Terminating Distributed Construction of Shapes and Patterns in a Fair Solution of Automata*

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

ABSTRACT

In this work, we consider a *solution of automata* similar to *Population Protocols* and *Network Constructors*. The automata, also called *nodes*, move passively in a well-mixed solution and can *cooperate* by interacting in pairs. During every such interaction, the nodes, apart from updating their states, may also choose to connect to each other in order to start forming some required structure. The model introduced here is a more applied version of Network Constructors, imposing *geometrical constraints* on the permissible connections. Each node can connect to other nodes only via a very limited number of *local ports*, which implies that at any given time it has only a *bounded number of neighbors*. Connections are always made at *unit distance* and are *perpendicular to connections of neighboring ports*. Though this variation can no longer form abstract networks, it is still capable of forming very practical *2D or 3D shapes*. We develop *new techniques* for determining the computational and constructive capabilities of our model. One of the main novelties, concerns our attempt to overcome the inherent inability of such systems to terminate. In particular, exploiting the assumptions that the system is well-mixed and has a unique leader, we *give terminating protocols that are correct with high probability* (w.h.p.). This allows us to develop terminating subroutines that can be *sequentially composed* to form larger *modular protocols*. One of our main results is a *terminating protocol counting the size n of the system* w.h.p.. We then use this protocol as a subroutine in order to develop our *universal constructors*, establishing that *it is possible for the nodes to self-organize w.h.p. into arbitrarily complex shapes and additionally always terminate*.

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; programmable matter; shape formation; population; distributed protocol; interacting automata; fairness; random schedule; self-organization

*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://arxiv.org/abs/1503.01913

PODC'15, July 21–23, 2015, Donostia-San Sebastián, Spain.

ACM 978-1-4503-3617-8/15/07 ...$15.00.
http://dx.doi.org/10.1145/2767386.2767402.

1. INTRODUCTION

Recent research in distributed computing theory and practice is taking its first timid steps on the pioneering endeavor of investigating the possible *relationships of distributed computing systems to physical and biological systems*. The first main motivation for this is the fact that a wide range of physical and biological systems are governed by underlying laws that are essentially *algorithmic*. The second is that the higher-level physical or behavioral properties of such systems are usually the outcome of the coexistence and constant interaction, which may include both cooperation and competition, of *very large numbers of relatively simple distributed entities* respecting such laws. This effort, to the extent that its perspective allows, is expected to promote our understanding of the algorithmic aspects of our (distributed) natural world and to develop innovative artificial systems inspired by these aspects.

Ulam's and von Neuman's Cellular Automata (cf. e.g. [16]), essentially a distributed grid network of 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. Population Protocols of Angluin *et al.* [1] were originally motivated by highly dynamic networks of simple sensor nodes that cannot control their mobility. Recently, Doty [4] demonstrated their formal equivalence to *chemical reaction networks* (CRNs), which model chemistry in a *well-mixed solution*. Moreover, the *Network Constructors* extension of population protocols [12], showed that a population of finite-automata that interact randomly like molecules in a well-mixed solution and that can establish bonds with each other according to the rules of a common small protocol, can construct arbitrarily complex stable networks. In the young area of DNA self-assembly it has been already demonstrated that it is possible to fold long, single-stranded

DNA molecules into arbitrary nanoscale two-dimensional shapes and patterns [14]. Recently, an interesting theoretical model was proposed, the *Nubot* model, for studying the complexity of self-assembled structures with active molecular components [17]. Finally a system, called the *Kilobot*, was reported recently [15], that demonstrates programmable self-assembly of complex two-dimensional shapes by a swarm consisting of *a thousand* small, cheap, and simple autonomous robots designed to operate in large groups and to cooperate through local interactions.

The established and ongoing research seems to have opened the road towards a vision that will further reshape society to an unprecedented degree. This vision concerns our ability to *manipulate matter* via information-theoretic and computing mechanisms and principles. It will be the jump from amorphous information to the *incorporation of information to the physical world*. Information will not only be part of the physical environment: it will constantly interact with the surrounding environment and will have the ability to reshape it. *Matter will become programmable* [7] which is a plausible future outcome of progress in high-volume nanoscale assembly that makes it feasible to inexpensively produce millimeter-scale units that integrate computing, sensing, actuation, and locomotion mechanisms. This will enable the astonishing possibility of transferring the discrete dynamics from the computer memory black-box to the real world and to achieve a *physical realization of any computer-generated object*. It will have profound implications for how we think about chemistry and materials. Materials will become user-programmed and smart, adapting to changing conditions in order to maintain, optimize or even create a whole new functionality using means that are intrinsic to the material itself. It will even change the way we think about engineering and manufacturing. We will for the first time be capable of building smart machines that adapt to their surroundings, such as an airplane wing that adjusts its surface properties in reaction to environmental variables [18], or even further realize machines that can self-built autonomously.

1.1 Our Approach-Contribution

We imagine here a "solution" of automata (also called *nodes* or *processes* throughout the paper), a setting similar to that of Population Protocols and Network Constructors. Due to its highly restricted computational nature and its very local perspective, each individual automaton can practically achieve nothing on its own. However, when many of them cooperate, each contributing its meager computational capabilities, impressive global outcomes become feasible. This is also, for example, the case in the Kilobot system, where each individual robot is a remarkably simple artifact that can perform only primitive locomotion via a simple vibration mechanism. Still, when a thousand of them work together, their global dynamics and outcome resemble the complex functions of living organisms. From our perspective, cooperation involves the capability of the nodes to communicate by interacting in pairs and to bind to each other in an algorithmically controlled way. In particular, during an interaction, the nodes can update their local states according to a small common program that is stored in their memories and may also choose to connect to each other in order to start forming some required structure. Later on, if needed, they may choose to drop their connection, e.g. for rearrangement purposes. We may think of such nodes as the smallest possible programmable pieces of matter. For example, they could be tiny nanorobots or programmable molecules (e.g. DNA strands). Naturally, such elementary entities are not (yet) expected to be equipped with some internal mobility mechanism. Still, it is reasonable to expect that they could be part of some dynamic environment, like a boiling liquid or the human circulatory system, providing an external (to the nodes) interaction mechanism. This, together with the fact that the dynamics of such models have been recently shown to be equivalent to those of CRNs, motivate the idea of regarding such systems as *a solution of programmable entities*. We model such an environment by imagining an *adversary scheduler* operating in discrete steps and selecting in every step a pair of nodes to interact with each other.

