and $V'' = Z_1$. The output of $\mathcal{A}$ is a subset Z_2 of cardinality $O(|Z_1| \log n/\Delta) = O(\sqrt{n} \log n)$ that hits all the subsets in $\mathcal{S}_2$. The 2^{nd} cluster-center $c_2(v)$ of a vertex $v \in V'_{dense}$ is chosen to be an arbitrary $s \in N_{k/2}(v) \cap Z_2$. The vertex v then adds some edge $(v, u) \in E(v, C_1(s), G)$ to the spanner H_{dense}. Hence, the trees spanning rooted at $s \in Z_2$ are now extended by one additional layer resulting in a $(k/2)$-depth clustering.

Connecting adjacent clusters. Finally, the algorithm adds to the spanner H_{dense} a single edge between each pairs of adjacent clusters $C, C' \in \mathcal{C}_1 \times \mathcal{C}_2$, this can be done in $O(1)$ rounds as follows. Each vertex broadcasts its cluster ID in $\mathcal{C}_2$. Every vertex $v \in C$ for every cluster $C \in \mathcal{C}_1$ picks one incident edge to each cluster $C' \in \mathcal{C}_2$ (if such exists) and sends this edge to the corresponding center of the cluster of C' in $\mathcal{C}_2$. Since a vertex sends at most one message for each cluster center in $\mathcal{C}_2$, this can be done in $O(1)$ rounds. Each cluster center r of the cluster C' in $\mathcal{C}_2$ picks one representative edge among the edges it has received for each cluster $C \in \mathcal{C}_1$ and sends a notification about the selected edge to the endpoint of the edge in C. Since the cluster center sends at most one edge for every vertex this take one round. Finally, the vertices in the clusters $C \in \mathcal{C}_1$ add the notified edges (that they received from the centers of $\mathcal{C}_2$) to the spanner. This completes the description of the algorithm. We now complete the proof of Lemma 11.

Proof. Recall that we assume $k \geq 6$ and thus $|\Gamma_{k/2-1}(v)| \geq n^{1/2-1/k}$, for every $v \in V_{dense}$. We first show that for every $(u, v) \in E_{dense}$, $\text{dist}(u, v, H_{dense}) \leq 2k - 1$. The clustering $\mathcal{C}_1$ covers all the dense vertices. If u and v belong to the same cluster C in $\mathcal{C}_1$, the claim follows as H_{dense} contains an $(k/2 - 1)$-depth tree that spans all the vertices in C, thus $\text{dist}(u, v, H_{dense}) \leq k - 2$. From now on assume that $c_1(u) \neq c_1(v)$. We first consider the case that for both of the endpoints it holds that $|N_{k/2}(v)|, |N_{k/2}(u)| \leq n^{1/k} \log n$. In such a case, since v is adjacent to the cluster C_1 of u, the algorithm adds to H_{dense} at least one edge in $E(v, C_1, G)$, let it be (x, v). We have that $\text{dist}(v, u, H_{dense}) \leq \text{dist}(v, x, H_{dense}) + \text{dist}(x, u, H_{dense}) \leq k - 1$ where the last inequality holds as x and u belong to the same cluster C_1 in $\mathcal{C}_1$. Finally, it remains to consider the case where for at least one endpoint, say v, it holds that $|N_{k/2}(v)| > n^{1/k} \log n$. In such a case, v is clustered in $\mathcal{C}_2$. Let C_1 be the cluster of u in $\mathcal{C}_1$ and let C_2 be the cluster of v in $\mathcal{C}_2$. Since C_1 and C_2 are adjacent, the algorithm adds an edge in $E(C_1, C_2, G)$, let it be (x, y) where $x, u \in C_1$ and $y, v \in C_2$. We have that $\text{dist}(u, v, H_{dense}) \leq \text{dist}(u, x, H_{dense}) + \text{dist}(x, y, H_{dense}) + \text{dist}(y, v, H_{dense}) \leq 2k - 1$, where the last inequality holds as u, x belong to the same $(k/2-1)$-depth cluster C_1, and v, y belong to the same $(k/2)$-depth cluster C_2. Finally, we bound the size of H_{dense}. Since the clusters in $\mathcal{C}_1, \mathcal{C}_2$ are vertex-disjoint, the trees spanning these clusters contain $O(n)$ edges. For each unclustered vertex in $\mathcal{C}_2$, we add $O(n^{1/k} \log n)$ edges. By the properties of the hitting-set algorithm $\mathcal{A}$ it holds that $|Z_1| = O(n^{1/2-1/k} \cdot \log n)$ and $|Z_2| = O(n^{1/2} \cdot \log n)$. Thus adding one edge between each pair of clusters adds $|Z_1| \cdot |Z_2| = O(n^{1+1/k} \cdot \log^2 n)$ edges. ◀

