Message Reduction in the LOCAL Model Is a Free Lunch

Shimon Bitton
Technion – Israel Institute of Technology, Haifa, Israel
sbitton@technion.ac.il

Yuval Emek
Technion – Israel Institute of Technology, Haifa, Israel
yemek@technion.ac.il

Taisuke Izumi
Nagoya Institute of Technology, Japan
t-izumi@nitech.ac.jp

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel
kutten@ie.technion.ac.il

---- **Abstract** ----

A new *spanner* construction algorithm is presented, working under the *LOCAL* model with unique edge IDs. Given an n-node communication graph, a spanner with a constant stretch and $O(n^{1+\varepsilon})$ edges (for an arbitrarily small constant $\varepsilon > 0$) is constructed in a constant number of rounds sending $O(n^{1+\varepsilon})$ messages whp. Consequently, we conclude that every t-round LOCAL algorithm can be transformed into an $O(t)$-round LOCAL algorithm that sends $O(t \cdot n^{1+\varepsilon})$ messages whp. This improves upon all previous message-reduction schemes for LOCAL algorithms that incur a $\log^{\Omega(1)} n$ blow-up of the round complexity.

1 Introduction

What is the minimum number of messages that must be sent by *distributed graph algorithm* for solving a certain task? Is there a tradeoff between the message and time complexities of such algorithms? How do the message complexity bounds depend on the exact model assumptions? These questions are among the most fundamental ones in distributed computing with a vast body of literature dedicated to their resolution.

A graph theoretic concept that plays a key role in this regard is that of *spanners*. Introduced by Peleg and Ullman [33] (see also [32]), an *α-spanner*, or a spanner with *stretch bound* α, of a connected graph $G = (V, E)$ is a (spanning) subgraph $H = (V, S)$ of G where

the distance between any two vertices is at most α times their distance in G.[1] More general spanners, called (α, β)-*spanners*, are also considered, where the spanner distance between any two nodes is at most α times their distance in G plus an additive β-term ([17]).

Sparse low stretch spanners are known to provide the means to save on message complexity in the *LOCAL* model [27, 31] without a significant increase in the round complexity. This can be done via the following classic simulation technique: Given an n-node communication graph $G = (V, E)$ and a LOCAL algorithm $\mathcal{A}$ whose run $\mathcal{A}(G)$ on G takes t rounds, (1) construct an α-spanner $H = (V, S)$ of G; and (2) simulate each communication round of $\mathcal{A}(G)$ by α communication rounds in H so that a message sent over the edge $(u, v) \in E$ under $\mathcal{A}(G)$ is now sent over a (u, v)-path of length at most α in H. The crux of this approach is that the simulating algorithm executed in stage (2) runs for αt rounds and sends at most $2\alpha t \cdot |S|$ messages. Therefore, if α and $|S|$ are "small", then the simulating algorithm incurs "good" round and message bounds. In particular, the performance of the simulating algorithm does not depend on the number $|E|$ of edges in the underlying graph G.

What about the performance of the spanner construction in the "preprocessing" stage (1) though? A common thread among distributed spanner construction algorithms is that they all send $\Omega(|E|)$ messages when running on graph $G = (V, E)$. Consequently, accounting for the messages sent during this preprocessing stage, the overall message complexity of the aforementioned simulation technique includes a seemingly inherent $\Omega(|E|)$ term. The following research question that lies at the heart of distributed message reduction schemes is therefore left open.

▶ **Question 1.** *Given a LOCAL algorithm $\mathcal{A}$ whose run $\mathcal{A}(G)$ on G takes t rounds, is it possible to simulate $\mathcal{A}(G)$ in $O(t)$ rounds while sending only $O(n^{1+\varepsilon})$ messages for an arbitrarily small constant $\varepsilon > 0$, irrespective of the number $|E|$ of edges in G?*

This question would be resolved on the affirmative if one could design a LOCAL algorithm that constructs an α-spanner $H = (V, S)$ of G with stretch $\alpha = O(1)$ and $|S| = O(n^{1+\varepsilon})$ edges in $O(1)$ rounds sending $O(n^{1+\varepsilon})$ messages. Despite the vast amount of literature on distributed spanner construction algorithms [5, 9–11, 14, 16, 18, 35], it is still unclear if such a LOCAL spanner construction algorithm exists.

Some progress towards the positive resolution of Question 1 has been obtained by Censor-Hillel et al. [8] and Haeupler [22] who introduced techniques for simulating LOCAL algorithms by *gossip* processes. Using this approach, one can transform any t-round LOCAL algorithm into a LOCAL algorithm that runs in $O(t \log n + \log^2 n)$ rounds while sending n messages per round [22]. This transformation provides a dramatic message complexity improvement if one is willing to accept algorithms that run for $\log^{O(1)} n$ many rounds, e.g., if the the bound t on the round complexity of the original algorithm is already in the $\log^{O(1)} n$ range. However, if $t = \log^{o(1)} n$, then the gossip based message reduction scheme of [8, 22] significantly increases the round complexity and this increase seems to be inherent to that technique.

1.1   Definitions and Results

Throughout, we consider a communication network represented by a connected unweighted undirected graph $G = (V, E)$ and denote $n = |V|$. The nodes of G participate in a distributed algorithm under the (fully synchronous) *LOCAL* model [27, 31] with the following two model

---

[1] An equivalent definition requires that it admits a path of length at most α between any two nodes adjacent in G.

assumptions: (i) the nodes know an $O(1)$-approximate upper bound on $\log n$ (equivalently, a poly(n)-approximate upper bound of n) at all times; and (ii) the graph admits *unique edge IDs* so that the ID of an edge is known to both its endpoints at all times.[2] Other than that, the nodes have no a priori knowledge of G's topology. Our main technical contribution is a new algorithm for constructing sparse spanners, called `Sampler`, whose guarantees are cast in the following Theorem.