Our main focus in this work, building upon the findings of [12], is to further investigate the cooperative structure formation capabilities of such systems. Our first main goal is to introduce a more realistic and more applicable version of network constructors by adjusting some of the abstract parameters of the model of [12]. In particular, we introduce some physical (or geometrical) constraints on the connections that the processes are allowed to form. In the network constructors model of [12], there were no such imposed restrictions, in the sense that, at any given step, any two processes were candidates for an interaction, independently of their relative positioning in the existing structure/network. This was very convenient for studying the capability of such systems to self-organize into abstract networks and it helped show that arbitrarily complex networks are in principle constructible. On the other hand, this is not expected to be the actual mechanism of at least the first potential implementations. First implementations will most probably be characterized by physical and geometrical constraints. To capture this in our model, we assume that each device can connect to other devices only via a very limited number of ports, usually four or six, which implies that, at any given time, a device has only a bounded number of neighbors. Moreover, we further restrict the connections to be always made at unit distance and to be perpendicular to connections of neighboring ports. Though such a model can no longer form abstract networks, it may still be capable of forming very practical 2D or 3D shapes. This is also in agreement with natural systems, where the complexity and physical properties of a system are rarely the result of an unrestricted interconnection between entities.

It can be immediately observed that the universal constructors of [12] do not apply in this case. In particular, those constructors cannot be adopted in order to characterize the constructive power of the model considered here. The reason is that they work by arranging the nodes in a long line and then exploiting the fact that connections are "elastic" and allow any pair of nodes of the line to interact independently of the distance between them. In contrast, no elasticity is allowed in the more local model considered here, where a long line can still be formed but only adjacent nodes of the line are allowed to interact with each other. As a result, we have to develop new techniques for determining the computational and constructive capabilities of our model. The other main novelty of our approach, concerns our attempt to overcome the inability of such systems to detect termination due to their limited global knowledge and their limited computational resources. For example, it can

be easily shown that deterministic termination of population protocols can fail even in determining whether there is a single a in an input assignment, mainly because the nodes do not know and cannot store in their memories neither the size of the network nor some upper bound on the time it takes to meet (or to influence or to be influenced by) every other node. To overcome the storage issue, we exploit the ability of nodes to self-assemble into larger structures that can then be used as distributed memories of any desired length. Moreover, we exploit the common (and natural in several cases) assumption that the system is *well-mixed*, meaning that, at any given time, all permissible pairs of node-ports have an equal probability to interact, in order to give *terminating protocols that are correct with high probability*. This is crucial not only because it allows to improve eventual stabilization to eventual termination but, most importantly, because it allows to develop terminating subroutines that can be sequentially composed to form larger modular protocols. Such protocols are more efficient, more natural, and more amenable to clear proofs of correctness, compared to existing protocols that are based on composing all subroutines in parallel and "sequentializing" them eventually by perpetual reinitializations. To the best of our knowledge, [13] is the only work that has considered this issue but with totally different and more deterministic assumptions. Several other papers [1, 2, 12] have already exploited a uniform random interaction model, but in all cases for analyzing the expected time to convergence of stabilizing protocols and not for maximizing the correctness probability of terminating protocols, as we do here.

Our model for shape construction is strongly inspired by the Population Protocol model [1] and the Mediated Population Protocol model [10]. For introductory texts on population protocols see [3, 11]. For further literature related to the present work, consult the full paper and [12].

Section 2 formally defines the model under consideration and brings together all definitions and basic facts that are used throughout the paper. Section 3 introduces our technique for counting the size n of the system with high probability. The main result of that section (i.e. Theorem 1) is of particular importance as it underlies all sequential composition arguments that follow in the paper. In particular, the protocol of Section 3.1 is used as a subroutine in our *universal constructors*, establishing that *it is possible to construct with high probability arbitrarily complex shapes (and patterns) by terminating protocols*. These *universality* results are discussed in Section 4. Finally, in Section 5 we conclude and give further research directions that are opened by our work. In the full paper we also provide direct constructors for some basic shape construction problems and study the problem of shape self-replication.

2. THE MODEL

The system consists of a population V of n distributed *processes* (finite-state machines), called *nodes* throughout. Every node has a bounded number of ports which it uses to interact with other nodes. In the 2-dimensional (2D) case, there are four ports p_y, p_x, p_{-y}, and p_{-x}, which for notational convenience are usually denoted u, r, d, and l, respectively (for *up*, *right*, *down*, and *left*, respectively). Neighboring ports are perpendicular to each other, forming local axes; that is $u \perp r$, $r \perp d$, $d \perp l$, and $l \perp u$. Similar assumptions hold for the 3D case. An important remark is that the above coordinates are only for local purposes and do not necessarily represent the actual orientation of a node in the system. Nodes may interact in pairs, whenever a port of one node w is at unit distance and in straight line (w.r.t. to the local axes) from a port of another node v.

DEFINITION 1. *A 2D protocol is 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* $\delta : (Q \times P) \times (Q \times P) \times \{0,1\} \to Q \times Q \times \{0,1\}$ *is the* transition function*, where* $P = \{u, r, d, l\}$ *is the set of* ports*. When required, also a special* initial leader-state $L_0 \in Q$ *may be defined.*

If $\delta((a, p_1), (b, p_2), c) = (a', b', c')$, we call $(a, p_1), (b, p_2), c \to (a', b', c')$ a *transition* (or *rule*). A transition $(a, p_1), (b, p_2), c \to (a', b', c')$ is called *effective* if $x \neq x'$ for at least one $x \in \{a, b, c\}$ and *ineffective* otherwise.

Let $E = \{(v_1, p_1)(v_2, p_2) : v_1 \neq v_2 \in V \text{ and } p_1, p_2 \in P\}$ be the set of all unordered pairs of node-ports (cf. [12] for more details on unordered interactions). A *configuration* C is a pair (C_V, C_E), where $C_V : V \to Q$ specifies the state of each node and $C_E : E \to \{0,1\}$ specifies the state of every possible pair of node-ports (or edge). In particular, an edge in state 0 is called *inactive* and an edge in state 1 is called *active*. The initial configuration is always the one in which all nodes are in state q_0 (apart possibly from a unique leader in state L_0) and all edges are *inactive*. Execution of the protocol proceeds in discrete steps. In every step, a pair of node-ports $(v_1, p_1)(v_2, p_2)$ is selected by an *adversary scheduler* and these nodes interact via the corresponding ports and update their states and the state of the edge joining them according to the transition function δ.

