Simple and Efficient Local Codes for Distributed Stable Network Construction*

Othon Michail
Computer Technology Institute & Press "Diophantus" (CTI)
Patras, Greece
michailo@cti.gr

Paul G. Spirakis
Department of Computer Science, University of Liverpool, Liverpool, UK
& CTI, Patras, Greece
P.Spirakis@liverpool.ac.uk

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.

Categories and Subject Descriptors:
C.2.4 [Computer-communication Networks]: Distributed Systems; C.2.1 [Computer-communication Networks]: Network Architecture and Design—*distributed networks, network communications, network topology*; F.1.1 [Computation By Abstract Devices]: Models of Computation—*automata, computability theory, unbounded-action devices*; F.1.2 [Computation By Abstract Devices]: Modes of Computation—*parallelism and concurrency, probabilistic computation*; J.2 [Computer Applications]: Physical Sciences and Engineering—*Chemistry, Physics*

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

1. INTRODUCTION

1.1 Motivation

Suppose a set of tiny computational devices (possibly at the nanoscale) is 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 (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 con-

*Supported in part by the project FOCUS, implemented under the "ARISTEIA" Action of the OP EdLL co-funded by the EU (ESF) and Greek National Resources. Full version: http://ru1.cti.gr/aigaion/?page=publication&kind=single&ID=989

PODC'14, July 15–18, 2014, Paris, France.

ACM 978-1-4503-2944-6/14/07 ...$15.00.
http://dx.doi.org/10.1145/2611462.2611466.

nection 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 $i \in \{1, 2, \ldots, k\}$, 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. 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 such a generic scheduler gives no information about 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), anonymous, 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. 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. Equivalently, given a (possibly 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 thus the constructed spanning star is stable. It is worth noting that this very simple protocol is optimal both w.r.t. to the number of states that it uses and w.r.t. to 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 [3] and the Mediated Population Protocol model [18]. 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 [18]. 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* and are widely used to describe information processing occurring in natural cellular regulatory networks [13]. However, CRNs and population protocols can only capture the dynamics of molecular counts and not of structure formation. Our model then may also be 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). 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* [4]. Semilinearity persists up to $o(\log \log n)$ local space but not more than this [9]. If addi-

tionally 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 $\mathbf{NSPACE}(n^2)$ [18]. Other important works include [16] and [7]. For a very recent introductory text see [19].

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. We remark that the spanning line problem is of outstanding importance because it constitutes a basic ingredient of universal constructors. We give three different protocols for this problem each improving on the running time but using more states to this end. Additionally, we establish a $\Omega(n \log n)$ generic lower bound on the expected running time of all constructors that construct a spanning network and a $\Omega(n^2)$ 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, (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. In Section 3, we formally define the model of network constructors and the network construction problems that are considered in this work. 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 trying to capture the self-assembly capability of natural processes with the purpose of engineering systems and developing algorithms inspired by such processes. For example, [12] proposes to learn how to program molecules to manipulate themselves, grow into machines and at the same time control their own growth. The model guiding the study in algorithmic self-assembly is the Abstract Tile Assembly Model (aTAM) [24, 21] and variations. In contrast to those models that try to incorporate the exact molecular mechanisms, 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 that may be obtained from our model by imposing restrictions on the scheduler, the degree, and the number of local states.

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 [1], 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, most of the time under the assumption of unique identities and unbounded memories. An exception is the area of geometric pattern formation by mobile robots (cf. [23, 10]). 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. Additionally, as observed in [11], in the area of self-organizing particle systems, only very little theoretical work has been done. This further supports the importance of introducing a simple yet sufficiently generic model for distributed network construction.

Cellular Automata. A cellular automaton (cf. e.g. [22]) 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. Cellular automata have been used as models for self-replication, for modeling several physical systems, 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. 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, called players, which usually have incentives and connections between them indicate some social relationship. The network is formed by allowing the individuals to form or delete connections usually selfishly by trying to maximize their own utility (see e.g. [17, 8]). 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 (see e.g. [6]). 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.

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 e.g. molecules) automatically assemble into large, and usually complex structures (like e.g. a crystal)

[12]. There is an abundance of such examples in the physical world. 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 which 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 [25]. 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 [5] and miniature (nano)robots for surgical instrumentation, diagnosis and drug delivery in medical applications (e.g. it has very recently been reported that DNA nanorobots could even kill cancer cells [14]) and monitoring in extreme conditions (e.g. in toxic environments). However, the road towards this vision passes first through 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 engineer artificial systems that achieve this complexity.