▶ **Theorem 2.** *Fix integer parameters* $1 \leq k \leq \log \log n$ *and* $0 \leq h \leq \log n$. *Algorithm* `Sampler` *constructs an edge set* $S \subseteq E$ *of size* $|S| \leq \tilde{O}(n^{1+1/(2^{k+1}-1)})$ *such that* $H = (V, S)$ *is an* $O(3^k)$-*spanner of* G *whp.*[3][4] *The round complexity of* `Sampler` *is* $O(3^k h)$ *and its message complexity is* $\tilde{O}(n^{1+1/(2^{k+1}-1)+(1/h)})$ *whp.*

By setting the parameters k and h so that $1/(2^{k+1} - 1) = 1/h = \varepsilon/2$ for an arbitrarily small constant $\varepsilon > 0$ and utilizing the aforementioned spanner based simulation technique, we obtain a message-reduction scheme that transforms any LOCAL algorithm $\mathcal{A}$ whose run on G takes t rounds into a (randomized) LOCAL algorithm that runs in $O(t)$ rounds and sends $\tilde{O}(tn^{1+\varepsilon})$ messages whp. This resolves Question 1 on the affirmative provided that one is willing to tolerate a $1/\text{poly}(n)$ error probability. In fact, we can improve the message reduction scheme even further via the following two-stage process: first, use the α-spanner $H = (V, S)$ constructed by `Sampler` to simulate the run on G of some off-the-shelf LOCAL algorithm that constructs an α'-spanner $H' = (V, S')$ with a better tradeoff between α' and $|S'|$; then, use H' to simulate the run of $\mathcal{A}$ on G. In Section 6, we show that with the right choice of parameters, this two-stage process leads to the following theorem.

▶ **Theorem 3.** *Every distributed task solvable by a t-round LOCAL algorithm can be solved with any one of the following pairs of time and message complexities:*
- $\tilde{O}(tn^{1+2/(2^{\gamma+1}-1)})$ *message complexity and* $O(3^\gamma t + 6^\gamma)$ *round complexity for any* $1 \leq \gamma \leq \log \log n$,
- $\tilde{O}(t^2 n^{1+O(1/\log t)})$ *message complexity and* $O(t)$ *round complexity.*

1.2 Related Work and Discussion

Model Assumptions

The current paper considers the fully synchronous message passing LOCAL model [27, 31] that ignores the message size and focuses only on locality considerations. This model has been studied extensively (at least) since the seminal paper of Linial [27], with special attention to the question of what can be computed efficiently, including some recent interesting developments, see, e.g., the survey in [20, Section 1]. The more restrictive *CONGEST* model [31], where message size is bounded (typically to $O(\log n)$ bits), has also been extensively studied.

Many variants of the LOCAL model have been addressed over the years, distinguished from each other by the exact model assumptions, the most common such assumptions being unique node IDs and knowledge of n. Another important distinction addresses the exact knowledge held by any node v regarding its incident edges when the execution commences.

---

[2] Alternatively, the algorithm can run under the rather common KT_1 model variant [3], where the nodes are associated with unique IDs and each node knows the ID of the other endpoint of each one of its incident edges; see the discussion in Section 1.2.
[3] We say that an event occurs *with high probability*, abbreviated by *whp*, if the probability that it does not occur is at most n^{-c} for an arbitrarily large constant c.
[4] The asymptotic notation $\tilde{O}(\cdot)$ may hide $\log^{O(1)} n$ factors.

Two common choices in this regard are the KT_0 variant, where v knows only its own degree, and the KT_1 variant, where v knows the ID of e's other endpoint for each incident edge e [3]. The authors of [3] advocate KT_1, arguing that it is the more natural among the two model variants, but papers have been published about each of them.

In the current paper, it is assumed that each *edge* $(u, v) \in E$ is equipped with a unique ID, known to both u and v. In general, this assumption lies (strictly) between the KT_0 and KT_1 model variants. Note that the unique edge IDs assumption is no longer weaker than the KT_1 assumption when the communication graph admits *parallel* edges. However, our algorithm and analysis apply also to such graphs (assuming that $|E| \leq n^{O(1)}$) under either of the two assumptions.

Message Complexity $o(|E|)$

As discussed in Section 1.1, the main conceptual contribution of this paper is that on graphs with $m = |E| \gg n$ edges, many distributed tasks can now be solved by sending $o(m)$ messages while keeping the round complexity unharmed. The challenge of reducing the message complexity below $O(m)$ has already received significant attention. In particular, it has been proved in [25] that under the CONGEST KT_0 model, intensively studied *global* tasks, namely, distributed tasks that require $\Omega(D)$ rounds, where D is the graph's diameter (e.g., broadcasting, leader election, etc.), cannot be solved unless $\Omega(m)$ messages are sent (in the worst case).

This is no longer true under more relaxed models. For example, under the LOCAL KT_1 model, DFS and leader election can be solved by sending $O(n)$ and $O(n \log n)$ messages, respectively [24]. This implies similar savings in the number of messages required for most global tasks (trivially, by collecting all the information to the leader). Under the CONGEST KT_1 model, it has been recently proved that a *minimum spanning tree* can be constructed, sending $o(m)$ messages [19, 21, 23, 29].

Restricted graph classes have also been addressed in this regard. In particular, the authors of [26] proved that under the CONGEST KT_0 model, the message complexity of leader election is $O(\sqrt{n} \log^{\frac{3}{2}} n)$ whp in the complete graph and more generally, $O(\tau \sqrt{n} \log^{\frac{3}{2}} n)$ whp in graphs with mixing time $\tau(G)$.

Spanners

Graph spanners have been extensively studied and papers dealing with this fundamental graph theoretic object are too numerous to cite. Beyond the role that sparse spanners play in reducing the message complexity of distributed (particularly LOCAL) algorithms as discussed in Section 1.1, spanners have many applications in various different fields, some of the more relevant ones include synchronization [1, 33], routing [2, 34], and distance oracles [36].

Many existing distributed spanner algorithms have a node collect the topology of the graph up to a distance of some r from itself [9, 12] or employ more sophisticated bounded diameter graph decomposition techniques involving the node's neighborhood [13, 16, 18, 30] such as the techniques presented in [6, 15, 28]. This approach typically requires sending messages over every edge at distance at most r from some subset of the nodes which leads to a large number of messages. Another approach to constructing sparse spanners in a distributed manner is to recursively grow local clusters [4, 5, 7, 10, 35]. Although this approach does not require the (explicit) exploration of multi-hop neighborhoods, the existing algorithms operating this way also admit large message complexity because too many nodes have to explore their 1-hop neighborhoods. Our algorithm `Sampler` is inspired by the algorithm of [5] and adheres to the latter approach, but it is designed in a way that drastically reduces the message complexity – see Section 1.3.

1.3   Techniques Overview

Algorithm `Sampler` employs *hierarchical node sampling*, where a sampled node u in level j of the hierarchy forms the *center* of a cluster that includes (some of) its non-sampled neighbors v. An edge connecting u and v is added to the spanner. The clusters are then contracted into the nodes of the next level $j + 1$. Also added to the spanner are all incident edges of every non-center node that has no adjacent center.

This hierarchical node sampling is used also in the distributed spanner construction of Baswana and Sen [5] and similar recursive clustering techniques were used in other papers as well (see Section 1.2). Common to all these papers is that the centers in each level communicate directly with *all* their neighbors to facilitate the cluster forming task.[5] As this leads inherently to $\Omega(|E|)$ messages, we are forced to follow a different approach, also based on (a different) sampling process.