Putting All Together: Randomized spanners in $O(\log k)$ rounds. We now complete the proof of Theorem 1. For an edge $(u,v) \in E_{sparse}$, the correctness follows by the correctness of Alg. LocalSpanner. We next consider the dense case. Let $\mathcal{A}$ be the algorithm where each $v \in V'$ is added into Z with probability of $\log/\Delta$. By Chernoff bound, we get that w.h.p. $|Z| = O(|V'| \log n/\Delta)$ and $Z \cap S_i \neq \emptyset$ for every $S_i \in \mathcal{S}$. The correctness follows by applying Lemma 11. ◀

Algorithm SpannerDenseRegion

1. Compute an $(k/2-1)$ clustering $\mathcal{C}_1 = \{C(s) \mid s \in Z_1\}$ centered at subset Z_1.
2. For every $v \in V_{dense}$, let $N_{k/2}(v) = \{s \in Z_1 \mid E(v, C_1(s), G) \neq \emptyset\}$.
3. For every $v \in V_{dense}$ with $|N_{k/2}(v)| \leq n^{1/k} \log n$, add to the spanner one edge in $E(v, C(s), G)$ for every $s \in N_{k/2}(v)$.
4. Compute an $(k/2)$ clustering $\mathcal{C}_2$ centered at Z_2 to cover the remaining dense vertices.
5. Connect (in the spanner) each pair of adjacent clusters $C, C' \in \mathcal{C}_1 \times \mathcal{C}_2$.

5 Derandomization of Hitting Sets

5.1 Hitting Sets with Short Seeds

The main technical part of the deterministic construction is to completely derandomize the randomized hitting-set algorithm using short seeds. We show two hitting-set constructions with different tradeoffs. The first construction is based on pseudorandom generators (PRG) for DNF formulas. The PRG will have a seed of length $O(\log n(\log\log n)^3)$. This would serve the basis for the construction of Theorem 2. The second hitting-set construction is based on $O(1)$-wise independence, it uses a small seed of length $O(\log n)$ but yields a larger hitting-set. This would be the basis for the construction of Theorem 3.

We begin by setting up some notation. For a set S we denote by $x \sim S$ a uniform sampling from S. For a function PRG and an index i, let $\mathsf{PRG}(s)_i$ the i^{th} bit of $\mathsf{PRG}(s)$.

▶ **Definition 12** (Pseudorandom Generators)**.** A generator $\mathsf{PRG}\colon \{0,1\}^r \to \{0,1\}^n$ is an ϵ-pseudorandom generator (PRG) for a class $\mathcal{C}$ of Boolean functions if for every $f \in \mathcal{C}$:

$$\left| \mathop{\mathbf{E}}_{x\sim\{0,1\}^n}[f(x)] - \mathop{\mathbf{E}}_{s\sim\{0,1\}^r}[f(\mathsf{PRG}(s))] \right| \leq \epsilon.$$

We refer to r as the seed-length of the generator and say PRG is explicit if there is an efficient algorithm to compute PRG that runs in time $poly(n, 1/\epsilon)$.

▶ **Theorem 13.** *For every $\epsilon = \epsilon(n) > 0$, there exists an explicit pseudoranom generator, $\mathsf{PRG}\colon \{0,1\}^r \to \{0,1\}^n$ that fools all read-once DNFs on n-variables with error at most ϵ and seed-length $r = O((\log(n/\epsilon)) \cdot (\log\log(n/\epsilon))^3)$.*

Using the notation above, and Theorem 13 we formulate and prove the following Lemma:

▶ **Lemma 14.** *Let S be subset of $[n]$ where $|S| \geq \Delta$ for some parameter $\Delta \leq n$ and let c be any constant. Then, there exists a family of hash functions $\mathcal{H} = \{h\colon [n] \to \{0,1\}\}$ such that choosing a random function from $\mathcal{H}$ takes $r = O(\log n \cdot (\log\log n)^3)$ random bits and for $Z_h = \{u \in [n] : h(u) = 0\}$ it holds that:*
(1) $\Pr_h\left[|Z_h| \leq \widetilde{O}(n/\Delta)\right] \geq 2/3$, and (2) $\Pr_h[S \cap Z_h \neq \emptyset] \geq 1 - 1/n^c$.