3. 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. In particular, we assume that, for all distinct node-states $a, b \in Q$ and for all edge-states $c \in \{0,1\}$, δ specifies either (a,b,c) or (b,a,c). 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 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'.

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 $C \xrightarrow{e} C'$, 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 $C \xrightarrow{e} C'$ 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.

Note that NETs are *uniform* and *anonymous*, that is the size of the protocol does not depend on the number of nodes n and there is not enough room in the states of the nodes to store unique identifiers.

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 $t \geq 0$ s.t. $G(C_i) = G$ 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. In this case, the running time becomes a random variable (abbreviated "r.v.") 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.

3.1 Problem Definitions

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

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. (Already discussed in Section 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 $G = (V, E)$ on a subset V_1 of the processes ($|V_1| = |V|$). The processes in V_1 are initially in state q_0 and the edges of E are the active edges between them. All other edges in E_I are initially inactive. The processes in $V_2 = V_I \backslash V_1$ are initially in state r_0. The goal is to create a *replica* of G 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.

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 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 (always under the uniform random scheduler) of any protocol that constructs a spanning network is $\Omega(n \log n)$.*

PROOF. Consider the time at which the last edge is activated. By that time, all nodes have some active edge incident to them, thus every node has interacted at least once. The latter can be shown to require an expected number of $\Theta(n \log n)$ steps. □

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 $\Omega(n^2)$.*

We proceed by presenting our simplest protocol for the spanning line problem.
Protocol Simple-Global-Line: $(q_0, q_0, 0) \to (q_1, l, 1)$, $(l, q_0, 0) \to (q_2, l, 1)$, $(l, l, 0) \to (q_2, w, 1)$, $(w, q_2, 1) \to (q_2, w, 1)$, $(w, q_1, 1) \to (q_2, l, 1)$

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. *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 active lines and isolated nodes. Additionally, every active 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.

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 active lines with unique leaders and isolated nodes. 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 $(l, q_0, 0) \to (q_2, l, 1)$ to expand the line towards all isolated nodes. Thus we may now w.l.o.g. consider a collection of lines without isolated nodes. By successively applying the rule $(l, l, 0) \to (q_2, w, 1)$ to pairs of lines while always moving the internal leaders that appear towards an endpoint

it is not hard to see that the 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 further 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 (until state w reaches one endpoint of the line) to finish and to have again the system ready for progress. $O(n^4)$ follows because we have a random walk on a line with two absorbing barriers (see e.g. [15] pages 348-349) delayed on average by a factor of $O(n^2)$. As progress is made $n-2$ times, we conclude that the expected running time of the protocol is upper bounded from above by $(n-2)[O(n^2)+O(n^4)] = O(n^5)$.

Running Time Lower Bound. We next prove that we cannot hope to improve the above upper bound by a better analysis by more than a factor of n. For this we first prove that the protocol w.h.p. constructs $\Theta(n)$ different lines of length 1 during its course. A set of k disjoint lines implies that k distinct merging processes have to be executed in order to merge them all in a common line and each single merging results in the execution of another random walk. We exploit all these to 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 R be the r.v. of 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 (e.g. until no further successes are possible or until stabilization). 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 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. [20], 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, we have $\exp(-\mu\delta^2/2) = \epsilon \Leftrightarrow \delta = \sqrt{\frac{2\ln 1/\epsilon}{\mu}}$. 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 $c = O(1)$, $\mathrm{P}[Y < \frac{1}{16}\left(n-2\sqrt{cn\ln n}-2\right)] < n^{-c} \Rightarrow \mathrm{P}[Y \geq \frac{1}{16}\left(n-2\sqrt{cn\ln n}-2\right)] > 1-n^{-c} \Rightarrow \mathrm{P}[Y = \Theta(n)] > 1-n^{-c}$ and as X dominates Y, we have $\mathrm{P}[X = \Theta(n)] > 1-n^{-c}$. In words, w.h.p. we expect $\Theta(n)$ lines of length 1 to be constructed by the protocol.