The first thing to notice in this regard is that it is enough for a non-center node v to find a *single* center u in its neighborhood, so perhaps there is no need for the center nodes to announce their status to *all* their neighbors? Indeed, we invoke an *edge sampling* process in the non-center nodes v that identifies a subset of v's incident edges over which *query* messages are sent. Our analysis shows that (1) the number of query messages is small whp; (2) if v does have an adjacent center, then the edge sampling process finds such a center whp; and (3) if v does not have an adjacent center, then v's degree is small; in this case, all its incident edges are queried and join the spanner whp.

However, this edge sampling idea by itself does not suffice: Note that the graph (in some level of the hierarchical construction) is constructed via cluster contraction (into a single node) in lower levels of the hierarchy. Hence this graph typically exhibits edge multiplicities (even if the original communication graph is simple). This means that some neighbors w of v may have many more (parallel) (w, v)-edges than others. Informally, this can bias the probabilities for finding additional neighbors in the edge sampling process. The key idea in resolving this issue is to run the edge sampling process (in each level) in a carefully designed *iterative* fashion. Intuitively, the first iterations in each level "peels off" the neighbors to which a large fraction of v's incident edges lead. This increases the probability of finding one of the rest of the neighbors in later iterations. Note that once v found a neighbor u, node v can identify all v's edges leading to u (and so "peel them off" from the next iterations), by having u report to v the IDs of all the edges touching u.

1.4   Paper Organization

The rest of the paper is organized as follows. Following some preliminary definitions provided in Section 2, `Sampler` is presented in Section 3 and analyzed in Section 4. For clarity of the exposition, we first present `Sampler` as a centralized algorithm and then, in Section 5, explain how it can be implemented under the LOCAL model with the round and message complexities promised in Theorem 2. Section 6 describes how algorithm `Sampler` is used to obtain the message-reduction schemes we mentioned in the introduction. Finally, we conclude the paper in Section 7.

---

[5] Some of these papers consider weighted communication graphs and as such, have to deal with other issues that are spared in the current paper.

2   Preliminaries

Consider some graph $G = (V, E)$. When convenient, we may denote the node set V and edge set E by $V(G)$ and $E(G)$, respectively. Unless stated otherwise, the graphs considered throughout this paper are undirected and not necessarily simple, namely, the edge set E may include edge multiplicities (a.k.a. *parallel* edges). Given disjoint node subsets $U, U' \subseteq V$, let $E(U)$ denote the subset of edges with (exactly) one endpoint in U and let $E(U, U') = E(U) \cap E(U')$ denote the subset of edges with one endpoint in U and the other in U'. If $U = \{u\}$ and $U' = \{u'\}$ are singletons, then we may write $E(u)$ and $E(u, u')$ instead of $E(\{u\})$ and $E(\{u\}, \{u'\})$, respectively (notice that $E(u, u')$ may contain multiple edges when G is not a simple graph).

Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a collection of non-empty pairwise disjoint node subsets referred to as *clusters* of G.[6] The *cluster graph* (cf. [31]) induced by $\mathcal{C}$ on G, denoted by $G(\mathcal{C})$, is the undirected graph whose nodes are identified with the clusters in $\mathcal{C}$, and the edges connecting nodes C_i and C_j, $1 \leq i \neq j \leq \ell$, correspond to the edges crossing between clusters C_i and C_j in G, that is, the edges in $E(C_i, C_j)$. Observe that $G(\mathcal{C})$ may include edge multiplicities even if G is a simple graph. We denote by $\mathrm{Ind}_G(C)$ the subgraph of G induced by C. When convenient, the term "cluster C applies also to $\mathrm{Ind}_G(C)$[7] For $u, v, \in V$, the distance between u and v in G is denoted by $\mathrm{dist}_G(u, v)$.

As stated in the introduction, the assumption about global parameters is that each node knows an $O(1)$-approximate upper bound on $\log n$. For the sake of simplicity, we treat the algorithm as if each node knows the exact value of $\log n$, however, this is not essential.

3   Constructing an $O(3^k)$-Spanner

In the following argument, let $\delta = 1/(2^{k+1} - 1)$ and $\epsilon = 1/h$ for short. Denote the simple graph input to the algorithm by $G_0 = (V_0, E_0)$. Algorithm `Sampler` (see Pseudocode 1) generates a sequence $G_1, \ldots, G_k$ of graphs, where $G_j = (V_j, E_j)$. Let n_j and m_j be the numbers of nodes and edges in G_j respectively, $N_j(v)$ be the set of neighbors in G_j of node $v \in V_j$, and $E_j(v, u)$ be the set of edges connecting v to u in G_j. The process is executed in an iterative fashion, where in each iteration $j = 0, \ldots, k$, the algorithm constructs a collection $\mathcal{C} \subset 2^{V_j}$ of pairwise disjoint clusters and an edge set $F \subseteq E_j$ that is added to the spanner edge set S (as an exception, in the final iteration of $j = k$, only F is constructed but the cluster collection $\mathcal{C}$ is not created). The graph G_{j+1} is then defined to be the cluster graph $G_j(\mathcal{C})$ induced by $\mathcal{C}$ on G_j. The construction of $\mathcal{C}$ and F is handled by procedure `Cluster`$_j$ that is described soon. To avoid confusion, in what follows, we fix $n = n_0 = |V_0|$.

Procedure `Cluster`$_j$

On input graph G_j ($0 \leq j \leq k$), this procedure constructs the cluster collection $\mathcal{C} \subset 2^{V_j}$ and edge set $F = \cup_{v \in V_j} F_v$ ($F_v \subseteq E_j$) to be added to the spanner edges. The procedure (see Pseudocode 2) consists of two steps. In the first step, each node v tries to identify $\min\{c \exp_n(2^j \delta) \log n, |N_j(v)|\}$ neighbors by an iterative random-edge sampling process[8],

---

[6] Notice that the union of the clusters is not required to be the whole node set V.
[7] Each cluster here will be a connected component.
[8] For ease of writing long exponents, define $\exp_a(x) = a^x$.

■ **Algorithm 1** Sampler.

