Distributed Computing manuscript No.
(will be inserted by the editor)

Simple and Efficient Local Codes for Distributed Stable Network Construction

Othon Michail · Paul G. Spirakis

Received: date / Accepted: date

Abstract In this work, we study protocols so that populations of distributed processes can *construct networks*. In order to highlight the basic principles of distributed network construction, we keep the model minimal in all respects. In particular, we assume *finite-state processes* that all begin from the same initial state and all execute the same protocol. Moreover, we assume *pairwise interactions* between the processes that are scheduled by a fair adversary. In order to allow processes to construct networks, we let them *activate* and *deactivate* their pairwise connections. When two processes interact, the protocol takes as input the states of the processes and the state of their connection and updates all of them. Initially all connections are inactive and the goal is for the processes, after interacting and activating/deactivating connections for a while, to end up with a desired *stable network*. We give protocols (optimal in some cases) and lower bounds for several basic network construction problems such as *spanning line*, *spanning ring*, *spanning star*, and *regular network*. The *expected time to convergence* of our protocols is analyzed under a *uniform random scheduler*. Finally, we prove several *universality* results by presenting generic protocols that are capable of simulating a Turing Machine (TM) and exploiting it in order to construct a large class of networks. We additionally show how to partition the population into k *supernodes*, each being a line of $\log k$ nodes, for the largest such k. This amount of local memory is sufficient for the supernodes to obtain unique names and exploit their names and their memory to realize nontrivial constructions.

Keywords distributed network construction · stabilization · homogeneous population · distributed protocol · interacting automata · fairness · random schedule · structure formation · self-organization

1 Introduction

1.1 Motivation

Suppose a set of tiny computational devices (possibly at the nanoscale) are injected into a human circulatory system for the purpose of monitoring or even treating a disease. The devices are incapable of controlling their mobility. The mobility of the devices, and consequently the interactions between them, stems solely from the dynamicity of the environment, the blood flow inside the circulatory system in this case. Additionally, each device alone is incapable of performing any useful computation, as the small scale of the device highly constrains its computational capabilities. The goal is for the devices to accomplish their task via cooperation. To this end, the devices are equipped with a mechanism that allows them to create bonds with other devices

Supported in part by (i) the project “Foundations of Dynamic Distributed Computing Systems” (FOCUS) which is implemented under the “ARISTEIA” Action of the Operational Programme “Education and Lifelong Learning” and is co-funded by the European Union (European Social Fund) and Greek National Resources and (ii) the FET EU IP project MULTIPLEX under contract no 317532. A preliminary version of the results in this paper has appeared in [MS14].

Othon Michail · Paul G. Spirakis
Computer Technology Institute & Press “Diophantus” (CTI), N. Kazantzaki Str., Patras University Campus, Rion, P.O. Box 1382, 26504, Patras, Greece Tel.: +30 2610 960300
Fax: +30 2610 960490

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
E-mail: michailo@cti.gr, P.Spirakis@liverpool.ac.uk

(mimicking nature's ability to do so). So, whenever two devices come sufficiently close to each other and interact, apart from updating their local states, they may also become connected by establishing a physical connection between them. Moreover, two connected devices may at some point choose to drop their connection. In this manner, the devices can organize themselves into a desired global structure. This network-constructing self-assembly capability allows the artificial population of devices to evolve greater complexity, better storage capacity, and to adapt and optimize its performance to the needs of the specific task to be accomplished.

1.2 Our Approach

In this work, we study the fundamental problem of *network construction* by a distributed computing system. The system consists of a set of processes that are capable of performing local computation (via pairwise interactions) and of forming and deleting connections between them. Connections between processes can be either *physical* or *virtual* depending on the application. In the most general case, a connection between two processes can be in one of a finite number of possible states. For example, state 0 could mean that the connection does not exist while state $i \in \{1, 2, \dots, k\}$, for some finite k, that the connection exists and has strength i. We consider here the simplest case, which we call the *on/off* case, in which, at any time, a connection can either exist or not exist; that is, there are just two states for the connections, 1 and 0, respectively. If a connection exists we also say that it is *active* and if it does not exist we say that it is *inactive*. Initially all connections are inactive and the goal is for the processes, after interacting and activating/deactivating connections for a while, to end up with a desired *stable network*. In the simplest case, the output-network is the one induced by the active connections and it is stable when no connection changes state any more.

Our aim in this work is to initiate this study by proposing and studying a very *simple*, yet sufficiently generic, model for distributed network construction. To this end, we assume the computationally weakest type of processes. In particular, the processes are finite automata that all begin from the same initial state and all execute the same finite program which is stored in their memory (i.e., the system is *homogeneous*). The communication model that we consider is also very minimal. In particular, we consider processes that are inhabitants of an *adversarial environment* that has total control over the inter-process interactions. We model such an environment by an adversary scheduler that operates in discrete steps, selecting in every step a pair of processes which then interact according to the common program. This represents very well systems of (not necessarily computational) entities that interact in pairs whenever two of them come sufficiently close to each other. When two processes interact, the program takes as input the states of the interacting processes and the state of their connection and outputs a new state for each process and a new state for the connection. The only restriction that we impose on the scheduler, in order to study the constructive power of the model, is that it is *fair*, by which we mean the weak requirement that, at every step, it assigns to every reachable configuration of the system a non-zero probability to occur. In other words, a fair scheduler cannot forever conceal an always reachable configuration of the system. Note that under such a generic scheduler, we cannot bound the running time of our constructors. Thus, to estimate the efficiency of our solutions we assume a *uniform random scheduler*, one of the simplest fair probabilistic schedulers. The uniform random scheduler selects in every step independently and uniformly at random a pair of processes to interact from all such pairs. What renders this model interesting is its ability to achieve complex global behavior via a set of notably simple, uniform (i.e., with codes that are independent of the size of the system), homogeneous, and cooperative entities.

We now give a simple illustration of the above. Assume a set of n very weak processes that can only be in one of two states, "black" or "red". Initially, all processes are black. We can think of the processes as small particles that move randomly in a fair solution. The particles are capable of forming and deleting physical connections between them, by which we mean that, whenever two particles interact, they can read and write the state of their connection. Moreover, for simplicity of the model, we assume that fairness of the solution is independent of the states of the connections. This is in contrast to schedulers that would take into account the geometry of the active connections and would, for example, forbid two non-neighboring particles of the same component to interact with each other. [1] In particular, we assume that throughout the execution every pair of processes may be selected for interaction.

Consider now the following simple problem. We want to identically program the initially disorganized particles so that they become self-organized into a *spanning star*. In particular, we want to end up with a unique black particle connected (via active connections) to $n-1$ red particles and all other connections (between red particles) being inactive. Conversely, given a (possi-

[1] Such a geometrically restricted variant has been studied in [Mic15].

bly physical) system that tends to form a spanning star we would like to unveil the code behind this behavior.

Consider the following program. When two black particles that are not connected interact, they become connected and one of them becomes red. When two connected red particles interact they become disconnected (i.e., reds repel). Finally, when a black and a red that are not connected interact they become connected (i.e., blacks and reds attract).

The protocol forms a spanning star as follows. As whenever two blacks interact only one survives and the other becomes red, eventually a unique black will remain and all other particles will be red (we say "eventually", meaning "in finite time", because we do not know how much time it will take for all blacks to meet each other but from fairness we know that this has to occur in a finite number of steps). As blacks and reds attract while reds repel, it is clear that eventually the unique black will be connected to all reds while every pair of reds will be disconnected. Moreover, no rule of the program can modify such a configuration, so the constructed spanning star is stable (see Figure 1). It is worth noting that this very simple protocol is optimal both with respect to (abbreviated "w.r.t." throughout) the number of states that it uses and w.r.t. the time it takes to construct a stable spanning star under the uniform random scheduler.

Our model for network construction is strongly inspired by the Population Protocol model [AAD$^+$06] and the Mediated Population Protocol model [MCS11a]. In the former, connections do not have states. States on the connections were first introduced in the latter. The main difference to our model is that *in those models the focus was on the computation of functions of some input values and not on network construction.* Another important difference is that we allow the edges to choose between *only two possible states* which was not the case in [MCS11a]. Interestingly, when operating under a uniform random scheduler, population protocols are formally equivalent to *chemical reaction networks* (CRNs) which model chemistry in a *well-mixed solution* [Dot14]. "CRNs are widely used to describe information processing occurring in natural cellular regulatory networks, and with upcoming advances in synthetic biology, CRNs are a promising programming language for the design of artificial molecular control circuitry" [Dot14]. However, CRNs and population protocols can only capture the dynamics of molecular counts and not of structure formation. Our model then may be also viewed as an extension of population protocols and CRNs aiming to capture the stable structures that may occur in a well-mixed solution. From this perspective, our goal is to determine what stable structures can result in such systems (natural or artificial), how fast, and under what conditions (e.g., by what underlying codes/reaction-rules).

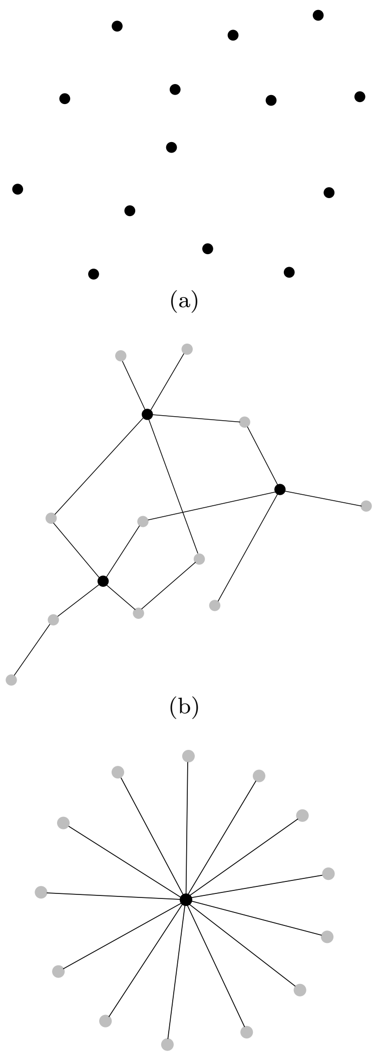

Fig. 1 (a) Initially all particles are black and no active connections exist. (b) After a while, only 3 black particles have survived each having a set of red neighbors (red particles appear as gray here). Note that some red particles are also connected to red particles. The tendency is for the red particles to repel red particles and attract black particles. (c) A unique black has survived, it has attracted all red particles, and all connections between red particles have been deactivated. The construction is a stable spanning star.

Most computability issues in the area of population protocols have now been resolved. Finite-state processes on a complete interaction network, i.e., one in which every pair of processes may interact, (and several variations) compute the *semilinear predicates* [AAER07]. Semilinearity persists up to $o(\log \log n)$ local space but not more than this [CMN$^+$11]. If, additionally, the connections between processes can hold a state from a finite domain (note that this is a stronger requirement than the on/off that the present work assumes) then the computational power dramatically increases to the commutative subclass of $\mathbf{NSPACE}(n^2)$ [MCS11a]. Other important works include [GR09] which equipped the nodes of population protocols with unique ids and [BBCK10] which introduced a (weak) notion of speed of the nodes that allowed the design of fast converging protocols with only weak requirements. For introductory texts see [AR09, MCS11b].

The paper essentially consists of two parts. In the first part, we give simple (i.e., small) and efficient (i.e., polynomial-time) protocols for the construction of several fundamental networks. In particular, we give protocols for spanning lines, spanning rings, cycle-covers, partitioning into cliques, and regular networks and we also provide a protocol that replicates a given input network (formal definitions of all problems considered can be found in Section 3.2). We remark that the spanning line problem is of outstanding importance because it constitutes a basic ingredient of universal constructors. We give two different protocols for this problem, the second improving on the running time of the first but using more states to this end. Additionally, we establish an $\Omega(n \log n)$ generic lower bound on the expected running time of all constructors that construct a spanning network and an $\Omega(n^2)$ lower bound for the spanning line, where n throughout this work denotes the number of processes. Our fastest protocol for the problem runs in $O(n^3)$ expected time and uses 9 states while our simplest uses only 5 states but pays in an expected time which is between $\Omega(n^4)$ and $O(n^5)$.

In the second part, we investigate the more generic question of *what is in principle constructible by our model*. We arrive there at several satisfactory characterizations establishing some sort of universality of the model. The main idea is as follows. To construct a decidable graph-language L we (i) construct on k of the processes (called the *waste*) a network G_1 capable of simulating a Turing Machine (abbreviated "TM" throughout the paper) and of constructing a random network on the remaining $n-k$ processes (called the *useful space*), (ii) use G_1 to construct a random network $G_2 \in G_{n-k,1/2}$ on the remaining $n-k$ processes, [2] (iii) execute on G_1 the TM that decides L, with G_2 as input. If the TM accepts, then we output G_2 (note that this is not a terminating step - the reason why will become clear in Section 6; the protocol just freezes and its output forever remains G_2), otherwise we go back to (ii) and repeat. Using this core idea we prove several universality results for our model. Additionally, we show how to organize the population into a distributed system with names and logarithmic local memories.

In Section 2, we discuss further related literature. Section 3 brings together all definitions and basic facts that are used throughout the paper. In particular, in Section 3.1 we formally define the model of network constructors, Section 3.2 formally defines all network construction problems that are considered in this work, and in Section 3.3 we identify and analyze a set of basic probabilistic processes that are recurrent in the analysis of the running times of network constructors. In Section 4, we study the spanning line problem. In Section 5, we provide direct constructors for all the other basic network construction problems. Section 6 presents our *universality* results. Finally, in Section 7 we conclude and give further research directions that are opened by our work.

2 Further Related Work

Algorithmic Self-Assembly. There are already several models that try to capture the self-assembly capability of natural processes with the purpose of engineering systems and developing algorithms inspired by such processes. For example, [Dot12] proposes to learn how to program molecules to manipulate themselves, grow into machines and at the same time control their own growth. The research area of "algorithmic self-assembly" belongs to the field of "molecular computing". The latter was initiated by Adleman [Adl94], who designed interacting DNA molecules to solve an instance of the Hamiltonian path problem. The model guiding the study in algorithmic self-assembly is the Abstract Tile Assembly Model (aTAM) [Win98, RW00] and variations (e.g., see [WCG$^+$13] for a very recent interesting variation allowing DNA tiles to actively control their mobility and to self-replicate).

In contrast to most of the work in algorithmic self-assembly, that tries to incorporate the exact molecular mechanisms (like temperature, energy, and bounded degree), we propose a very abstract combinatorial rule-based model, free of specific application-driven assumptions, with the aim of revealing the fundamental laws governing the distributed (algorithmic) generation of networks. Our model may serve as a common substructure to more applied models (like assembly models or models with geometry restrictions) that may be obtained from our model by imposing restrictions on the scheduler, the degree, and the number of local states (see Section 7 for several interesting variations

[2] The $G_{n,p}$ random graph model consists of all graphs with node set $V = \{1, 2, \ldots, n\}$ in which the edges are chosen independently and with probability p (for more details, cf. [Bol01] pages 34-35).

of our model).

Distributed Network Construction. To the best of our knowledge, classical distributed computing has not considered the problem of constructing an actual communication network from scratch. From the seminal work of Angluin [Ang80] that initiated the theoretical study of distributed computing systems up to now, the focus has been more on assuming a given communication topology and constructing a virtual network over it, e.g., a spanning tree for the purpose of fast dissemination of information. Moreover, these models usually assume unique identities, unbounded memories, and message-passing communication. Additionally, a process always communicates with its neighboring processes (see [Lyn96] for all the details).

An exception is the area of geometric pattern formation by mobile robots (cf. [SY99,DFSY15] and references therein). A great difference, though, to our model is that in mobile robotics the computational entities have complete control over their mobility and thus over their future interactions. That is, the goal of a protocol is to result in a desired interaction pattern while in our model the goal of a protocol is to construct a network while operating under a totally unpredictable interaction pattern.

Very recently, a model inspired by the behavior of ameba that allows algorithmic research on self-organizing particle systems was proposed [DGRS13, DDG$^+$14]. The goal is for the particles to self-organize in order to adapt to a desired shape without any central control, which is quite similar to our objective, but the two models seem to have little in common. The authors also observe that, in contrast to the considerable work that has been performed w.r.t. systems, like in self-reconfigurable robotic systems [3], only very little theoretical work has been done in this area. This further supports the importance of introducing a simple yet sufficiently generic model for distributed network construction, as we do in this work.

Cellular Automata. A cellular automaton (cf., e.g., [Sch11]) consists of a grid of cells each cell being a finite automaton. A cell updates its own state by reading the states of its neighboring cells (e.g., 2 in the 1-dimensional case and 4 in the 2-dimensional case). All cells may perform the updates in discrete synchronous steps or updates may occur asynchronously. Cellular automata have been used as models for self-replication, for modeling several physical systems (e.g., neural activity, bacterial growth, pattern formation in nature), and for understanding emergence, complexity, and self-organization issues.

Though there are some similarities there are also significant differences between our model and cellular automata. One is that in our model the interaction pattern is nondeterministic as it depends on the scheduler and a process may interact with any other process of the system and not just with some predefined neighbors. Moreover, our model has a direct capability of forming networks whereas cellular automata can form networks only indirectly (an edge between two cells u and v has to be represented as a line of cells beginning at u, ending at v and all cells on the line being in a special edge-state). In fact, cellular automata are more suitable for studying the formation of patterns on e.g., a discrete surface of static cells while our model is more suitable for studying how a totally dynamic (e.g., mobile) and initially disordered collection of entities can self-organize into a network.

Social Networks. There is a great amount of work dealing with networks formed by a group of interacting individuals. Individuals, also called players, which may, for example, be people, animals, or companies, depending on the application, usually have incentives and connections between individuals indicate some social relationship, like for example friendship. The network is formed by allowing the individuals to form or delete connections, usually selfishly trying to maximize their own utility. The usual goal there is to study how the whole network affects the outcome of a specific interaction, to predict the network that will be formed by a set of selfish individuals, and to characterize the quality of the network formed (e.g., its efficiency). See, e.g., [Jac05,BEK$^+$13]. This is a game-theoretic setting which is very different from the setting considered here as the latter does not include incentives and utilities.

Another important line of research considers random social networks in which new links are formed according to some probability distribution. For example, in [BA99] it was shown that growth and preferential attachment that characterize a great majority of social networks (like, for example, the Internet) results in scale-free properties that are not predicted by the Erdös-Rényi random graph model [ER59,Bol01]. Though, in principle, we allow processes to perform a coin tossing during an interaction, our focus is not on the formation of a random network but on cooperative (algorithmic) construction according to a common set of rules. In summary, our model looks more like a standard dynamic distributed computing system in which the interacting entities are

[3] See [RCN14] for a very recently reported system that demonstrates programmable self-assembly of complex two-dimensional shapes with a thousand-robot swarm.

computing processes that all execute the same program.