Now, given that $X = \Theta(n)$, we distinguish two cases: (i) At some point during the course of the protocol two lines both of length $\Theta(n)$ get merged. In this case, the corresponding random walk takes on average $\Theta(n^2)$ transitions involving the leader and on average the leader is selected every $\Theta(n^2)$ steps to interact with one of the 2 active edges incident to it. That is, the expected number of steps for the completion of such a random walk is $\Theta(n^4)$ and the expected running time of the protocol is $\Omega(n^4)$. (ii) In every merging process, at least one of the two participating lines has length at most $d_{max} = o(n)$. We have already shown that the protocol w.h.p. forms $k = \Theta(n)$ distinct lines of length 1. Consider now the interval $I = \{k/2 - d_{max} + 1, \ldots, k/2\}$. As $h = \Theta(n) > d_{max}$ for all $h \in I$, only a single line can ever have length $h \in I$ and one, call it l_1, will necessarily fall in this interval due to the fact that the length of l_1 will increase by at most d_{max} in every merging until it becomes n. Consider now the time at which l_1 has length $h \in I$. As the total length due to lines of length 1 (ever to appear) is k and the length of l_1 is h there is still a remaining length of at least $k-h \geq k-k/2 = k/2 = \Theta(n)$ to be merged to l_1. As the maximum length of any line different than l_1 is d_{max}, l_1 will get merged to the $k-h$ remaining length via at least $j = \Theta(n)/d_{max}$ distinct mergings with lines of length at most d_{max}. These mergings, and thus also the resulting random walks, cannot occur in parallel as all of them share l_1 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_1, for $1 \leq i \leq j$. If l_1 has length $d(l_1)$ just before the i-th merging, then the expected duration of the resulting random walk is $n^2 \cdot d(l_1) \cdot d_i$ and the new l_1 resulting from merging will have length $d(l_1) + d_i$. 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_1 is $\mathrm{E}[Y] = \mathrm{E}[\sum_{i=1}^{j} Y_i] = \sum_{i=1}^{j} \mathrm{E}[Y_i] = \sum_{i=1}^{j} n^2(h + d_1 + \ldots + d_{i-1})d_i \geq n^2 \sum_{i=1}^{j} h d_i = n^2 h \sum_{i=1}^{j} d_i = n^2\Theta(n)\Theta(n) = \Theta(n^4)$. The fifth equality follows from the fact that $\sum_{i=1}^{j} d_i = k - h = \Theta(n)$. We conclude that the expected running time of the protocol is also in this case $\Omega(n^4)$.

Now 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: $\mathrm{E}[W] = \mathrm{E}[W \mid X = \Theta(n)] \cdot \mathrm{P}[X = \Theta(n)] + \mathrm{E}[W \mid X = o(n)] \cdot \mathrm{P}[X = o(n)] \geq \mathrm{E}[W \mid X = \Theta(n)] \cdot \mathrm{P}[X = \Theta(n)] > \Theta(n^4)(1-n^{-c}) = \Theta(n^4 - n^{4-c}) = \Theta(n^4)$. Thus, the expected running time of the protocol is $\Omega(n^4)$. □

We now give our fastest protocol for the global line construction. [1] The main difference to the previous protocol is that we now totally avoid mergings as they seem to consume much time. The intuition is that when two disjoint lines in-

[1] In the full paper, we also give an intermediate protocol that performs a more "deterministic" merging. That protocol uses 3 more states than *Simple-Global-Line* but improves the expected running time to between $\Omega(n^3)$ and $O(n^4)$.

teract, instead of merging, the corresponding leaders play a pairwise game. The winner grows by one towards the other line and the loser sleeps. A sleeping line cannot increase any more and only loses nodes by lines that are still awake. A single leader is guaranteed to always win and eventually remain unique and this occurs quite fast. Then the leader makes progress (by one) in most interactions and every such progress is in turn quite fast.
Protocol Fast-Global-Line: $(q_0, q_0, 0) \to (q_1, l, 1)$, $(l, q_0, 0) \to (q_2, l, 1)$, $(l, l, 0) \to (q_2', l', 1)$, $(l', q_2, 1) \to (l'', f_1, 0)$, $(l', q_1, 1) \to (l'', f_0, 0)$, $(l'', q_2', 1) \to (l, q_2, 1)$, $(l, f_0, 0) \to (q_2, l, 1)$, $(l, f_1, 0) \to (q_2', l', 1)$

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* $O(n^3)$.