Proof. We first describe the construction of $\mathcal{H}$. Let $p = c' \log n/\Delta$ for some large constant c' (will be set later), and let $\ell = \lfloor \log 1/p \rfloor$. Let $\mathsf{PRG}\colon \{0,1\}^r \to \{0,1\}^{n\ell}$ be the PRG constructed in Theorem 13 for $r = O(\log n\ell \cdot (\log\log n\ell)^3) = O(\log n \cdot (\log\log n)^3)$ and

for $\epsilon = 1/n^{10c}$. For a string s of length r we define the hash function $h_s(i)$ as follows. First, it computes $y = \mathsf{PRG}(s)$. Then, it interprets y as n blocks where each block is of length ℓ bits, and outputs 1 if and only if all the bits of the i^{th} block are 1. Formally, we define $h_s(i) = \bigwedge_{j=(i-1)\ell+1}^{i\ell} \mathsf{PRG}(s)_j$. We show that properties 1 and 2 hold for the set Z_{h_s} where $h_s \in \mathcal{H}$. We begin with property 1. For $i \in [n]$ let $X_i = h_s(i)$ be a random variable where $s \sim \{0,1\}^r$. Moreover, let $X = \sum_{i=1}^{n} X_i$. Using this notation we have that $|Z_{h_s}| = X$. Thus, to show property 1, we need to show that $\Pr_{s\sim\{0,1\}^r}[X \leq \widetilde{O}(n/\Delta)] \geq 2/3$. Let $f_i \colon \{0,1\}^{n\ell} \to \{0,1\}$ be a function that outputs 1 if the i^{th} block is all 1's. That is, $f_i(y) = \bigwedge_{j=(i-1)\ell+1}^{i\ell} y_j$. Since f_i is a read-once DNF formula we have that

$$\left| \mathop{\mathbf{E}}_{y\sim\{0,1\}^{n\ell}}[f_i(y)] - \mathop{\mathbf{E}}_{s\sim\{0,1\}^{r}}[f_i(\mathsf{PRG}(s))] \right| \leq \epsilon.$$

Therefore, it follows that

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} \mathop{\mathbf{E}}_{s\sim\{0,1\}^{r}}[f_i(\mathsf{PRG}(s))] \leq \sum_{i=1}^{n} (\mathop{\mathbf{E}}_{y\sim\{0,1\}^{n\ell}}[f_i(y)] + \epsilon) = n(2^{-\ell} + \epsilon) = \widetilde{O}\left(\frac{n}{\Delta}\right).$$

Then, by Markov's inequality we get that $\Pr_{s\sim\{0,1\}^r}[X > 3\,\mathbf{E}[X]] \leq 1/3$ and thus

$$\Pr_{s\sim\{0,1\}^r}\left[X \leq \widetilde{O}(n/\Delta)\right] \geq 1 - \Pr_{s\sim\{0,1\}^r}[X > 3\,\mathbf{E}[X]] \geq 2/3.$$

We turn to show property 2. Let S be any set of size at least Δ and let $g \colon \{0,1\}^{n\ell} \to \{0,1\}$ be an indicator function for the event that the set S is covered. That is,

$$g(y) = \bigvee_{i\in S} \bigwedge_{j=(i-1)\ell+1}^{i\ell} y_j.$$

Since g is a read-once DNF formula, and thus we have that

$$\left| \mathop{\mathbf{E}}_{y\sim\{0,1\}^{n\ell}}[g(y)] - \mathop{\mathbf{E}}_{s\sim\{0,1\}^{r}}[g(\mathsf{PRG}(s))] \right| \leq \epsilon.$$