Network Formation in Nature. Nature has an intrinsic ability to form complex structures and networks via a process known as *self-assembly*. By self-assembly, small components (like molecules) automatically assemble into large, and usually complex structures (like a crystal). There is an abundance of such examples in the physical world. Lipid molecules form a cell's membrane, ribosomal proteins and RNA coalesce into functional ribosomes, and bacteriophage virus proteins self-assemble a capsid that allows the virus to invade bacteria [Dot12]. "Mixtures of RNA fragments that self-assemble into self-replicating ribozymes spontaneously form cooperative catalytic cycles and networks". Such cooperative networks grow faster than selfish autocatalytic cycles "indicating an intrinsic ability of RNA populations to evolve greater complexity through cooperation" [VMC+12]. "Through billions of years of prebiotic molecular selection and evolution, nature has produced a basic set of molecules". By combining these simple elements, "natural processes are capable of fashioning an enormously diverse range of fabrication units, which can further self-organize into refined structures, materials and molecular machines that not only have high precision, flexibility and error-correction capacity, but are also self-sustaining and evolving". In fact, "nature shows a strong preference for bottom-up design" [Zha03].

Systems and solutions inspired by nature have often turned out to be extremely practical and efficient. For example, the bottom-up approach of nature inspires the fabrication of biomaterials by attempting to "mimic these phenomena with the aim of creating new and varied structures with novel utilities well beyond the gifts of nature" [Zha03]. Moreover, there is already a remarkable amount of work envisioning our future ability to engineer computing and robotic systems by manipulating molecules with nanoscale precision. Ambitious long-term applications include molecular computers [BPS+10] and miniature (nano)robots for surgical instrumentation, diagnosis and drug delivery in medical applications and monitoring in extreme conditions (e.g., in toxic environments). We believe that the success of this ambitious effort depends to some extent on our ability to discover *the laws governing the capability of distributed systems to construct networks*. The gain of developing such a theory will be twofold: It will give some insight to the role (and the mechanisms) of network formation in the complexity of natural processes and it will allow us to engineer artificial systems that achieve this complexity.

3 Preliminaries

3.1 A Model of Network Constructors

Definition 1 A *Network Constructor* (NET) is a distributed protocol defined by a 4-tuple $(Q, q_0, Q_{out}, \delta)$, where Q is a finite set of *node-states*, $q_0 \in Q$ is the *initial node-state*, $Q_{out} \subseteq Q$ is the set of *output node-states*, and $\delta : Q \times Q \times \{0,1\} \to Q \times Q \times \{0,1\}$ is the *transition function*.

If $\delta(a,b,c) = (a',b',c')$, we call $(a,b,c) \to (a',b',c')$ a *transition* (or *rule*) and we define $\delta_1(a,b,c) = a'$, $\delta_2(a,b,c) = b'$, and $\delta_3(a,b,c) = c'$. A transition $(a,b,c) \to (a',b',c')$ is called *effective* if $x \neq x'$ for at least one $x \in \{a,b,c\}$ and *ineffective* otherwise. When we present the transition function of a protocol we only present the effective transitions. Additionally, we agree that the *size* of a protocol is the number of its states, i.e., $|Q|$.

The system consists of a population V_I of n distributed *processes* (also called *nodes* when clear from context). In the generic case, there is an underlying *interaction graph* $G_I = (V_I, E_I)$ specifying the permissible interactions between the nodes. Interactions in this model are always pairwise. In this work, G_I is a *complete undirected interaction graph*, i.e., $E_I = \{uv : u, v \in V_I \text{ and } u \neq v\}$, where $uv = \{u, v\}$. Initially, all nodes in V_I are in the initial node-state q_0.

A central assumption of the model is that edges have binary states. An edge in state 0 is said to be *inactive* while an edge in state 1 is said to be *active*. All edges are initially inactive.

Execution of the protocol proceeds in discrete steps. In every step, a pair of nodes uv from E_I is selected by an *adversary scheduler* and these nodes interact and update their states and the state of the edge joining them according to the transition function δ. Due to the fact that the interactions are undirected, we restrict δ to be a *partial* function which, for all edge-states $c \in \{0,1\}$: (i) is defined at (a,a,c), for all node-states $a \in Q$ and (ii) is defined at either (a,b,c) or (b,a,c), for all distinct node-states $a,b \in Q$. [4] So, if a, b, and c are the states of nodes u, v, and edge uv, respectively, then the unique rule corresponding to these states, let it be $(a,b,c) \to (a',b',c')$, is applied, the edge that was in state c updates its state to c' and if $a \neq b$, then u updates its state to a' and v updates its state to b', if $a = b$ and $a' = b'$, then both nodes update their states

[4] An equivalent way is to assume that it is defined at both (a,b,c) and (b,a,c) but require that it satisfies *symmetry w.r.t. node-states*, i.e., $\delta_1(a,b,c) = \delta_2(b,a,c)$ and $\delta_2(a,b,c) = \delta_1(b,a,c)$, and *equality w.r.t. edge-states*, i.e., $\delta_3(a,b,c) = \delta_3(b,a,c)$.

to a', and if $a = b$ and $a' \neq b'$, then the node that gets a' is drawn equiprobably from the two interacting nodes and the other node gets b'. The latter is the only case in which the protocol has no other means of breaking the symmetry apart from making a random choice, because in this case the two interacting nodes are in the same state, the edge between them has no direction but the new states are not the same, so the protocol has no means of knowing where to assign each of the new states. In all other cases, the protocol can make the distinction because either symmetry is broken by the fact that the interacting nodes are in different states or the new states are the same so there is no choice to be made.

A *configuration* is a mapping $C : V_I \cup E_I \to Q \cup \{0, 1\}$ specifying the state of each node and each edge of the interaction graph. Let C and C' be configurations, and let u, v be distinct nodes. We say that *C goes to C' via encounter $e = uv$*, denoted $C \overset{e}{\to} C'$, if $(C'(u), C'(v), C'(e)) = \delta(C(u), C(v), C(e))$ or $(C'(v), C'(u), C'(e)) = \delta(C(v), C(u), C(e))$ and $C'(z) = C(z)$, for all $z \in (V_I \backslash \{u, v\}) \cup (E_I \backslash \{e\})$. We say that *$C'$ is reachable in one step from C*, denoted $C \to C'$, if $C \overset{e}{\to} C'$ for some encounter $e \in E_I$. We say that C' is *reachable* from C and write $C \rightsquigarrow C'$, if there is a sequence of configurations $C = C_0, C_1, \ldots, C_t = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < t$.

An *execution* is a finite or infinite sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. A *fairness condition* is imposed on the adversary to ensure the protocol makes progress. An infinite execution is *fair* if for every pair of configurations C and C' such that $C \to C'$, if C occurs infinitely often in the execution then so does C'. In what follows, every execution of a NET will by definition considered to be fair.

We define the *output of a configuration C* as the graph $G(C) = (V, E)$ where $V = \{u \in V_I : C(u) \in Q_{out}\}$ and $E = \{uv : u, v \in V,\ u \neq v, \text{ and } C(uv) = 1\}$. In words, the output-graph of a configuration consists of those nodes that are in output states and those edges between them that are active, i.e., the active subgraph induced by the nodes that are in output states. The output of an execution $C_0, C_1, \ldots$ is said to *stabilize* (or *converge*) to a graph G if there exists some step $t \geq 0$ such that (abbreviated "s.t." in several places) $G(C_i) = G$ for all $i \geq t$, i.e., from step t and onwards the output-graph remains unchanged. Every such configuration C_i, for $i \geq t$, is called *output-stable*. The *running time* (or *time to convergence*) of an execution is defined as the minimum such t (or ∞ if no such t exists). Throughout the paper, whenever we study the running time of a NET, we assume that interactions are chosen by a *uniform random scheduler* which, in every step, selects independently and uniformly at random one of the $|E_I| = n(n-1)/2$ possible interactions. [5] In this case, the running time becomes a random variable (abbreviated "r.v." throughout) X and our goal is to obtain bounds on the expectation $\mathrm{E}[X]$ of X. Note that the uniform random scheduler is fair with probability 1.

Definition 2 We say that an execution of a NET on n processes *constructs a graph* (or *network*) G, if its output stabilizes to a graph isomorphic to G.

Definition 3 We say that a NET $\mathcal{A}$ *constructs a graph language L with useful space $g(n) \leq n$*, if $g(n)$ is the greatest function for which: (i) for all n, every execution of $\mathcal{A}$ on n processes constructs a $G \in L$ of order at least $g(n)$ (provided that such a G exists) and, additionally, (ii) for all $G \in L$ there is an execution of $\mathcal{A}$ on n processes, for some n satisfying $|V(G)| \geq g(n)$, that constructs G. Equivalently, we say that $\mathcal{A}$ *constructs L with waste $n - g(n)$*.

Definition 4 Define $\mathbf{REL}(g(n))$ to be the class of all graph languages that are constructible with useful space $g(n)$ by a NET. We call $\mathbf{REL}(\cdot)$ the *relation* or *on/off* class.

Also define $\mathbf{PREL}(g(n))$ in precisely the same way as $\mathbf{REL}(g(n))$ but in the extension of the above model in which every pair of processes is capable of tossing an unbiased coin during an interaction between them. In particular, in the weakest probabilistic version of the model, we allow transitions that with probability 1/2 give one outcome and with probability 1/2 another. Additionally, we require that all graphs have the same probability to be constructed by the protocol.

We denote by $\mathbf{DGS}(f(l))$ (for "Deterministic Graph Space") the class of all graph languages that are decidable by a TM of (*binary*) space $f(l)$, where l is the length of the adjacency matrix encoding of the input graph.

3.2 Problem Definitions

We here provide formal definitions of all the network construction problems that are considered in this work.

[5] We should emphasize, in order to avoid confusion, that in this work "time" is *sequential*, as a time-step consists of a single interaction selected by the scheduler. Such a sequential estimate can then be easily translated to some estimate of *parallel* time. For example, assuming that $\Theta(n)$ interactions occur in parallel in every step, one could obtain an estimation of parallel time by dividing sequential time by n. In contrast, there are some papers, like [CCDS14], that perform their analysis directly in terms of parallel time.

Protocols and bounds for these problems are presented in Sections 4 and 5.

Global line. The goal is for the n distributed processes to construct a spanning line, i.e., a connected graph in which 2 nodes have degree 1 and $n-2$ nodes have degree 2.

Cycle cover. Every process in V_I must eventually have degree 2. The result is a collection of node-disjoint cycles spanning V_I.

Global star. The processes must construct a spanning star, i.e., a connected graph in which 1 node, called the *center*, has degree $n-1$ and $n-1$ nodes, called the *peripheral nodes*, have degree 1.

Global ring. The processes must construct a spanning ring, i.e., a connected graph in which every node has degree 2.

k-regular connected. The generalization of global ring in which every node has degree $k \geq 2$ (note that k is a constant and a protocol for the problem must run correctly on any number n of processes).

c-cliques. The processes must partition themselves into $\lfloor n/c \rfloor$ cliques of order c each (again c is a constant).

Replication. The protocol is given an input graph $G_1 = (V_1, E_1)$ on a subset V_1 of the processes. The input graph is provided as follows. All processes in V_1 are initially in state q_0 and all other processes, in $V_2 = V_I \backslash V_1$, are initially in state r_0. Every edge of E_1 is initially active and all other edges, in $E_I \backslash E_1$, are initially inactive (that is, the only active edges, initially, are the edges of E_1). The goal is to create a *replica* of G_1 on V_2, provided that $|V_2| \geq |V_1|$. Formally, we want, in every execution, the output induced by the active edges between the nodes of V_2 to stabilize to a graph isomorphic to G_1.

Keep in mind that the above definitions (apart from the replication problem) assume no waste. In case of a waste x the definitions must be updated in such a way that the target-construction refers to the useful space. For example, a cycle cover with waste x is a cycle cover on at least $n-x$ of the nodes.

3.3 Basic Probabilistic Processes

We now present a set of very fundamental probabilistic processes that are recurrent in the analysis of the running times of network constructors. All these processes assume a uniform random scheduler and are applications of the standard coupon collector problem. In most of these processes, we ignore the states of the edges and focus only on the dynamics of the node-states, that is, we consider rules of the form $\delta : Q \times Q \rightarrow Q \times Q$. Throughout this section, we call a step a *success* if an effective rule applies on the interacting nodes and we denote by X the r.v. of the running time of the processes. We should mention that many of these processes have been used before in the relevant literature, usually implicitly in the running-time analysis of other more complicated protocols. We believe that the reader and the further growth of the subject may benefit from a clear identification and analysis of these processes, since they are recurrent in the analyses of protocols' running times.

One-way epidemic. Consider the protocol in which the only effective transition is $(a, b) \rightarrow (a, a)$. Initially, there is a single a and $n-1$ bs and we want to estimate the expected number of steps until all nodes become as.

Proposition 1 *The expected time to convergence of a one-way epidemic (under the uniform random scheduler) is $\Theta(n \log n)$.*

Proof Let the r.v. X be the number of steps until all n nodes are in state a. Call a step a success if an effective rule applies and a new a appears on some node. Divide the steps of the protocol into *epochs*, where epoch i begins with the step following the $(i-1)$st success and ends with the step at which the ith success occurs. Let also the r.v. X_i, $1 \leq i \leq n-1$, be the number of steps in the i-th epoch. Let p_i be the probability of success at any step during the i-th epoch. We have $p_i = \frac{i(n-i)}{m} = \frac{2i(n-i)}{n(n-1)}$, where $m = |E_I| = n(n-1)/2$ denotes the total number of possible interactions and $\mathrm{E}[X_i] = 1/p_i = \frac{n(n-1)}{2i(n-i)}$. By linearity of expectation we have

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}[\sum_{i=1}^{n-1} X_i] = \sum_{i=1}^{n-1} \mathrm{E}[X_i] = \sum_{i=1}^{n-1} \frac{n(n-1)}{2i(n-i)} \\
&= \frac{n(n-1)}{2} \sum_{i=1}^{n-1} \frac{1}{i(n-i)} \\
&= \frac{n(n-1)}{2} \sum_{i=1}^{n-1} \frac{1}{n} \left(\frac{1}{i} + \frac{1}{n-i} \right) \\
&= \frac{(n-1)}{2} \left[\sum_{i=1}^{n-1} \frac{1}{i} + \sum_{i=1}^{n-1} \frac{1}{n-i} \right] \\
&= \frac{(n-1)}{2} 2H_{n-1} = (n-1)[\ln(n-1) + \Theta(1)] \\
&= \Theta(n \log n),
\end{aligned}$$

where H_n denotes the nth Harmonic number. □

One-to-one elimination. All nodes are initially in state a. The only effective transition of the protocol is $(a, a) \to (a, b)$. We are now interested in the expected time until a single a remains. We call the process one-to-one elimination because as are only eliminated with themselves. A straightforward application is in protocols that elect a unique leader by beginning with all nodes in the leader state and eliminating a leader whenever two leaders interact.

Proposition 2 *The expected time to convergence of a one-to-one elimination is $\Theta(n^2)$.*

Proof Epoch i begins with the step following the ith success and ends with the step at which the $(i+1)$st success occurs. The probability of success during the ith epoch, for $0 \le i \le n-2$, is $p_i = [(n-i)(n-i-1)/2]/[n(n-1)/2] = [(n-i)(n-i-1)]/[n(n-1)]$ and

$$\begin{aligned}
\mathrm{E}[X] &= n(n-1)\sum_{i=0}^{n-2}\frac{1}{(n-i)(n-i-1)} \\
&= n(n-1)\sum_{i=2}^{n}\frac{1}{i(i-1)} \\
&< n(n-1)\sum_{i=2}^{n}\frac{1}{(i-1)^2} \\
&= n(n-1)\sum_{i=1}^{n-1}\frac{1}{i^2} < 2n(n-1) < 2n^2.
\end{aligned}$$

The above uses the fact that $\sum_{i=1}^{n-1} 1/i^2$ is less than 2. This holds because $\sum_{i=1}^{n-1} 1/i^2 < 1 + \int_{s=1}^{n}(1/s^2)\mathrm{d}s = 1 + \left[-s^{-1}\right]_{s=1}^{n} = 2 - 1/n < 2$.

Now, for the lower bound, observe that the last two as need on average $n(n-1)/2$ steps to meet each other. As $n(n-1)/2 \le \mathrm{E}[X] < n^2$, we conclude that $\mathrm{E}[X] = \Theta(n^2)$. □

Maximum matching. A slight variation of the one-to-one elimination protocol constructs a *maximum matching*, i.e., a matching of cardinality $\lfloor n/2 \rfloor$ (which is a perfect matching in case n is even). The variation is $(a, a, 0) \to (b, b, 1)$ and its running time is again $\Theta(n^2)$, which we now prove.

Proposition 3 *The expected time to convergence of a maximum matching is $\Theta(n^2)$.*

Proof For the upper bound, we shall prove that the running time of a one-to-one elimination, i.e., $\Theta(n^2)$, is an upper bound on the maximum matching variation. Note first that this cannot be proved by executing the two processes side-by-side on the same schedule, because there are rare schedules for which one-to-one elimination stabilizes much faster than maximum matching. An extreme such example is the schedule of length $n-1$ in which a particular a eliminates one after the other all other as (here, we have also included in the schedule the random choice of the winner of an elimination). At the end of this schedule, one-to-one elimination has stabilized, having eliminated $n-1$ as, while maximum matching has only managed to eliminate 2 as.

A way to establish the upper bounding relation is the following. Both protocols begin from n as and they stabilize when at least $n-1$ as have been eliminated. Both eliminate as by an (a, a) interaction: maximum matching eliminates both as while one-to-one elimination eliminates only one of them. Take now the sequence C_{n-2i}, for $0 \le i \le \lfloor n/2 \rfloor$, of distinct node-configurations from which maximum matching passes (here, the index of configuration C represents the number of as in C) and observe that one-to-one elimination cannot skip any of these configurations. Finally, observe that for any C_j in the sequence, both protocols have the same probability of making progress under C_j. When maximum matching makes progress it moves to C_{j-2}. On the other hand, when one-to-one elimination makes progress it moves to a C_{j-1} not in the sequence and needs one or more additional steps to reach C_{j-2} and catch up the other process.