Every configuration C defines a *set of shapes* $G[A(C)]$, where $A(C) = C_E^{-1}[1]$; i.e. the network induced by the active edges of C. Observe that not all possible $A(C)$ are valid given our geometrical restrictions, that connections are made at unit distance and are perpendicular whenever they correspond to consecutive ports of a node. For example, if $(v_1, r)(v_2, l) \in A(C)$ then $(v_1, l)(v_2, r) \notin A(C)$. In general, $A(C)$ is *valid* if any connected component defined by it (when arranged according to the geometrical constraints) is a subnetwork of the *2D grid network with unit distances*. A valid $A(C_{t-1})$ also restricts the possible selections of the scheduler at step $t \geq 1$. In particular, $(v_1, p_1)(v_2, p_2) \in E$ can be selected for interaction (or *is permitted*) at step t iff $A(C_{t-1}) \cup \{(v_1, p_1)(v_2, p_2)\}$ is valid. Observe that any edge that is active before step t is trivially permitted at step t. From now on, we call a 2D (3D) *shape* any connected subnetwork of the 2D (3D) grid network with unit distances.

Throughout the paper we restrict attention to configurations C in which $A(C)$ is valid. We write $C \to C'$ if C' *is reachable in one step from* C (meaning via a single interaction that is permitted on C). 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 the initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. We only consider *fair* executions, so we require that 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 most cases, we assume that interactions are chosen by a *uniform random scheduler* which in every step selects independently

and uniformly at random one of the permitted interactions. The uniform random scheduler is fair with probability 1. In this work, with high probability (abbreviated w.h.p.) means with probability at least $1 - 1/n^c$ for some constant $c \geq 1$.

We define the *output of a configuration* C as the set of shapes $G_{out}(C) = (V_s, E_s)$ where $V_s = \{u \in V : C_V(u) \in Q_{out}\}$ and $E_s = A(C) \cap \{(v_1, p_1)(v_2, p_2) : v_1 \neq v_2 \in V_s \text{ and } p_1, p_2 \in P\}$. In words, the output shapes of a configuration consist of those nodes that are in output states and those edges between them that are active. Throughout this work, we are interested in obtaining a single shape as the final output of the protocol. As already mentioned, our main focus will be on terminating protocols. In this case, we assume that $Q_{out} \subseteq Q_{halt} \subseteq Q$, where, for all $q_{halt} \in Q_{halt}$, every rule containing q_{halt} is ineffective.

DEFINITION 2. *We say that an execution of a protocol on n processes* constructs (stably constructs) a shape G, *if it terminates (stabilizes, resp.) with output G.*

Every 2D shape G has a unique minimum 2D rectangle R_G enclosing it. R_G is a shape with its nodes labeled from $\{0, 1\}$. The nodes of G are labeled 1, the nodes in $V(R_G) \backslash V(G)$ are labeled 0, and all edges are active. It is like filling G with additional nodes and edges to make it a rectangle. In fact, R_G can also be constructed by a protocol, given G (see the full paper). The dimensions of R_G are defined by h_G, which is the horizontal distance between a leftmost node and a rightmost node of the shape (x-dimension), and v_G, which is the vertical distance between a highest and a lowest node of the shape (y-dimension). Let also $max\_dim_G := \max\{h_G, v_G\}$ and $min\_dim_G := \min\{h_G, v_G\}$. Then R_G can be extended by $max\_dim_G - min\_dim_G$ extra rows or columns, depending on which of its dimensions is smaller, to yield a $max\_dim_G \times max\_dim_G$ square S_G enclosing G (we mean here a $\{0, 1\}$-node-labeled square, as above, in which G can be identified). Observe, that such a square is not unique. For example, if G is a horizontal line of length d (i.e. $h_G = d$ and $v_G = 1$) then it is already equal to R_G and has to be extended by $d - 1$ rows to become S_G. These rows can be placed in d distinct ways relative to G, but all these squares have the same size $max\_dim_G \times max\_dim_G$ denoted by $|S_G|$.

A 2D (3D) *shape language* $\mathcal{L}$ is a subset of the set of all possible 2D (3D) shapes. We restrict our attention here to shape languages that contain a unique shape for each possible maximum dimension of the shape. In this case, it is equivalent, and more convenient, to translate $\mathcal{L}$ to a language of labeled squares. In particular, we define in this work a *shape language* $\mathcal{L}$ by providing for every $d \geq 1$ a single $d \times d$ square with its nodes labeled from $\{0, 1\}$. Such a square may also be defined by a d^2-sequence $S_d = (s_0, s_1, \ldots, s_{d^2-1})$ of bits or *pixels*, where $s_j \in \{0, 1\}$ corresponds to the jth node as follows: We assume that the pixels are indexed in a "zig-zag" fashion, beginning from the bottom left corner of the square, moving to the right until the bottom right corner is encountered, then one step up, then to the left until the node above the bottom left corner is encountered, then one step up again, then right, and so on. The shape G_d defined by S_d, called *the shape of* S_d, is the one induced by the nodes of the square that are labeled 1 and throughout this work we assume that $max\_dim_{G_d} = d$.

For simulation purposes, we also need to introduce appropriate shape-constructing Turing Machines (TMs). We now describe such a TM M: M's goal is to construct a shape on the pixels of a $\sqrt{n} \times \sqrt{n}$ square, which are indexed in the zig-zag way described above. M takes as input an integer $i \in \{0, 1, \ldots, n-1\}$ and the size n or the dimension $\sqrt{n}$ of the square (all in binary) and decides whether pixel i should belong or not to the final shape, i.e. if it should be *on* or *off*, respectively. [1] Moreover, in accordance to our definition of a shape, the construction of the TM, consisting of the pixels that M accepts (as *on*) and the active connections between them, should be *connected* (i.e. it should be a single shape).

DEFINITION 3. *We say that a shape language $\mathcal{L} = (S_1, S_2, S_3, \ldots)$ is* TM-computable *(or* TM-constructible*) in space $f(d)$, if there exists a TM M (as defined above) such that, for every $d \geq 1$, when M is executed on the pixels of a $d \times d$ square results in S_d (in particular, on input (i, d), where $0 \leq i \leq d^2 - 1$, M gives output $S_d[i]$), by using space $O(f(d))$ in every execution.*

DEFINITION 4. *We say that a protocol $\mathcal{A}$* constructs a shape language $\mathcal{L}$ 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 shape $G \in \mathcal{L}$* [2] *of order at least $g(n)$ (provided that such a G exists) and, additionally, (ii) for all $G \in \mathcal{L}$ 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 $\mathcal{L}$ with waste $n - g(n)$.