Let $Y_i = \bigwedge_{j=(i-1)\ell+1}^{i\ell} y_j$, and let $Y = \sum_{i\in S} Y_i$. Then $\mathbf{E}[Y] = \sum_{i\in S} \mathbf{E}[Y_i] \geq \Delta 2^{-\ell} \geq \Delta p = c' \log n$. Thus, by a Chernoff bound we have that $\Pr[Y = 0] \leq \Pr[\mathbf{E}[Y] - Y \geq c' \log n] \leq 1/n^{2c}$, for a large enough constant c' (that depends on c). Together, we get that $\Pr_s[S \cap Z_{h_s} \neq \emptyset] = \mathbf{E}_{s\sim\{0,1\}^r}[g(\mathsf{PRG}(s))] \geq \mathbf{E}_{y\sim\{0,1\}^{n\ell}}[g(y)] - \epsilon = \Pr_{y\sim\{0,1\}^{n\ell}}[Y \geq 1] - \epsilon \geq 1 - 1/n^c$. ◀

We turn to show the second construction of dominating sets with short seed. In this construction the seed length of shorter, but the set is larger. By a direct application of Lemma 2.2 in [6], we get the following lemma which becomes useful for showing Theorem 3.

▶ **Lemma 15.** *Let S be a subset of $[n]$ where $|S| \geq \Delta$ for some parameter $\Delta \leq n$ and let c be any constant. Then, there exists a family of hash functions $\mathcal{H} = \{h \colon [n] \to \{0,1\}\}$ such that choosing a random function from $\mathcal{H}$ takes $r = O(\log n)$ random bits and for $Z_h = \{u \in [n] : h(u) = 0\}$ it holds that: (1) $\Pr_h\left[|Z_h| \leq O(n^{17/16}/\sqrt{\Delta})\right] \geq 2/3$, and (2) $\Pr_h[S \cap Z_h \neq \emptyset] \geq 1 - 1/n^c$.*

5.2 Deterministic Hitting Sets in the Congested Clique

We next present a deterministic construction of hitting sets by means of derandomization. The round complexity of the algorithm depends on the number of random bits used by the randomized algorithms.

▶ **Theorem 16.** *Let $G = (V, E)$ be an n-vertex graph, let $V' \subset V$, let $\mathcal{S} = \{S_u \subset V : u \in V'\}$ be a set of subsets such that each node $u \in V'$ knows the set S_u and $|S_u| \geq \Delta$, and let c be a constant. Let $\mathcal{H} = \{h : [n] \to \{0,1\}\}$ be a family of hash functions such that choosing a random function from $\mathcal{H}$ takes $g_{\mathcal{A}}(n,\Delta)$ random bits and for $Z_h = \{u \in [n] : h(u) = 0\}$ it holds that: (1) $\Pr[|Z_h| \leq f_{\mathcal{A}}(n,\Delta)] \geq 2/3$ and (2) for any $u \in V'$: $\Pr[S_u \cap Z_h \neq \emptyset] \geq 1 - 1/n^c$. Then, there exists a deterministic algorithm $\mathcal{A}_{det}$ that constructs a hitting set of size $O(f_{\mathcal{A}}(n,\Delta))$ in $O(g_{\mathcal{A}}(n,\Delta)/\log n)$ rounds.*

Proof. Our goal is to completely derandomize the process of finding Z_h by using the method of conditional expectation. We follow the scheme of [7] to achieve this, and define two bad events that can occur when using a random seed of size $g = g_{\mathcal{A}}(n,\Delta)$. Let A be the event where the hitting set Z_h consists of more than $f_{\mathcal{A}}(n,\Delta)$ vertices. Let B be the event that there exists an $u \in V'$ such that $S_u \cap Z_h = \emptyset$. Let X_A, X_B be the corresponding indicator random variables for the events, and let $X = X_A + X_B$.

Since a random seed with $g_{\mathcal{A}}(n,\Delta)$ bits avoids both of these events with high probability, we have that $\mathbf{E}[X] < 1$ where the expectation is taken over a seed of length g bits. Thus, we can use the method of conditional expectations in order to get an assignment to our random coins such that no bad event occurs, i.e., $X = 0$. In each step of the method, we run a distributed protocol to compute the conditional expectation. Actually, we will compute a pessimistic estimator for the conditional expectation.

