On the Power of Graphical Reconfigurable Circuits

Yuval Emek
Technion - Israel Institute of Technology, Haifa, Israel

Yuval Gil
Technion - Israel Institute of Technology, Haifa, Israel

Noga Harlev
Technion - Israel Institute of Technology, Haifa, Israel

Abstract

We introduce the *graphical reconfigurable circuits (GRC)* model as an abstraction for distributed graph algorithms whose communication scheme is based on local mechanisms that collectively construct long-range reconfigurable channels (this is an extension to general graphs of a distributed computational model recently introduced by Feldmann et al. (JCB 2022) for hexagonal grids). The crux of the GRC model lies in its modest assumptions: (1) the individual nodes are computationally weak, with state space bounded independently of any global graph parameter; and (2) the reconfigurable communication channels are highly restrictive, only carrying information-less signals (a.k.a. *beeps*). Despite these modest assumptions, we prove that GRC algorithms can solve many important distributed tasks efficiently, i.e., in polylogarithmic time. On the negative side, we establish various runtime lower bounds, proving that for other tasks, GRC algorithms (if they exist) are doomed to be slow.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases graphical reconfigurable circuits, bounded uniformity, beeping

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.22

Related Version *Full Version*: `https://arxiv.org/pdf/2408.10761` [6]

1 Introduction

The *reconfigurable circuits* model was introduced recently by Feldmann et al. [7] and studied further by Padalkin et al. [14, 13]. It extends the popular *geometric amoebot* model for (synchronous) distributed algorithms running in the hexagonal grid by providing them with an opportunity to form long-range communication channels. This is done by means of a distributed mechanism that allows each node to bind together a subset of its incident edges (which can be thought of as installing internal "wires" between the corresponding ports); the long-range channels, a.k.a. *circuits*, are then formed by taking the transitive closure of these local bindings (see Sec. 1.1 for details). The circuits serve as *beeping channels*, enabling their participating nodes to communicate via information-less signals. The crux of the model is that the distributed mechanism that controls the circuit formation is invoked in every round (of the synchronous execution) so that the circuits can be reconfigured.

In contrast to the original geometric amoebot model which is tailored specifically to planarly embedded (hexagonal) grids, the reconfigurable circuits model can be naturally generalized to arbitrary graph topologies. The starting point of the current paper is the formulation of such a generalization that we refer to as the *graphical reconfigurable circuits (GRC)* model (formally defined in Sec. 1.1).

An important feature of the GRC model is that it is *uniform*: the actions of each node v in the (general) communication graph G are dictated by a (possibly randomized) state machine whose description is fully determined by the degree of v (and the local input provided to v if there is such an input), independently of any global parameter of G [2]. A clear advantage of

uniform algorithms is that they can be deployed in a "one size fits all" fashion, without any global knowledge of the graph on which they run. We further require that the aforementioned state machines admit a finite description, which means, in particular, that the state space of the state machines are bounded independently of any global graph parameter. This requirement is an obvious necessary condition for practical implementations; we subsequently refer to uniform distributed algorithms subject to this requirement as *boundedly uniform*.

Combining the bounded uniformity with the light demands of the beeping communication scheme, demands which are known to be easy to meet in practice [4, 8], we conclude that the GRC model provides an abstraction for distributed (arbitrary topology) graph algorithms that can be implemented over devices with slim computation and communication capabilities. In particular, the GRC model may open the gate for a rigorous investigation of distributed algorithms operating in (natural or artificial) biological cellular networks whose communication mechanism is based on bioelectric signaling, known to be the basis for long range (low latency) communication in such networks.

The main technical contribution of this paper is the design of GRC algorithms for various classic distributed tasks that terminate in polylogarithmic time. Some of these tasks (e.g., the construction of a minimum spanning tree) are inherently global and are known to be subject to congestion bottlenecks, thus demonstrating that despite their limited computation and communication power, GRC algorithms can overcome both "locality" and "bandwidth" barriers. In fact, as far as we know, these are the first distributed algorithms that solve such tasks in polylogarithmic time under any boundedly uniform model.

While GRC algorithms can bypass the congestion bottlenecks of some distributed tasks, other tasks turn out to be much harder: We prove that under certain conditions, runtime lower bound constructions, developed originally for the CONGEST model [15], can be translated, almost directly, to the GRC model, thus establishing runtime lower bounds for a wide class of tasks.

1.1 The GRC Model

In the current section, we introduce the distributed computational model used throughout this paper, referred to as the *graphical reconfigurable circuits (GRC)* model. A GRC algorithm `Alg` runs over a (finite simple) undirected graph $G = (V, E)$ so that each node $v \in V$ is associated with its own copy of a (possibly randomized) state machine defined by `Alg`; for clarity of the exposition, we often address node v and the state machine that dictates v's actions as the same entity (our intention will be clear from the context).

We adopt the *port numbering* convention [2, 10] stating that from the perspective of a node $v \in V$, each edge $e \in E(v)$ is identified by a unique port number taken from the set $\{1, \ldots, \deg(v)\}$.[1] Every edge $e \in E$ is associated with k *pins*, where $k \in \mathbb{Z}_{>0}$ is a constant determined by the algorithm designer;[2] these pins are represented as pairs of the form $p = (e, i)$ for $i \in [k]$. Let $P = E \times [k]$ denote the set of all pins. For a node $v \in V$, let $P(v) = E(v) \times [k]$ denote the set of pins associated with the edges incident on v. The GRC model is defined so that for each pin $p = (e, i) \in P(v)$, node v is aware of the (local) port

[1] Given an edge subset $F \subseteq E$ and a node $v \in V$, we denote the set of edges in F incident on v by $F(v) = \{e \in F \mid e \ni v\}$ and the degree of v by $\deg(v) = |E(v)|$.

[2] For the (asymptotic) upper bounds established in the current paper, it is actually sufficient to use $k = 1$ pins per edge. However, this is not true in general (see, e.g., [7, Sections 3.4 and 4.4]) and regardless, using multiple (yet, $O(1)$) pins per edge often facilitates the algorithm's exposition. In any case, we do not make an effort to optimize the value of k.