We now give as an illustrating example, a stabilizing protocol for the problem of constructing a spanning square. More protocols are given in the full paper. We assume, for simplicity, that $\sqrt{n}$ is integer and that there is a preelected unique leader in state L_u and all other nodes are initially in state q_0. The effective rules are $(L_u, u), (q_0, d), 0 \to (q_1, L_r, 1)$, $(L_r, r), (q_0, l), 0 \to (q_1, L_d, 1)$, $(L_d, d), (q_0, u), 0 \to (q_1, L_l, 1)$, $(L_l, l), (q_0, r), 0 \to (q_1, L_u, 1)$, $(L_u, u), (q_1, d), 0 \to (L_l, q_1, 1)$, $(L_r, r), (q_1, l), 0 \to (L_u, q_1, 1)$, $(L_d, d), (q_1, u), 0 \to (L_r, q_1, 1)$, and $(L_l, l), (q_1, r), 0 \to (L_d, q_1, 1)$.

The protocol first constructs a 2×2 square. When it is done, the leader is at the bottom-right corner and is in state L_d. In general, whenever the leader is at the left (the up, right, and down cases are symmetric) of the already constructed square it tries to move right in order to walk above the square. If it does not succeed, it is because it has not yet passed over the upper boundary, so it activates the edge to the right, takes another step up and then tries again to move right. In this way, the leader always grows the square perimetrically and clockwise.

3. PROBABILISTIC COUNTING

In this section, we consider the problem of counting n. In particular, we assume a uniform random scheduler and we want to give protocols that always terminate but still w.h.p. count n correctly (or a satisfactory upper bound on n). The importance of such protocols is further supported by the fact that we cannot guarantee anything much better than this. In particular, if we require a population protocol

[1] Note that if the TM is not provided with the square size together with the pixel, then it can only compute uniform/symmetric shapes that are independent of n.

[2] G is the shape of a labeled square $S \in \mathcal{L}$ in case $\mathcal{L}$ is defined in terms of such squares.

to always terminate and additionally to always be correct, then we immediately obtain an impossibility result.

In Section 3.1, we present a protocol with a unique leader that solves w.h.p. the counting problem and always terminates. To the best of our knowledge, this is the first protocol of this sort in the relevant literature. Additionally, this protocol is crucial because all of our generic constructors, that are developed in Section 4, are terminating by assuming knowledge of n (stored distributedly on a line of length $\log n$). They obtain access to this knowledge w.h.p. by executing the counting protocol as a subroutine. Finally, knowing n w.h.p. allows to develop protocols that exploit sequential composition of (terminating) subroutines, which makes them much more natural and easy to describe than the protocols in which all subroutines are executed in parallel and perpetual reinitializations is the only means of guaranteeing eventual correctness (the latter is the case e.g. in [8, 10, 12], but not in [13] which was the first extension to allow for sequential composition based on some non-probabilistic assumptions). In Section 3.2 we establish that if the nodes have unique ids (UIDs) then it is possible to solve the problem without a unique leader.

3.1 Fast Probabilistic Counting With a Leader

Keep in mind that in order to simplify the discussion, a sort of population protocol is presented here. So, there are no ports, no geometry, and no activations/deactivations of connections. In every step, a uniform random scheduler selects equiprobably one of the $n(n-1)/2$ possible node pairs, and the selected nodes interact and update their states according to the transition function. The only difference from the classical population protocols is that a distinguished leader node has unbounded local memory (of the order of n). Later, in Section 4.1, we will adjust the protocol to make it work in our model (thus also dropping this unbounded local memory assumption).

Counting-Upper-Bound Protocol: There is initially a unique leader l and all other nodes are in state q_0. Assume that l has two n-counters in its memory, initially both set to 0. So, the state of l is denoted as $l(r_0, r_1)$, where r_0 is the value of the first counter and r_1 the value of the second counter, $0 \leq r_0, r_1 \leq n$. The rules of the protocol are $(l(r_0, r_1), q_0) \to (l(r_0+1, r_1), q_1)$, $(l(r_0, r_1), q_1) \to (l(r_0, r_1+1), q_2)$, and $(l(r_0, r_1), \cdot) \to (halt, \cdot)$ if $r_0 = r_1$.

Observe that r_0 counts the number of q_0s in the population while r_1 counts the number of q_1s. Initially, there are $n-1$ q_0s and no q_1s. Whenever l interacts with a q_0, r_0 increases by 1 and the q_0 is converted to q_1. Whenever l interacts with a q_1, r_1 increases by 1 and the q_1 is converted to q_2. The process terminates when $r_0 = r_1$ for the first time. We also give to r_0 an initial head start of b, where b can be any desired constant. So, initially we have $r_0 = b$, $r_1 = 0$ and $i = \#q_0 = n-b-1$, $j = \#q_1 = b$ (this can be easily achieved by the protocol). So, we have two competing processes, one counting q_0s and the other counting q_1s, the first one begins with an initial head start of b and the game ends when the second catches up the first. We now prove that when this occurs the leader will almost surely have already counted at least half of the nodes.

THEOREM 1. *The above protocol halts in every execution. Moreover, if the scheduler is a uniform random one, when this occurs, w.h.p. it holds that* $r_0 \geq n/2$.

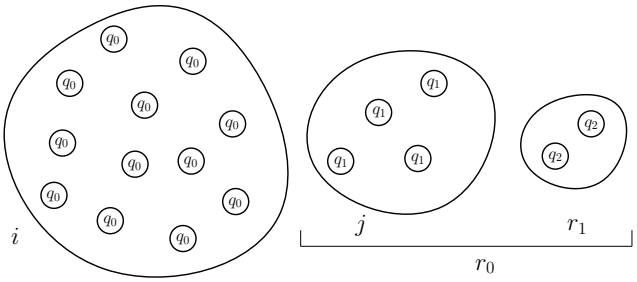

Figure 1: A configuration of the system (excluding the leader).

PROOF. The random variable i denotes the number of q_0s and j denotes the number of q_1s in the configuration, where initially $i = n-b-1$ and $j = b$. Observe also that all the following hold: $j = r_0 - r_1$, $r_0 \geq r_1$, $r_1 = (n-1) - (i+j)$, and $r_0 + r_1$ is equal to the number of effective interactions (see Figure 1).