For the lower bound, notice that when only two (or three) as remain the expected number of steps for a success is $n(n-1)/2$ ($n(n-1)/6$, respectively), that is, the running time is also $\Omega(n^2)$. We conclude that the protocol constructs a maximum matching in an expected number of $\Theta(n^2)$ steps. □

One-to-all elimination. All nodes are initially in state a. The effective rules of the protocol are $(a, a) \to (b, a)$ and $(a, b) \to (b, b)$. We are now interested in the expected time until no a remains. The process is called one-to-all elimination because as are eliminated not only when they interact with as but also when they interact with bs. At a first sight, it seems to run faster than a one-way epidemic as bs still propagate towards as as in a one-way epidemic but now bs are also created when two as interact. We show that this is not the case.

Proposition 4 *The expected time to convergence of a one-to-all elimination is $\Theta(n \log n)$.*

Proof The probability of success during the ith epoch, for $0 \le i \le n-1$, is $p_i = 1 - [i(i-1)/2]/[n(n-1)/2] = [n(n-1) - i(i-1)]/[n(n-1)]$ and

$$\mathrm{E}[X] = n(n-1)\sum_{i=0}^{n-1}\frac{1}{n(n-1) - i(i-1)}.$$

For the upper bound, we have

$$\begin{aligned}
\mathrm{E}[X] &= n(n-1)\sum_{i=0}^{n-1}\frac{1}{n(n-1)-i(i-1)} \\
&< n(n-1)\left[\sum_{i=0}^{n-2}\frac{1}{(n-1)^2-i^2}\right]+\frac{n}{2} \\
&= \frac{n}{2}\left(\sum_{i=0}^{n-2}\frac{1}{n-i-1}+\sum_{i=0}^{n-2}\frac{1}{n+i-1}+1\right) \\
&= \frac{n}{2}\left(\sum_{i=1}^{n-1}\frac{1}{i}+\sum_{i=1}^{2n-3}\frac{1}{i}-\sum_{i=1}^{n-2}\frac{1}{i}+1\right) \\
&= \frac{n}{2}\left(\frac{1}{n-1}+\sum_{i=1}^{2n-3}\frac{1}{i}+1\right) \\
&= \frac{n}{2}H_{2n-3}+\frac{n}{2}+\frac{n}{2(n-1)} \\
&< n(H_{2n}+1) = n[\ln 2n+\Theta(1)].
\end{aligned}$$

For the lower bound, we have

$$\begin{aligned}
\mathrm{E}[X] &= n(n-1)\sum_{i=0}^{n-1}\frac{1}{n(n-1)-i(i-1)} \\
&> n(n-1)\sum_{i=0}^{n-1}\frac{1}{n^2-(i-1)^2} \\
&= \frac{n-1}{2}\left(\sum_{i=0}^{n-1}\frac{1}{n-i+1}+\sum_{i=0}^{n-1}\frac{1}{n+i-1}\right) \\
&= \frac{n-1}{2}\left(\sum_{i=1}^{n+1}\frac{1}{i}+\sum_{i=1}^{2n-2}\frac{1}{i}-\sum_{i=1}^{n-2}\frac{1}{i}-1\right) \\
&= \frac{n-1}{2}\left(\sum_{i=1}^{2n-2}\frac{1}{i}+\frac{1}{n-1}+\frac{1}{n}+\frac{1}{n+1}-1\right) \\
&> \frac{n-1}{2}(H_{2n-2}-1) \\
&= \frac{n-1}{2}[\ln(2n-2)+\Theta(1)].
\end{aligned}$$

We conclude that $\mathrm{E}[X]=\Theta(n\log n)$. □

Meet everybody. A single node u is initially in state a and all other nodes are in state b. The only effective transition is $(a,b)\to(a,c)$. We study the time until all bs become cs which is equal to the time needed for u to interact with every other node.

Proposition 5 *The expected time to convergence of a meet everybody is $\Theta(n^2\log n)$.*

Proof Assume that in every step u participates in an interaction. Then u must collect the $n-1$ coupons which are $n-1$ different nodes that it must interact with. Clearly, in every step, every node has the same probability to interact with u, i.e., $1/(n-1)$, and this is the classical coupon collector problem that takes average time $\Theta(n\log n)$. But on average u needs $\Theta(n)$ steps to participate in an interaction, thus the total time is $\Theta(n^2\log n)$. □

Node cover. All nodes are initially in state a. The only effective transitions are $(a,a)\to(b,b)$, $(a,b)\to(b,b)$. We are interested in the number of steps until all nodes become bs, i.e., the time needed for every node to interact at least once.

Proposition 6 *The expected time to convergence of a node cover is $\Theta(n\log n)$.*

Proof For the upper bound, simply observe that the running time of a one-to-all elimination, i.e., $\Theta(n\log n)$, is an upper bound on the running time of a node cover. The reason is that a node cover is a one-to-all elimination in which in some cases we may get two new bs by one effective transition (namely $(a,a)\to(b,b)$) while in one-to-all elimination all effective transitions result in at most one new b.

For the lower bound, if i is the number of bs then the probability of success is $p_i=1-[i(i-1)]/[n(n-1)]$. Observe now that a node cover process is slower than the artificial variation in which whenever rule $(a,b)\to(b,b)$ applies we pick another a and make it a b. This is because, given i bs, this artificial process has the same probability of success as a node cover but additionally in every success the artificial process is guaranteed to produce two new bs while a node cover may in some cases produce only one new b. Define $k=\lceil n/2\rceil+1$. Then, taking into account what we already proved in the lower bound of one-to-all elimination (see Proposition 4), we have

$$\begin{aligned}
\mathrm{E}[X] &\geq n(n-1)\sum_{i=0}^{\lceil n/2\rceil}\frac{1}{n(n-1)-2i(2i-1)} \\
&= \frac{n(n-1)}{4}\sum_{i=0}^{k-1}\frac{1}{\frac{n(n-1)}{4}-\frac{2i(2i-1)}{4}} \\
&= \frac{n(n-1)}{4}\sum_{i=0}^{k-1}\frac{1}{\frac{n}{2}(\frac{n}{2}-\frac{1}{2})-i(i-\frac{1}{2})} \\
&> \frac{n(n-1)}{4}\sum_{i=0}^{k-1}\frac{1}{k(k-1)-i(i-1)} \\
&> \frac{n(n-1)}{8k}(H_{2k-2}-1) > \frac{n-1}{8}(H_n-1) \\
&= \frac{n-1}{8}[\ln n+\Theta(1)].
\end{aligned}$$

We conclude that $\mathrm{E}[X]=\Theta(n\log n)$. □

Edge cover. All nodes are in state a throughout the execution of the protocol. The only effective transition is $(a, a, 0) \to (a, a, 1)$ (we now focus on edge-state updates), i.e., whenever an edge is found inactive it is activated (recall that initially all edges are inactive). We study the number of steps until all edges in E_I become activated, which is equal to the time needed for all possible interactions to occur.

Proposition 7 *The expected time to convergence of an edge cover is $\Theta(n^2 \log n)$.*

Proof Given that $m = n(n-1)/2$ and given that j successes (i.e., j distinct interactions) have occurred the corresponding probability for the coupon collector argument is $p_j = (m-j)/m$ and the expected number of steps is $\mathrm{E}[X] = \sum_{i=0}^{m-1} m/(m-i) = m\sum_{i=0}^{m-1} 1/(m-i) = m\sum_{i=1}^{m} 1/i = m(\ln m + \Theta(1)) = \Theta(n^2 \log n)$. Another way to see this is to observe that it is a classical coupon collector problem with m coupons each selected in every step with probability $1/m$, thus $\mathrm{E}[X] = m \ln m + O(m) = \Theta(n^2 \log n)$. □

Table 1 summarizes the expected time to convergence of each of the above fundamental probabilistic processes.

Protocol	Expected Time
One-way epidemic	$\Theta(n \log n)$
One-to-one elimination	$\Theta(n^2)$
Maximum matching	$\Theta(n^2)$
One-to-all elimination	$\Theta(n \log n)$
Meet everybody	$\Theta(n^2 \log n)$
Node Cover	$\Theta(n \log n)$
Edge cover	$\Theta(n^2 \log n)$

Table 1 Our results for the expected time to convergence of several fundamental probabilistic processes.

4 Constructing a Global Line

In this section, we study probably the most fundamental network-construction problem, which is the problem of constructing a spanning line. Its importance lies in the fact that a spanning line provides an ordering on the processes which can then be exploited (as shown in Section 6) to simulate a TM and thus to establish universality of our model. We give two different protocols for the spanning line problem, a simple (w.r.t. the number of states) and a fast one.

We begin with a generic lower bound holding for all protocols that construct a spanning network.

Theorem 1 (Generic Lower Bound) *The expected time to convergence of any protocol that constructs a spanning network, i.e., one in which every node has at least one active edge incident to it, is $\Omega(n \log n)$. Moreover, this is the best lower bound for general spanning networks that we can hope for, as there is a protocol that constructs a spanning network in $\Theta(n \log n)$ expected time.*

Proof Consider the time at which the last edge is activated. Clearly, by that time, all nodes must have some active edge incident to them which implies that every node must have interacted at least once. Thus the running time is lower bounded by a node cover, which by Proposition 6 takes an expected number of $\Theta(n \log n)$ steps.

Now consider the variation of node cover which in every transition that is effective w.r.t. node-states additionally activates the corresponding edge. In particular, the protocol consists of the rules $(a, a, 0) \to (b, b, 1)$ and $(a, b, 0) \to (b, b, 1)$. Clearly, when every node has interacted at least once, or equivalently when all as have become bs, every node has an active edge incident to it, and thus the resulting stable network is spanning. The reason is that all nodes are as in the beginning, every node at some point is converted to b, and every such conversion results in an activation of the corresponding edge. As a node-cover completes in $\Theta(n \log n)$ steps, the above protocol takes $\Theta(n \log n)$ steps to construct a spanning network. □

We now give an improved lower bound for the particular case of constructing a spanning line.

Theorem 2 (Line Lower Bound) *The expected time to convergence of any protocol that constructs a spanning line is $\Omega(n^2)$.*

Proof Take any protocol $\mathcal{A}$ that constructs a spanning line and any execution of $\mathcal{A}$ on n nodes. It suffices to show that any execution necessarily passes through a "bottleneck" transition [6], by which we mean a transition that requires $\Omega(n^2)$ expected number of steps to occur. The idea is that in any execution the set of active edges eventually stabilizes (in this case, to a spanning line), which implies that there is always a *last* activation/deactivation of an edge. We shall show that either this last operation is a bottleneck transition or an immediately previous operation is a bottleneck transition. In both cases, any execution passes through a bottleneck transition, thus paying at that point an $\Omega(n^2)$ expected number of steps.

[6] To the best of our knowledge, the term "bottleneck" to characterize such types of slow transitions in the context of population protocols, was first used in [CCDS14].

Consider the step t at which $\mathcal{A}$ performed the last modification of an edge. Observe that the construction after step t must be a spanning line. We distinguish two cases.

(i) The last modification was an activation. In this case, the construction just before step t was either a line on $n-1$ nodes and an isolated node or two disjoint lines spanning all nodes. To see this, observe that these are the only constructions that can be turned into a line by a single additional activation. In the first case, the probability of obtaining an interaction between the isolated node and one of the endpoints of the line is $4/[n(n-1)]$ and in the second the probability of obtaining an interaction between an endpoint of one line and an endpoint of the other line is $8/[n(n-1)]$. In both cases, the expected number of steps until the last edge becomes activated is $\Omega(n^2)$.

(ii) The last modification was a deactivation. This implies that the construction just before step t was a spanning line with an additional active edge between two nodes, u and v, that are not neighbors on the line. If one of these nodes, say u, is an internal node, then u has degree 3 and we can only obtain a line by deactivating one of the edges incident to u. Clearly, the probability of getting one of these edges is $6/[n(n-1)]$ and it is even smaller if both nodes are internal. Thus, if at least one of u and v is internal, the expected number of steps is $\Omega(n^2)$. It remains to consider the case in which the construction just before step t was a spanning ring, i.e., the case in which u and v are the endpoints of the spanning line. In this case, consider the step $t' < t$ of the last modification of an edge that resulted in the ring. To this end notice that all nodes of a ring have degree 2. If t' was an activation then exactly two nodes had degree 1 and if t' was a deactivation then two nodes had degree 3. In both cases, there is a single interaction that results in a ring, the probability of success is $2/[n(n-1)]$ and the expectation is again $\Omega(n^2)$. □

We proceed by presenting protocols for the spanning line problem.

4.1 1st Protocol

We present now our simplest protocol (Protocol 1) for the spanning line problem.

Theorem 3 *Protocol* Simple-Global-Line *constructs a spanning line. It uses 5 states and its expected running time is $\Omega(n^4)$ and $O(n^5)$.*

Proof We begin by proving that, for any number of processes $n \geq 2$, the protocol correctly constructs a spanning line under any fair scheduler. Then we study the running time of the protocol under the uniform random scheduler.

Protocol 1 *Simple-Global-Line*

```
$Q = \{q_0, q_1, q_2, l, w\}$
$\delta$:

$(q_0, q_0, 0) \rightarrow (q_1, l, 1)$
$(l, q_0, 0) \rightarrow (q_2, l, 1)$
$(l, l, 0) \rightarrow (q_2, w, 1)$
$(w, q_2, 1) \rightarrow (q_2, w, 1)$
$(w, q_1, 1) \rightarrow (q_2, l, 1)$

// All transitions that do not appear have no effect
```

Correctness. In the initial configuration C_0, all nodes are in state q_0 and all edges are inactive, i.e in state 0. Every configuration C that is reachable from C_0 consists of a collection of lines and isolated nodes. Additionally, every line has a unique leader which either occupies an endpoint and is in state l or occupies an internal node, is in state w, and moves along the line. Whenever the leader lies on an endpoint of its line, its state is l and whenever it lies on an internal node, its state is w. Lines can expand towards isolated nodes and two lines can connect their endpoints to get merged into a single line (with total length equal to the sum of the lengths of the merged lines plus one). Both of these operations only take place when the corresponding endpoint of every line that takes part in the operation is in state l. Figure 2 gives an illustration of a typical configuration of the protocol.

We have to prove two things: (i) there is a set $\mathcal{S}$ of output-stable configurations whose active network is a spanning line, (ii) for every reachable configuration C (i.e., $C_0 \rightsquigarrow C$) it holds that $C \rightsquigarrow C_s$ for some $C_s \in \mathcal{S}$. For (i), consider a spanning line, in which the non-leader endpoints are in state q_1, the non-leader internal nodes in q_2, and there is a unique leader either in state l if it occupies an endpoint or in state w if it occupies an internal node. For (ii), note that any reachable configuration C is a collection of lines with unique leaders and isolated nodes in state q_0. We present a (finite) sequence of transitions that converts C to a $C_s \in \mathcal{S}$. If there are isolated nodes, take any line and if its leader is internal make it reach one of the endpoints by selecting the appropriate interactions. Then successively apply the rule $(l, q_0, 0) \rightarrow (q_2, l, 1)$ to expand the line towards all isolated nodes. Thus we may now without loss of generality (abbreviated "w.l.o.g." throughout) consider a collection of lines without isolated nodes. By successively applying the rule $(l, l, 0) \rightarrow (q_2, w, 1)$ to pairs of lines while always moving the internal leaders that appear towards an endpoint it is not hard to see that the

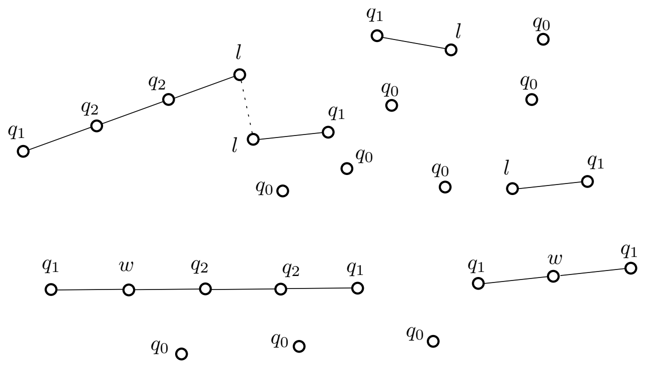

Fig. 2 This is a typical configuration of Protocol *Simple-Global-Line* (after some time has passed). Lines with a w internal-leader only wait until the random walk of w reaches one endpoint and becomes an l leader. Lines with an l leader can expand towards isolated nodes in state q_0 or merge to other such lines. An example of the latter is the interaction over the dotted edge. The result will be the activation of the edge (merging the two lines into a longer one) and the replacement of the l leaders by a q_2 and a w internal-leader that will perform a random walk until it reaches one of the two endpoints of the new line.

process results in an output-stable configuration from $\mathcal{S}$, i.e., one whose active network is a spanning line.

Running Time Upper Bound. For the running time upper bound, we have an expected number of $O(n^2)$ steps until progress is made (i.e., for another merging to occur given that at least two l-leaders exist) and $O(n^4)$ steps for the resulting random walk (walk of state w until it reaches one endpoint of the line) to finish and to have the system again ready for progress. The $O(n^4)$ bound holds because we have a random walk on a line with two absorbing barriers (see, e.g., [Fel68] pages 348-349) delayed on average by a factor of $O(n^2)$. The delay is $O(n^2)$ because there is a unique walking state on one of the n nodes, so it is selected on average every n steps. But, additionally, the state actually walks only if it interacts with one of its (at most) two neighbors on the line. As only 2 interactions over the $\Theta(n^2)$ possible interactions allow the state to walk, the walk is delayed by a factor of $O(n^2)$. As progress must be made $n-2$ times, we conclude that the expected running time of the protocol is bounded from above by $(n-2)[O(n^2)+O(n^4)] = O(n^5)$.

We next prove that we cannot hope to improve the upper bound on the expected running time by a better analysis by more than a factor of n. For this, we first prove that the protocol with high probability (abbreviated "w.h.p." throughout) constructs $\Theta(n)$ disjoint lines of length 1 during its course. A set of k disjoint lines implies that $k-1$ distinct merging processes have to be executed in order to merge them all into a common line and each single merging results in the execution of another random walk. Based on these, we prove the desired $\Omega(n^4)$ lower bound.