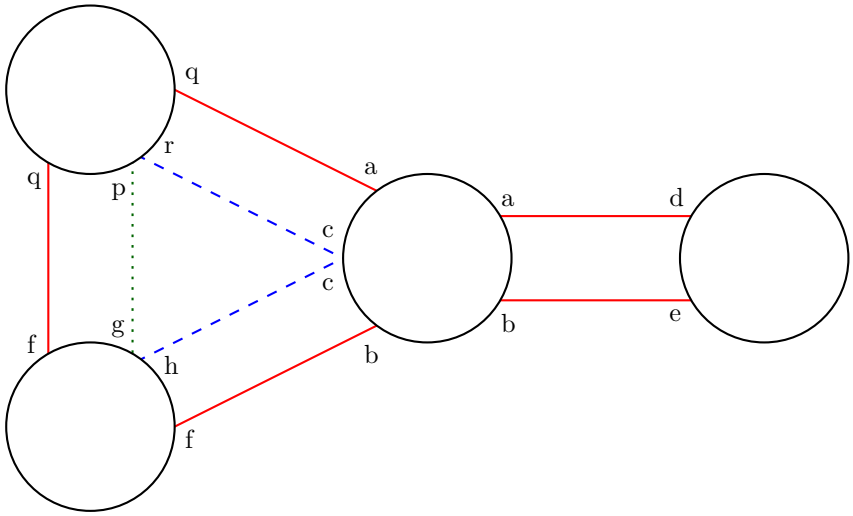

Figure 1 The circuits formed on a communication graph by the local node decisions. The graph includes 4 nodes, depicted by the black cycles, and 4 edges (not shown explicitly in the figure), each one of them is associated with $k = 2$ pins, depicted by the straight lines. The local pin partitions are presented by the lower-case letters. These local pin partitions result in forming three circuits, consisting of the red (solid) pins, the blue (dashed) pins, and the green (dotted) pin.

number of edge e as well as the (global) index $i \in [k]$. In particular, the other endpoint of edge e agrees with v on the index i of pin p although the two nodes may identify e by different port numbers.

The execution of algorithm `Alg` advances in synchronous *rounds*. Each round $t = 0, 1, \ldots$ is associated with a partition $\mathcal{C}^t$ of the pin set P into non-empty pairwise disjoint parts, called *circuits*. The partition $\mathcal{C}^0$ is defined so that each pin forms its own singleton circuit; for $t \geq 1$, the partition $\mathcal{C}^t$ is determined by the nodes according to a distributed mechanism explained soon.

For a round $t \geq 0$, a node $v \in V$ is said to *partake* in a circuit $C \in \mathcal{C}^t$ if $P(v) \cap C \neq \emptyset$. Let $\mathcal{C}^t(v) = \{C \in \mathcal{C}^t \mid P(v) \cap C \neq \emptyset\}$ denote the set of circuits in which node v partakes.

The communication scheme of the GRC model is defined on top of the circuits so that each circuit $C \in \mathcal{C}^t$ serves (during round t) as a *beeping channel* [4] for the nodes that partake in C. Before getting into the specifics of this communication scheme, let us explain how the partition $\mathcal{C}^t$ is formed based on the actions of the nodes in round $t - 1$.

Fix some round $t \geq 1$. Towards the end of round $t - 1$, each node $v \in V$ decides on a partition $\mathcal{R}^t(v)$ of $P(v)$, referred to as the *local pin partition* of v. Let $\mathcal{L}^t$ be the symmetric binary relation over P defined so that pins $p = (e, i)$ and $p' = (e', i')$ are related under $\mathcal{L}^t$ (i.e., $(p, p'), (p', p) \in \mathcal{L}^t$) if and only if there exists a node $v \in V$ (incident on both e and e') such that p and p' belong to the same part of $\mathcal{R}^t(v)$. Let $\mathrm{tc}(\mathcal{L}^t)$ be the reflexive transitive closure of $\mathcal{L}^t$, which is, by definition, an equivalence relation over P. The circuits in $\mathcal{C}^t$ are taken to be the equivalence classes of $\mathrm{tc}(\mathcal{L}^t)$. See Figure 1 for an illustration.[3]

[3] As presented by Feldmann et al. [7], the physical interpretation of the abstract circuit forming process is that each node v internally "wires" all pins belonging to the same part $R \in \mathcal{R}^t(v)$ to each other, thus ensuring that a signal transmitted over one pin in R is disseminated to all pins in R (and through them, to the entire circuit that contains R).

We are now ready to formally define the operation of each node $v \in V$ in round $t = 0, 1, \ldots$ This includes the following three steps, where we denote the state of v in round t by $S^t(v)$:
(1) Node v decides (possibly in a probabilistic fashion), based on $S^t(v)$, on a pin subset $B^t(v) \subseteq P(v)$ and *beeps* – namely, emits an information-less signal – on every pin in $B^t(v)$; we say that v *beeps* on a circuit $C \in \mathcal{C}^t(v)$ if v beeps on (at least) one of the pins in C.
(2) For each pin $p \in P(v)$, node v obtains a bit of information revealing whether at least one node beeps (in the current round) on the (unique) circuit $C \in \mathcal{C}^t$ to which p belongs.
(3) Node v decides (possibly in a probabilistic fashion), based on $S^t(v)$ and the information obtained in step (2), on the next state $S^{t+1}(v)$ and the next local pin partition $\mathcal{R}^{t+1}(v)$.
We emphasize that for each circuit $C \in \mathcal{C}^t(v)$ and pin $p \in P(v) \cap C$, node v can distinguish, based on the information obtained in step (2) for p, between the scenario in which zero nodes beep on C and the scenario in which a positive number of nodes beep on C, however, node v cannot tell how large this positive number is. In fact, if v itself decides (in step (1)) to beep on pin p, then v does not obtain any meaningful information from p in step (2) (in the beeping model terminology [1], this is referred to as lacking "sender collision detection").[4]

An important feature of the GRC model is that `Alg` is required to be *boundedly uniform*, namely, the number of states in the state machine associated with a node $v \in V$, as well as the description of the transition functions that determine the next state $S^{t+1}(v)$ and the next local pin partition $\mathcal{R}^{t+1}(v)$, are finite and fully determined by the local parameters of v, independently of any global parameter of the graph G on which `Alg` runs. These local parameters include the degree $\deg(v)$ of v and, depending on the specific task, any local input provided to v at the beginning of the execution (e.g., the weights of the edges incident on v).[5] In particular, node v does not "know" (and generally, cannot encode) the number $n = |V|$ of nodes, the number $m = |E|$ of edges, the maximum degree $\Delta = \max_{v \in V} \deg(v)$, or the diameter $D = \max_{u,v \in V} d_G(u, v)$.[6] Notice that the uniformity in n means that the nodes are also *anonymous*, i.e., they do not (and cannot) have unique identifiers.

The primary performance measure applied to our algorithms is their *runtime* defined to be the number of rounds until termination. When the algorithm is randomized, its runtime may be a random variable, in which case we aim towards bounding it whp.[7]

Relation to CONGEST. An adversity faced by GRC algorithms is the limited amount of information that can be sent/received by each node in a single round. Such limitations lie at the heart of the popular *CONGEST* [15] model that operates in synchronous message passing rounds, using messages of size B, where the typical choices for B are $B = O(1)$, $B = \Theta(\log n)$, or $B = \text{polylog}(n)$ (by definition, the uniform version of CONGEST adopts the former choice). An important point of similarity between the two models is that per round, both CONGEST and GRC algorithms can communicate $\tilde{O}(s)$ bits of information over a cut of

[4] The reader may wonder why the decisions made in step (1) and the information obtained in step (2) are centered on the pins in $P(v)$, rather than on the circuits in $\mathcal{C}^t(v)$. The reason is that node v is not necessarily aware of the partition induced on $P(v)$ by $\mathcal{C}^t(v)$ (i.e., the exact assignment of the pins in $P(v)$ to the circuits in $\mathcal{C}^t(v)$); indeed, the latter partition depends on the local pin partitions $\mathcal{R}^t(u)$ of other nodes $u \in V$, some of which may be far away from v. For example, in Figure 1, the local pin partition of the rightmost node separates between its two incident pins; nevertheless, both pins belong to the same (red) circuit due to local pin partitions decided upon in the other side of the graph.
[5] To maintain strict uniformity, we adhere to the convention that numerical values included in the local inputs (e.g., edge weights) are encoded as bitstrings without "leading zeros", thus ensuring that the length of such a bitstring by itself does not reveal any global information.
[6] The notation $d_G(u, v)$ denotes the distance (in hops) between nodes u and v in G.
[7] An event A holds *with high probability (whp)* if $\mathbb{P}(A) \geq 1 - n^{-c}$ for an arbitrarily large constant c.