Letting X_u be indicator random variable for the event that S_u is not hit by Z_h, we can write our expectation as follows: $\mathbf{E}[X] = \mathbf{E}[X_A] + \mathbf{E}[X_B] = \Pr[X_A = 1] + \Pr[X_B = 1] = \Pr[X_A = 1] + \Pr[\vee_u X_u = 1]$ Suppose we have a partial assignment to the seed, denoted by Y. Our goal is to compute the conditional expectation $\mathbf{E}[X|Y]$, which translates to computing $\Pr[X_A = 1|Y]$ and $\Pr[\vee_u X_u = 1|Y]$. Notice that computing $\Pr[X_A = 1|Y]$ is simple since it depends only on Y (and not on the graph or the subsets $\mathcal{S}$). The difficult part is computing $\Pr[\vee_u X_u = 1|Y]$. Instead, we use a pessimistic estimator of $\mathbf{E}[X]$ which avoids this difficult computation. Specifically, we define the estimator: $\Psi = X_A + \sum_{u \in V'} X_u$. Recall that for any $u \in V'$ for a random g-bit length seed, it holds that $\Pr[X_u = 1] \leq 1/n^c$ and thus by applying a union bound over all n sets, it also holds that $\mathbf{E}[\Psi] = \Pr[X_A = 1] + \sum_u \Pr[X_u = 1] < 1$. We describe how to compute the desired seed using the method of conditional expectation. We will reveal the assignment of the seed in chunks of $\ell = \lfloor \log n \rfloor$ bits. In particular, we show how to compute the assignment of ℓ bits in the seed in $O(1)$ rounds. Since the seed has g many bits, this will yield an $O(g/\log n)$ round algorithm.

Consider the i^{th} chunk of the seed $Y_i = (y_1, \ldots, y_\ell)$ and assume that the assignment for the first $i-1$ chunks $Y_1 \ldots, Y_{i-1}$ have been computed. For each of the n possible assignments to Y_i, we assign a node v that receives the conditional probability values $\Pr[X_u = 1|Y_1, \ldots, Y_i]$ from all nodes $u \in V'$. Notice that a node u can compute the conditional probability values $\Pr[X_u = 1|Y_1, \ldots, Y_i]$, since u knows the IDs of the vertices in S_u and thus has all the information for this computation. The node v then sums up all these values and sends them to a global leader w. The leader w can easily compute the conditional probability $\Pr[X_A = 1|Y]$, and thus using the values it received from all the nodes it can compute $\mathbf{E}[X|Y]$ for of the possible n assignments to Y_i. Finally, w selects the assignment $(y_1^*, \ldots, y_\ell^*)$ that minimizes the pessimistic estimator Ψ and broadcasts it to all nodes in the graph. After $O(g/\log n)$ rounds Y has been completely fixed such that $X < 1$. Since X_A and X_B get binary values, it must be the case that $X_A = X_B = 0$, and a hitting set has been found. ◀

Combining Lemma 14 and Lemma 15 with Theorem 16, yields:

▶ **Corollary 17.** *Let $G = (V, E)$ be an n-vertex graph, let $V', V'' \subset V$, let $\mathcal{S} = \{S_u \subset V : u \in V'\}$ be a set of subsets such that each node $u \in V'$ knows the set S_u, such that $|S_u| \geq \Delta$ and $\bigcup S_u \subseteq V''$. Then, there exists deterministic algorithms $\mathcal{A}_{det}, \mathcal{A}'_{det}$ in the congested clique model that construct a hitting set Z for $\mathcal{S}$ such that: (1) $|Z| = \widetilde{O}(|V''|/\Delta)$ and $\mathcal{A}_{det}$ runs in $O((\log\log n)^3)$ rounds. (2) $|Z| = O(|V''|^{17/16}/\sqrt{\Delta})$ and $\mathcal{A}'_{det}$ runs in $O(1)$ rounds.*

Deterministic construction in $O(\log k + O((\log\log n)^3))$ Rounds. Theorem 2 follows by plugging Corollary 17(1) into Lemma 11.