Given that we have an effective interaction, the probability that it is an (l, q_0) is $p_{ij} = i/(i+j)$ and the probability that it is an (l, q_1) is $q_{ij} = 1 - p_{ij} = j/(i+j)$. This random process may be viewed as a random walk (r.w.) on a line with $n+1$ positions $0, 1, \ldots, n$ where a particle begins from position b and there is an absorbing barrier at 0 and a reflecting barrier at n. The position of the particle corresponds to the difference $r_0 - r_1$ of the two counters which is equal to j. Observe now that if $j \geq n/2$ then $r_0 - r_1 \geq n/2 \Rightarrow r_0 \geq n/2$, so it suffices to consider a second absorbing barrier at $n/2$. The particle moves forward (i.e. to the right) with probability p_{ij} and backward with probability q_{ij}. This is a "difficult" r.w. because the transition probabilities not only depend on the position j but also on the sum $i+j$ which decreases in time. In particular, the sum decreases whenever an (l, q_1) interaction occurs, in which case a q_1 becomes q_2. Observe also that, in our case, the duration of the r.w. can be at most $n-b$, in the sense that if the particle has not been absorbed after $n-b$ steps then we have success. The reason for this is that $n-b$ effective interactions imply that $r_0 + r_1 = n$, but as $r_0 \geq r_1$, we have $r_0 \geq n/2$. In fact, $r_0 \geq n/2 \Leftrightarrow j + r_1 \geq n/2$. We are interested in upper bounding P[failure] = P[reach 0 before $r_0 \geq n/2$ holds], which is in turn upper bounded by the probability of reaching 0 before reaching $n/2$ and before $n-b$ effective interactions have occurred (this is true because, in the latter event, we have disregarded some winning conditions). It suffices to bound the probability of reaching 0 before n effective interactions have occurred. Thus, we have $r_0 + r_1 \leq n$ but $r_1 \leq r_0 \Rightarrow 2r_1 \leq r_0 + r_1$, thus $2r_1 \leq n \Rightarrow r_1 \leq n/2 \Rightarrow (n-1) - (i+j) \leq n/2 \Rightarrow i + j \geq (n/2) - 1 = n'$. And if we set $n' = (n/2) - 1$ we have $i + j \geq n'$. Moreover, observe that when $r_0 + r_1 = n+1$ we have $n+1 = r_0 + r_1 \leq 2r_0 \Rightarrow r_0 \geq n/2$. In summary, during the first n effective interactions, it holds that $i + j \geq n' = (n/2) - 1$ and when interaction $n+1$ occurs it holds that $r_0 \geq n/2$, that is, if the process is still alive after time n, then r_0 has managed to count up to $n/2$ and the protocol has succeeded.

Now, $i + j \geq n'$ implies that $p_j \geq (n' - j)/n'$ and $q_j \leq j/n'$ so that now the probabilities only depend on the position j. This is the well-studied Ehrenfest r.w. coming from the theory of brownian motion [5, 9]. We will reduce this walk

to one in which the probabilities do not depend on j by restricting it to the prefix $[0, b]$ of the line. In this part, it holds that $j \leq b$ which implies that $p \geq (n' - b)/n'$ and $q \leq b/n'$. Now we set $p = (n' - b)/n'$ and $q = b/n'$. Observe that this may only increase the probability of failure. Recall that initially the particle is on position b. Imagine now an absorbing barrier at 0 and another one at b. Whenever the r.w. is on $b-1$ it will either return to b before reaching 0 or it will reach 0 (and fail) before returning to b. Due to symmetry it is equivalent to assume that the particle begins from position 1, moves forward with probability $p' = q$, backward with probability $q' = p$, and it fails at b. Thus, it is equivalent to bound P[reach b before 0 (when beginning from 1)], i.e. the probability of winning in the classical ruin problem analyzed e.g. in [6] page 345. If we set $x = q'/p' = p/q = (n' - b)/b$ we have that: P[reach b before 0] $= 1 - \frac{x^b - x}{x^b - 1} = \frac{x-1}{x^b-1} \leq \frac{x}{x^b-1} \approx \frac{1}{x^{b-1}} \approx \frac{1}{n^{b-1}}$. Thus, whenever the original walk is on $b-1$, the probability of reaching 0 before reaching b again, is at most $1/n^{b-1}$. Now assume that we repeat the above walk n times, i.e. we place the particle on $b-1$, play the game, then if it returns to b we put again the particle on $b-1$ and play the game again, and so on. From Boole-Bonferroni inequality, we have that:
P[fail at least once] $\leq \sum_{m=1}^{n}$ P[fail at repetition m] $\leq \sum_{m=1}^{n} \frac{1}{n^{b-1}} = \frac{n}{n^{b-1}} = \frac{1}{n^{b-2}}$. □

REMARK 1. *For the Counting-Upper-Bound protocol to terminate, it suffices for the leader to meet every other node twice. This takes twice the expected time of a* meet everybody *(cf. [12]), thus the expected running time of Counting-Upper-Bound is $O(n^2 \log n)$ (interactions).*

REMARK 2. *When the Counting-Upper-Bound protocol terminates, w.h.p. the leader knows an r_0 which is between $n/2$ and n. So any subsequent routine can use directly this estimation and pay in an* a priori *waste which is at most half of the population. In practice, this estimation is expected to be much closer to n than to $n/2$ (in all of our experiments for up to 1000 nodes, the estimation was always close to $(9/10)n$ and usually higher). On the other hand, if we want to determine the exact value of n and to have no* a priori *waste, then we can have the leader wait an additional large polynomial (in r_0) number of steps, to ensure that the leader has met every other node w.h.p..*

In the full paper, we also give some evidence that the unique leader assumption *is probably necessary*, but leave it as an interesting open question. In contrast, we show in the next section that it can be dropped if UIDs are available.

3.2 Counting Without a Leader but With UIDs

Now nodes have unique ids from a universe $\mathcal{U}$. Nodes initially do not know the ids of other nodes nor n. The goal is again to count n w.h.p.. All nodes execute the same program and no node can initially act as unique leader, because nodes do not know which ids from $\mathcal{U}$ are actually present in the system. Nodes have unbounded memory but we try to minimize it, e.g. if possible store only up to a constant number of other nodes' ids. We show that under these assumptions, the counting problem can be solved without the necessity of a unique leader.

The idea of the protocol is to have the node with the maximum id in the system to perform the same process as the unique leader in the protocol with no ids of Theorem 1. Of course, initially all nodes have to behave as if they were the maximum (as they do not know in advance who the maximum is) and we must also guarantee that no other node ever terminates (with sufficiently large probability) early, giving as output a wrong count.

Informal description: Every node u has a unique id id_u and tries to simulate the behavior of the unique leader of the protocol of Theorem 1. In particular, whenever it meets another node for the first time it wants to mark it once and the second time it meets that node it wants to mark it twice, recording the number of first-meetings and second-meetings in two local counters. The problem is that now many nodes may want to mark the same node. One idea, of course, could be to have a node remember all the nodes that have marked it so far but we want to avoid this because it requires a lot of memory and communication. Instead, we allow a node to only remember a single other node's id at a time. Every node tries initially to increase its first-meetings counter to b so that it creates an initial b head start of this counter w.r.t. the other. Every node that succeeds, starts executing its main process. The main idea is that whenever a node u interacts with another node that either has or has been marked by an id greater than id_u, u becomes sleeping and stops counting. This guarantees that only u_{max} will forever remain awake. Moreover, every node u always remembers the maximum id that has marked it so far, so that the probabilistic counting process of a node u can only be affected by nodes with id greater than id_u and, as a result, no one can affect the counting process of u_{max}. The formal protocol and the proof that it correctly simulates the counting process of Theorem 1, thus providing w.h.p. an upper bound on n, can be found in the full paper.