Table 1 Our runtime upper bounds. The corresponding GRC algorithms are randomized and their correctness and runtime guarantees hold whp; the one exception is the spanner construction, where the number of edges is bounded in expectation.

	task	runtime
construction	minimum spanning tree (integral edge weights $\in [1, W]$)	$O(\log(n) \cdot \log(n + W))$
	$(2\kappa - 1)$-spanner with $O(n^{1+(1+\varepsilon)/\kappa})$ edges in expectation	$O(\kappa + \log n)$
verification	minimum spanning tree (integral edge weights $\in [1, W]$)	$O(\log(n) \cdot \log(n + W))$
	simple path, connectivity, (s, t)-connectivity, connected spanning subgraph, cut, (s, t)-cut, Hamiltonian cycle, e-cycle containment, edge on all (s, t)-paths	$O(\log n)$

size s.[8] As explained in Sec. 3, from the perspective of message exchange per se (regardless of local computation), T CONGEST rounds can be simulated by $O(\log n + T \cdot B)$ GRC rounds whp, so, ignoring the additive logarithmic term, GRC algorithms are at least as strong as the boundedly uniform version of CONGEST algorithms. In fact, they are strictly stronger: the crux of GRC algorithms is that they enjoy the advantage of reconfigurable long-range communication channels (though highly restrictive ones); this advantage materializes in some of the GRC algorithms developed in the sequel whose runtime is significantly smaller than their corresponding (not necessarily uniform) CONGEST lower bounds.

1.2 Our Contribution

The main takeaway from this paper is that many important distributed tasks admit highly efficient GRC algorithms – see Table 1. Notice that with the exception of the sparse spanner construction, all tasks mentioned in Table 1 admit $\tilde{\Omega}(\sqrt{n} + D)$ runtime lower bounds under the (not necessarily uniform) CONGEST model [17, 16], demonstrating that reconfigurable beeping channels are a powerful tool even for boundedly uniform algorithms.

The polylogarithmic runtime upper bounds presented in Table 1 imply that the $\tilde{\Omega}(\sqrt{n}+D)$ CONGEST lower bounds for the corresponding tasks fail to transfer to the GRC model (refer to the full version [6] for further discussion of this "failed transfer"). CONGEST lower bounds for other distributed tasks on the other hand do transfer, almost directly, to GRC. Indeed, we develop a generic translation, from CONGEST runtime lower bounds to GRC runtime lower bounds, which applies to a large class of CONGEST lower bound constructions.

1.3 Paper's Outline

The remainder of this paper is organized as follows. We start in Sec. 2 with a discussion of the main technical challenges encountered towards establishing our results and the ideas used to overcome them. Sec. 3 introduces some preliminary definitions, as well as several basic procedures used in the later technical sections. The GRC algorithms promised in Table 1 for the tasks of constructing a minimum spanning tree and a spanner are presented and analyzed in Sec. 4 and 5, respectively. (Throughout, missing proofs and constructions are deferred to the full version [6].)

[8] The asymptotic notations $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ hide polylog(n) expressions.

2 Technical Overview

In this section, we discuss the different challenges that arise in our constructions and present a brief overview of the technical ideas used to overcome these challenges; see Sec. 4 and 5 for the full details.

Minimum Spanning Tree. The minimum spanning tree (MST) construction follows the structure of Boruvka's classic algorithm [3]. The algorithm maintains a partition of the node set into *clusters* that correspond to the connected components of the subgraph induced by the edges which were already selected for the MST. It operates in phases, where the main algorithmic task in a phase is to identify a *lightest outgoing edge* for each cluster. The clusters are then merged over the identified edges, adding those edges to the output edge set.

If the edge weights are distinct, then no cycles are formed by the cluster merging process and Boruvka's algorithm is guaranteed to return an MST of the original graph. This well known fact is utilized by the existing distributed implementations of Boruvka's algorithm that typically use the unique node IDs to "enforce" distinct edge weights.

Unfortunately, obtaining distinct edge weights under our boundedly uniform model is hopeless. This means that the set L of lightest outgoing edges (of all clusters) cannot be safely added to the output edge set without the risk of forming cycles, thus forcing us to come up with an alternative mechanism. The key technical idea here is a procedure that runs in each phase independently and constructs (whp) a *total order* $\mathcal{T}$ over the set L. Following that, we identify a $\mathcal{T}$-minimal outgoing edge for each cluster and perform the cluster merger over the identified edges. As we prove in Sec. 4, selecting the $\mathcal{T}$-minimal outgoing edges ensures that no cycles are formed, resulting in a valid MST. Notice that for this argument to work, it is crucial that $\mathcal{T}$ is defined *globally* over all edges in L which is ensured by a careful design of the aforementioned procedure.

Spanner. The spanner construction is based on the elegant random shifts method of [12]. Particularly, the idea is similar to the distributed algorithm of [9] that uses random shifts to obtain a $(2\kappa-1)$-spanner of expected size $O(n^{1+1/\kappa})$. The heart of the random shift method is a probabilistic clustering process based on a random variable δ_v drawn independently by each node $v \in V$. Specifically, in [9], each node $v \in V$ samples δ_v from the capped geometric distribution (see Sec. 3 for a definition) with parameters $p = 1 - n^{-1/\kappa}$ and $r = \kappa - 1$. The main challenge of adapting the algorithm to the boundedly uniform GRC model lies in the fact that the nodes are unable to sample from a distribution whose parameters depend on n. Nevertheless, we present a sampling procedure that allows each node $v \in V$ to sample δ_v from a distribution that is *sufficiently close* to the aforementioned capped geometric distribution.

As we prove in Sec. 5, the sampling procedure allows us to construct a spanner with nearly the same properties as those of [9]. More concretely, we extend and adapt the analysis of [9] to show that our algorithm constructs a spanner with stretch $2\kappa - 1$ whp, and size $O(n^{1+(1+\varepsilon)/\kappa})$ in expectation, where $\varepsilon > 0$ is a constant parameter that can be made desirably small.

3 Preliminaries

Graph Theoretic Definitions. Consider a connected graph $G = (V, E)$. Given an edge-weight function $w : E \to \mathbb{R}$, a *minimum spanning tree (MST)* of G with respect to w is an edge subset $T \subseteq E$ such that (V, T) is a spanning tree of G that minimizes the weight $w(T) = \sum_{e \in T} w(e)$.