5.3 Deterministic $O(k)$-Spanners in $O(\log k)$ Rounds

In this subsection, we provide a proof sketch of Theorem 3. The complete proof appears in the full version. Let $k \geq 10$. According to Section 3, it remains to consider the construction of H_{dense} for the dense edge set E_{dense}. Recall that for every dense vertex v, it holds that $|\Gamma_{k/2}(v, G)| \geq n^{1/2-1/k}$. Similarly to the proof of Lemma 11, we construct a $(k/2-1)$ dominating set Z for the dense vertices. However, to achieve the desired round complexity, we use the $O(1)$-round hitting set construction of Corollary 17(2) with parameters of $\Delta = n^{1/2-1/k}$ and $V' = V$. The output is then a hitting set Z of cardinality $O(n^{13/16+1/(2k)})$ that hits all the $(k/2-1)$ neighborhoods of the dense vertices. Then, as in Alg. SpannerDenseRegion, we compute a $(k/2-1)$-depth clustering $\mathcal{C}_1$ centered at Z. The key difference to Alg. SpannerDenseRegion is that $|Z|$ is too large for allowing us to add an edge between each pair of adjacent clusters, as this would result in a spanner of size $O(|Z|^2)$. Instead, we essentially contract the clusters of $\mathcal{C}_1$ (i.e., contracting the intra-cluster edges) and construct the spanner recursively in the resulting contracted graph G''. Every contracted node in G'' corresponds to a cluster with a small strong diameter in the spanner. Specifically, G'' is decomposed into sparse and dense regions. Handling the sparse part is done deterministically by applying Alg. SpannerSparseRegion. To handle the dense case, we apply the hitting-set algorithm of Corollary 17(2) to cluster the dense nodes (which are in fact, contracted nodes) into $|V(G'')|/\sqrt{\Delta}$ clusters for $\Delta = n^{1/2-1/k}$. After $O(1)$ repetitions of the above, we will be left with a contracted graph with $o(\sqrt{n})$ vertices. At this point, we connect each pair of clusters (corresponding to these contracted nodes) in the spanner.

A naïve implementation of such an approach would yield a spanner with stretch $k^{O(1)}$, as the diameter of the clusters induced by the contracted nodes is increased by a k-factor in each of the phases. To avoid this blow-up in the stretch, we enjoy the fact that already after the first phase, the contracted graph G' has $O(n^{13/16+o(1)})$ nodes and hence we can allow to compute a $(2k'-1)$ spanner for G' with $k' = 8$ as this would add $O(n)$ edges to the final spanner. Since in each of the phases (except for the first one) the stretch parameter is *constant*, the stretch will be bounded by $O(k)$, and the number of edges by $O(k \cdot n^{1+1/k})$.

References

1 Leonid Barenboim and Victor Khazanov. Distributed symmetry-breaking algorithms for congested cliques. *arXiv preprint arXiv:1802.07209*, 2018.

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

3 Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *DISC*, 2017.

4 Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *arXiv preprint arXiv:1802.10297*, 2018.

5 Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P Woodruff. Transitive-closure spanners. *SIAM Journal on Computing*, 41(6):1380–1425, 2012.

6 L Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. *SIAM Journal on Computing*, 42(3):1030–1050, 2013.

7 Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31 International Symposium on Distributed Computing*, 2017.

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

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

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

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

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

13 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. *Manuescript*, 2018.

14 Mohsen Ghaffari, Themis Gouleakis, Slobodan Mitrović, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. *PODC*, 2018.

15 Parikshit Gopalan, Raghu Meka, Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Better pseudorandom generators from milder pseudorandom restrictions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 120–129, 2012.

16 Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners. In *DISC*, 2017.

17 James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science*, 608:268–281, 2015.

18 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *the Proc. of the Int'l Symp. on Princ. of Dist. Comp. (PODC)*, pages 42–50, 2013.

19 Christoph Lenzen and Roger Wattenhofer. Brief announcement: exponential speed-up of local algorithms using non-local communication. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 295–296, 2010.

20 Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. MST construction in $O(\log \log n)$ communication rounds. In *the Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 94–100. ACM, 2003.

21 Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. *arXiv preprint*, 2018. `arXiv:1805.05404`.

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

23 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

24 David Peleg and Jeffrey D Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on computing*, 18(4):740–747, 1989.

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

26 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 2005.

27 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–10. ACM, 2001.

28 Virginia Vassilevska Williams. Graph algorithms – Fall 2016, MIT, lecture notes 5, 2016. URL: `http://theory.stanford.edu/~virgi/cs267/lecture5.pdf`.