1: $n \leftarrow |V_0|$; $\delta \leftarrow 1/(2^{k+1} - 1)$; $S \leftarrow \emptyset$
2: **for** $j = 0, \ldots, k$ **do**
3:    $\langle \mathcal{C}, F \rangle \leftarrow \text{Cluster}_j$
4:    $S \leftarrow S \cup F$
5:    **if** $j < k$ **then**
6:       $G_{j+1} \leftarrow G_j(\mathcal{C})$
7: **return** S

---

where c is a sufficiently large constant to guarantee high success probability of the algorithm. For this process, v maintains a set $X_v \subseteq E_j(v)$ of the edges that have not been explored yet. Initially, X_v is set to $X_v = E_j(v)$; the content of X_v is then gradually eliminated by running $2/\epsilon = 2h$ *trials*. In every trial, each node $v \in V_j$ chooses $c^2 \exp_n(2^j\delta + \epsilon) \log^3 n$ edges from X_v independently and uniformly at random (possibly choosing the same edge twice or more). Each chosen edge is said to be a *query* edge. For a neighbor $u \in N_j(v)$ such that $E_j(v, u)$ contains a query edge, we say that u is *queried* by v. For each queried node $u \in N_j(v)$, v adds one arbitrary query edge $e \in E_j(u, v)$ to the edge set F_v, and eliminates all the edges in $E_j(u, v)$ from X_v (Figure 1 (a)-(c)). Then the procedure advances to the next trial unless $|F_v| \geq c \exp_n(2^j\delta) \log n$ holds or X_v is emptied. Let $\hat{N}_j(v) \subseteq N_j(v)$ be the set of the nodes queried by v after finishing $2h$ trials. A node $v \in V_j$ is called *light* if $\hat{N}_j(v) = N_j(v)$ holds, or called *heavy* if $|N_j(v)| > |\hat{N}_j(v)| \geq c \exp_n(2^j\delta) \log n$. It is proved later that every node becomes either light or heavy whp.

In the second step, the algorithm creates (only if $j < k$) the vertex set V_{j+1} by clustering the nodes in V_j. The algorithm marks each node $v \in V_j$ as a *center* w.p. $p_j = \exp_n(-2^j\delta)$. Each node v having a center u contained in $\hat{N}_j(v)$ is merged into u (if two or more centers are contained, an arbitrary one is chosen). A merged cluster corresponds to a node in G_{j+1} (Figure 1 (d)-(f)). As a result, letting $\mathcal{C} \subset 2^{V_j}$ be the clusters inducing the cluster graph G_{j+1}, each cluster $C = C(u) \in \mathcal{C}$ contains exactly one center $u \in V_j$ and some subset of $N_j(u)$. The node which is not merged into any center is said to be an *unclustered* node. Due to some technical reason, all the nodes in G_k are defined to be unclustered. It is shown that every heavy node is merged into some center whp., and that every node in G_k is light (proved later). Thus, every unclustered node is light.

4 Analysis

Throughout this section, we refine the definition of terminology "whp." to claim that the probabilistic event considered in the context holds with probability $1 - 1/n^{\Theta(c)}$ for parameter c defined in the algorithm. With a small abuse of probabilistic arguments, we treat those events as if they necessarily occur (with probability one). Since we only handle a polynomially-bounded number of probabilistic events in the proof, the standard union-bound argument ensures that any consequence of the analysis also holds whp. for a sufficiently large c. We begin the analysis by bounding the number of nodes in graph G_j. Denote $\hat{p}_j = \prod_{0 \leq i \leq j} p_i$. It is easy to check that $\hat{p}_{j-1} = \exp_n(-(2^j - 1)\delta)$ for $1 \leq j \leq k$.

▶ **Lemma 4.** *Graph G_j satisfies $n\hat{p}_{j-1}/2 \leq n_j \leq 3n\hat{p}_{j-1}/2$ whp. for $1 \leq j \leq k$.*

■ **Algorithm 2** Cluster$_j$.

1: **for all** $v \in V_j$ **do**
2:     $F'_v \leftarrow \emptyset$; $F_v \leftarrow \emptyset$; $X_v \leftarrow E_j(v)$; $i \leftarrow 1$
3:     /* First Step */
4:     **while** $(i \leq 2h) \ \wedge \ (|F'_v| < c \exp_n(2^j \delta) \log n) \ \wedge \ (|X_v| \neq \emptyset)$ **do**   ▷ run (at most) $2h$
   trials
5:         **for** $x = 1$ to $c^2 \exp_n(2^j \delta + \epsilon) \log^3 n$ **do**
6:             add an edge e selected uniformly at random from X_v to F'_v
7:         **while** $(F'_v \setminus F_v) \neq \emptyset$ **do** ▷ $F'_v \setminus F_v$ is the set of edges newly added in the current
   trial
8:             Pick an arbitrary edge $e = (v, u) \in F'_v \setminus F_v$
9:             Remove all the edges incident to u from X_v
10:            Remove all the edges incident to u other than e from F'_v
11:            $F_v \leftarrow F_v \cup \{e\}$
12:        $i \leftarrow i + 1$
13: $F \leftarrow \cup_{v \in V_j} F_v$
14: /* Second Step */
15: **if** $j < k$ **then**
16:     **for all** $v \in V_j$ **do**
17:         mark v as a center and create $C(v) = \{v\}$ w.p. $\exp_n(-2^j \delta)$
18:     **for all** non-center $v \in V_j$ **do**
19:         **if** $\exists (v, u) \in F : u$ is a center **then**
20:             $C(u) \leftarrow C(u) \cup \{v\}$
21: **return** $\langle \{C(u) \mid u \text{ is a center}\}, F \rangle$

Proof. The value n_j follows the binomial distribution of n_{j-1} trials and success probability p_{j-1}. Applying Chernoff bound (under conditioning n_{j-1}), the inequality below holds for all $1 \leq j \leq k$ whp.[9]

$$\left(1 - \sqrt{\frac{c \log n}{n_{j-1} p_{j-1}}}\right) n_{j-1} p_{j-1} \leq n_j \leq \left(1 + \sqrt{\frac{c \log n}{n_{j-1} p_{j-1}}}\right) n_{j-1} p_{j-1}.$$

By the definition, n_j and p_j are both non-increasing with respect to j. Hence we have

$$\left(1 - \sqrt{\frac{c \log n}{n_{j-1} p_{j-1}}}\right)^{j-1} n \hat{p}_{j-1} \leq n_j \leq \left(1 + \sqrt{\frac{c \log n}{n_{j-1} p_{j-1}}}\right)^{j-1} n \hat{p}_{j-1}.$$

We prove the lemma inductively (on j) following this inequality. For $j = 1$, $n_{j-1} = n$ and thus $|\sqrt{c \log n / n_{j-1} p_{j-1}}| \leq 1/2$ holds. From the inequality above, we obtain $n_0 p_0 / 2 \leq n_1 \leq 3 n_0 p_0 / 2$. Suppose $n \hat{p}_{j-2} / 2 \leq n_{j-1} \leq 3 n \hat{p}_{j-1} / 2$. Since $n \hat{p}_{j-1} = \exp_n(1 - (2^j - 1)\delta) \geq n^{1/2}$ holds for $j \leq k$, we have $\sqrt{c \log n / (n_{j-1} p_{j-1})} \leq c \log n / n^{1/2} \ll 1/2(j-1)$ for sufficiently large n. Applying the approximation of $(1 + x)^y \approx 1 + xy$ for $|x| \ll 1$ to the inequality, we obtain the lemma.   ◀