For an edge subset $H \subseteq E$, let $d_H(u,v)$ denote the distance in the graph (V,H) between two nodes $u,v \in V$. For an integer $\sigma > 0$, we say that $H \subseteq E$ is a *σ-spanner* of G if $d_H(u,v) \leq \sigma \cdot d_G(u,v)$ for all $u,v \in V$. Equivalently, H is a σ-spanner if and only if $d_H(u,v) \leq \sigma$ for every edge $(u,v) \in E$. The *stretch* of H is defined as the smallest value σ for which H is a σ-spanner.

The parts of a partition $\mathcal{P}$ of the node set V are often referred to as *clusters*. We say that clusters S and S', $S \neq S'$, are *neighboring clusters* if there exists an edge $(v,v') \in E$ such that $v \in S$ and $v' \in S'$. In this case, we say that edge (v,v') *bridges* the clusters S and S', and more broadly, refer to (v,v') as a *bridging* edge of $\mathcal{P}$. We say that an edge $(u,v) \in E$ is an *outgoing* edge of cluster S if $u \in S$ and $v \notin S$. For a cluster S, let $\partial_S \subseteq E$ denote the set of edges outgoing from S.

Capped Geometric Distribution. For parameters $p \in [0,1]$ and $r \in \mathbb{Z}_{>0}$, the *capped geometric distribution*, denoted by $GeomCap(p,r)$, is defined by taking $\mathbb{P}[GeomCap(p,r) = i]$ to be $p(1-p)^i$ if $i \in \{0,\ldots,r-1\}$; $(1-p)^r$ if $i = r$; and 0 otherwise. Intuitively, the distribution relates to r Bernoulli experiments indexed by $0,\ldots,r-1$, each with success probability p. A random variable sampled from the capped geometric distribution represents the index of the first successful experiment, whereas it is equal to r if all experiments fail. The capped geometric distribution admits a memoryless property for the values $0 \leq i \leq r-1$. In particular, a useful identity that follows is $\mathbb{P}[X = i \mid X \geq i] = \mathbb{P}[X = 0] = p$ for a random variable $X \sim GeomCap(p,r)$ and an index $0 \leq i \leq r-1$.

3.1 Auxiliary Procedures

Global Circuits. The algorithms presented in this paper utilize a *global circuit*, i.e., a circuit in which every node $v \in V$ partakes. A global circuit can be constructed in round $t \geq 0$ as follows. For some index $1 \leq i \leq k$, every node $v \in V$ partitions its pin set in round t such that $E(v) \times \{i\} \in \mathcal{R}^t(v)$.

Procedure `CountingToLogn`. We next present a procedure referred to as `CountingToLogn`, whose runtime is $\Theta(\log n)$ rounds whp. While the uniformity in n prevents the nodes from counting $\log n$ rounds individually, the duration of this procedure can indicate to the nodes that whp, $\Theta(\log n)$ rounds have passed. The nodes first construct a global circuit, as described above. Throughout the procedure, the nodes maintain a node set $M \subseteq V$ of *competitors*, where initially $M = V$. In each round, each competitor $v \in M$ tosses a fair coin and beeps through the global circuit if the coin lands heads. If the coin lands tails, v removes itself from M. The procedure terminates when no competitor beeps through the global circuit.

We show the following useful property regarding the runtime of the described procedure.

▶ **Lemma 3.1.** *For an integer $r > 0$, consider $2r-1$ independent executions of* `CountingToLogn` *and let τ be the median runtime of these executions (i.e., the r-th fastest runtime). For any constant $0 < \rho < 1$, it holds that $\mathbb{P}[(1-\rho)\log n \leq \tau \leq (1+\rho)\log n] \geq 1 - 2n^{-\rho r}$.*

Simulating a Message-Passing Network. In a *message-passing* network, in each round, every pair of neighboring nodes may exchange single bit messages with each other (cf. the CONGEST(1) model [15]). One can simulate a message-passing network in the GRC model using relatively standard techniques as cast in the following theorem.

▶ **Theorem 3.2.** *Let* `Alg` *be a GRC algorithm where additionally, in each round, each node is able to exchange* 1*-bit messages with its neighbors. If the runtime of* `Alg` *is* T*, then it can be transformed into an algorithm* `Alg`$'$ *in the GRC model (without messages between neighbors) with a runtime of* $O(\log n) + 4T$ *whp.*

For simplicity of presentation, we subsequently utilize Thm. 3.2 and describe our algorithms as if the nodes can exchange 1-bit messages with their neighbors in each round.

Leader Election. In the leader election task, the goal is for a single node in a given node set $I \subseteq V$ to be selected as a *leader*, whereas all other nodes of I are selected to be *non-leaders*. Leader election is used as a procedure in some of our algorithms. To that end, we use a leader election algorithm presented by Feldmann et al. [7] in the context of reconfigurable circuits in the geometric amoebot model. We note that this leader election algorithm only uses a global circuit (as described above) and thus can be applied as-is in the GRC model. Hence, the following theorem is established.

▶ **Theorem 3.3** ([7])**.** *The leader election task can be solved within* $O(\log n)$ *rounds whp.*

Outgoing Edge Detection. Consider a graph $G = (V, E)$ and let $H \subseteq E$ be a subset of edges such that each node $v \in V$ knows the set of incident edges $H(v)$. Define a partition of V into clusters according to the connected components of (V, H). The objective of this procedure is for each node $v \in V$ to determine for each neighbor $u \in N(v)$, whether u belongs to the same cluster as v. To that end, the nodes first construct a circuit for each cluster. This is done by each node $v \in V$ including the pin subset $H(v) \times \{i\}$ as part of its local pin partition for some $i \in [k]$ (i is the same for all nodes). Then, each cluster elects a leader utilizing the leader election algorithm mentioned above. The selected leader of each cluster tosses $\Theta(\log n)$ bits and beeps them through the cluster's circuit, one at a time (a beep represents 1 and silence represents 0). Since the nodes cannot count $\Theta(\log n)$ rounds, Proc. `CountingToLogn` is executed in parallel through a global circuit for (a sufficiently large) $c > 1$ times, indicating to the clusters' leaders how long to continue with the bit tossing process. Every node $v \in V$ sends every bit received through its cluster's circuit in a direct message to all its neighbors (messages between neighbors are executed by means of the simulation method described in Sec. 3.1). For every incident edge $e \in E(v)$, node v checks if the bit received differs from the bit sent. If so, e is classified by v as an outgoing edge.

▶ **Lemma 3.4.** *In the outgoing edge detection procedure, every edge* $e = (u, v) \in E$ *is classified correctly whp by both* u *and* v*.*

▶ **Lemma 3.5.** *The outgoing edge detection procedure takes* $\Theta(\log n)$ *rounds whp.*

4 A Fast Minimum Spanning Tree Algorithm

In this section, we present a randomized MST algorithm that operates in the GRC model. As common in the distributed setting, we assume the edge-weights are integers from the set $\{1, \ldots, W\}$ for some positive integer W. Each node $v \in V$ initially knows only the weights of edges in $E(v)$. In particular, as dictated by the GRC model, node v does not know the value of W or any other information about W.