4. GENERIC CONSTRUCTORS

In this section, we give a characterization for the class of constructible 2D shape languages. In particular, we establish that shape constructing TMs (defined in Section 2), can be simulated by our model and therefore we can realize their output-shape in the actual distributed system. To this end, we begin in Section 4.1 by adapting the Counting-Upper-Bound protocol of Section 3.1 to work in our model. The result is w.h.p. a line of length $\Theta(\log n)$, containing n in binary, and having a unique leader. Then, in Section 4.2 the leader exploits the knowledge of n to construct a $\sqrt{n} \times \sqrt{n}$ square. In the sequel (Section 4.3), it simulates the TM on the square n distinct times, one for each pixel of the square. Each time, the input provided to the TM is the index of the pixel and $\sqrt{n}$, both in binary. Each simulation decides whether the corresponding pixel should be *on* or *off*. When all simulations have completed, the leader releases in the solution the connected shape consisting of the *on* pixels and the active edges between them. The connections of all other (*off*) pixels become deactivated and the corresponding nodes become free (isolated) nodes in the solution.

4.1 Storing the Count on a Line

We begin by adapting the Counting-Upper-Bound protocol of Theorem 1 so that when the protocol terminates the final correct count is stored distributedly in binary on an active line of length $\log n$.

Counting-on-a-Line Protocol: The probabilistic process that is being executed is essentially the same as that of

the Counting-Upper-Bound protocol. Again the protocol assumes a unique leader that forever controls the process. A difference now is that every node has four ports (in the 2D case). The leader operates as a TM that stores the r_0 and r_1 counters in its tape in binary. The ith cell of the tape has two components, one storing the ith bit of r_0 and the other storing the ith bit of r_1. We say that the tape is *full*, if the bits of all r_0 components of the tape are set to 1. The tape of the TM is the active line that the leader has formed so far, each node in the line implementing one cell of the tape. Initially the tape consists of a single cell, stored in the memory of the unique leader node. As in Counting-Upper-Bound, the leader first tries to obtain an initial advantage of b for the r_0 counter. To achieve the advantage, the leader does not count the q_1s that it interacts with until it holds that $r_0 \geq b$. Observe that the initial length of the tape is not sufficient for storing the binary representation of b (of course b is constant so, in principle, it could be stored on a single node, however we prefer to keep the description as uniform as possible). In order to resolve this, the leader does the following. Whenever it meets the left port of a q_0 from its right port, if its tape is not full yet, it switches the q_0 to q_1, leaving it free to move in the solution, and increases the r_0 counter by one. To increase the counter, it freezes the probabilistic process (that is, during freezing it ignores all interactions with free nodes), and starts moving on its tape, which is a distributed line attached to its left port. After incrementing the counter, the leader keeps track of whether the tape is now full and then it moves back to the right endpoint of the line to unfreeze and continue the probabilistic process. On the other hand, if the tape is full, it binds the encountered q_0 to its right by activating the connection between them (thus increasing the length of the tape by one), then it reorganizes the tape, it again increases r_0 by one, and finally moves back to the right endpoint to continue the probabilistic process. This time, the leader also records that it has bound a q_0 that should have been converted to q_1. This *debt* is also stored on the tape in another counter r_2. Whenever the leader meets a q_2, if $r_2 \geq 1$, it converts q_2 to q_1 and decreases r_2 by one. So, q_2s may be viewed as a *deposit* that is used to pay back the debt. In this manner, the q_0s that are used to form the tape of the leader are not immediately converted to q_1 when first counted. Instead, the missing q_1s are introduced at a later time, one after every interaction of the leader with a q_2, and all of them will be introduced eventually, when a sufficient number of q_2s will become available. Finally, whenever the leader interacts with the left port of a q_1 from its right port, it freezes, increases the r_1 counter by one (observe that $r_0 \geq r_1$ always holds, so the length of the tape is always sufficient for r_1 increments), and checks whether $r_0 = r_1$. If equality holds, the leader terminates, otherwise it moves back to the right endpoint and continues the process. Correctness is captured by the following lemma.

LEMMA 1. *Counting-on-a-Line protocol terminates in every execution. Moreover, when the leader terminates, w.h.p. it has formed an active line of length* $\log n$ *containing* n *in binary in the* r_0 *components of the nodes of the line (each node storing one bit).*

PROOF. We begin by showing that the probabilistic process of the Counting-Upper-Bound protocol is not negatively affected in the Counting-on-a-Line protocol. This implies that the high probability argument of Theorem 1 holds also for Counting-on-a-Line. First, observe that the four ports of the nodes introduce more choices for the scheduler in every step. However, these new choices, if treated uniformly, result in the same multiplicative factor for both the "positive" (an (l, q_0) interaction) and the "negative" (an (l, q_1) interaction) events, so the probabilities of the process are not affected at all by this. Moreover, neither the debt affects the process. The reason is that the only essential difference w.r.t. to the process is that the conversion of some counted q_0s to the corresponding q_1s is delayed. But this only decreases the probability of early termination and thus of failure. It remains to show that not even a single q_1 remains forever as debt, because, otherwise, some executions of the protocol would not terminate. The reason is that the protocol cannot terminate before converting all the q_1s plus the debt to q_2. To this end, observe that the line of the leader has always length $\lfloor \lg r_0 \rfloor + 1$, thus $r_2 \leq \lfloor \lg r_0 \rfloor$, because the debt is always at most the length of the line excluding the initial leader. So, at least $r_0 - \lfloor \lg r_0 \rfloor$ nodes have been successfully converted from q_0 to q_1 which implies that there is an eventual deposit of at least $r_0 - \lfloor \lg r_0 \rfloor$ nodes in state q_2. These q_2s are not immediately available, but they will become available in the future, because every interaction of the leader with a q_1 results in a q_2. Finally, observe that $r_0 - \lfloor \lg r_0 \rfloor \geq \lfloor \lg r_0 \rfloor$ holds for all $r_0 \geq 1$. Thus, $r_0 - \lfloor \lg r_0 \rfloor \geq r_2$, which means that the eventual deposit is not smaller than the debt, so the protocol eventually pays back its debt and terminates. □

4.2 Constructing a $\sqrt{n} \times \sqrt{n}$ Square

We now show how to organize the nodes into a spanning square, i.e. a $\sqrt{n} \times \sqrt{n}$ one. As before, we assume that $\sqrt{n}$ is integer and that the leader has n stored in its line (w.h.p.) after executing the Counting-on-a-Line subroutine. Observe that knowledge of n allows the protocol to terminate after constructing the square and to know that the square has been successfully constructed.