Recall that initially all nodes are in q_0. Every interaction between two q_0-nodes constructs another line of length 1. Call the random interaction of step i a *success* if both participants are in q_0. Let the r.v. R be the number of nodes in state q_0; i.e., initially $R = n$. Note that, at every step, R decreases by at most 2, which happens only in a success (it may also remain unchanged, or decrease by 1 if a leader expands towards a q_0). Let the r.v. X_i be the number of successes up to step i and X be the total number of successes throughout the course of the protocol, that is, until at least $n-1$ q_0s have been converted to something else. Our goal is to calculate the expectation of X as this is equal to the number of distinct lines of length 1 that the protocol is expected to form throughout its execution (note that these lines do not necessarily have to coexist). Given R, the probability of success at the current step is $p_R = [R(R-1)]/[n(n-1)] \geq (R-1)^2/n^2$. As long as $R \geq (n/2)+1 = z$ it holds that $p_R \geq (n^2/4)/n^2 = 1/4$. Moreover, as R decreases by at most 2 in every step, there are at least $(n-z)/2 = [(n/2)-1]/2 = (n/4)-1/2$ steps until R becomes less than or equal to z. Thus, our process *dominates* a Bernoulli process Y with $(n/4)-1/2$ trials and probability of success $p' = 1/4$ in each trial. For this process we have $\mathrm{E}[Y] = [(n/4)-1/2](1/4) = (n/16)-1/8 = \Theta(n)$.

We now exploit the following Chernoff bound (cf. [MR95], page 70) establishing that w.h.p. Y does not deviate much below its mean $\mu = \mathrm{E}[Y]$:

Chernoff Bound. Let $Y_1, Y_2, \ldots, Y_t$ be independent Poisson trials such that, for $1 \leq i \leq t$, $\mathrm{P}[Y_i = 1] = p_i$, where $0 < p_i < 1$. Then, for $Y = \sum_{i=1}^t Y_i$, $\mu = \mathrm{E}[Y] = \sum_{i=1}^t p_i$, and $0 < \delta < 1$,

$$\mathrm{P}[Y < (1-\delta)\mu] < \exp(-\mu\delta^2/2).$$

Additionally, it holds that $\exp(-\mu\delta^2/2) = \epsilon \Leftrightarrow \delta = \sqrt{\frac{2\ln 1/\epsilon}{\mu}}$. Thus $\exp(-\mu\delta^2/2) = n^{-c}$ implies $\delta^2 = \frac{2c\ln n}{\mu} = \frac{2c\ln n}{(1/8)(n/2-1)} = \frac{16c\ln n}{n/2-1} \Rightarrow \delta = \sqrt{\frac{16c\ln n}{n/2-1}} \Rightarrow$

$$(1-\delta)\mu = \frac{1}{8}\left(1-\sqrt{\frac{16c\ln n}{n/2-1}}\right)\left(\frac{n}{2}-1\right)$$
$$> \frac{1}{16}\left(n-2\sqrt{cn\ln n}-2\right) = \Theta(n).$$

So, for all $c = O(1)$,

$$\mathrm{P}[Y < \frac{1}{16}\left(n-2\sqrt{cn\ln n}-2\right)] < n^{-c} \Rightarrow$$
$$\mathrm{P}[Y \geq \frac{1}{16}\left(n-2\sqrt{cn\ln n}-2\right)] > 1-n^{-c}$$

and as X dominates Y, we have $\mathrm{P}[X \geq (1/16)(n - 2\sqrt{cn\ln n} - 2)] > 1 - n^{-c}$. In words, w.h.p. we expect at least $k = (1/16)(n - 2\sqrt{cn\ln n} - 2) = \Theta(n)$ disjoint lines of length 1 to be constructed by the protocol.

Now, let us focus on those executions, on a population of size n, that satisfy $X \geq k$. Given such an execution, consider the first time t_{min} at which (after a merging or an expansion) there is a line L of length at least $k/4$. If we denote by h the length of L at t_{min}, it must also hold that $h \leq k/2 - 1$, because the maximum growth before time t_{min} is via a merging of two lines both of length $k/4 - 1$, which (by also taking into account the new edge between them) gives length $k/2 - 1$. Thus, we have $k/4 \leq h \leq k/2 - 1$.

The total length due to lines of length 1 (ever to appear) is at least k and, at t_{min}, L can have already obtained at most h of this length. Therefore, at t_{min} there is still a remaining length of at least $k - h \geq k - (k/2 - 1) = k/2 + 1$ to get merged to L via $j \geq 1$ distinct mergings. These mergings, and thus also the resulting random walks, cannot occur in parallel as all of them share L as a common participant (and a line can only participate in one merging at a time). Let d_i denote the length of the i-th line merged to L, for $1 \leq i \leq j$. If L has length $d(L)$ just before the i-th merging, then the expected duration of the resulting random walk is $n^2 \cdot d(L) \cdot d_i$ and the new L resulting from merging will have length $d(L) + d_i$. Let Y denote the duration of all random walks, and Y_i, $1 \leq i \leq j$, the duration of the i-th random walk. In total, the expected duration of all random walks resulting from the j mergings of L is

$$\begin{aligned}
\mathrm{E}[Y] &= \mathrm{E}[\sum_{i=1}^{j} Y_i] = \sum_{i=1}^{j} \mathrm{E}[Y_i] \\
&= \sum_{i=1}^{j} n^2(h + d_1 + \ldots + d_{i-1})d_i \\
&\geq n^2 \sum_{i=1}^{j} h d_i = n^2 h \sum_{i=1}^{j} d_i \\
&\geq n^2 \cdot \frac{k}{4} \cdot (\frac{k}{2} + 1) \\
&= n^2 \cdot \Theta(n) \cdot \Theta(n) \\
&= \Theta(n^4).
\end{aligned}$$

The second inequality follows from the fact that $\sum_{i=1}^{j} d_i = k - h \geq \frac{k}{2} + 1$. We conclude that, in case $X \geq k$, the expected running time of the protocol is $\Omega(n^4)$.

Finally, for calculating the total expected running time of the protocol, we take into account all possible executions and not only those that satisfy $X \geq k$. If we define the r.v. W to be the total running time of the protocol (until convergence), by the law of total probability and for every constant $c \geq 1$, we have that:

$$\begin{aligned}
\mathrm{E}[W] = {} & \mathrm{E}[W \mid X \geq k] \cdot \mathrm{P}[X \geq k] + \\
& \mathrm{E}[W \mid X < k] \cdot \mathrm{P}[X < k] \\
\geq {} & \mathrm{E}[W \mid X \geq k] \cdot \mathrm{P}[X \geq k] \\
> {} & \left(n^2 \cdot \frac{k}{4} \cdot (\frac{k}{2} + 1)\right)(1 - n^{-c}) \\
= {} & n^2 \cdot \Theta(n) \cdot \Theta(n) \cdot (1 - n^{-c}) \\
= {} & \Theta(n^4).
\end{aligned}$$

Thus, the expected running time of the protocol is $\Omega(n^4)$. □

4.2 2nd Protocol

We now give our fastest protocol (Protocol 2) for the global line construction. The main difference between this and the previous protocol is that we now totally avoid mergings as they seem to consume much time. In fact, merging two lines of total length $\Theta(n)$ requires $\Theta(n^3)$ time as every step takes an average of $\Theta(n^2)$ time and if, for example, $\Theta(n)$ such mergings have to be performed to obtain a spanning line, then the time-complexity becomes $\Omega(n^4)$, which is quite big.

We first give the intuition behind Protocol 2. As in Protocol 1, when the leaders of two lines interact, one of them becomes eliminated and the edge is activated. But in contrast in Protocol 1, the leader that has survived does not initiate a merging process. Instead, it steals a node from the eliminated leader's line and disconnects the two new lines: its own line, which has increased by one and is called *awake*, and the eliminated leader's line, which has decreased by one and is called *sleeping*.

In more detail, when two lines L_1 and L_2 interact via their l-leader endpoints, one of the leaders, say w.l.o.g. that of L_2, becomes l' and the other becomes q_2'. We can interpret this operation as expanding L_1 on the endpoint of L_2 and obtaining two new lines (still attached to each other): L_1' which is awake and L_2' which is sleeping. Now, the l'-leader of L_1' waits to interact with its neighbor from L_2' (which is either a q_2 or a q_1) to deactivate the edge between them and disconnect L_1' from L_2'. This operation leaves L_1' with an l''-leader and L_2' with a sleeping leader f_1 (it can also be the case that L_2' is just a single isolated f_0, in case L_2 consisted only of 2 nodes). Then l'' waits to meet its q_2' neighbor to convert it to q_2 and update itself to l. This completes the operation of a line growing one step towards another line and making the other line sleep. A sleeping line cannot increase any more and only loses nodes to lines that are still awake by a similar operation as the one just described. A single leader is guaranteed to always win and this occurs quite fast. Then the unique leader does not need much time to collect all nodes from the sleeping lines to its own line and make the latter spanning.

Protocol 2 *Fast-Global-Line*

```
$Q = \{q_0, q_1, q_2, q_2', l, l', l'', f_0, f_1\}$
$\delta$:

$(q_0, q_0, 0) \to (q_1, l, 1)$
$(l, q_0, 0) \to (q_2, l, 1)$
$(l, l, 0) \to (q_2', l', 1)$
$(l', q_2, 1) \to (l'', f_1, 0)$
$(l', q_1, 1) \to (l'', f_0, 0)$
$(l'', q_2', 1) \to (l, q_2, 1)$
$(l, f_0, 0) \to (q_2, l, 1)$
$(l, f_1, 0) \to (q_2', l', 1)$
```

Theorem 4 *Protocol* Fast-Global-Line *constructs a spanning line. It uses 9 states and its expected running time under the uniform random scheduler is $O(n^3)$.*

Proof Correctness is straightforward. The configuration is always a collection of awake (with a unique l, l', or l'' leader) and sleeping (with a unique f_1 leader) lines and isolated nodes (either awake in q_0 or sleeping in f_0). As long as there are at least two awake lines, eventually another line becomes sleeping, so eventually a single awake line will remain with all other nodes being sleeping (either part of a sleeping line or isolated). The protocol ensures that an awake line can always grow towards sleeping nodes (either by stealing them from sleeping lines or by expanding towards isolated nodes), so eventually the unique awake line will become spanning.

For the time analysis, observe first that in $O(n^2)$ steps all q_0s become something else. To see this let the r.v. X be the total number of steps until all q_0s disappear and let the r.v. X_i be the number of steps between the ith and the $(i+1)$st interaction between two nodes in state q_0 (assume no other interactions can change the state of a q_0). Let $p_i = [(n-2i)(n-2i-1)]/[n(n-1)]$ be the probability that such an interaction occurs. Then $\mathrm{E}[X_i] = 1/p_i = \Theta(n^2/(n-i)^2)$ and $\mathrm{E}[X] \simeq n^2 \sum_{i=1}^{n/2} 1/(n-i)^2 = \Theta(n^2)$. The last equation follows from the fact that $\sum_{i=1}^{n/2} 1/(n-i)^2 \leq \sum_{i=1}^{n^2} 1/i - \sum_{i=1}^{(n/2)^2} 1/i \simeq 2\ln n + \Theta(1) - 2\ln n + 2\ln 2 - \Theta(1) = O(1)$, i.e., it is bounded. Finally, observe that q_0s that become leaders can also turn other q_0s to something else thus the actual expectation is in fact $O(n^2)$ (i.e., what we have ignored can only help the process end faster).

Now notice that after this $O(n^2)$ time we have a set of at most $O(n)$ leaders and no new leader can ever appear. Moreover, in every interaction between two leaders only one survives and the other becomes a follower. Clearly, a single leader must win all the pairwise games in which it will participate. Consider that leader and observe that it takes it an average of n^2 steps to participate to another game in the worst case and another n^2 steps to win it. As it may have to eliminate up to $O(n)$ other leaders, in $O(n^3)$ steps on average there is a unique leader and every other node is either isolated in state f_0 or part of a line that has a unique follower f_1. Every interaction of a leader with a follower increases the length of the leader's line by 1 in $O(n^2)$ steps. Thus an increment occurs every $O(n^2)$ steps as the leader needs $O(n^2)$ steps to meet a follower and then $O(n^2)$ steps to increase by 1 towards that follower. As the leader needs to make at most $O(n)$ increments to make its own line global, we conclude that the expected time for this to occur is $O(n) \cdot O(n^2) = O(n^3)$. □

5 Other Basic Constructors

In this section, we present direct constructors and some lower bounds for several other basic network construction problems (defined in Section 3.2). We have analyzed the running times of most of our protocols. Those missing are left as open problems.

Cycle Cover

Protocol 3 *Cycle-Cover*

```
$Q = \{q_0, q_1, q_2\}$
$\delta$:

$(q_0, q_0, 0) \to (q_1, q_1, 1)$
$(q_1, q_0, 0) \to (q_2, q_1, 1)$
$(q_1, q_1, 0) \to (q_2, q_2, 1)$
```

Theorem 5 *Protocol* Cycle-Cover *constructs a cycle cover with waste 2 (i.e., a cycle cover on a subset of V_I of $n-2$ nodes). It uses 3 states, its expected running time under the uniform random scheduler is $\Theta(n^2)$, and it is optimal w.r.t. time.*

Proof The protocol preserves the following invariant: the degree of a node in state q_i, $0 \leq i \leq 2$, is i. Moreover, all interactions $(q_i, q_j, 0)$ with $i, j \in \{0, 1\}$ result in $(q_{i+1}, q_{j+1}, 1)$, that is, in an activation and a corresponding increase in the recorded degrees. As a result, as long as there are at least two disconnected nodes with degrees smaller than two, these two nodes can become connected. It follows that any component with at least three nodes eventually becomes a cycle and in the final stable configuration there can be at most one component that is not a cycle: either an isolated node, or two nodes connected by an active edge. So, the waste is indeed 2.

Note that the protocol stabilizes when at least $n-2$ nodes have become q_2 (the rest is the waste which consists of at most 2 nodes). In $O(n^2)$ time (by dominating a maximum matching) all q_0s have become q_1 and in another $O(n^2)$ steps all q_1s have become q_2s. We now give a lower bound that holds for *any protocol* that constructs a cycle cover, so we have to also take into account the possibility that the protocol deactivates some edges (even though our protocol never does this). To this end, consider the last edge modification that ever occurs. Due to the symmetry of cycle cover, both if it was an activation or a deactivation only a single edge satisfies the fact that after its activation or deactivation we get a cycle cover, which requires $\Theta(n^2)$ rounds. □

Global Star

Theorem 6 (Star Lower Bound) *Any protocol that constructs a spanning star has at least 2 states and its expected time to convergence is $\Omega(n^2 \log n)$.*

Proof Clearly, with a single state we cannot make the necessary distinction of a center and a peripheral node. More formally, if there is a single state q_0 then $(q_0, q_0, 0)$ must necessarily activate the edge (otherwise no edges will be ever activated) which implies that eventually all edges will become activated, i.e., instead of a star we will end up with a global clique. So every protocol that constructs a global star must have at least 2 states.

For the lower bound on the expected running time we argue as follows. Take any execution of a protocol that constructs a global star. Consider the node u that will become the center in that execution. When the execution stabilizes, u must be connected to every other node by an active edge. This implies that u must have interacted with every other node. Clearly, the time it takes for the eventually unique center, u in this case, to meet every other node is a lower bound on the total running time. This is a meet everybody that, as proved in Proposition 5, takes $\Theta(n^2 \log n)$ time. □

Protocol 4 *Global-Star*

```
$Q = \{c, p\}$, $q_0 = c$
$\delta$:

$(c, c, 0) \rightarrow (c, p, 1)$
$(p, p, 1) \rightarrow (p, p, 0)$
$(c, p, 0) \rightarrow (c, p, 1)$
```

Theorem 7 *Protocol* Global-Star *constructs a spanning star. It uses 2 states and its expected running time under the uniform random scheduler is $O(n^2 \log n)$, which is optimal both w.r.t. size and time.*

Proof Correctness. At any given time during the execution of the protocol, a node may be playing one of the following two roles: a *center* (state c) or a *peripheral* (state p). The unique output-stable configuration C_f whose active network is a spanning star, has one center and $n-1$ peripheral nodes, and a uv edge is active iff one of u, v is the center. Initially all nodes are centers. When two centers interact one of them remains a center and the other becomes a peripheral. No other interactions eliminate a center, which implies that not all centers can be eliminated, and once a center becomes a peripheral it can never become a center again. Due to fairness, eventually all pairs of centers will interact and, as no new centers appear, eventually a single center will remain. Thus from some point on there is a single center and $n-1$ peripheral nodes. The idea from now on is that c-p attract while p-p repel. In particular, rule $(c, p, 0) \rightarrow (c, p, 1)$ guarantees that any inactive edges joining the center to the peripherals will become activated and rule $(p, p, 1) \rightarrow (p, p, 0)$ guarantees that any active edges joining two peripherals will become deactivated. At the same time active edges between the center and the peripherals remain active and inactive edges between two peripherals remain inactive. This clearly leads to the construction of a spanning star.

Running Time. Forget for a while the edge updates and consider the rule $(c, c) \rightarrow (c, p)$, which is the only effective interaction of the protocol w.r.t. the states of the nodes. We are interested in the time needed for a single c to remain. This is clearly an original application of one-to-one elimination and as proved in Proposition 2 it takes $\Theta(n^2)$ time.

Notice now that once the states of the nodes have stabilized, the constructed network will for sure stabilize to a global star after all p-nodes have interacted with each other in order to deactivate any active edges between them and after the c has interacted with all ps in order to activate any inactive edges, i.e., after all pairs of interactions have occurred. This is an edge cover that, as proved in Proposition 7, takes $\Theta(n^2 \log n)$

time. Thus the total expected running time is at most $\Theta(n^2) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$. □

Global Ring

Theorem 8 (Ring Lower Bound) *The expected time to convergence of any protocol that constructs a spanning ring is $\Omega(n^2)$.*

Proof Take any protocol $\mathcal{A}$ that constructs a spanning ring and any execution of $\mathcal{A}$ on n nodes. Consider the step t at which $\mathcal{A}$ performed the last modification of an edge. Observe that the construction after step t must be a spanning ring. We distinguish two cases.

(i) The last modification was an activation. It follows that the previous active network should be a spanning line $u_1, u_2, \ldots, u_n$. But the only activation that can convert this spanning line into a spanning ring is $u_1 u_n$ which occurs with probability $2/[n(n-1)]$, i.e., in an expected number of $\Theta(n^2)$ steps.