Our algorithm can be seen as an adaptation of Boruvka's classical MST algorithm [3] to the GRC model. Throughout its execution, Boruvka's algorithm maintains an edge set T and a cluster partition defined such that each cluster is a connected component of (V, T).

Initially, $T = \emptyset$ (and each node is a cluster). At each iteration of the algorithm, each cluster S adds a lightest outgoing edge $e^* = \arg\min_{e \in \partial_S}\{w(e)\}$ to T. This means that S merges with the neighboring cluster S' that is incident on e^*. It is well-known that if the edge weights are unique, then Boruvka's algorithm computes an MST of G. Notice that in our case, edge weights are not necessarily unique, so we construct a symmetry-breaking mechanism based on a total order of the lightest outgoing edges as explained later on.

The algorithm begins with an empty set of *tree edges* and operates in phases. The goal of each phase is to add tree edges similarly to Boruvka's algorithm. Let $T_i \subseteq E$ denote the tree edges at the end of phase $i \geq 0$. As in Boruvka's algorithm, the connected components of (V, T_i) are defined to be the *clusters* at the beginning of phase $i + 1$. The nodes construct a designated circuit for each cluster formed during the algorithm. Additionally, the nodes communicate through a global circuit and exchange messages with their neighbors using the methods described in Sec. 3. The operation of each phase is divided into the following stages.

Outgoing Edge Detection. The purpose of this stage is to allow the nodes to identify which of their incident edges is an outgoing edge. To that end, the nodes execute the outgoing edge detection procedure described in Sec. 3.1. When the procedure terminates, each node detecting an outgoing edge beeps through the global circuit. The algorithm terminates if no node beeps in this round through the global circuit. Otherwise, the nodes advance to the next stage. Denote by $\texttt{Out}(v)$ the set of edges classified as outgoing by node $v \in V$.

Lightest Edge Detection. In this stage, each cluster searches for its lightest outgoing edges. Fix some cluster S. At the beginning of this stage, every node $v \in S$ such that $\texttt{Out}(v) \neq \emptyset$ marks a single edge $e \in \texttt{Out}(v)$ with weight $w(e) = \min_{e' \in \texttt{Out}(v)} w(e')$ as a candidate. The comparison between weights of the candidate edges incident on the nodes of S is done in two steps.

First, the nodes compare the lengths of the candidate edge weights (i.e., the number of bits in the edge-weight representation). Consider a node $v \in S$ incident on a candidate edge e, and let $\ell_v = \lfloor \log w(e) \rfloor + 1$ be the length of $w(e)$. Node v counts $\ell_v - 1$ rounds. If v hears a beep on the cluster's circuit during those $\ell_v - 1$ rounds, then v unmarks e as a candidate. Otherwise, v beeps through the cluster's circuit in round ℓ_v and keeps e as a candidate edge. Following the first step, all remaining candidate edges of S have weights of the same length. In the second step, the weights of the candidate edges of S are compared bit by bit, starting from the most significant bit. Let $v \in S$ be a node that still has an incident candidate edge e. The second step runs for ℓ_v rounds indexed by $j = 1, \ldots, \ell_v$. In round j, if e is still a candidate, then v beeps through the cluster's circuit if and only if the j-th most significant bit of $w(e)$ is 0. If v did not beep but heard a beep through the cluster's circuit, it unmarks e as a candidate edge. Notice that at the end of the second step, only the lightest edges that were classified as outgoing remain candidates.

In parallel, v beeps through the global circuit at every round of the stage in which e is still a candidate. Once v finishes the stage (either because e was marked as a lightest outgoing edge or e was unmarked as a candidate), it stops beeping through the global circuit. The stage terminates when no beep is transmitted through the global circuit.

Single Edge Selection. At this point, only the edges marked as lightest outgoing edges of each cluster remained candidates. However, there may be more than one candidate edge for some clusters. The goal of this stage is to select a single edge for each cluster while avoiding the formation of a cycle in the output edge set (as we will show in the analysis). To that

end, every node $v \in V$ with an incident candidate edge (u, v) informs u that (u, v) is still a candidate. Then, each of u and v draws a random bit denoted by $u.bit$ and $v.bit$, respectively. Node u sends $u.bit$ to v and v calculates the bitwise XOR of $u.bit$ and $v.bit$. Node v beeps through the cluster's circuit if the XOR result is 1. If node v does not beep for edge e but hears a beep through the cluster's circuit, it unmarks e as a candidate. Notice that if (u, v) is lightest with regard to u's cluster as well, then the same operation is performed also by u using the same drawn bits. This edge selection process is done in parallel to Proc. `CountingToLogn` over the global circuit, executed (a sufficiently large) $c > 1$ times. The nodes continue to draw bits for their incident candidate edges as long as Proc. `CountingToLogn` continues. If a node $v \in V$ has an incident candidate edge $e = (u, v)$ at the end of this stage, then it informs u, and both endpoints mark e as a tree edge.

Updating the Local Pin Partition. Every node $v \in V$ sets its local pin partition to include the pin subset $T(v) \times \{j\}$ for some $j \in [k]$, where $T(v)$ is the set of edges incident on v that were marked as tree edges (either in the current or a prior phase). Observe that this local pin partition by the nodes constructs a circuit for every cluster.

The output of the algorithm is the set of all tree edges.

4.1 Analysis

In this section, we prove the correctness and analyze the runtime of the MST algorithm presented above, establishing the following theorem.

▶ **Theorem 4.1.** *The algorithm constructs an MST of G whp and runs in $O(\log n \cdot \log(n+W))$ rounds whp.*

Recall that $T_i \subseteq E$ is the set of tree edges at the end of phase $i = 0, 1, \ldots$ and let i^* be the last phase of the algorithm. Let q_i be the number of clusters maintained by the algorithm at the beginning of phase i, that is, the number of connected components in (V, T_i).

▶ **Lemma 4.2.** *Consider a phase $0 \leq i \leq i^*$. If $q_i = 1$, then the algorithm terminates in phase i whp; otherwise, $q_{i+1} \leq \frac{1}{2} q_i$ whp.*

Notice that since the algorithm starts with n clusters, Lem. 4.2 implies the following corollary.

▶ **Corollary 4.3.** *The algorithm terminates after $i^* = O(\log n)$ phases whp. Moreover, the subgraph (V, T_{i^*}) is connected whp.*

Denote by $D_i \subseteq E$ the set of edges that are candidates for some (at least one) cluster at the end of the single edge selection stage of phase i (to be marked as tree edges).

▶ **Lemma 4.4.** *The subgraph (V, T_{i^*}) is a spanning tree of G whp.*

Proof. By Cor. 4.3, (V, T_{i^*}) is connected whp. So, it is left to show that (V, T_{i^*}) is a forest whp. We prove by induction over the phases that (V, T_i) is a forest whp for all $0 \leq i \leq i^*$. Cor. 4.3 also guarantees that there are $O(\log n)$ phases whp; hence the statement follows by applying union bound over the phases.