---

[9] Chernoff bound for the binomial distribution X of m trials and success probability p is $\mathbb{P}[|X - mp| \geq \alpha mp] \leq 2e^{-\frac{\alpha^2 mp}{3}}$.
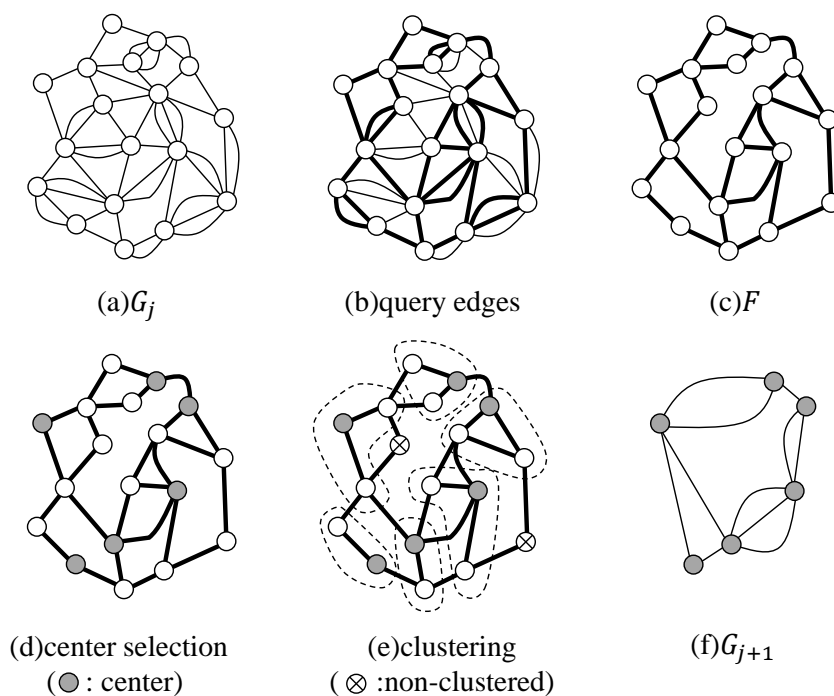
Figure 1 Procedure $\texttt{Cluster}_j$.

We next prove the facts mentioned in the explanation of the procedure $\texttt{Cluster}_j$.

▶ **Lemma 5.** *For any $0 \leq j \leq k - 1$, any heavy node $v \in V_j$ contains at least one center in $\hat{N}_j(v)$ whp.*

Proof. The probability that no center is contained in $\hat{N}_j$ is $(1 - p_j)^{|\hat{N}_j(v)|}$. Since $|\hat{N}_j(v)| \geq c \exp_n(2^j \delta) \log n = c \log n / p_j$ holds for any heavy node v, the probability is at most $1/n^c$. ◀

▶ **Lemma 6.** *For any $0 \leq j \leq k$, any node $v \in V_j$ becomes light or heavy whp. Furthermore, any node $v \in V_k$ becomes light whp.*

Proof. Let $\alpha = (3c \exp_n(2^j \delta) \log^2 n)/h$ for short. For $W \subseteq N_j(v)$, define $E_j(v, W) = \bigcup_{u \in W} E_j(v, u)$. Let $N_j^i(v) \subseteq N_j(v)$ be the set of the nodes not queried by v at the beginning of the i-th trial, and m_i be the numbers of edges in X_v at the beginning of the i-th trial. For any node $u \in N_j^i(v)$, the value of $|E_j(v, u)|$ is called the *volume* of u. Similarly, for any $X \subseteq N_j^i(v)$, we call the value of $|E_j(v, X)|$ the volume of X. Divide $N_j^i(v)$ into $2/\epsilon = 2h$ classes: A node $u \in N_j^i(v)$ belongs to the x-th class $\mathcal{K}_x^i(v)$ if $|E_j(v, u)| \in (\exp_n(x\epsilon), \exp_n((x + 1)\epsilon)]$ holds $(0 \leq x \leq 2h - 1)$. Let $\mathcal{K}^i(v)$ be the maximum-volume class of all at the beginning of the i-th trial. Since the volume of $\mathcal{K}^i(v)$ is at least $m_i/2h$, there exists at least one node $u \in \mathcal{K}^i(v)$ satisfying $|E_j(v, u)| \geq m_i/(2h|\mathcal{K}^i(v)|)$. By the definition of class $\mathcal{K}^i(v)$, it implies that the volume of any node $u \in \mathcal{K}^i(v)$ is at least $m_i/(2h|\mathcal{K}^i(v)|n^\epsilon)$.

Let β be the non-negative integer satisfying $(\beta - 1)\alpha \leq |K^i(v)| \leq \beta\alpha$. Then we consider an arbitrary partition of $\mathcal{K}^i(v)$ into $q = \lfloor |\mathcal{K}^i(v)|/\beta \rfloor$ groups $K_1, K_2, \ldots, K_q$ of size β. Note that βq is not necessarily equal to $|\mathcal{K}^i(v)|$, but the residuals are omitted. Since any node in $\mathcal{K}^i(v)$ has a volume at least $m_i/(2h|\mathcal{K}^i(v)|n^\epsilon)$, the volume of K_ℓ $(1 \leq \ell \leq q)$ is at least

$\beta m_i/(2h|\mathcal{K}^i(v)|n^\epsilon)$. Thus the probability that a query edge is sampled from $E_j(v, K_\ell)$ is at least $\beta/(2h|\mathcal{K}^i(v)|n^\epsilon) \geq 1/(2h\alpha n^\epsilon)$. Letting Z_ℓ be the number of query edges in $E_j(v, K_\ell)$ created at the i-th trial, for any $1 \leq \ell \leq q$, we have

$$\mathbb{P}[Z_\ell = 0] \leq \left(1 - \frac{1}{2h\alpha n^\epsilon}\right)^{c^2 \exp_n(2^j\delta+\epsilon) \log^3 n} \leq \exp_e\left(-\frac{c^2 \exp_n(2^j\delta) \log^3 n}{2h\alpha}\right) \leq e^{-c \log n/6}.$$