(ii) The last modification was a deactivation. It follows that the previous active network should be a spanning ring $u_1, u_2, \ldots, u_n, u_1$ with an additional active edge $u_i u_j$ for $1 \le i < j \le n$ and $j \neq i+1$ (i.e., a chord). Clearly, the only interaction that can convert such an active network into a spanning ring is $u_i u_j$ which takes an expected number of $\Theta(n^2)$ steps to occur. □

Protocol 5 *Global-Ring*

```
$Q = \{q_0, q_1, q_2, l, w, l', l'', q_2', q_2'', \bar{l}\}$
$\delta$:

  // normal behavior begins only after a line has length
  // 2 (edges)
  $(q_0, q_0, 0) \to (q_1, \bar{l}, 1)$
   $(x, q_0, 0) \to (q_2, l, 1)$, for $x \in \{l, \bar{l}\}$
  // merging: random walk of a $w$-leader begins
   $(x, y, 0) \to (q_2, w, 1)$, for $x, y \in \{l, \bar{l}\}$
  $(w, q_2, 1) \to (q_2, w, 1)$
  $(w, q_1, 1) \to (q_2, l, 1)$
  // $l$ connecting to a $q_1$ endpoint, possibly turning its
  // own line to a cycle
   $(l, q_1, 0) \to (l', q_2', 1)$
  // another component detected: a closed cycle must open
   $(x', y, 0) \to (x'', y, 0)$, for $x \in \{l, q_2\}$, $y \in \{l, \bar{l}, w, q_1, q_0\}$
  $(x', y', 0) \to (x'', y'', 0)$, for $x \in \{l, q_2\}$, $y \in \{l, q_2\}$
  // opening closed cycles
  $(l'', q_2', 1) \to (l, q_1, 0)$
  $(l', q_2'', 1) \to (l, q_1, 0)$
 $(l'', q_2'', 1) \to (l, q_1, 0)$
```

Theorem 9 *Protocol* Global-Ring *(see Protocol 5) constructs a spanning ring.* [7]

Proof The protocol is essentially the same as the *Simple-Global-Line* protocol (Protocol 1) but additionally we allow the endpoints of a line to become connected. This occurs whenever one endpoint is in state l and the other is in state q_1 and the two endpoints interact. In this case, rule $(l, q_1, 0) \to (l', q_2', 1)$ applies and the two endpoints become blocked. If any of the two endpoints detects the existence of another component, then, in the next interaction between them, the two endpoints backtrack, by which we mean that they deactivate the connection between them and both become unblocked again by returning to their original states. The existence of another component can be eventually detected due to the fact that every component is either an isolated node in state q_0 or has at least one leader.

Now take an arbitrary reachable configuration C with at least 2 components. We may w.l.o.g. assume that C has no blocked nodes, as if it has there is a sequence of interactions that unblocks them all. Thus, as in the *Simple-Global-Line* protocol we have a collection of lines and isolated nodes. This may very well lead to the formation of a spanning line with a single leader. It is now clear that at some point the leader will occupy one endpoint of the line, will interact with the other endpoint, the spanning line will close to form a spanning ring and the previous endpoints will become blocked. As there is a single component in the network, these two nodes will remain blocked forever and therefore the constructed ring is stable.

Finally, observe that we have not allowed a line to participate to the normal operation of the protocol until its length becomes 2 (edges). In particular, we have not allowed the existence of lines consisting of a single edge with endpoints q_1 and l. The reason is that such lines could connect to each other, forming chains of the form $q_2', l', q_2', l', q_2', l', \ldots$. In such a chain, all q_2's will eventually become q_2'' and all l's will become l''. So, it is possible for an l'' to disconnect from the q_2'' of its original line (as it cannot distinguish between its two q_2'' neighbors) and this may result in isolated l-leaders and blocked lines consisting of a single edge with endpoints l'' and q_1. In such a case, the protocol would not manage to form a spanning ring. Actually, this was the bug of [MS14] that has now been fixed. □

Global Ring: A Generic Approach

[7] We should remark that the corresponding protocol in [MS14] contained a small error (making it fail to construct a ring in a small fraction of its executions) that was detected via experimentation and fixed in this journal version.

We now follow an alternative approach (Protocol 6) for the global ring problem, mainly because it can be generalized to a protocol for the k-regular connected problem. We present the generalization for the latter problem in the sequel (Protocol 7).

Protocol 6 *2RC*

```
$Q = \{q_0, q_1, q_2, l_1, l_2, l_3\}$
$\delta$:

$(q_0, q_0, 0) \to (q_1, l_1, 1)$
$(q_1, q_0, 0) \to (q_2, q_1, 1)$
$(q_1, q_1, 0) \to (q_2, q_2, 1)$
$(l_1, l_1, 0) \to (l_2, q_2, 1)$
$(l_1, q_i, 0) \to (q_2, l_{i+1}, 1)$, for $i \in \{0, 1\}$
// swapping: leaders keep moving inside components
$(l_i, q_j, 1) \to (q_i, l_j, 1)$, for $i, j \in \{1, 2\}$
// leader elimination: eventually a single leader will
// remain in every component
$(l_i, l_j, 1) \to (q_i, l_j, 1)$, for $i, j \in \{1, 2\}$
// opening cycles in the presence of other components
$(l_2, q_0, 0) \to (l_3, q_1, 1)$
$(l_2, l_1, 0) \to (l_3, q_2, 1)$
$(l_2, l_2, 0) \to (l_3, l_3, 1)$
$(l_3, q_1, 1) \to (l_2, q_0, 0)$
$(l_3, q_2, 1) \to (l_2, l_1, 0)$
$(l_3, l_1, 1) \to (l_2, q_0, 0)$
$(l_3, l_2, 1) \to (l_2, l_1, 0)$
$(l_3, l_3, 1) \to (l_2, l_2, 0)$
```

Theorem 10 *Protocol* 2RC *(see Protocol 6) constructs a connected spanning 2-regular network (i.e., a spanning ring).*

Proof Sketch The set $\mathcal{S}$ of output-stable configurations whose active network is a spanning ring consists of those configurations that have one node in state l_2 and all other nodes in state q_2. The index of a state indicates the number of active neighbors of a node. A first goal is for all nodes to have degree 2 which implies a cycle cover, i.e., a partitioning of the nodes into disjoint cycles. The protocol achieves this by allowing every node with degree smaller than 2 to increase its degree. The final goal is to end up with a unique spanning ring. To achieve this, the protocol allows nodes with degree 2 to drop an existing neighbor and pick a new one provided that there are at least 2 components in the network. Clearly, this implies that any closed cycle coexisting with other components, which are cycles, lines, or isolated nodes, may open to form a line. As any collection of lines and isolated nodes can always be merged to a global line and any global line can close to form a global ring, the theorem follows. □

Generalizing to k-Regular Connected

Protocol 7 *kRC*

```
$Q = \{q_0, q_1, \ldots, q_k, l_1, l_2, \ldots, l_{k+1}\}$, i.e., $|Q| = 2(k+1)$
$\delta$:

    $(q_0, q_0, 0) \to (q_1, l_1, 1)$
    $(q_i, q_j, 0) \to (q_{i+1}, q_{j+1}, 1)$, for $1 \le i < k$ and $j < k$
    $(l_i, l_j, 0) \to (l_{i+1}, q_{j+1}, 1)$, for $1 \le i, j < k$
    $(l_i, q_j, 0) \to (q_{i+1}, l_{j+1}, 1)$, for $1 \le i < k$ and $j < k$
// swapping: leaders keep moving inside components
    $(l_i, q_j, 1) \to (q_i, l_j, 1)$, for $1 \le i, j \le k$
// leader elimination: eventually a single leader will
// remain in every component
    $(l_i, l_j, 1) \to (q_i, l_j, 1)$, for $1 \le i, j \le k$
// opening $k$-regular components in the presence of
// other components
    $(l_k, q_0, 0) \to (l_{k+1}, q_1, 1)$
    $(l_k, l_i, 0) \to (l_{k+1}, q_{i+1}, 1)$, for $1 \le i < k$
    $(l_k, l_k, 0) \to (l_{k+1}, l_{k+1}, 1)$
  $(l_{k+1}, q_1, 1) \to (l_k, q_0, 0)$
  $(l_{k+1}, q_i, 1) \to (l_k, l_{i-1}, 0)$, for $2 \le i \le k$
  $(l_{k+1}, l_i, 1) \to (l_k, l_{i-1}, 0)$, for $1 \le i \le k$
$(l_{k+1}, l_{k+1}, 1) \to (l_k, l_k, 0)$
```

Using almost the same ideas as in the proof of Theorem 10, one can prove the following.

Theorem 11 *For every fixed integer $k \ge 2$ and population of size $n \ge k+1$, Protocol* kRC *(see Protocol 7) constructs a connected spanning network in which at least $n-k+1$ nodes have degree k and each of the remaining $l \le k-1$ nodes has degree at least $l-1$ and at most $k-1$.*

It is interesting to point out that the number of states can be substantially reduced in some cases by relying on the computability of the target-degree k. For an example, we show that we can make a node u obtain 2^d neighbors by using only $2(d+2)$ states, for all fixed integers d. Node u is initially in state q_0 and all other nodes are in state a_0. The protocol is $(q_0, a_0, 0) \to (q_0', a_1, 1)$, $(q_0', a_0, 0) \to (q, a_1, 1)$, $(q, a_i, 1) \to (q_{i+1}, a_{i+1}, 1)$, $(q_j, a_0, 0) \to (q, a_j, 1)$ for all $1 \le i \le d-1$, $2 \le j \le d$. Note that u initially collects 2 neighbors (by activating edges) which go to state a_1. Then for every a_1 neighbor that it encounters it makes it an a_2 and collects another neighbor which goes to state a_2. Eventually both a_1 neighbors will become a_2

and there will be another 2 neighbors in state a_2, so in total 4 a_2 neighbors. This process is repeated d times (the 4 a_2s will become 8 a_3s, and so on), each time doubling the number of neighbors, thus eventually u will have obtained 2^d neighbors. The protocol uses only $2(d+1)$ states for the indices of the q_is and the a_is and another 2 states, namely q and q_0'. Clearly, it follows that the target-degree of the nodes is not a lower bound on the size of the protocol.

Many Small Components

We show here how to partition the population into small cliques. This construction is of special value as such a partitioning may serve as a means of maintaining non-interfering clusters. In particular, given such a partitioning, we can easily have a node u perform effective interactions only with nodes belonging to the same component as u. This can be easily determined by the state of the connection between the interacting nodes.

Theorem 12 *For every fixed positive integer c, Protocol c-Cliques constructs $\lfloor n/c \rfloor$ cliques of order c each.*

Proof Sketch The protocol tries to construct $\lfloor n/c \rfloor$ components of order c, each having a unique leader (states l_i, for $i \geq 1$, $\bar{l}_j$, l, and l'_j) directly connected to $c-1$ followers (states f, $i \in \{1,2,\ldots,c-1\}$, and f_j). This is done via $c-2$ successive applications of rule $(l_i, l_0, 0) \to (l_{i+1}, f, 1)$ and then a single application of rule $(l_{c-2}, l_0, 0) \to (\bar{l}_1, 1, 1)$. The role of state $\bar{l}_i$ is to convert its $c-2$ remaining state-f followers to state-1 followers, via $c-3$ successive applications of rule $(\bar{l}_i, f, 1) \to (\bar{l}_{i+1}, 1, 1)$ and then a single application of rule $(\bar{l}_{c-2}, f, 1) \to (l, 1, 1)$. Then each state-$i$ follower, for $1 \leq i < c-1$, tries to become connected to the other $c-1$ followers of the component via rule $(i, j, 0) \to (i+1, j+1, 1)$. As it cannot distinguish the followers of its component from the followers of other components, several of these connections may be wrong.

It suffices to prove that the protocol recognizes wrong connections and deactivates them. Then, as followers always try to make their degree $c-1$ when it is still less than $c-1$ and as wrong connections between different components are always corrected, it follows (by fairness) that eventually each component will become a clique (having only correct connections). At that time, no new connections may be created and no existing connection can be deactivated (as they are all correct), and the correctness of the protocol follows.

To recognize erroneous connections, the leader of a component constantly visits the followers of its component, via rule $(l, i, 1) \to (r, l'_i, 1)$, and checks any active connections that it may encounter during its stay. The duration of its stay is nondeterministic, as it depends on the chosen interactions. In particular, the leader returns to its original position nondeterministically via rule $(l'_i, r, 1) \to (i, l, 1)$, in order to avoid waiting forever in case there are no connections to be fixed. If, instead, during its stay it encounters another leader over an active connection, then this is clearly a connection between different components and the leaders deactivate that connection and decrease the counters of the corresponding followers. This is done via rule $(l'_i, l'_j, 1) \to (l'_{i-1}, l'_{j-1}, 0)$. Clearly, by fairness, every wrong connection will eventually be selected for interaction while having a leader in each of its endpoints. Finally, note that correct connections (between nodes of the same component) are never deactivated as at any time at most one of their endpoints may be occupied by a leader. □

Protocol 8 *c-Cliques*

```
$Q = \{l_0, l_1, \ldots, l_{c-2}, f_1, \ldots, f_{c-2}, f, \bar{l}_0, \ldots, \bar{l}_{c-2}, l,$
    $1, 2, \ldots, c-1, l'_1, \ldots, l'_{c-1}, r\}$, $q_0 = l_0$
$\delta$:

// for $i = 0$, a new component initiated; for $i \geq 1$, a
// leader tries to increase the size of its component to
// $c$ by attracting isolated nodes to its neighborhood
$(l_i, l_0, 0) \to (l_{i+1}, f, 1)$, if $0 \leq i < c-2$
          $\to (\bar{l}_1, 1, 1)$, if $i = c-2$
// nondeterministic elimination of incomplete components
// to avoid deadlock of all components having size $< c$
$(l_i, l_j, 0) \to (l_{i+1}, f_j, 1)$, if $j \leq i < c-2$
          $\to (\bar{l}_0, f_j, 1)$, if $i = c-2$
$(f_i, f, 1) \to (f_{i-1}, l_0, 0)$, if $i > 1$
          $\to (f, l_0, 0)$, if $i = 1$
// the leader of a component with $c$ nodes begins to
// inform its followers to connect to other followers
 $(\bar{l}_i, f, 1) \to (\bar{l}_{i+1}, 1, 1)$, if $i < c-2$
          $\to (l, 1, 1)$, if $i = c-2$
// followers keep track of their number of connections
 $(i, j, 0) \to (i+1, j+1, 1)$, if $i < c-1$ and $j < c-1$
// a leader temporarily takes the place of a follower
// in order to check for wrong connections
 $(l, i, 1) \to (r, l'_i, 1)$
// two leaders deactivating a wrong connection joining
// distinct components
$(l'_i, l'_j, 1) \to (l'_{i-1}, l'_{j-1}, 0)$
// the leader returns to its original position nonde-
// terministically, after performing 0 or more checks
 $(l'_i, r, 1) \to (i, l, 1)$
```

Replication

We now study the related problem of replicating a given input graph $G_1 = (V_1, E_1)$. Let $V_2 = V_I \backslash V_1$ be the set of the remaining nodes. A protocol must construct on V_2 a replica G_2 of G_1, thus it must hold that $|V_2| \geq |V_1|$. In what follows, we assume that nodes in V_1 are in different initial states than nodes in V_2. In particular, we use q_0 and r_0 as the initial states of nodes in V_1 and V_2, respectively. Additionally, E_1 is defined by the active edges between nodes in V_1. We assume that G_1 is connected.

We present a very simple protocol (Protocol 9) which, by exploiting the election of a unique leader, successfully copies G_1 on any V_2 satisfying $|V_2| \geq |V_1|$. The protocol never introduces waste in V_2. Actually, it always modifies the state of precisely $|V_1|$ nodes from V_2 always leaving the remaining $|V_2| - |V_1|$ nodes of V_2 to their initial states. Note that, unlike all other protocols in this section, this one is a randomized protocol.

Initially, all nodes of V_1 are in q_0 and all nodes of V_2 are in r_0. The protocol matches every node of V_1 to a distinct node of V_2 (that is, creates a maximum matching between the two sets) and in parallel it starts pairwise eliminations between leaders, that is, when two leaders (nodes in state l) interact one of them survives (i.e., remains l) and the other becomes a follower (state f). Eventually the protocol ends up with a unique leader and $|V_1| - 1$ followers. Moreover, when a leader and a follower meet they swap their states with probability 1/2. With the remaining 1/2 probability they become either l_a, f_a or l_d, f_d depending on whether the edge joining them was active or inactive, respectively. In both cases they mark their matched nodes from V_2 to either activate or deactivate the edge between them in V_2 accordingly. Once there is a unique leader, the leader moves nondeterministically over the nodes of V_1 and again nondeterministically applies this copying process on the edges of E_1. Thus it will eventually apply this copying process to all edges of E_1 and as there are no conflicts with other activations/deactivations (as no other leaders exist) G_2 eventually becomes isomorphic to G_1. Finally, note that the active edges of the matching between V_1 and V_2 are never deactivated but this is not a problem, provided that $Q_{out} = \{r, r_a, r_d\}$, as every such edge uv has an endpoint $u \in V_1$ in a state from $Q \backslash Q_{out}$ and is not considered as part of the output.

Theorem 13 *Protocol* Graph-Replication *constructs a copy of any connected input graph $G_1 = (V_1, E_1)$ with no waste. It uses 12 states and its expected running time under the uniform random scheduler is $\Theta(n^4 \log n)$.*

Protocol 9 *Graph-Replication*

```
$Q = \{q_0, r_0, l, l_a, l_d, f, f_a, f_d, r, r_a, r_d, r'\}$
$\delta$:

// matching every $u \in V_1$ to a distinct $v \in V_2$
$(q_0, r_0, 0) \to (l, r, 1)$
// leader election in $V_1$
  $(l, l, x) \to (l, f, x)$
// a non-edge (inactive) of $G_1$ detected: with prob.
// 1/2 copying to $G_2$ initiated and with prob. 1/2
// the leader $l$ continues its random walk in $V_1$
  $(l, f, 0) \overset{1/2}{\to} (l_d, f_d, 0)$
            $\overset{1/2}{\to} (f, l, 0)$
// an edge (active) of $G_1$ detected: with prob. 1/2
// copying to $G_2$ initiated and with prob. 1/2 the
// leader $l$ continues its random walk in $V_1$
  $(l, f, 1) \overset{1/2}{\to} (l_a, f_a, 1)$
            $\overset{1/2}{\to} (f, l, 1)$
// informing the matched nodes from $V_2$ to apply copying
$(x_i, r, 1) \to (x_i, r_i, 1)$, for $x \in \{l, f\}$ and $i \in \{a, d\}$
// an activation copying applied in $G_2$
$(r_a, r_a, \cdot) \to (r', r', 1)$
// a deactivation copying applied in $G_2$
$(r_d, r_d, \cdot) \to (r', r', 0)$
// informing the matched nodes from $V_1$ that the
// requested copying has been performed; as long as
// there are more than one leaders, copying may have
// been performed on a wrong pair of nodes of $V_2$
$(r', x_i, 1) \to (r, x, 1)$, for $x \in \{l, f\}$ and $i \in \{a, d\}$
// leader election applies also to $l_a$s and $l_d$s in
// order to prevent blocking
  $(l_i, l, x) \to (l_i, f, x)$, for $i \in \{a, d\}$
$(l_i, l_j, x) \to (l_i, f_j, x)$, for $i, j \in \{a, d\}$
```

Proof First observe that the maximum matching between V_1 and V_2 is eventually constructed. The reason is that any node can only be matched once, because when a q_0 is matched to an r_0 both change states to l and r, respectively (so they cannot be matched any more). Moreover, as long as a q_0 or an r_0 has not been matched, it does not change state so it remains forever a candidate for matching. Then $|V_2| \geq |V_1|$ and fairness imply that eventually the matching becomes maximum, i.e., each $u \in V_1$ is matched to a distinct $v \in V_2$. Note also that there are always $|V_2| - |V_1| \geq 0$ nodes of V_2 that will never participate in the protocol, because all $v \in V_2$ are initially in r_0, an r_0 can only participate if it encounters a q_0, but any such encounter decreases the number of q_0s by one (and no new q_0s are never created). Clearly, after the first $|V_1|$ such encounters there are no q_0s left, therefore $|V_2| - |V_1|$ nodes of V_2 cannot participate in the protocol any more. So, we can w.l.o.g. restrict our analysis to the special case of populations in which $|V_1| = |V_2|$. For this protocol, correct copying for $|V_2| = |V_1|$ implies correct copying for all $|V_2| \geq |V_1|$ and also implies that the waste (from V_2) is indeed zero, as every graph can be copied in the absence of auxiliary nodes. In what follows, we assume that $|V_1| = |V_2|$. Note that, in this case, the constructed matching between V_1 and V_2 is actually a perfect matching.