For the base of the induction, notice that $T_0 = \emptyset$, and thus (V, T_0) is a forest. Now, suppose that (V, T_i) is a forest for some $0 \leq i < i^*$. We show that $(V, T_{i+1}) = (V, T_i \cup D_i)$ is a forest whp. For every edge $e \in D_i$, let $B_i(e)$ be the integer obtained from the binary

representation of the bit sequence drawn for e by its endpoints (i.e., the sequence of XORed bits) in the single edge selection stage of phase i. Define the binary relation $\prec_i$ for every two edges $e, e' \in D_i$ as:

$$e \prec_i e' \iff w(e) < w(e') \vee (w(e) = w(e') \wedge B_i(e) > B_i(e')) \ .$$

Notice that by repeating the `CountingToLogn` for a sufficiently large number of times, we get that the $B_i(\cdot)$ values are unique whp. By the construction of the single edge selection stage, this means that each cluster selects exactly one outgoing edge whp – the lightest outgoing edge which is minimal with respect to $\prec_i$. To complete our proof, we show that if the $B_i(\cdot)$ values are unique and (V, T_i) is a forest, then $(V, T_{i+1}) = (V, T_i \cup D_i)$ is a forest.

Assume by contradiction that there exists at least one cycle in $(V, T_i \cup D_i)$ and let Y be a simple cycle in $(V, T_i \cup D_i)$. By the induction hypothesis we know that (V, T_i) is a forest, therefore $Y \cap D_i \neq \emptyset$. Let $e \in Y \cap D_i$ be the (unique) largest edge (with respect to $\prec_i$) of $Y \cap D_i$, and let S be the cluster that selected e. Observe that since (V, T_i) is a forest and Y is a cycle, there exists another edge $e' \in Y \cap D_i - \{e\}$ which is an outgoing edge of S. However, by the choice of e, we know that $e' \prec_i e$, in contradiction to the selection of e by S. ◀

The following lemma asserts the correctness of our MST algorithm.

▶ **Lemma 4.5.** *The graph (V, T_{i^*}) is an MST of G whp.*

Proof. By Lem. 4.4, the graph (V, T_{i^*}) is a spanning tree of G whp. The proof of Lem. 4.4 shows that every cluster selects a single lightest outgoing edge in each phase whp. The statement then follows from the correctness of Boruvka's algorithm [3]. ◀

It remains to analyze the runtime of the algorithm.

▶ **Lemma 4.6.** *The MST algorithm runs in $O(\log n \cdot \log(n + W))$ rounds whp.*

Proof. By Corollary 4.3, the algorithm runs for $O(\log n)$ phases whp. We are left to bound the runtime of each phase. Every execution of the leader election algorithm and Proc. `CountingToLogn` takes $O(\log n)$ rounds whp. Hence, the outgoing edge detection and single edge selection stages each take $O(\log n)$ rounds whp. The lightest edge detection stage completes in $O(\log W)$ rounds, and updating the local pin partition does not require any communication. Therefore, every phase of the algorithm completes in $O(\log n + \log W)$ rounds whp. Overall, we get a runtime bound of $O(\log n(\log n + \log W)) = O(\log n \cdot \log(n \cdot W)) = O(\log n \cdot \log(n + W))$ rounds whp, where the last equality hods because $\log(n \cdot W) = \log n + \log W = O(\log(n + W))$. ◀

5 A Sparse Spanner Algorithm

In this section, we present a randomized spanner algorithm that operates in the GRC model. Given a parameter $\kappa \in \mathbb{Z}_{>0}$ and a constant $0 < \varepsilon < 1$, the algorithm constructs a spanner with a stretch of $(2\kappa - 1)$ whp and $O(n^{1+(1+\varepsilon)/\kappa})$ edges in expectation. More concretely, we prove the following theorem.

▶ **Theorem 5.1.** *There exists an algorithm in the GRC model that computes a set $H \subseteq E$ of edges such that H is a $(2\kappa - 1)$-spanner whp, and $\mathbb{E}[|H|] = O(n^{1+\frac{1+\varepsilon}{\kappa}})$. The runtime of the algorithm is $O(\kappa + \log n)$ rounds whp, and the memory space used by each node $v \in V$ is $O(\deg(v) + \kappa)$.*

In the full version [6], we present a modification of our algorithm to accommodate a memory space of only $O(\deg(v) + \log \kappa)$ for each node $v \in V$, at the cost of a slightly slower $O(\kappa \log n)$-round algorithm. We now describe the algorithm stated in Thm. 5.1.

The algorithm is based on the random shift concept introduced by Miller et al. in [12] and studied further in various works (see, e.g., [11, 5, 9]). We now give a high-level overview of a spanner construction algorithm based on the random shift approach (see [9] for the full details).

The algorithm starts with each node $v \in V$ sampling a value $\delta_v \sim GeomCap(1-n^{-1/\kappa}, \kappa-1)$ (see Sec. 3 for the capped geometric distribution definition). Then, the nodes conceptually add a virtual node s. Each node $v \in V$ adds an edge (s, v) of weight $w(s, v) = \kappa - \delta_v$ to form the graph G', where all other edges are assigned a unit weight. Following that, the nodes construct a shortest path tree T rooted at s. The nodes of G are partitioned into clusters defined by the connected components of T after removing s and its incident edges. To construct the spanner H, the nodes first add the (non-virtual) edges of T. Then, the nodes add edges to H such that for each edge $(u, v) \in E - T$, at least one of the following is satisfied: (1) H contains exactly one edge between u and a node in v's cluster; or (2) H contains exactly one edge between v and a node in u's cluster. As discussed in [9], the constructed edge-set H is a $(2\kappa - 1)$-spanner of expected size $O(n^{1+1/\kappa})$.

Our algorithm works in three stages as described below.

Sampling Procedure. Recall that the algorithm of [9] begins with each node $v \in V$ sampling $\delta_v \sim GeomCap(1 - n^{-1/\kappa}, \kappa - 1)$. Note that sampling from $GeomCap(1 - n^{-1/\kappa}, \kappa - 1)$ requires the nodes to know the value of n, which is not possible in the GRC model. Hence, we devise a designated sampling procedure for each node $v \in V$.

Let us first present the intuition behind the sampling procedure. The idea is for each node $v \in V$ to simulate $\kappa - 1$ *experiments*, each with success probability close to $1 - n^{-1/\kappa}$, and compute δ_v accordingly. To achieve such success probability without knowing n, Proc. `CountingToLogn` is utilized. In order to enhance the proximity to $1 - n^{-1/\kappa}$, Proc. `CountingToLogn` is executed numerous times in parallel, and δ_v is computed based on the run with median runtime.

For ease of presentation, we describe the sampling procedure in two stages. First, a sub-procedure referred to as the *basic* scheme is described. We later explain how this basic scheme is used in the sampling procedure. The basic scheme runs during an execution of Proc. `CountingToLogn`. For each node $v \in V$, let $b_v = (b_v[0], \ldots, b_v[\kappa - 2])$ be a vector of $\kappa - 1$ bits initialized to $b_v = (0, \ldots, 0)$. The purpose of entry $b_v[j]$ is to represent the success/failure of the i-th experiment for each $0 \leq j \leq \kappa - 2$. Let ε' be the largest value such that $1/(1-\varepsilon')$ is an integer and $\varepsilon' \leq \varepsilon/(2+\varepsilon)$. In each round j such that $j \bmod \kappa \neq 0$, each node v draws $1/(1-\varepsilon')$ bits uniformly at random and sets $b_v[(j-1) \bmod \kappa] = 1$ if any of those bits are 1.