Thus, in every trial, at least one node in each group K_ℓ is queried by v whp. If $|\mathcal{K}^i(v)| \leq \alpha$ holds, $\beta = 1$ holds and thus each group consists of a single node in $\mathcal{K}^i(v)$. Thus all nodes in $\mathcal{K}^i(v)$ are queried by v whp. in the i-th trial (note that no node becomes a residual in the case of $\beta = 1$). Otherwise, $q = \lfloor|\mathcal{K}^i(v)|/\beta\rfloor \geq \lfloor(\beta-1)\alpha/\beta\rfloor \geq \alpha/3 = (c \exp_n(2^j\delta) \log^2 n)/h \geq c \exp_n(2^j\delta) \log n$ holds, and thus v queries at least $c \exp_n(2^j\delta) \log n$ nodes in the i-th trial. Consequently, if $|\mathcal{K}^i(v)| \leq \alpha$ holds for all $1 \leq i \leq 2h$, v queries all nodes in $\mathcal{K}^i(v)$ at the i-th trial, that is, queries all nodes in $N_j(v)$ throughout the run of $\texttt{Cluster}_j$. Then v becomes light. If $|\mathcal{K}^i(v)| > \alpha$ holds for some i, v queries at least $c \exp_n(2^j\delta) \log n$ nodes in $\mathcal{K}^i(v)$, which implies v becomes heavy or light.

Finally, let us show that any node $v \in V_k$ is light. Since $n_k \leq 3 \exp_n(1 - (2^k - 1)\delta)/2 = 3 \exp_n(2^k\delta)/2 \leq (3c \exp_n(2^k\delta) \log^2 n)/h = \alpha$ holds from Lemma 4, we have $|N_j(v)| \leq n^k \leq \alpha$. Since $\mathcal{K}^i(v)$ is a subset of $N_j(v)$, $|\mathcal{K}^i(v)| \leq \alpha$ holds for all i. By the argument above, then v is light.   ◄

The rest of the analysis is divided into two parts: First, in Section 4.1, we analyze the stretch of H, proving that it is at most $\kappa = O(3^k)$. Section 4.2 then establishes an $\tilde{O}(n^{1+\delta})$ upper bound on the number of edges in H.

4.1   Bounding the Stretch

The following lemma is a well-known fact.

▶ **Lemma 7** ([32]). *Let $H = (V, X)$ be any (spanning) subgraph of $G = (V, E)$ and $\mathcal{C}$ be the partition of V such that for any $C \in \mathcal{C}$, $\mathrm{Ind}_H(C)$ has a diameter at most ℓ. If X contains at least one edge in $E(C_i, C_j)$ for any pair $(C_i, C_j) \in \mathcal{C}^2$ such that $E(C_i, C_j)$ is nonempty, H is a $(2\ell + 1)$-spanner.*

Let $V_j' \subseteq V_j$ $(1 \leq j \leq k - 1)$ be the set of the nodes unclustered in the run of $\texttt{Cluster}_j$, and $V' = \bigcup_{1 \leq j \leq k-1} V_j'$. We define $C_j(v) \subseteq V$ as the set of nodes in V which are clustered into $v \in V_j$, and also define $r(v)$ as the value j satisfying $v \in V_j'$ for any $v \in V'$. Let $C(v) = C_{r(v)}(v)$ for short.

▶ **Lemma 8.** *Let $H = (V, S)$ be the (spanning) subgraph output by $\texttt{Sampler}$. The diameter of $\mathrm{Ind}_H(C_j(v))$ for any $v \in V_j$ is at most $r = 3^j - 1$.*

Proof. We show that $\mathrm{Ind}_H(C_j(v))$ contains a spanning tree of $\mathrm{Ind}_G(C_j(v))$ with height at most $3^j - 1$. The proof follows the induction on j. For $j = 0$, $\mathrm{Ind}_H(C_j(v)) = \mathrm{Ind}_G(C_j(v))$ is a graph consisting of a single node, and thus its diameter is zero. Suppose as the induction hypothesis that the lemma holds for some j, and consider the case of $j + 1$. Since any node $v \in V_{j+1}$ corresponds to a center node $v \in V_j$ and a star-based connection with its neighbors in V_j, $\mathrm{Ind}_H(C_j(v))$ is obviously contains a spanning tree of $\mathrm{Ind}_G(C_j(v))$ with a diameter at most $3(3^j - 1) + 2 = 3^{j+1} - 1$. The lemma follows.   ◄

Finally the following theorem is deduced.

▶ **Theorem 9.** *The graph $H = (V, S)$ output by* Sampler *is an $O(2 \cdot 3^k - 1)$-spanner of G whp.*

Proof. Since any node in G_k is unclustered, $\mathcal{C} = \{C(v) \mid v \in V'\}$ is a partition of V. If $C(u)$ and $C(v)$ ($u, v \in V'$) are neighboring and $r(u) \leq r(v)$ holds, there exists a node $w \in V_{r(u)}$ such that u and w are neighboring in $G_{r(u)}$ and $C_{r(u)}(w) \subseteq C(v)$ holds. Since every unclustered node is light (by Lemmas 5 and 6), $C(u)$ is light. Then at least one edge in $E(C_{r(u)}(u), C_{r(u)}(w))$ is added to S, which implies that at least one edge in $E(C(u), C(v))$ is added to S. Consequently, the edge set S constructed by Sampler satisfies the condition of Lemma 7 w.r.t. $\mathcal{C}$. The remaining issue is to bound the diameter of $C(v) \in \mathcal{C}$ for all $v \in V'$, which is shown by Lemma 8. ◀

4.2 Bounding the Number of Edges

Using Lemma 4, we can bound the number of edges in S output by Sampler.

▶ **Lemma 10.** *The output edge set S contains $\tilde{O}(n^{1+\delta})$ edges.*

Proof. Each trial of the first step in the run of Cluster$_j$ adds $O(\exp_n(2^j \delta) \log^3 n)$ edges to S per node in V_j, and $n_j = O(\exp_n(1 - (2^j - 1)\delta))$ holds by Lemma 4. Then the total number of edges added to S in Cluster$_j$ is $h \cdot O(\exp_n(2^j \delta) \log^3 n) \cdot n_j = O(hn^{1+\delta} \log^3 n)$, and thus the size of S is $O(khn^{1+\delta} \log^3 n)$. Since $k, h \leq \log n$, we obtain the lemma by omitting all the logarithmic factors. ◀

5 Distributed Implementation

In this section, we explain how the centralized algorithm presented in Section 3 is implemented in a distributed fashion over the (simple) communication graph $G = (V, E)$ with round complexity $O(3^k h)$ and message complexity $\tilde{O}(n^{1+1/(2^{k+1}-1)+(1/h)}) = \tilde{O}(n^{1+\delta+\epsilon})$ whp.