Assume now that there is a unique leader in V_1 in state l, all other nodes in V_1 are in state f, all nodes in V_2 are in state r, and there is an arbitrary active graph on V_2. We prove that the graph of V_2 eventually becomes isomorphic to G_1. Take any edge $u'v'$, where $u', v' \in V_2$ and let u, v be their corresponding matched nodes from V_1. The unique leader l performs a random walk on the nodes of V_1 and fairness guarantees that the following must eventually occur: l reaches one of u, v, say u, its next interaction is with v (which is in state f), and it is a non-swapping interaction. The result of the interaction is then that u goes to l_i and v to f_i where $i \in \{a, d\}$ represents the state of uv. From that point on the following "deterministic" operations occur: u' and v' will eventually interact with their matched nodes and will both go to state r_i, and then they will eventually interact with each other and will activate or deactivate $u'v'$ depending on i. In both cases, $u'v'$ copies the state of uv. This proves that any $u'v'$ will eventually copy the value of its corresponding edge uv. The claim follows by observing that, given that V_1 has a unique leader, once a $u'v'$ has the same state as uv it cannot change state any more.

Next observe that indeed eventually a unique leader leader remains in V_1. After it has been matched (which eventually occurs), a node of V_1 can only be in one of the states l, l_i, f, f_i. As long as there are at least two leaders, there is always an interaction that eliminates one of them. So, it remains to show that the system will eventually reach a configuration, as described above, in which all other nodes in V_1 are in state f and all nodes in V_2 are in state r. Clearly, any remaining f_i has a corresponding r_i (if, instead, it has an r' then there is an eventual interaction between them that will convert them to f and r, respectively, so we need not consider this case). If the f_i, r_i pairs are even, then each r_i will eventually meet another r_i, which will make them both r', and their corresponding f_is will become converted to fs (this holds regardless of the additional r_is introduced by the unique leader, since they always come in pairs). So, the only case remaining to consider is the one in which there is an odd number of r_is. In this case, however, there must also be an odd number of f_is that have not yet informed their matched nodes, due to the following invariant: the number of r_is plus the number of rs with an x_i matched node is always even. So, again, eventually every r_i will have another r_i to interact with.

Now, for the running time we consider three phases: the *matching formation*, the *leader election*, and the *unique-leader replication*.

The matching formation phase begins from step 1 and ends when the last q_0 becomes l, i.e., when all nodes in V_1 have been matched to the nodes of V_2. It is not hard to see that the probability of the ith edge of the matching to be established (given $(i-1)$ established matches) is $p_i = [2(n/2 - i)^2]/[n(n-1)]$ and the corresponding expectation is $\mathrm{E}[X_i] = 1/p_i = \Theta(n^2/(n-i)^2)$. Then similarly to the coupon collector's application in the running time of Protocol *Fast-Global-Line* in Theorem 4 we have that the expected running time of this phase is $\mathrm{E}[X] = \Theta(n^2)$.

An almost identical analysis yields that the expected running time of the leader election phase is also $\Theta(n^2)$.

Thus, it remains to estimate the time it takes for the unique leader to copy every edge of E_1. Given that the leader has marked the endpoints of a particular edge of E_1 then copying and restoring the state of the leader takes on average $\Theta(n^2)$ time (as a constant number of particular interactions must occur and each one occurs with probability $1/n^2$). Now we consider the time for copying as constant and try to estimate the time it takes for the leader to "collect" (i.e., visit and mark) all edges of E_1. Assume also that the leader is selected in every step to interact with one of its neighbors (the truth is that it is selected every $\Theta(n)$ steps on average). If p_e is the probability that a specific edge e is selected after two subsequent interactions then $p_e \simeq (1/n)(1/2)(1/n)(1/2) = \Theta(1/n^2)$, where $(1/n)(1/2)$ is the probability that the leader interacts with and decides to move on one endpoint of e and $(1/n)(1/2)$ the probability that it then interacts with and decides to mark the other endpoint of e. Let the r.v. Y_i be the number of steps between the $(i-1)$th and ith edge collected and p_i be the probability of a success in two consecutive steps of the ith epoch. Clearly, $p_i \simeq (n^2 - i)/n^2$, $\mathrm{E}[Y_i] = 1/p_i = n^2/(n^2 - i)$, and $\mathrm{E}[Y] = \mathrm{E}[\sum_{i=0}^{n^2-1} Y_i] = n^2 \sum_{i=0}^{n^2-1} 1/(n^2 - i) = n^2 \sum_{i=1}^{n^2} 1/i = \Theta(n^2 \log n)$. Thus, provided that the leader always interacts and that every copying that it performs takes constant time, the expected time until the unique-leader replication phase ends is $\Theta(n^2 \log n)$. Now, notice that on average it takes $\Theta(n)$ steps for the leader to interact and that in half of its interactions the leader performs a copying that takes $\Theta(n^2)$ steps to complete. That is, each of the above $\Theta(n^2 \log n)$ steps is charged on average by n and half of them are charged by n^2, i.e., half of the steps are charged by $\Theta(n)$ and the other half are charged by $n^2+n=\Theta(n^2)$. We conclude that the expected running time of the unique-leader replication phase is $\Theta(n^3 \log n)+\Theta(n^4 \log n)=\Theta(n^4 \log n)$. This is clearly the dominating factor of the total running time of the protocol. □

Table 2 summarizes all upper and lower bounds that we established in Sections 4 and 5.

6 Generic Constructors

In this section, we ask whether there is a generic constructor capable of constructing a large class of networks. We answer this in the affirmative by presenting (i) constructors that simulate a Turing Machine (TM) and (ii) a constructor that simulates a distributed system with names and logarithmic local memories. Let us denote by l the binary length of the input of a TM and by n the size of a population. All of our protocols construct a random graph G on $\Theta(n)$ nodes and use the remaining nodes (and in one case also the edges between them) to simulate a TM on input G. Thus, due to the fact that G is provided to the TM in adjacency matrix encoding, in what follows it always holds that the input of the TM has size $\Theta(n^2)$, i.e., it happens that $l=\Theta(n^2)$. This allows us to use in all of our theorems $\Theta(n^2)$ in place of l and avoid any confusion that could result by presenting them in terms of two parameters, l and n. Moreover, it is also useful to keep in mind that the TM can use space at most $O(n^2)$, as this is the total distributed memory available (including nodes and edges).

We now briefly describe the main idea behind all of our generic constructors that simulate a TM (see also Figure 3). Assume that we are given a decidable graph-language L and we are asked to provide a NET that constructs L. The NET that we give works as follows:

1. It constructs on k of the nodes a network G_1 capable of simulating a TM and of constructing a random network on the remaining $n-k$ nodes. Let $V_1 \subseteq V$ be the set of the k nodes and $V_2 = V \backslash V_2$ the set of the remaining $n-k$ nodes. G_1 is usually a sufficiently long line or a bounded degree network as these networks can be operated as TMs. A line also serves as a measure of order as we can match a line of length k with k other nodes and by exploiting the ordering of the line we may achieve an ordering of the other nodes.
2. The NET exploits G_1 to construct a random network on V_2. The idea is to exploit the structure of G_1 so that it can perform a random coin tossing on each edge between nodes of V_2 exactly once. In this manner, it constructs a random network G_2 from $G_{n-k,1/2}$ on the nodes of V_2 (if required, recall the definition of the $G_{n,p}$ random graph model from a footnote of Section 1). It is worth noting that all networks of $G_{n-k,1/2}$ have an equal probability to occur and this results in an equiprobable constructor (the only exception to this is the constructor of Theorem 17, which doesn't produce all networks with the same probability).
3. The NET simulates on G_1 the TM that decides L with G_2 as its input. The only constraint is that the space used by the TM should be at most the space that the constructor can allocate in G_1. If the TM rejects, then the protocol goes back to 2, that is, it draws another random network and starts a new simulation. Otherwise, its output stabilizes to G_2.

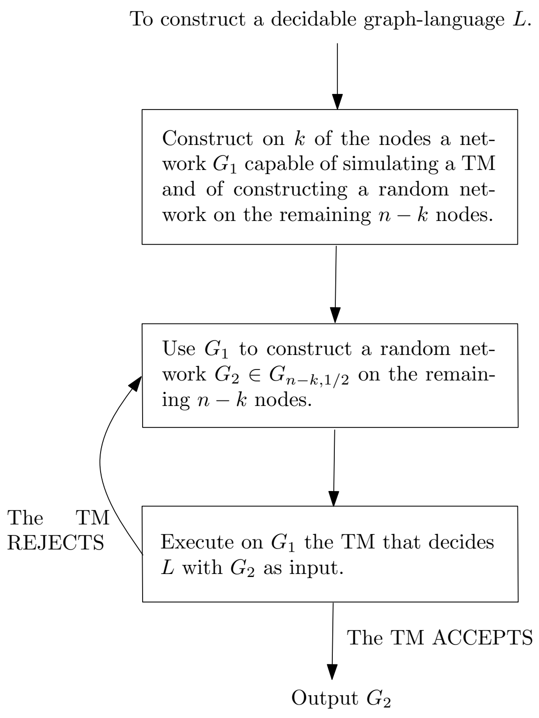
To construct a decidable graph-language L.
Construct on k of the nodes a network G_1 capable of simulating a TM and of constructing a random network on the remaining $n-k$ nodes.
Use G_1 to construct a random network $G_2 \in G_{n-k,1/2}$ on the remaining $n-k$ nodes.
Execute on G_1 the TM that decides L with G_2 as input.
The TM REJECTS
The TM ACCEPTS
Output G_2

Fig. 3 The main mechanism used by all generic constructors in this section. The loop repeats until the TM accepts for the first time. When this occurs, the random graph G_2 constructed belongs to L and thus the protocol may output G_2. Note that this is not a terminating step. The protocol just does not repeat the loop and thus its output forever remains G_2.

6.1 Linear Waste

Theorem 14 (Linear Waste-Half) $\mathbf{DGS}(O(n)) \subseteq \mathbf{PREL}(\lfloor n/2 \rfloor)$. *In words, for every graph language L*

Protocol	# states	Expected Time	Lower Bound
Simple-Global-Line	5	$\Omega(n^4)$ and $O(n^5)$	$\Omega(n^2)$
Fast-Global-Line	9	$O(n^3)$	$\Omega(n^2)$
Cycle-Cover	3	$\Theta(n^2)$ (optimal)	$\Omega(n^2)$
Global-Star	2 (optimal)	$\Theta(n^2 \log n)$ (optimal)	$\Omega(n^2 \log n)$
Global-Ring	9		$\Omega(n^2)$
2RC	6		$\Omega(n \log n)$
kRC	$2(k+1)$		$\Omega(n \log n)$
c-Cliques	$5c-3$		$\Omega(n \log n)$
Graph-Replication	12	$\Theta(n^4 \log n)$	

Table 2 All upper and lower bounds established in Sections 4 and 5. *Graph-Replication* is a randomized protocol thus it concerns class **PREL**, while all other protocols do not rely on randomization thus they concern **REL**.

that is decidable by an $O(n)$-space TM, there is a protocol that constructs L equiprobably with useful space $\lfloor n/2 \rfloor$.

Proof We give a high-level description of the protocol, call it $\mathcal{A}$. Let us begin by briefly presenting the main idea. Given a population of size n, $\mathcal{A}$ partitions the population (apart from one node when n is odd) into two equal sets U and D such that all nodes in U are in state q_u, all nodes in D are in state q_d and each $u \in U$ is matched via an active edge to a $v \in D$, i.e., there is a perfect matching between U and D (see Figure 4). By using the *Simple-Global-Line* protocol (see Protocol 1 in Section 4.1) on the nodes of set U, $\mathcal{A}$ constructs a spanning line in U which has the endpoints in state q_1, the internal nodes in state q_2, and has additionally a unique leader on some node. We should mention that, though we use protocol Simple-Global-Line here as our reference, any protocol that constructs a spanning line would work. Given such a construction, $\mathcal{A}$ organizes the line into a TM. The goal is for the TM to compute a graph from L and construct it on the nodes of set D. To achieve this, the TM implements a binary counter ($\log n$ bits long) in its memory and uses it in order to uniquely identify the nodes of set D according to their distance from one endpoint, say the left one. Whenever it wants to modify the state of edge (i, j) of the network to be constructed, it marks by a special activating or deactivating state the D-nodes at distances i and j from the left endpoint, respectively. Then an interaction between two such marked D-nodes activates or deactivates, respectively, the edge between them. To compute a graph from L equiprobably, the TM performs the following random experiment. It activates or deactivates each edge of D equiprobably (i.e., each edge becomes active/inactive with probability 1/2) and independently of the other edges. In this manner, it constructs a random graph G in D and all possible graphs have the same probability to occur. Then it simulates on input G the TM that decides L in $\Theta(n)$ space to determine whether $G \in L$. Notice that the $n/2$ space of the simulator is sufficient to decide on an input graph encoded by an adjacency matrix of $(n/2)^2$ binary cells (which are the edges of U). If the TM rejects, then $G \notin L$ and the protocol repeats the random experiment to produce a new random graph G' and starts another simulation on input G' this time. When the TM accepts for the first time, the constructed random network belongs to L and the protocol releases the constructed network by deactivating one after the other the active (q_u, q_d) edges and at the same time updates the state of each D-node to a special q_{out} state. Finally, we should point out that, whenever the global line protocol makes progress, all edges in D are deactivated and the TM-configuration is *reinitialized* to ensure that, when the final progress is made (resulting in the final line spanning U) the TM will be executed from the beginning on a correct configuration (free of residues from previous partial simulations).

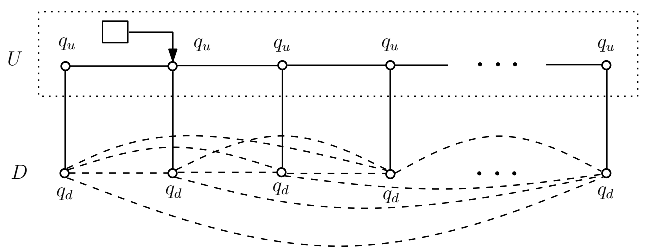

Fig. 4 The population partitioned into sets U and D. The vertical active edges (solid) match the nodes of the two sets. The horizontal active edges between nodes in U form a spanning line that is used to simulate a TM. The TM will construct the desired network on the nodes of set D by activating the appropriate edges between them (dashed edges that are initially inactive).

We now proceed with a more detailed presentation of the various subroutines of the protocol.

Simulating the direction of the TM's head. We begin by assuming that the spanning line has been constructed somehow (we defer for the end of the proof the actual mechanism of this construction), as in Figure 4, and that each node has three components (c_1, c_2, c_3) in its state. c_1 is used to store the head of the TM, i.e., the actual state of the control of the TM; assume that initially the head lies on an arbitrary node, e.g., on the second one from the left as in Figure 4. c_2 is used to store the symbol written on each cell of the TM. c_3 is l, r, t for "left", "right", and "temporary" respectively, or $\sqcup$ (for "empty") and we assume that initially the left endpoint is l, the right endpoint is r, and all internal nodes are $\sqcup$. As initially the head cannot have any sense of direction, it moves towards an arbitrary neighbor, say w.l.o.g. the right one, and leaves a t on its previous position. The t mark gives to the head a sense of direction on the line. Now the head can continue its progress towards the right endpoint by just moving only towards the unmarked neighbor (avoiding the one marked by t). Once the head reaches the right endpoint for the first time, it starts moving towards the left endpoint by leaving r marks on the way. Once it reaches the left endpoint it is ready to begin working as a TM. Now every time it wants to move to the right it moves onto the neighbor that is marked by r while leaving an l mark on its previous position. Similarly, to move to the left, it moves onto the l neighbor and leaves an r mark on its previous position. In this way, no matter what the position of the head will be, there will be always l marks to its left and r marks to its right, as in Figure 5, and the head can exploit them to move correctly. Additionally, we ensure that the endpoints are in special states, e.g., l_e and r_e, to ensure that the head recognizes them in order to start moving in the opposite direction.