In the sampling procedure, the nodes perform $c' = 2 \cdot \lceil c/\varepsilon' \rceil - 1$ executions of the basic scheme, where $c > 0$ is a constant. Let us index these executions by $i = 0, \ldots, c' - 1$. Starting from the execution indexed 0, the rounds of the executions are done alternately, i.e., a round of the run indexed by i is followed by a round of the run indexed by $(i+1) \bmod c'$. Accordingly, each node $v \in V$ maintains c' vectors, $b_v^0, \ldots, b_v^{c'-1}$, each of size $\kappa - 1$ bits, such that b_v^i is the vector maintained by v during the i-th execution of the basic scheme. Additionally, v maintains a counter initialized to 0, whose goal is to count the executions that terminated. Whenever an execution terminates, the counter is increased by 1. Following the termination, during the rounds that are associated with that execution, the nodes do

nothing. The nodes halt the executions when the counter reaches $\lceil c/\varepsilon' \rceil$ (notice that the counter is updated in the same manner for all nodes, thus they halt at the same time). Let $\tilde{i}$ denote the index of the execution in which the counter reached $\lceil c/\varepsilon' \rceil$. Observe that this is the $\lceil c/\varepsilon' \rceil$-th fastest execution, i.e., the execution with median runtime. Each node $v \in V$ defines δ_v to be the smallest index $0 \leq j \leq \kappa - 2$ for which $b_v^{\tilde{i}}[j] = 1$ if such an index exists, or $\delta_v = \kappa - 1$ otherwise.

Partition Into Clusters. Let G' be the graph formed by adding a virtual node s and edge (s, v) of weight $w(s, v) = \kappa - \delta_v$ for every $v \in V$. To compute the cluster partition, the nodes first construct a shortest path tree T rooted at s. The idea is simple: If $w(s, v) = 1$, then v sends a message to all its neighbors and marks itself as the center of its cluster. Otherwise, assume first that v receives a message in at least one of the rounds $2, \ldots, w(s, v) - 1$ and let $2 \leq i < w(s, v) - 1$ be the first such round. After receiving a message in round i, node v (arbitrarily) chooses a neighbor u that sent v a message in that round and adds the edge (u, v) into T. Then, in round $i + 1$, node v sends a message to all neighbors from which it did not receive a message in round i. Otherwise, if v does not receive a message after $w(s, v) - 1$ rounds, then in round $w(s, v)$ node v sends a message to all its neighbors and sets itself as the center of its cluster. Notice that after at most κ rounds, T is a shortest path tree rooted at s. The edges of T are added to the spanner H. The clusters are defined to be the connected components of (V, T) (i.e., the connected components formed by removing s and its incident edges). The nodes then construct a circuit for each cluster (similarly to the MST algorithm of Sec. 4). Observe that by design, each cluster has exactly one center. Note that every message sent in each round of this stage is of size one bit.

Addition of Bridging Edges. The construction of H is completed by the following procedure whose goal is to augment H with some of the edges that bridge between clusters. This is done by each cluster randomly drawing an ID. Then, each node $v \in V$ identifies its neighboring clusters with smaller IDs and adds a single edge to each such cluster into H.

Formally, each node $v \in V$ maintains a set $S^{\text{eq}}(v)$ initialized to be $N(v)$, and a set $S^{\text{sml}}(v)$ initialized to be $\emptyset$. Additionally, throughout the execution, v maintains a partition of $S^{\text{sml}}(v)$ into subsets according to the (randomly drawn) cluster IDs. The nodes engage in a process that runs in parallel to $4c + 7$ iterations of Proc. `CountingToLogn`. In each round of this process, every cluster center tosses a coin and communicates the outcome through the cluster's circuit to all the nodes in its cluster. Then, every node $v \in V$ sends a message with the coin toss received from its cluster's center to all neighbors. Let $S_i^{\text{eq}}(v)$ be the set $S^{\text{eq}}(v)$ at the beginning of round i. For each $u \in S_i^{\text{eq}}(v)$, if u and v sent the same bit, then u stays in $S^{\text{eq}}(v)$; otherwise, u is removed. Additionally, if u's bit is smaller than v's, then u is added to $S^{\text{sml}}(v)$. The partition of the nodes in $S^{\text{sml}}(v)$ is defined so that u and u' are in the same subset by the end of round i if and only if they were in the same subset at the beginning of round i and sent the same bit in round i. Let $s_1, \ldots, s_q$ be the partition of $S^{\text{sml}}(v)$ at the end of the process. For each $j \in [q]$, node v (arbitrarily) selects a single node $u \in s_j$ and adds the edge (u, v) into H.

This completes the construction of H. We now turn to analyzing the algorithm.

5.1 Analysis

This section is dedicated to proving Thm. 5.1. To that end, we start with a structural lemma about the capped geometric distribution.

▶ **Lemma 5.2.** *For arbitrary values $q_1, \ldots, q_n$ and for $X_1, \ldots, X_n \sim GeomCap(\phi, \kappa - 1)$, define $M = \max_{i \in [n]}\{X_i - q_i\}$. For the set $I = \{i \mid X_i < \kappa - 1 \ \wedge \ X_i - q_i \in \{M-1, M\}\}$, it holds that $\mathbb{E}[|I|] \leq \frac{2}{1-\phi}$.*

Recall that in the sampling procedure of our algorithm, the value δ_v is computed for each node $v \in V$ based on the $\tilde{i}$-th execution of the basic scheme, i.e., the execution that admits the median runtime. Particularly, within that execution, δ_v is defined as the first successful experiment out of $0, \ldots, \kappa - 2$; or $\kappa - 1$ if all experiments failed. Let ϕ be the success probability of each such experiment and notice that ϕ itself is a random variable that depends on the execution's length. Define A to be the event that $1 - n^{-1/\kappa} \leq \phi \leq 1 - n^{-(1+\varepsilon)/\kappa}$. We prove the following lemma.

▶ **Lemma 5.3.** $\mathbb{P}[A] \geq 1 - 2n^{-c}$.

We now consider the bridging edges addition stage of the algorithm. Let B denote the event that for every edge $(u, v) \in E - T$, at least one of the following is satisfied: (1) H contains exactly one edge between u and a node in v's cluster; or (2) H contains exactly one edge between v and a node in u's cluster. We state the following.

▶ **Lemma 5.4.** $\mathbb{P}[B] \geq 1 - 3n^{-c}$.

For each node $v \in V$, let $M_v = \max_{u \in V}\{\delta_u - d_G(u, v)\}$ and $R(v) = \{u \in V \mid M_v - 1 \leq \delta_u - d_G(u, v) \leq M_v\}$. We obtain the following observation.

▶ **Observation 5.5.** *Consider an edge $(u, v) \in H$ such that u and v belong to clusters centered at nodes u' and v', respectively. Then, $u' \in R(v)$ or $v' \in R(u)$.*

We are now prepared to bound the expected number of edges in the spanner.