The key observation in this regard is that procedures Cluster$_j$ would have been naturally distributed if the nodes in $V_0, V_1, \ldots, V_k$ could have performed local computations and exchanged messages over their incident edges in $G_0, G_1, \ldots, G_k$, respectively (recall that graphs $G_1, \ldots, G_k$ are virtual, defined only for the sake of the algorithm's presentation). Indeed, the action of marking a node as a center and the action of marking an edge as a query/probe edge are completely local and do not require any communication (in G_j). The action of checking whether a query edge e leads to a center and the action of identifying all the edges parallel to a query/probe edge e are easily implemented under the LOCAL model with unique edge IDs by sending a constant number of messages over edge e in G_j.

So it remains to explain how local actions in graph G_j, $1 \leq j \leq k$, are simulated in the actual communication graph G. Let $T_j(v)$ be the spanning tree of $\text{Ind}_G(C_j(v))$ shown in the proof of Lemma 8. Once a cluster $C_j(v)$ is formed, no further edge is added to the inside of the cluster. Thus $T_j(v)$ is already contained in $C_j(v)$ at the beginning of the run of Cluster$_j$. In the distributed implementation, the local actions of node v in G_j are simulated by nodes $C_j(v)$ in G via a constant number of broadcast-convergecast sessions over $T_j(v)$ rooted at $v \in V$. (This is made possible by the choice of the LOCAL model with unique edge IDs). This process requires sending $O(1)$ (additional) messages over each edge in $T_j(v)$, and by Lemma 8, it takes at most $O(3^j)$ rounds.

▶ **Theorem 11.** *Algorithm* Sampler *has round complexity $O(3^k h)$ and message complexity $\tilde{O}(n^{1+\delta+\epsilon})$ whp.*

Proof. In the first step of $\texttt{Cluster}_j$, each trial takes $O(1)$ rounds in G_j. The second step takes $O(1)$ rounds in G_j. Hence the total running time of $\texttt{Cluster}_j$ takes $O(h(3^j - 1))$ rounds in G. Summing up it over all $0 \le j \le k$, the bound on the round complexity is $\sum_{j=0}^{k} O(h(3^j - 1)) = O(3^k h)$.

For the message complexity, the simulation of one round in G_j is implemented with an additive overhead incurred by a constant-number sessions of broadcast and convergecast in each cluster $C(v)$ for $v \in V_j$, which use $O(n)$ messages in total. Each trial of the first step in $\texttt{Cluster}_j$ uses $O(\exp_n(2^j \delta + \epsilon) \log^3 n)$ messages per node in V_j. Thus the total message complexity in $\texttt{Cluster}_j$ is $O(\exp_n(2^j \delta + \epsilon) \log^3 n) \cdot n_j = O(hn^{1+\delta+\epsilon} \log^3 n)$ by Lemma 4. Summing up this over $0 \le j \le k$, we can conclude that the message complexity is $O(khn^{1+\delta+\epsilon} \log^3 n) = \tilde{O}(n^{1+\delta+\epsilon})$ (recall $k, h \le \log n$). ◀

6   Message-Efficient Simulation of Local Algorithms

In this section, we provide a new and versatile message-reduction scheme for LOCAL algorithms based on the new $\texttt{Sampler}$ algorithm. The technical ingredients of this scheme consist of a message-efficient t-local broadcast algorithm built on top of constructed spanners, which is commonly used in the past message-reduction schemes [8, 22].

Consider the initial configuration where each node $v \in V$ has a message M_v, and let $B_{G,t}(v) = \{u \mid \text{dist}_G(v, u) \le t\}$. The task of the t-local broadcast is that each $v \in V$ delivers M_v to all the nodes $u \in B_{G,t}(v)$. In any t-round LOCAL algorithm, the computation at node $v \in V$ relies only on the initial knowledge (i.e., its ID, initial state, and incident edge set) of the nodes in $B_{G,t}(v)$, and thus any t-local broadcast algorithm in the LOCAL model can simulate any t-round LOCAL algorithm. The core of the scheme is the following theorem.

▶ **Lemma 12.** *There exist two t-local broadcast algorithms respectively achieving the following time and message complexities:*

- $\tilde{O}(tn^{1+2/(2^{\gamma+1}-1)})$ *message complexity and $O(3^\gamma t + 6^\gamma)$ round complexity for any $1 \le \gamma \le \log \log n$ and $t \ge 1$,*
- $\tilde{O}(t^2 n^{1+1/O(t^{1/\log \log t}) \log t})$ *message complexity and $O(t)$ round complexity for $t \ge 1$.*

Proof. Consider the realization of the first algorithm. For any v, all the nodes in $B_{G,t}(v)$ are within αt-hop away from v in any α-spanner. Thus, once we got any α-spanner $H = (V, S)$, the local flooding within distance αt in H trivially implements t-local broadcast. Setting $k = \gamma$ and $h = (2^{\gamma+1} - 1)$ of Theorem 2 implements the spanner satisfying the first condition, where the additive $O(6^\gamma)$ term is the time for spanner construction. For the second algorithm, we utilize the spanner-construction algorithm by Derbel et al. [11] which provides a $(3, O(3^k))$-spanner H with $\tilde{O}(3^k n^{1+1/O(k)})$ edges within $O(3^k)$ rounds for any $k \ge 1$. Consider the algorithm by Derbel et al. with parameter $k = \lceil \log_3 t - \log \log_3 t \rceil$, which results in the $O(t/\log_3 t)$-round algorithm of constructing $(3, O(t))$-spanner with $\tilde{O}(tn^{1+1/O(\log t)})$ edges. We run this algorithm on top of the first simulation scheme with parameter $\gamma = \log_3 \log_3 t$. The simulated algorithm constructs a $(3, O(t))$-spanner H' with $\tilde{O}(tn^{1+1/O(\log t)})$ edges spending $O(3^{\log_3 \log_3 t} \cdot t/\log_3 t + 6^{\log_3 \log_3 t}) = O(t)$ rounds and $\tilde{O}(tn^{1+1/O(\log t)})$ messages. The local flooding within distance $3t + O(t)$ on top of H' implements the t-local broadcast, which takes $O(t)$ rounds and $\tilde{O}(t^2 n^{1+1/O(\log t)})$ messages. The lemma is proved. ◀

As stated above, Theorem 3 is trivially deduced from Lemma 12.

7 Concluding Remarks

In this paper, we present an efficient spanner construction as well as two message-reduction schemes for LOCAL algorithms that preserve the asymptotic time complexity of the original algorithm. The reduced message complexity is close to linear (in n). Is this the best possible in constructing a spanner? Similarly, some open questions still lie on the line of developing efficient message-reduction schemes: (1) While our scheme only sends $\tilde{O}(t^2 n^{1+O(1/\log t)})$ messages for simulating t-round algorithms, it is not clear whether the additive $O(1/\log t)$ term in the exponent can be improved further. Can one have a message-reduction scheme with $\tilde{O}(\text{poly}(t)n^{1+o(1/\log t)})$ message complexity and no overhead in the round complexity? (2) Algorithm `Sampler` inherently relies on randomized techniques for probing the neighbors in G_j using only few messages. Is it possible to obtain a deterministic message-reduction scheme with no degradation of time?