Reading and Writing on the edges of set D. We now present the mechanism via which the TM reads or writes the state of an edge joining two D-nodes. The TM uniquely identifies a D-node by its distance from the left endpoint. To do this, it implements a binary counter on $\log n$ cells of its memory. Whenever it wants to read (write, resp.) the state of the edge joining the D-nodes i and j, it sets the counter to i, places a special mark on the left endpoint, and repeatedly moves the mark one position to the right while decrementing the counter by one. When the counter becomes 0, it knows that the mark is over the i-th U-node. Now by exploiting the corresponding active vertical edge it may assign a special mark to the i-th D-node (Figure 6 provides an illustration). By setting the counter to j and repeating the same process, another special mark may be assigned to the j-th D-node. Now the TM waits for an interaction to occur between the marked D-nodes i

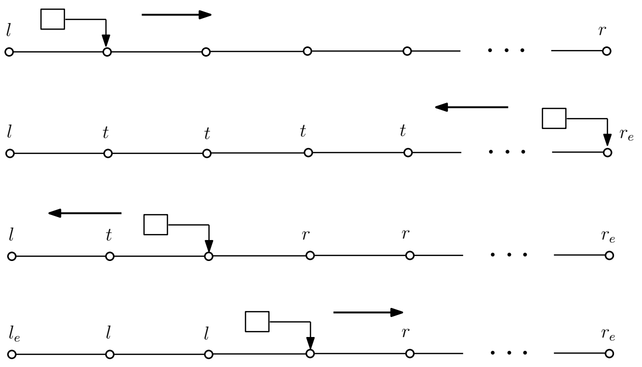

Fig. 5 The main idea of using l and r marks to simulate the movement of the head of a TM. The first three snapshots present the phase of the initialization of the marks where a temporary t mark is used to move for the first time towards an endpoint. In the fourth snapshot, after the head has visited both endpoints, the t marks have been removed and all nodes to the left of the head are marked l while all nodes to the right are marked r. Additionally, the endpoints have special marks to ensure that the head recognizes them.

and j. During that interaction edge (i, j) is read (written, resp.) by the corresponding endpoints. Then, in case of a read (and similarly for a write), the TM reads the value of the edge that the endpoints detected, and in both cases unmarks both endpoints resetting them to their original states.

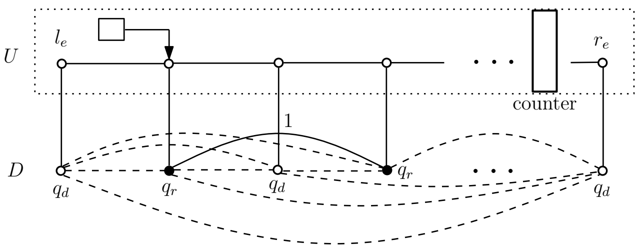

Fig. 6 By exploiting the implemented binary counter, the TM has managed to mark the desired nodes from set D, in this case the 2nd and the 4th ones counting from left, which are now in a special "reading" state q_r. An interaction between them will read the state of the edge joining them, which here happens to be an active one. Then the TM will read that value from one of these two nodes, in this case from the 2nd one. A write is implemented similarly.

Creating the input of the TM. We now describe how the network construction works. As already stated, to simplify the description and in order to present an equiprobable constructor we have allowed nodes to toss a fair coin during their interaction. In particular, we allow transitions that with probability 1/2 give one outcome and with probability 1/2 another. Now before executing the simulation, the simulating protocol does the following. It visits one after the other the edges of set D

and on each one of them performs the following random experiment: with probability $1/2$ it activates the edge and with probability $1/2$ it deactivates it. The result of this random process is an equiprobable construction of a random graph. In particular, all possible graphs have the same probability to occur. Note that the protocol can detect when all random experiments have been performed because it can detect the endpoints of the spanning line. For example, to visit all edges one after the other we may: (i) place two marks on the left endpoint; let i and j, $1 \leq i < j \leq n$, denote the positions of these marks on the line, (ii) for all $1 \leq i \leq n-1$, perform random experiments on all $i < j \leq n$ by starting the rightmost mark from position $i+1$ and moving it each time one position to the right, (iii) the process stops when i becomes n, i.e., when the leftmost mark occupies the right endpoint (which can be detected). Thus we can safely compose the process that draws the random graph to the process that simulates the TM. Once the random graph has been drawn, the protocol starts the simulation of the TM. Notice that the input to the TM is the random graph that has been drawn on the edges of D which provide an encoding equivalent to an adjacency matrix. There are $(n/2)^2$ edges and the simulator has available space $n/2$, which is sufficient for the simulation of a $\Theta(n)$-space TM. We now distinguish two cases, one for each possible outcome of the simulation.

1. The TM rejects: In this case, the constructed random graph does not belong to L. The protocol repeats the random experiment, i.e., draws another random graph, and starts over the simulation on the new input.
2. The TM accepts: The constructed graph belongs to L and the protocol enters the Releasing phase (see below).

Releasing. When the TM accepts for the first time, the simulating protocol updates the head to a special finalizing state f. Now the head moves to the left endpoint and starts releasing one after the other the nodes of set D by deactivating the vertical edges and updating the states of the released D-nodes to q_{out}. Now the network constructed over the nodes of set D is free to move in the "solution".

It remains to resolve the following issue. In the beginning, we made the assumptions that the population has been partitioned into sets U and D and that a spanning line in U has been constructed somehow. Though it is clear that the rule $(q_0, q_0, 0) \rightarrow (q_u, q_d, 1)$ can achieve the partitioning and that the *Simple-Global-Line* protocol can construct a spanning line in U, it is not yet clear whether these processes can be safely composed to the simulating process. To get a feeling of the subtlety, consider the following situation. It may happen that a small subset S of the nodes has been partitioned into sets U' and D' and that U' has been organized into a line spanning its nodes. If the nodes in S do not communicate for a while to the rest of the network, then it is possible that a graph is constructed in D', which on one hand belongs to L but on the other hand its order is much smaller than the desired $n/2$. To resolve this we introduce a reinitialization phase.

Reinitialization. A reinitialization phase is executed whenever a line on U-nodes expands (either by attracting free nodes or by merging with another line). At that point, the protocol "makes the assumption" that no further expansions will occur, restores the components of the simulation to their original values, ensures that each node in the updated set U has a D-neighbor (as it is possible that some of them have released their neighbors), and initiates the drawing of a new random graph on the new set D. Though the assumption of the protocol may be wrong as long as further expansions of the line may occur, at some point the last expansion will occur and the assumption of the protocol will be correct. From that point on, the simulation will be reinitialized and executed for the last time on the correct sets U and D. A final point that we should make clear is the following. During reinitialization we have two options: (i) block the line from further expansions until all components have been restored correctly and then unblock it again or (ii) leave it unblocked from the beginning. In the latter case, if another expansion occurs before completion of the previous reinitialization then another reinitialization will be triggered. However, if the two reinitialization processes ever meet then we can always kill one of them and restart a new single reinitialization process. Both options are correct and equivalent for our purposes. □

We now show an interesting trade-off between the space of the simulated TM and the order of the constructed network. In particular, we prove that if the constructed network is required to occupy $1/3$ instead of half of the nodes, then the available space of the TM-constructor dramatically increases to $O(n^2)$ from $O(n)$.

Theorem 15 (Linear Waste-Two Thirds) DGS($O(n^2) + O(n)$**)** $\subseteq$ **PREL(**$\lfloor n/3 \rfloor$**)**. *In words, for every graph language L that is decidable by a $(O(n^2)+O(n))$-space TM, there is a protocol that constructs L equiprobably with useful space $\lfloor n/3 \rfloor$.*

Proof The idea is to partition the population into three equal sets U, D, and M instead of the two sets of Theorem 14. The purpose of sets U and D is more or less as in Theorem 14. The purpose of the additional set M is to constitute a $\Theta(n^2)$ memory for the TM to be simulated. The goal is to exploit the $(n/3)(n/3-1)/2$ edges of set M as the binary cells of the simulated TM (see Figure 7). The set U now, instead of executing the simulation on its own nodes, uses for that purpose the edges of set M. Reading and writing on the edges of set M is performed in precisely the same way as reading/writing the edges of set D (described in Theorem 14).

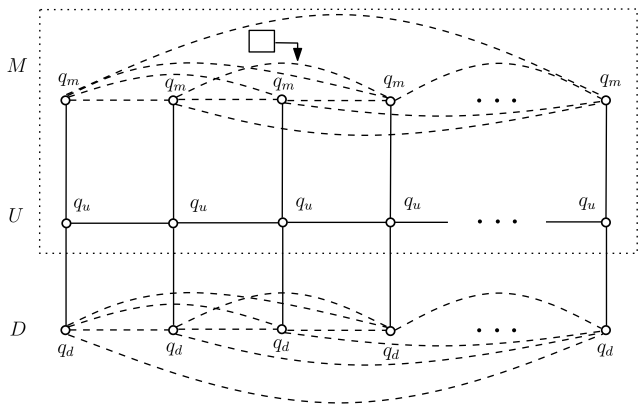

Fig. 7 A partitioning into three equal sets U, D, and M. The line of set U plays the role of an ordering that will be exploited both by the random graph drawing process and by the TM-simulation. The line of set U instead of using its $\Theta(n)$ memory as the memory of the TM it now uses the $\Theta(n^2)$ memory of set M for this purpose. Set D is again the useful space on which the output-network will be constructed. Sets U and M constitute the waste.

As everything works in precisely the same way as in Theorem 14, we only present the subroutine that constructs the (U, D, M) partitioning.

Constructing the (U, D, M) partitioning. The rules that guarantee the desired partitioning into the three sets are:

$$(q_0, q_0, 0) \to (q_u', q_d, 1)$$
$$(q_u', q_0, 0) \to (q_u, q_m, 1)$$
$$(q_u', q_u', 0) \to (q_u, q_m', 1)$$
$$(q_m', q_d, 1) \to (q_m, q_0, 0)$$

The idea is to consider a U-node as unsatisfied as long as it has not managed to obtain a q_m neighbor. The unsatisfied state of a U-node is q_u'. If a q_u' meets a q_0 then it makes that q_0 its q_m neighbor and becomes satisfied. Note that it is possible that at some point the population may only consist of q_u' nodes matched to D-nodes which is not a desired outcome. For this reason, we have allowed q_u' nodes to be capable of making other q_u' nodes their q_m neighbors. That is, when two q_u' nodes interact, one of them becomes satisfied, the other becomes q_m', and the edge joining them becomes active. A q_m' just waits to meet its active connection to a D-node, deactivates it, isolates the D-node by making it q_0 again, and becomes q_m. For an illustration, see Figure 8. Then, for the construction of the line spanning U, we only allow satisfied U-nodes to participate to the construction. As a satisfied U-node never becomes unsatisfied again, this choice is safe. □

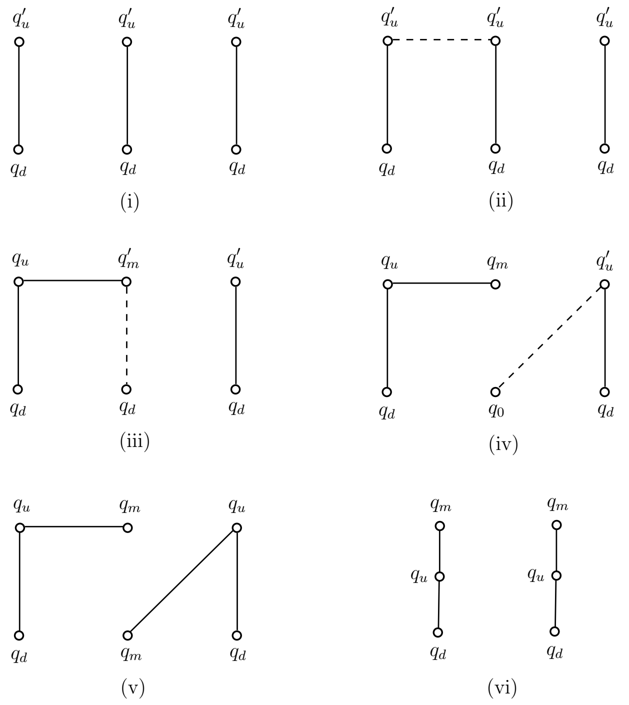

Fig. 8 An example construction of a (U, D, M) partitioning.

6.2 Logarithmic Waste

We now relax our requirement for simulation space in order to reduce the waste (which, in both of the previous two theorems, was of the order of n).

Theorem 16 (Logarithmic Waste) DGS$(O(\log n))$ $\subseteq$ **PREL**$(n - \log n)$. *In words, for every graph language L that is decidable in logarithmic space, there is a protocol that constructs L equiprobably with useful space $n - \log n$.*

Proof Sketch The protocol first constructs a spanning line. Let us for now assume that the spanning line has been somehow constructed by the protocol. Then the protocol exploits the line to count the number of nodes in the network. We may assume that counting is performed in the rightmost cells of the line. The head visits one after the other the nodes from left to right and for each next move it increments the binary counter by one. When the head reaches the right endpoint, counting stops and the binary counter will have occupied approximately $\log n$ nodes (in fact, the rightmost $\log n$ nodes). Now the protocol releases the counter without altering its line structure and additionally makes all remaining $n - \log n$ nodes isolated by resetting their states and deactivating the edges between them.

From now on, we may assume w.l.o.g. that there is a line of $\log n$ nodes with a unique leader and with a distributed variable containing a very good estimate of the number of isolated nodes (for this, we just compute in the logarithmic memory $n - \log n$, where n was already stored in binary and $\log n$ is the number of cells of the memory; another way to achieve this is to stop counting when the head - moving from left to right - reaches the first, i.e., leftmost, cell occupied by the counter). All nodes of the memory are in a special m state while all remaining nodes are in some other state, e.g., f, so the two sets are distinguishable.

Next the leader starts a random experiment in order to construct a random graph on the free nodes as follows. It picks the first free node that it sees, call it u_1, activates the edge between them and informs it to start tossing coins on each one of the edges joining it to other free nodes. Whenever u_1 tosses a coin on a new edge, it marks the corresponding node to avoid it in the future and informs the leader to decrement its $(n - \log n)$-counter by 1. When the counter becomes 0, u_1 has tossed coins on all its edges, by a similar counting process it removes all marks from the other free nodes, and remains marked so that the leader avoids picking it again in the future. Then the leader moves to some other free node u_2, repeating more or less the same process. At the same time the leader decrements another $(n - \log n)$-counter by one to know when all free u_is have been picked. In this manner, a random graph is drawn equiprobably on the set of free nodes. Next, the leader simulates a logarithmic TM in its memory trying to decide whether the random graph belongs to a given language L or not. If so, then we are done. If not, then the TM just repeats the random experiment and restarts the simulation.

Reinitialization. Clearly, the protocol cannot know when the line that it was initially trying to construct has become spanning. Due to this, after every expansion of the line it assumes that the line has become spanning and starts counting. It is clear that every counting process leads to the formation of a small line with a leader (of length logarithmic in the length of the original line) and several free nodes. The small line and its leader are kept forever by the simulation process. This implies that if there is more than one such line, they will eventually interact and detect that their original line was not spanning. At that point, the interacting lines may merge to form a new line. It is clear that the only stable case is the one in which the original line was spanning and this will eventually occur. □

6.3 No Waste

Going one step further, we prove that if we sacrifice the requirement of constructing all graphs in the language equiprobably, then a large class of graph-families can be constructed with no waste.

Theorem 17 (No Waste) *Let L be a graph language such that: (i) there exists a natural number d s.t. for all $G \in L$ there is a subgraph G' of G, of order logarithmic in the order of G, s.t. either G' or its complement is connected and has degree upper bounded by d and (ii) $L \in \mathbf{DGS}(O(\log n))$, i.e., L is decidable in logarithmic space. Then there is a randomized protocol that constructs L with useful space n.*

Proof Sketch As in Theorem 16, the protocol first constructs a spanning line used to separate a subpopulation S of V_I of size approximately $\log n$. Before deactivating the line of $T = V_I \backslash S$ of length $n - \log n$ the protocol first exploits it to construct a random graph in S of active or inactive degree (choosing randomly between these) upper bounded by d (note that d is finite and thus it is known in advance by the protocol). Then the line of T organizes the bounded-degree graph of S into a TM M (which is feasible due to the fact that the degree is bounded; see Theorem 7 of [AAC$^+$05]) of logarithmic space with a unique leader on some node. Next M draws (more or less as in Theorem 16) a random graph on the edges of $E_I \backslash E[S]$, i.e., on all edges apart from those between the nodes of S (to prevent destroying the structure of the TM). Note that, in order for the TM to be able to distinguish the nodes of S, the protocol has all these nodes in a special state that is not present in T. Observe now that, in this manner, the protocol has constructed on V_I a random graph from those having a connected subgraph of logarithmic order and degree upper bounded by d. It remains to verify whether the one constructed indeed belongs to L. To do this, M simulates the TM N that decides L in logarithmic space. If N accepts, then we are done (given that the final reinitialization has occurred, as in the previous theorems). If N rejects, then M builds another line in T that repeats the whole process, i.e., draws a new random graph in S and so on.

Observe that this construction has an important difference from the previous ones. The TM does not work on a separate part of the population, which will be then thrown away as waste. It works on a part of the input graph that it tries to decide. Still the graph can be processed more or less as in Theorem 16. The only difference is that now the TM also takes into account the edges that involve at least one node in S. This can be easily achieved by using separate components, in the states of the nodes of S, for the simulation and the reading of the input (while, on the other hand, nodes outside S need only have a reading component). □

The above protocol constructs every $G \in L$ with non-zero probability but not all graphs in L have the same probability to be constructed. For example, if a graph G_1 has more distinct subgraphs satisfying condition (i) of the theorem than a graph G_2, then the random bounded-degree graph constructed by the protocol is more often a subgraph of G_1 than it is of G_2. Therefore we cannot claim that $L \in \mathbf{PREL}(n)$ (the latter was erroneously reported in [MS14]). We leave this as an interesting open problem.

Remark 1 If the graph-property L (in any of the above results) happens to occur with probability at least $1/f(n)$, where $f(n)$ is polynomial on n, in the $G_{n,1/2}$ random graph model, then its corresponding generic constructor runs in polynomial expected time. Connectivity is such an example as every $G \in G_{n,\Theta(\log n/n)}$ is almost surely connected and the same holds for every $G \in G_{n,1/2}$ (hamiltonicity is another example).

Remark 2 All of the above generic results, but the last one, have been proved for **PREL**. The reason is that we have exploited a minimal internal randomness of the nodes in order to be able to draw random graphs (equiprobably). The only exception was Theorem 17, which does not concern **PREL**, however, it also relies on the use of internal randomness. Note that in **REL** we can again construct a sufficiently long line (as our protocols for global line are in **REL**, since they do not use internal randomness) and exploit it as a space-bounded TM of the following sort: on input $g(n)$ (i.e., the size, in number of nodes, of the useful space) the TM outputs a graph of order $g(n)$. By exploiting such graph-constructing TMs we can again construct a possibly large class of networks without giving to our protocols access to randomization. For example, it could be a TM, that on every input $i \in \mathbb{N}$ constructs (deterministically, and without any random experiment) a ring (or a clique or a planar graph) of size i. Alternatively, we could simulate the internal randomness of the nodes by marking half of the nodes as 0 and the other half as 1. Then the current probabilistic choice of a node would depend on whether its previous interaction was with a node marked 0 or with a node marked 1 (this is not 100% equiprobable but it can be made so by other simple tricks).