Square-Knowing-n Protocol: The initial leader L first computes $\sqrt{n}$ on its line by any plausible algorithm and then expands its line to the right by attaching free nodes to make its length $\sqrt{n}$. Then it exploits the down ports to create a replica of its line. The replica has also length $\sqrt{n}$ and has its own leader but in a distinguished state L_s. This new line plays the role of a *seed* that starts creating other self-replicating lines of length $\sqrt{n}$. In particular, the seed attaches free nodes to its down ports, until all positions below the line are filled by nodes and additionally all horizontal connections between those nodes are activated. Then it introduces a leader L_r to one endpoint of the replica and starts deactivating the vertical connections to release the new line of length $\sqrt{n}$. These lines with L_r leaders are *totally self-replicating*, meaning that their children also begin in state L_r. The initial leader L waits until the up ports of a non-seed replica r become totally aligned with the down ports of the square segment that has been constructed so far. So, initially it waits until a replica becomes attached to the lower side of its own line. When this occurs, it activates all intermediate vertical connections to make the construction rigid, increments a row-counter by one (initially 0), and moves to the new lowest row. If at the time of attachment r was in the middle of an incomplete replication, then there

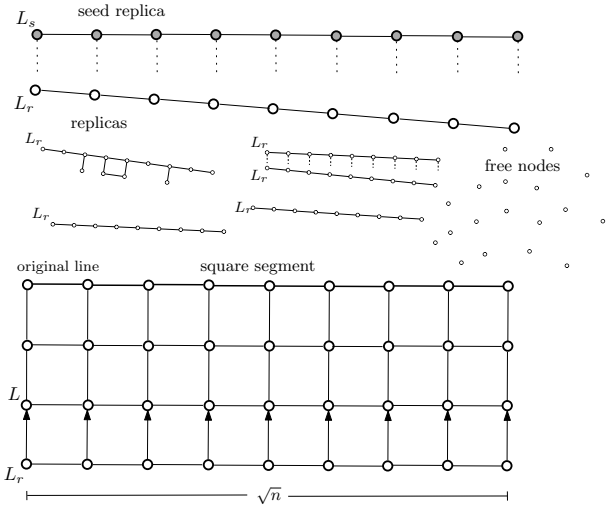

Figure 2: The seed at the top has created another replica which has just been released in the solution. At the bottom appears the square segment that has been constructed so far. A replica has just arrived (from below) and will be attached to the segment.

will be nodes attached to the down ports of r. L releases all these nodes, by deactivating the active connections of r to them, and then waits for another non-seed replica to arrive. When the row-counter becomes equal to $\sqrt{n}-1$, the leader for the first time accepts the attachment of the seed to its construction and when the seed is successfully attached the leader terminates. This completes the construction of the $\sqrt{n} \times \sqrt{n}$ square. See Figure 2 for an illustration.

The reason for attaching the seed last, and in particular when no further free nodes have remained, is that otherwise self-replication could possibly cease in some executions. Observe also that we have allowed the L-leader to accept the attachment of a replica to the square segment even though the replica may be in the middle of an incomplete replication. This is important in order to avoid reaching a point at which some free lines are in the middle of incomplete replications but there are no further free nodes for any of them to complete. For a simple example, consider the seed and a replica r and $\sqrt{n}$ free nodes (all other nodes have been attached to the square segment). It is possible that $\sqrt{n}-1$ of the free nodes become attached to the seed and the last free node becomes attached to r. We have overcome this deadlock by allowing L to accept the attachment of r to the square segment. When this occurs, the free node will be released and eventually it will be attached to the last free position below the seed.

We now give, in Protocol 1, one of the possible codes for replicating the lines. We present a version that does not use the line's leader in order to successfully self-replicate. A direct replication controlled by the leader is simpler to conceive but its code is lengthy (it can be found in the full paper). Without loss of generality, we assume that one line has e on both of its endpoints and i on the internal nodes, and every (free) node is in state q_0. The protocol works as follows. Free nodes are attached below the nodes of the original line. When a node is attached below an internal node i, both become i_1 and when a node is attached below an endpoint e, both become e_1. Moreover, adjacent nodes of the replica connect to each other and every such connection increases their index, so that their index counts their degree. An internal node of the replica can detach from the original line only when it has degree 3, that is when, apart from its vertical connection, it has also already become connected to both a left and a right neighbor on the replica. On the other hand, an endpoint detaches when it has a single internal neighbor. It follows that the replica can only detach when its length (counted in number of horizontal active connections) is equal to that of the original line. To see this, assume that a shorter line detaches at some point. Clearly, such a line must have at least one endpoint that corresponds to an internal node i_j of the replica. But this node is an endpoint of the shorter line, so its degree is less than 3, i.e. $j < 3$, and we conclude that it cannot have detached. To make the protocol also replicate the leader-state simply replace one e with L, L_s, or L_r and map it to L_s, L_r, L_r, resp., in the replica, when detaching.

Protocol 1 *No-Leader-Line-Replication*

$Q = \{q_0, e, e_1, i, i_1, i_2, i_3\}$
δ:

$$(i,d),(q_0,u),0 \to (i_1,i_1,1)$$
$$(e,d),(q_0,u),0 \to (e_1,e_1,1)$$
$$(i_j,r),(i_k,l),0 \to (i_{j+1},i_{k+1},1) \text{ for all } j,k \in \{1,2\}$$
$$(i_1,r),(e_1,l),0 \to (i_2,e_2,1)$$
$$(i_2,r),(e_1,l),0 \to (i_3,e_2,1)$$
$$(e_1,r),(i_1,l),0 \to (e_2,i_2,1)$$
$$(e_1,r),(i_2,l),0 \to (e_2,i_3,1)$$
$$(i_3,u),(i_1,d),1 \to (i,i,0)$$
$$(e_2,u),(e_1,d),1 \to (e,e,0)$$

LEMMA 2. *There is a protocol (described above) that when executed on n nodes (for all n with integer $\sqrt{n}$) w.h.p. constructs a $\sqrt{n} \times \sqrt{n}$ square and terminates.*

4.3 Simulating a TM

We now assume as given (from the discussion of the previous section) a $\sqrt{n} \times \sqrt{n}$ square with a unique leader L at the bottom left corner. However, keep in mind that, in principle, the simulation described here can begin before the construction of the $\sqrt{n} \times \sqrt{n}$ square is complete. The only difference in this case, is that the two processes are executed in parallel and if at some point the TM needs more space, it has to wait until it becomes available. The square may be viewed as a TM-tape of length n traversed by the leader in a "zig-zag" fashion, first moving to the right until the bottom right corner is encountered, then one step up, then to the left until the node above the bottom left corner is encountered, then one step up again, then right, and so on. To simplify this process, we may assume that a preprocessing has marked appropriately the turning points. The tape will be used to simulate a TM M of the form described in the Section 2. The n pixels of the square are numbered according to the above zig-zag process beginning from the bottom left node, each node corresponding to one pixel. The space available to the TM is exponential in the binary representation of the input (i,n) (or $(i,\sqrt{n})$), because

$i \leq n-1$ and therefore the length of its binary representation $|i| = O(\log n)$, thus $|(i,n)| = O(\log n)$, but the available space is $\Theta(n) = \Theta(2^{\log n}) = \Omega(2^{|(i,n)|})$ (still it is linear in the size of the whole shape to be constructed).