Very recently, the authors received a comment on the first question, which states that utilizing the spanner construction by Elkin and Neiman [16] will improve the message complexity. Unfortunately, due to lack of time, we do not completely check this idea, and thus the current version only states the result based on the algorithm by Derbel et al., but certainly it is a promising approach. If it actually works, the message complexity will be reduced to $\tilde{O}(t^2 n^{1+O(1/t^{1/\log\log t})})$.

Finally, we note that using an $o(m)$ messages spanner construction algorithm that does not increase the time can be useful also for global algorithms in the LOCAL model. It implies that any function can now be computed on the graph in strictly optimal $O(\text{diameter})$ time and $o(m)$ messages (for large enough m).

References

1   Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985. `doi:10.1145/4221.4227`.

2   Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing (SICOMP)*, 28(1):263–277, 1998. `doi:10.1137/S0097539794271898`.

3   Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990. `doi:10.1145/77600.77618`.

4   Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. *Theoretical Computer Science*, 751:2–23, 2018. `doi:10.1016/j.tcs.2016.07.005`.

5   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. `doi:10.1002/rsa.v30:4`.

6   Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-Linear Work Parallel SDD Solvers, Low-Diameter Decomposition, and Low-Stretch Subgraphs. *Theory of Computing Systems*, 55(3):521–554, 2014. `doi:10.1007/s00224-013-9444-5`.

7   Keren Censor-Hillel and Michal Dory. Distributed Spanner Approximation. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 139–148, 2018. `doi:10.1145/3212734.3212758`.

8   Keren Censor-Hillel, Bernhard Haeupler, Jonathan Kelner, and Petar Maymounkov. Global Computation in a Poorly Connected World: Fast Rumor Spreading with No Dependence on Conductance. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 961–970, 2012. `doi:10.1145/2213977.2214064`.

9   Bilel Derbel and Cyril Gavoille. Fast Deterministic Distributed Algorithms for Sparse Spanners. *Theoretical Computer Science*, 399(1):83–100, 2008. `doi:10.1016/j.tcs.2008.02.019`.

10   Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the Locality of Distributed Sparse Spanner Construction. In *Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–282, 2008. `doi:10.1145/1400751.1400788`.

11   Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. Local Computation of Nearly Additive Spanners. In *Proceedings of 23rd International Symposium on Distributed Computing (DISC)*, pages 176–190, 2009. `doi:10.1007/978-3-642-04355-0_20`.

12   Devdatt Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Aravind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. *Journal of Computer and System Sciences*, 71(4):467–479, 2005. `doi:10.1016/j.jcss.2005.04.002`.

13   Michael Elkin. Computing Almost Shortest Paths. *ACM Transactions on Algorithms (TALG)*, 1(2):283–323, 2005. `doi:10.1145/1103963.1103968`.

14   Michael Elkin. A near-optimal distributed fully dynamic algorithm for maintaining sparse spanners. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing (PODC)*, pages 185–194, 2007. `doi:10.1145/1281100.1281128`.

15   Michael Elkin and Ofer Neiman. Distributed Strong Diameter Network Decomposition: Extended Abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 211–216, 2016. `doi:10.1145/2933057.2933094`.

16   Michael Elkin and Ofer Neiman. Efficient Algorithms for Constructing Very Sparse Spanners and Emulators. *ACM Transactions on Algorithms (TALG)*, 15(1):4:1–4:29, 2018. `doi:10.1145/3274651`.

17   Michael Elkin and David Peleg. $(1 + \epsilon, \beta)$-Spanner Constructions for General Graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004. `doi:10.1137/S0097539701393384`.

18   Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1 + \epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006. `doi:10.1007/s00446-005-0147-2`.

19   Mohsen Ghaffari and Fabian Kuhn. Distributed MST and Broadcast with Fewer Messages, and Faster Gossiping. In *Proceedings of 32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 30:1–30:12, 2018. `doi:10.4230/LIPIcs.DISC.2018.30`.

20   Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 784–797, 2017. `doi:10.1145/3055399.3055471`.

21   Robert Gmyr and Gopal Pandurangan. Time-Message Trade-Offs in Distributed Algorithms. In *Proceedings of 32nd International Symposium on Distributed Computing, (DISC)*, volume 121, pages 32:1–32:18, 2018. `doi:10.4230/LIPIcs.DISC.2018.32`.

22   Bernhard Haeupler. Simple, Fast and Deterministic Gossip and Rumor Spreading. *Journal of the ACM (JACM)*, 62(6):47:1–47:18, December 2015. `doi:10.1145/2767126`.

23   Valerie King, Shay Kutten, and Mikkel Thorup. Construction and impromptu repair of an MST in a distributed network with $o(m)$ communication. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 71–80, 2015. `doi:10.1145/2767386.2767405`.

24   Ephraim Korach, Shay Kutten, and Shlomo Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):84–101, 1990. `doi:10.1145/323596.323611`.

25   Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the Complexity of Universal Leader Election. *Journal of the ACM (JACM)*, 62(1):7:1–7:27, 2015. `doi:10.1145/2699440`.

26 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theoretical Computer Science*, 561:134–143, 2015. `doi:10.1016/j.tcs.2014.02.009`.

27 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing (SICOMP)*, 21(1):193–201, 1992. `doi:10.1137/0221015`.

28 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993. `doi:10.1007/BF01303516`.

29 Ali Mashreghi and Valerie King. Broadcast and minimum spanning tree with $o(m)$ messages in the asynchronous CONGEST model. In *Proceedings of 32nd International Symposium on Distributed Computing (DISC)*, pages 37:1–37:17, 2018. `doi:10.4230/LIPIcs.DISC.2018.37`.

30 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved Parallel Algorithms for Spanners and Hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 192–201, 2015. `doi:10.1145/2755573.2755574`.

31 David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. `doi:10.1137/1.9780898719772`.

32 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. `doi:10.1002/jgt.3190130114`.

33 David Peleg and Jeffrey D. Ullman. An Optimal Synchronizer for the Hypercube. *SIAM Journal on Computing (SICOMP)*, 18(4):740–747, 1989. `doi:10.1137/0218050`.

34 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM (JACM)*, 36(3):510–530, 1989. `doi:10.1145/62212.62217`.

35 Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010. `doi:10.1007/s00446-009-0091-7`.

36 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005. `doi:10.1145/1044731.1044732`.