6.4 Constructing and Simulating Supernodes with Logarithmic Memories

We now show that a population consisting of n nodes can be partitioned into k *supernodes* each consisting of $\log k$ nodes, for the largest such k. The internal structure of each supernode is a line, thus it can be operated as a TM of memory logarithmic in the total number of supernodes. This amount of storage is sufficient for the supernodes to obtain unique names and exploit their names and their internal storage to realize nontrivial constructions. We are interested in the networks that can be constructed at the supernode abstraction layer. The following theorem establishes that such a construction is feasible and presents a network constructor that achieves it.

Theorem 18 (Partitioning into Supernodes) *For every network G that can be constructed by k nodes having local memories $\lceil \log k \rceil$ and unique names there is a NET that constructs G on $n = k\lceil \log k \rceil$ nodes.*

Proof We present a NET $\mathcal{A}$ that when executed on n nodes it is guaranteed to organize the nodes into k lines of length $\lceil \log k \rceil$ each for the maximum k for which $k\lceil \log k \rceil \leq n$. We assume a unique pre-elected leader in the initial configuration of the system and we will soon show how to drop this requirement. Assume also for simplicity that $n \geq 8$ (this is again not necessary). The protocol operates in phases. Variable j denotes the current phase number, r denotes the number of new lines that should be constructed in the current phase, and a is a line counter. We assume that the leader has somehow already created 4 lines of length 2 each (note that here we count the length of a line in terms of its nodes). One of them is the leader's line. Also the left endpoint of the leader's line is directly connected to the left endpoints of the other 3 lines. In fact, all these assumptions are trivial to achieve. Initially $j \leftarrow 2$. All variables are stored by the leader in the distributed memory of its line.

– A new phase starts when the leader manages to increase by one the length of its line by attaching an isolated node its right endpoint. When this occurs, the leader sets $j \leftarrow j+1$, $r \leftarrow 2^{j-1}$, and $a \leftarrow 2$. A phase is divided into two subphases: the *Increment existing lines* subphase and the *Create new lines* subphase.

– *Increment existing lines*: Initially, all existing lines, excluding the leader's line, are marked as *unvisited.* While $a \leq r$ the leader visits an unvisited line and tries to increment its length by one by attaching an isolated node to its right endpoint. When it succeeds, it marks the line as *visited*, sets $a \leftarrow a + 1$ and returns to its own line. When this subphase ends all existing lines have length j. Then the leader sets $a \leftarrow 1$ and the *Create new lines* subphase begins.
– *Create new lines*: While $a \leq r$ the leader becomes connected to an isolated node, it marks that node as the left endpoint of the new line and then starts creating the new line node-by-node, by attaching isolated nodes to its right. It stops increasing the length of the new line when it becomes equal to the length of its own line. This can be easily implemented by a mark on the leader's line that moves one step to the right every time the length of the new line increases by one. The new line has the right length when the mark reaches the right endpoint of the leader's line. When this subphase ends there is a total of $2r = 2^j$ lines of length j each and the leader is directly connected to the left endpoint of each one of them. Then the leader waits again to increase its own length by one and when this occurs a new phase begins.

Naming. We now show that it is not hard to keep the constructed lines named (in fact there are various strategies for achieving this). Initially, the leader has 4 lines of length 2 each and we may assume that these are uniquely named $0, 1, 2, 3$ in binary, that is, every line has its name stored in its own memory. During a phase, the leader keeps a variable *cname* storing the current name to be assigned, initially 0. Whenever the leader increases the length of an existing line (during the *increment* subphase) or creates a new line (during the *create* subphase) it assigns to it *cname* in binary and sets *cname* $\leftarrow$ *cname* $+ 1$. Clearly, at the end of phase j the lines are uniquely named $0, 1, \ldots, 2^j - 1$.

Electing the Leader. We now show how to circumvent the problem of not having initially a unique pre-elected leader. In fact, as we will soon discuss, the solution we develop may serve as a generic technique for simulating protocols that assume a pre-elected leader. Initially all nodes are leaders in state l_0. Rule $(l_0, l_0, 0) \rightarrow (l, q_0, 0)$ eliminates one of the two l_0 leaders and converts the other to l. These l leaders start executing the above protocol by attaching q_0 and l_0 nodes to their construction. Each l leader executes the protocol on its own constructed component until it meets another l leader. When this occurs, one of the two ls becomes w. The goal of a w leader is to revert its whole component to a set of isolated nodes in state q_0 (itself inclusive). Note that a leader can easily revert a single line by beginning from the right endpoint and releasing one after the other the nodes until it reaches the left endpoint.

The generic idea (that works for other constructions as well) is that in order to release a node it suffices to know its degree. Then the only possible difficulty in our case is the fact that the left endpoint of the leader's line may be connected to a non-constant number of other endpoints. To resolve this, the leader exploits the fact that it can count in its line's memory the number of lines. When the reversion process begins, the leader knows the number of lines, that is, it knows also the degree of the left endpoint of its line. Whenever it reverts another line it decreases the counter by one. So, when the counter becomes equal to 1, it knows that the only remaining line is its own line, thus it knows that when it comes to release the last two nodes of its own line (i.e., during the interaction between the left endpoint and the other remaining node of the line) it should make both q_0 as there is no other reversion to be performed. This is quite important as it guarantees that reverting does not introduce waste. Note that if the reversion process could not determine its completion then every such reversion would result in a node remaining forever in state w. Such zombie ws cannot be exploited by other leaders in their constructions, as allowing a leader to attach a w would introduce conflicts between constructing and reverting processes.

Reinitialization. Note that the simulated protocol that constructs G assuming memories and names must be executed from the beginning, because protocol $\mathcal{A}$, that gives the organization into lines, is not terminating, so the two protocols must be composed in parallel. It suffices to have every line remember the number of active edges that it has to other lines. Then, whenever a new phase begins (implying that what has been constructed so far by the simulated protocol is not valid), each line deactivates one after the other all those edges and starts the simulation over.

The only drawback is that the above protocol retains forever the connections between the left endpoint of the leader's line and the left endpoints of the other lines. However, if we agree that the output-network of the protocol is the one induced by the active edges joining the right endpoints of lines then this is not an issue. Additionally, it should not be that hard to circumvent this subtlety by having the leader periodically release the constructed lines and reattracting them only in case it manages to increase the length of one of them. □

Many network construction problems are substantially simplified given the supernodes with names and memories. For a simple example, consider the problem of partitioning the nodes into triangles. This construction is quite hard to achieve in the original setting without a leader, however, given the supernodes it becomes trivial. Each supernode with id i checks whether its id is a multiple of 3 and, if it is, it connects to id $(i+2)$, otherwise it connects to id $(i-1)$. This is a totally parallel and thus a very efficient solution.

Finally, the above approach introduces the idea of constructing disjoint stable structures and then looking at those structures from a higher level and considering them as units (supernodes). It is then challenging, interesting, and valuable to understand how these units behave, what is the dependence of their behavior to their internal structure and configuration, what is the outcome of an interaction between two such units, and what are their constructive capabilities. In fact, one can imagine a whole hierarchy of such layers where nodes self-assemble into supernodes, supernodes self-assemble into supersupernodes, and so on. Formalizing this hierarchy is a very promising and totally open research direction.

7 Conclusions and Further Research

There are many open problems related to the findings of the present work. Though our universal constructors show that a large class of networks is in principle constructible, they provide neither the simplest nor the most efficient protocol for each single network in the class. To this end, we have provided direct constructors for some of the most basic networks, but there are still many other constructions to be investigated like grids or planar graphs. Moreover, a look at Table 2 makes it evident that there is even more work to be done towards the probabilistic analysis of protocols and in particular towards the establishment of tight bounds. Of special interest is the spanning line problem as it is a key component of universal construction. All of our attempts to give a protocol asymptotically faster than $O(n^3)$ have failed. Observe that with a pre-elected leader in state l and all edges initially inactive, the straightforward protocol $(l, q_0, 0) \rightarrow (q_1, l, 1)$ produces a stable spanning line in an expected number of $\Theta(n^2 \log n)$ steps (follows from the meet everybody fundamental process). Moreover, by a one-to-one elimination we can elect a unique leader in an expected number of $\Theta(n^2)$ steps. If we could safely compose these two protocols, then we would obtain a $\Theta(n^2 \log n)$ constructor which is almost optimal as our present best lower bound for the spanning line is $\Omega(n^2)$. The problem is that the protocol cannot detect when the leader-election phase has completed, thus it has to activate edges while still having more than one leader but this gives an overhead for either merging the constructed disjoint lines or deactivating some wrong connections. A possible solution could be to consider Monte Carlo protocols that may err with some small probability, e.g., a protocol that would try somehow to estimate when w.h.p. the leader-election phase completes and only then start the line construction phase.

We should mention that there is an improvement (which is also supported by experimental evidence) to the *Fast-Global-Line* protocol, however it is not yet clear whether this improvement is also an asymptotic one. The code of the improvement is given in Protocol 10.

Protocol 10 *Faster-Global-Line*

```
$Q = \{q_0, q_1, q_2, q, l, f\}$
$\delta$:

$(q_0, q_0, 0) \rightarrow (q_1, l, 1)$
$(l, q_0, 0) \rightarrow (q_2, l, 1)$
$(l, q, 0) \rightarrow (q_2, l, 1)$
$(l, l, 0) \rightarrow (l, f, 0)$
$(f, q_2, 1) \rightarrow (q, f, 0)$
$(f, q_1, 1) \rightarrow (q, q, 0)$
```

As in our previous protocols for the problem, many lines grow in parallel. When the leaders of two lines interact, one of them becomes a follower f. The follower starts deactivating its own line, releasing its nodes, while the l that survived does not change its behavior. Observe the contrast to the *Fast-Global-Line* protocol: in that protocol sleeping lines could only lose nodes by interacting with awake leaders, while now sleeping lines keep releasing their own nodes to make them available to the awake leaders. Eventually, a single l will remain and all other lines will have an f. It could be the case that the parallel releasing of the nodes of the f-lines allows the l leader to be able to rapidly expand towards free nodes and it would be really valuable to have a formal analysis of the running time of this variation. Also observe that the description of this protocol is rather simpler than the description of *Fast-Global-Line*.

One of the problems that we considered in this work, was the problem of constructing any k-regular network. Note that this is a quite different problem than the problem of constructing a specific k-regular network. For example, given a population of 10 processes is there a protocol that stabilizes to the Petersen graph? In general, it is worth considering non-uniform protocols that when executed on the correct number of nodes are required to construct a unique network like the cubical graph or the Wagner graph on 8 processes.

Another very intriguing issue has to do with the size of network constructors. In particular, we would like to know whether there is some generic lower bound on the size of all constructors, to give problem-specific lower bounds, and to formalize the apparent relationship between the size and the running time of a protocol. Is there some sort of hierarchy showing that with more states we can produce faster protocols (until optimality is obtained)?

To this end, observe that neither the maximum degree nor the number of different degrees of the target-network are lower bounds on the number of states required to construct the network. For the former, it is not hard to show that $\Theta(x)$ states suffice to make a node obtain 2^x neighbors (stably). The idea is to have a node initially obtain 2 neighbors and then repeatedly double their number. For the latter, one can show that $\Theta(x)$ states suffice to have 2^x nodes with different degrees (stably) and in particular for all $i \in \{1, ..., 2^x\}$ we obtain a node with degree i. The idea is to mark a set of 2^x nodes as before and construct a line spanning these nodes. Then the protocol assigns to the ith node of the line, counting, e.g., from the left endpoint, i neighbors. This can be done by using only a constant number of states. The head begins from the left endpoint and moves step-by-step on the line towards u. For every step it takes it assigns to u a new neighbor and stops when it reaches u. In this manner, it assigns to u a number of neighbors equal to its distance from the endpoint without having to explicitly count the distance. Is there some other property of the target-network that determines the number of states that have to be used?

It is also worth noting that our results on universal construction indicate that the constructive power increases as a function of the available waste. A complete characterization of this dependence would be of special value.

There is also a practically unlimited set of variations of the proposed model that are worth considering. We mention a few of them. As already discussed, in this work we have considered a model of network construction with as minimal assumptions as possible to serve as a simple and clear starting point for more applied models to be defined. We now introduce such a model which seems to be of particular interest. Assume that every node is equipped with a predefined number of ports at specific positions of its "body". For example, in the 2-dimensional case these could be "North", "South", "East", "West" having the obvious angles between them. Nodes interact via their ports and they can detect which of their ports are used in an interaction. Moreover, when a connection is activated, it is always activated at a predetermined distance (i.e., all connections have the same length d) and it is always a straight line respecting the angles between itself and the (potentially active) lines of the other ports of the same node. Such a model (and possible variations of it, depending on the assumed hardware) seems particularly suitable for studying/designing very simple and local distributed protocols that are capable of constructing stable geometric objects (even in three dimensions), like squares, cubes, or more complex polyhedra, without any mobility-control mechanism (a first attempt towards this direction is [Mic15]).

Another immediate extension of our model is to allow the connections to have more than just the two states that we considered in this work. Recall also that, whenever we had to analyze the running time of a protocol, we did it under the uniform random scheduler, mainly because we wanted to keep this first model of network construction as simple as possible and because of its correspondence to a well-mixed solution. However, there are many other natural probabilistic scheduling models to be considered which would probably require different algorithmic developments and techniques to achieve efficiency. It is also natural to consider a variant in which connected nodes communicate much faster (even in synchronous rounds) than disconnected nodes. Moreover, it would be interesting to consider a model of network construction in which the behavior of a node depends on some input from the environment (this would allow the consideration of codes that exhibit different behaviors in different environments). The model in which a connected component has access to a *self-bit* indicating whether a given interaction involves two nodes of the same component or not, also seems interesting and natural. It is not yet clear whether this extra assumption increases the constructive power of the model but it is clear that it substantially simplifies the description of several protocols. It would also be of its own value to depart from cooperative models and consider an antagonistic scenario in which different sets of nodes try to construct different networks (by deterministic codes and not game-theoretic assumptions involving incentives). It would be interesting to discover cases in which the antagonism leads to unexpected stable formations.

Finally, a very valuable and challenging interdisciplinary goal is to further investigate and formalize the apparent applicability of the model proposed here (and potential variations of it) in physical and chemical (possibly biological) processes. As already stated, we envision that a potential usefulness of such models is to unveil the algorithmic properties underlying the structure/network formation capabilities of natural processes.

Acknowledgements. We would like to thank Leslie Ann Goldberg for bringing to our attention the importance of constructing regular networks and Dimitrios Amaxilatis and Marios Logaras for experimenting with our protocols and detecting a bug in the *Global-Ring* protocol. Finally, we would like to thank the anonymous reviewers of this article, and also those of some previous versions of it, for carefully reading every single line of our manuscript. Their thorough comments have helped us to improve our work substantially.

References

AAC+05. D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In *Proceedings of the 1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, volume 3560 of *LNCS*, pages 63–74. Springer-Verlag, June 2005.

AAD+06. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006.

AAER07. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20[4]:279–304, November 2007.

Adl94. L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266[11]:1021–1024, November 1994.

Ang80. D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th annual ACM symposium on Theory of computing (STOC)*, pages 82–93. ACM, 1980.

AR09. J. Aspnes and E. Ruppert. An introduction to population protocols. In B. Garbinato, H. Miranda, and L. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer-Verlag, 2009.

BA99. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286[5439]:509–512, 1999.

BBCK10. J. Beauquier, J. Burman, J. Clement, and S. Kutten. On utilizing speed in networks of mobile agents. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC)*, pages 305–314. ACM, 2010.

BEK+13. L. Blume, D. Easley, J. Kleinberg, R. Kleinberg, and É. Tardos. Network formation in the presence of contagious risk. *ACM Transactions on Economics and Computation*, 1[2]:6, 2013.

Bol01. B. Bollobás. *Random graphs*, volume 73. Cambridge university press, 2001.

BPS+10. A. Bandyopadhyay, R. Pati, S. Sahu, F. Peper, and D. Fujita. Massively parallel computing on an organic molecular layer. *Nature Physics*, 6[5]:369–375, 2010.

CCDS14. H.-L. Chen, R. Cummings, D. Doty, and D. Soloveichik. Speed faults in computation by chemical reaction networks. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, volume 8784 of *LNCS*, pages 16–30. Springer, 2014. Also to appear in *Distributed Computing*.

CMN+11. I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating machines that use restricted space. *Theoretical Computer Science*, 412[46]:6469–6483, October 2011.

DDG+14. Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Brief announcement: amoebot–a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 220–222. ACM, 2014.

DFSY15. S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28[2]:131–145, April 2015.

DGRS13. S. Dolev, R. Gmyr, A. W. Richa, and C. Scheideler. Ameba-inspired self-organizing particle systems. *arXiv preprint arXiv:1307.4259*, 2013.

Dot12. D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55:78–88, 2012.

Dot14. D. Doty. Timing in chemical reaction networks. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–784, 2014.

ER59. P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

Fel68. W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition, Revised Printing.* Wiley, 1968.

GR09. R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5556 of *LNCS*, pages 484–495. Springer-Verlag, 2009.

Jac05. M. O. Jackson. A survey of network formation models: Stability and efficiency. *Group Formation in Economics: Networks, Clubs and Coalitions, ed. G. Demange and M. Wooders*, pages 11–57, 2005.

Lyn96. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann; 1st edition, 1996.

MCS11a. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Mediated population protocols. *Theoretical Computer Science*, 412[22]:2434–2450, May 2011.

MCS11b. O. Michail, I. Chatzigiannakis, and P. G. Spirakis. *New Models for Population Protocols.* N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2011.

Mic15. O. Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. In *Proceedings of the 34th ACM Sym-*

posium on Principles of Distributed Computing (PODC), pages 37–46. ACM, 2015.

MR95. R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

MS14. O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 76–85. ACM, 2014.

RCN14. M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345[6198]:795–799, 2014.

RW00. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 459–468, 2000.

Sch11. J. L. Schiff. *Cellular automata: a discrete view of the world*, volume 45. Wiley-Interscience, 2011.

SY99. I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28[4]:1347–1363, March 1999.

VMC$^+$12. N. Vaidya, M. L. Manapat, I. A. Chen, R. Xulvi-Brunet, E. J. Hayden, and N. Lehman. Spontaneous network formation among cooperative RNA replicators. *Nature*, 491[7422]:72–77, 2012.

WCG$^+$13. D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 353–354. ACM, 2013.

Win98. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

Zha03. S. Zhang. Fabrication of novel biomaterials through molecular self-assembly. *Nature biotechnology*, 21[10]:1171–1178, 2003.