5. OTHER BASIC CONSTRUCTORS

Protocol Cycle-Cover: $(q_0, q_0, 0) \to (q_1, q_1, 1)$, $(q_1, q_0, 0) \to (q_2, q_1, 1)$, $(q_1, q_1, 0) \to (q_2, q_2, 1)$

THEOREM 5 (CYCLE COVER). *Protocol* Cycle-Cover *constructs a cycle cover with waste 2. It uses 3 states, its expected running time is* $\Theta(n^2)$*, and it is optimal w.r.t. time.*

THEOREM 6 (RING LOWER BOUND). *The expected time to convergence of any protocol that constructs a spanning ring is* $\Omega(n^2)$.

Protocol Global-Ring: The rules of Protocol *Simple-Global-Line* and additionally $(l, q_1, 0) \to (l', q_1', 1)$, $(x', y, 0) \to (x'', y, 0)$, for $x \in \{l, q_1\}$ and $y \in \{l, w, q_1, q_0\}$, $(x', y', 0) \to (x'', y'', 0)$, for $x \in \{l, q_1\}$ and $y \in \{l, q_1\}$, $(l'', q_1', 1) \to (l, q_1, 0)$, $(l', q_1'', 1) \to (l, q_1, 0)$, $(l'', q_1'', 1) \to (l, q_1, 0)$

THEOREM 7 (GLOBAL RING). *Protocol* Global-Ring *constructs a spanning ring using 9 states.*

The following protocols can be found in the full paper.

THEOREM 8 (k-REGULAR CONNECTED). *We provide a parameterized protocol (called* kRC*) which, for every fixed integer* $k \geq 2$ *and every population of size* $n \geq k+1$*, constructs a connected spanning network in which at least* $n-k+1$ *nodes have degree* k *and each of the remaining* $l \leq k-1$ *nodes has degree at least* $l-1$ *and at most* $k-1$.

THEOREM 9 (MANY SMALL COMPONENTS). *We provide a parameterized protocol (*c-Cliques*) which, for every fixed positive integer* c*, constructs* $\lfloor n/c \rfloor$ *cliques of order* c *each.*

THEOREM 10 (REPLICATION). *We provide a protocol (*Leader-Replication*) that 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 is* $\Theta(n^4 \log n)$.

Table 1 summarizes all upper and lower bounds that we established in Sections 1 (global star), 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. Denote by l the binary length of the input of a TM and by n the size of a population. Note that Theorems 11 to 14 construct a random graph in the useful space and that graph constitutes the input to a TM. Thus, if the useful space consists of h nodes, the input to the TM has size $l = \Theta(h^2)$ and, as all of our constructors have $h = \Theta(n)$, in what follows it holds that $l = \Theta(n^2)$. Moreover, the TM can use space at most $O((n-h)^2) = O(n^2)$, where $n - h$ is the waste.

THEOREM 11 (LINEAR WASTE-HALF). $\mathbf{DGS}(O(\sqrt{l})) \subseteq \mathbf{PREL}(\lfloor n/2 \rfloor)$. *In words, for every graph language* L *that is decidable by a* $O(\sqrt{l})$*-space TM, there is a protocol that constructs* L *equiprobably with useful space* $\lfloor n/2 \rfloor$.

PROOF. We present here the main idea. Given a population of size n, the protocol, call it $\mathcal{A}$, 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 an active perfect matching between U and D. By using the *Simple-Global-Line* protocol of Section 4 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 e.g. 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 $\sqrt{l}$ 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 *Simple-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

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		$\Omega(n^2)$
kRC	$2(k+1)$		$\Omega(n \log n)$
c-Cliques	$5c-3$		$\Omega(n \log n)$
Leader-Replication	12	$\Theta(n^4 \log n)$	

Table 1: All upper and lower bounds established in this work. *Leader-Replication* is a randomized protocol thus it concerns class PREL, while all other protocols do not use randomization and concern REL.

correct configuration (free of residues from previous partial simulations). □

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 12 (Linear Waste-Two Thirds). $\mathbf{DGS}(l + O(\sqrt{l})) \subseteq \mathbf{PREL}(\lfloor n/3 \rfloor)$.