▶ **Lemma 5.6.** $\mathbb{E}[|H|] \leq 2n^{1+(1+\varepsilon)/\kappa} + n^{1+1/\kappa} + 1$.

Proof. By the law of total expectation,

$$\mathbb{E}[|H|] \ = \ \mathbb{E}[|H| \mid A \wedge B] \cdot \mathbb{P}[A \wedge B] + \mathbb{E}[|H| \mid \neg A \vee \neg B] \cdot \mathbb{P}[\neg A \vee \neg B] \, .$$

Combining Lem. 5.3 with Lem. 5.4, we get $\mathbb{P}[\neg A \vee \neg B] \leq 5n^{-c}$, and since $\mathbb{E}[|H| \mid \neg A \vee \neg B] \leq m < n^2$, it follows that

$$\mathbb{E}[|H|] \leq \mathbb{E}[|H| \mid A \wedge B] \cdot \mathbb{P}[A \wedge B] + n^2 \cdot 5n^{-c} \leq \mathbb{E}[|H| \mid A \wedge B] + 1 \, ,$$

where the final inequality holds for, e.g., $c \geq 3$. Therefore, we are left to bound the term $\mathbb{E}[|H| \mid A \wedge B]$.

Obs. 5.5 implies that the sum $\sum_{v \in V} |R(v)|$ accounts for every edge in H at least once, i.e., $\sum_{v \in V} |R(v)| \geq |H|$. Fix some node $v \in V$, we seek to bound $\mathbb{E}[|R(v)|]$. Partition the set $R(v)$ into $R_1(v) = \{u \in R(v) \mid \delta_u = \kappa - 1\}$ and $R_2(v) = R(v) - R_1(v)$. Notice that the events $\delta_u = \kappa - 1$ and B are independent. Thus, we get

$$\mathbb{E}[|R_1(v)| \mid A \wedge B] \leq n \cdot \mathbb{P}[\delta_u = \kappa - 1 \mid A \wedge B] \ = \ n \cdot \mathbb{P}[\delta_u = \kappa - 1 \mid A] \, .$$

Observe that $\mathbb{E}[|R_1(v)|] \leq n \cdot \mathbb{P}[\delta_u = \kappa - 1] = n(1 - \phi)^{\kappa - 1}$, and recall that if event A occurs, then $\phi \geq 1 - n^{-1/\kappa}$. Hence, it follows that

$$n \cdot \mathbb{P}[\delta_u = \kappa - 1 \mid A] \ = \ n(1 - \phi)^{\kappa - 1} \leq n \cdot n^{(-1/\kappa) \cdot (\kappa - 1)} \ = \ n^{1/\kappa} \, .$$

As for R_2, applying Lem. 5.2, we get $\mathbb{E}[|R_2(v)|] \leq 2/(1-\phi)$. Once again, we condition on A and B to get

$$\mathbb{E}[|R_2(v)| \mid A \wedge B] = \mathbb{E}[|R_2(v)| \mid A] \leq 2/n^{-(1+\varepsilon)/\kappa} = 2n^{(1+\varepsilon)/\kappa}.$$

Overall, we conclude that

$$\begin{aligned}\mathbb{E}[|H|] &\leq n \cdot \mathbb{E}[R(v)] \leq n \cdot \mathbb{E}[|R_1(v)| \mid A \wedge B] + n \cdot \mathbb{E}[|R_2(v)| \mid A \wedge B] + 1 \\ &\leq 2n^{1+(1+\varepsilon)/\kappa} + n^{1+1/\kappa} + 1\,.\end{aligned}$$

◀

Next, we bound the stretch of H.

▶ **Lemma 5.7.** *H is a $(2\kappa-1)$-spanner whp.*

Proof. We now argue that if event B occurs, then H has stretch $2\kappa-1$, which implies the stated claim due to Lem. 5.4. To see that, consider an edge $(u,v) \in E$. Observe that the diameter within each cluster is at most $2\kappa-2$. This is because every node is at distance at most $\kappa-1$ from its cluster's center. Hence, if u and v belong to the same cluster, then $d_H(u,v) \leq 2\kappa-2$. Otherwise, if event B occurs, then either there is an edge $(\tilde{u},v) \in H$ between v and a node $\tilde{u}$ in u's cluster, or an edge $(u,\tilde{v}) \in H$ between u and a node $\tilde{v}$ in v's cluster. Assume w.l.o.g. that $(\tilde{u},v) \in H$. It follows that $d_H(u,v) \leq d_H(u,\tilde{u}) + d_H(\tilde{u},v) \leq 1 + 2\kappa - 2 = 2\kappa - 1$. ◀

This concludes the analysis of our algorithm.

References

1. Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Comput.*, 26(4):195–208, 2013. doi:10.1007/S00446-012-0175-7.
2. Dana Angluin. Local and global properties in networks of processors (extended abstract). In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC)*, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.
3. Otakar Boruvka. Contribution to the solution of a problem of economical construction of electrical networks. *Elektronickỳ Obzor*, 15:153–154, 1926.
4. Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010. doi:10.1007/978-3-642-15763-9_15.
5. Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019. doi:10.1145/3274651.
6. Yuval Emek, Yuval Gil, and Noga Harlev. On the power of graphical reconfigurable circuits, 2024. arXiv:2408.10761.
7. Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *Journal of Computational Biology*, 29(4):317–343, 2022. doi:10.1089/CMB.2021.0363.
8. Roland Flury and Roger Wattenhofer. Slotted programming for sensor networks. In Tarek F. Abdelzaher, Thiemo Voigt, and Adam Wolisz, editors, *Proceedings of the 9th International Conference on Information Processing in Sensor Networks, IPSN 2010, April 12-16, 2010, Stockholm, Sweden*, pages 24–34. ACM, 2010. doi:10.1145/1791212.1791216.

9 Sebastian Forster, Martin Grösbacher, and Tijn de Vos. An improved random shift algorithm for spanners and low diameter decompositions. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.OPODIS.2021.16.

10 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Comput.*, 28(1):31–53, 2015. doi:10.1007/S00446-013-0202-3.

11 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In Guy E. Blelloch and Kunal Agrawal, editors, *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 192–201. ACM, 2015. doi:10.1145/2755573.2755574.

12 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In Guy E. Blelloch and Berthold Vöcking, editors, *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 196–203. ACM, 2013. doi:10.1145/2486159.2486180.

13 Andreas Padalkin and Christian Scheideler. Polylogarithmic time algorithms for shortest path forests in programmable matter. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2024. To appear.

14 Andreas Padalkin, Christian Scheideler, and Daniel Warner. The structural power of reconfigurable circuits in the amoebot model. In Thomas E. Ouldridge and Shelley F. J. Wickham, editors, *28th International Conference on DNA Computing and Molecular Programming, DNA 28, August 8-12, 2022, University of New Mexico, Albuquerque, New Mexico, USA*, volume 238 of *LIPIcs*, pages 8:1–8:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DNA.28.8.

15 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

16 David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000. doi:10.1137/S0097539700369740.

17 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.