Proof. The idea is to partition the population into three equal sets U, D, and M instead of the two sets of Theorem 11. The purpose of sets U and D is more or less as in Theorem 11. 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 1). 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 11).

As everything works in precisely the same way as in Theorem 11, 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: $(q_0, q_0, 0) \to (q'_u, q_d, 1)$, $(q'_u, q_0, 0) \to (q_u, q_m, 1)$, $(q'_u, q'_u, 0) \to (q_u, q'_m, 1)$, and $(q'_m, q_d, 1) \to (q_m, q_0, 0)$. 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. 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. □

We now relax our requirement for simulation space in order to reduce the waste.

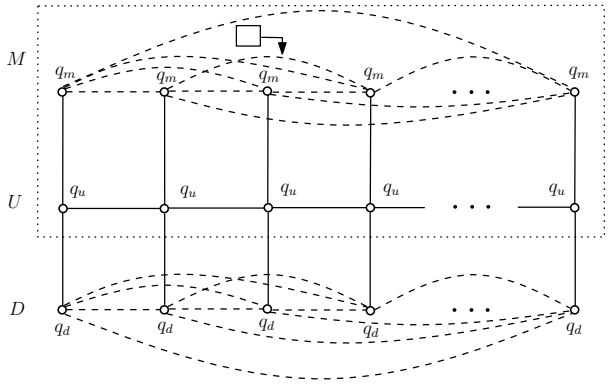

Figure 1: A partitioning into three equal sets U, D, and M. The line of set U plays the role of an ordering that will be exploited both by the random graph drawing process and by the TM-simulation. The line of set U instead of using its $\Theta(n)$ memory as the memory of the TM it now uses the $\Theta(n^2)$ 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.

Theorem 13 (Logarithmic Waste). $\mathbf{DGS}(O(\log l)) \subseteq \mathbf{PREL}(n - \log n)$.

Proof. We give the main idea. 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 or not to a given language L. If yes, then we are done. If not, then the TM just repeats the random experiment and restarts the simulation.

Reinitialization. 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 are more than one such lines, 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. □

We next prove that a large class of graph-families can be constructed with no waste.

THEOREM 14 (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 logarithmic order s.t. either G' or its complement is connected and has degree upper bounded by d and (ii) L is decidable in logarithmic space. Then $L \in$ **PREL**(n), i.e. there is a protocol that constructs L equiprobably with useful space n.*

PROOF. We give again the main idea. As in Theorem 13, 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 [2]) of logarithmic space with a unique leader on some node. Next M draws (more or less as in Theorem 13) 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 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. When N first accepts, the protocol sleeps (in the sense that it does not terminate but does not alter edges anymore either). □

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 (e.g. connectivity, hamiltonicity).*

Finally, we establish 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.

THEOREM 15 (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.*

7. CONCLUSION AND OPEN PROBLEMS

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 do not imply neither the simplest nor the most efficient protocols. 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 1 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 (our best lower bound for the problem is $\Omega(n^2)$). Is there a $\Theta(n^2 \log n)$ constructor?

Another very intriguing issue has to do with the size of network constructors. In particular, we would like 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)? Moreover, it is worth considering non-uniform protocols that when executed on the correct number of nodes are required to construct a unique network. For example, given a population of 10 processes is there a protocol that stabilizes to the Petersen graph?

Another interesting open problem is to characterize the class **REL** in which protocols do not have access to (internal) randomness. Note that in **REL** we can again construct a sufficiently long line (as our protocols for global line are in **REL**) and exploit it as a space-bounded TM. 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 is worth considering. An immediate extension of our model is to allow the connections to have more than just the two states that we considered in this work. Another route is to assume other natural probabilistic scheduling models, than the uniform considered here, which would probably require different algorithmic developments and techniques to achieve efficiency.

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.

Acknowledgments: We would like to thank Leslie Ann Goldberg for bringing to our attention the importance of constructing regular networks and also the reviewers of this work and some previous versions of it whose comments have helped us to improve our work substantially.

8. REFERENCES

[1] 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.

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC)*, pages 267–276, 2010.

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

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

[13] D. Doty. Timing in chemical reaction networks. In *Proc. of the 25th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 772–784, 2014.

[14] S. M. Douglas, I. Bachelet, and G. M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge university press, 1995.

[21] 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.

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

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

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

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