The protocol invokes n distinct simulations of M, one for each of the pixels $i \in \{0, 1, \ldots, n-1\}$, beginning from $i = 0$ and every time incrementing i by one. The leader maintains the current value of i in binary, in a pixel-counter *pixel* stored in the $O(\log n)$ leftmost cells of the tape. Recall that the leader knows n from the procedures of the previous sections. So, we may assume that the tape also holds in advance n and $\sqrt{n}$ in binary (again in the leftmost cells). Initially $pixel = 0$ and the leader marks the 0th node, that is the bottom left corner of the square. Then it simulates M on input $(pixel, \sqrt{n})$. When M decides, if its decision is *accept*, the leader marks the node corresponding to *pixel* as *on*, otherwise it marks it as *off*. Then the leader increments *pixel* by one, marks the node corresponding to the new value of *pixel* (which is the next node on the tape), clears the tape from residues of the previous simulation, invokes another simulation of M on the new value of *pixel*, and marks the corresponding node as *on* or *off* according to M's decision. The process stops when $pixel = n$, in which case no further simulation is executed. When this occurs, the leader starts walking the tape in the opposite direction until it reaches the bottom left corner. In the way, it deactivates all connections involving at least one *off* node, leaving active only the connected 2D shape consisting of the *on* nodes.

The following theorem states the lower bound implied by the construction described in this section.

THEOREM 2. *Let $\mathcal{L} = (S_1, S_2, \ldots)$ be a connected 2D shape language, such that $\mathcal{L}$ is TM-computable in space d^2. Then there is a protocol (described above) that w.h.p. constructs $\mathcal{L}$. In particular, for all $d \geq 1$, whenever the protocol is executed on a population of size $n = d^2$, w.h.p. it constructs S_d and terminates. In the worst case, when G_d (that is, the shape of S_d) is a line of length d, the waste is $(d-1)d = O(d^2) = O(n)$.*

REMARK 3. *If the system designer knew n in advance, then he/she could preprogram the nodes to simulate a TM that constructs a specific shape of size n, e.g. the TM corresponding to the Kolmogorov complexity of the shape. In this work, n* is not known in advance*, so we had to preprogram the nodes with a TM that can work for all n.*

REMARK 4. *The above results can be immediately modified to construct* patterns *instead of shapes. The idea is to keep the same constructor as above and simulate TMs that for every pixel output a color from a set of colors $\mathcal{C}$.*

REMARK 5. *In all the above constructions the unique leader assumption can be dropped in the price of sacrificing termination. In this case, the constructions become stabilizing by the reinitialization technique, as e.g. in [12], but should be carefully rewritten.*

4.4 Parallelizing the Simulations

We now present an approach for parallelizing the simulations. Assume that there is a TM M deciding each pixel in space k, that k and d are computable in space $O(n)$, and that $n = k \cdot d^2$.

The leader, instead of constructing a square, constructs now a spanning line of length d^2, say in the x dimension, corresponding to a linear expansion of the pixels of a $d \times d$ square. Moreover, the leader creates a seed of length $k-1$ and uses it to partition the rest of the nodes into lines of length $k-1$ in the y dimension. Each such line will be attached below one of the nodes of the x-line. When all y-lines have been attached, the leader, for all $0 \leq i \leq d^2 - 1$, initializes the memory of the line attached below pixel i with (i, d). Then all simulations of M are executed in parallel and eventually each one of them sets its x-pixel to either *on* or *off*. When all simulations have ended, the leader releases the y-lines and then partitions the x-line into consecutive segments of length d (see Figure 3(a)). Each segment corresponds to a row of the $d \times d$ square to be constructed. In particular, segment $i \geq 1$ (counting from left) corresponds to row i (bottom-up). Moreover, if i is even (odd), segment i should match with its upper (lower) side to the upper (lower) side of segment $i-1$. The leader marks appropriately the nodes of each segment to make them aware of the orientation that they should have in the square. Moreover, it assigns a unique key-marking to each segment so that segment i can easily and locally detect segment $i-1$. In particular, if i is odd (even), it marks nodes i and $i-1$ of the segment counting from left to right (right to left); see Figure 3(b). Then the leader releases, one after the other, all segments. The segments are free to move in the solution until they meet and recognize their counterpart, and when this occurs the two segments bind together. Eventually, the $d \times d$ square is constructed and every pixel is in the correct position. The leader can detect this and release the constructed shape consisting of the *on* pixels.

5. CONCLUSION AND OPEN PROBLEMS

There are several interesting open problems related to the findings of this work. A very intriguing problem is to give a proof, or strong experimental evidence (we have already some preliminary such evidence), that there is no analogue of Theorem 1 if all processes are identical (i.e. no unique leader). A possibility left open then would be to achieve high probability counting with $f(n)$ leaders. There is also work to be done w.r.t. analyzing the running times of our protocols and our generic constructors and proposing more efficient solutions. Also it is not yet clear whether the protocol of Section 3.1 is the fastest possible nor that its success probability or the upper bound on n that it guarantees cannot be improved; a proof would be useful. Moreover, it is not obvious what is the class of shapes and patterns that the TMs considered here compute. Of course, it was sufficient as a first step to draw the analogy to such TMs because it helped us establish that our model is quite powerful. However, still we would like to have a characterization that gives some more insight to the actual shapes and patterns that the model can construct.

A possible refinement of the model could be a distinction between the *speed of the scheduler* and the *internal operation speed of a component*. Such a distinction is very natural, because a connected component should operate at a different speed than it takes for the scheduler to bring two nodes into contact. It would also be interesting to consider for the first time a *hybrid model* combining active mobility controlled by the protocol and passive mobility controlled by the environment.

There is also work to be done towards the development of even more applied models (e.g. variations of the one pro-

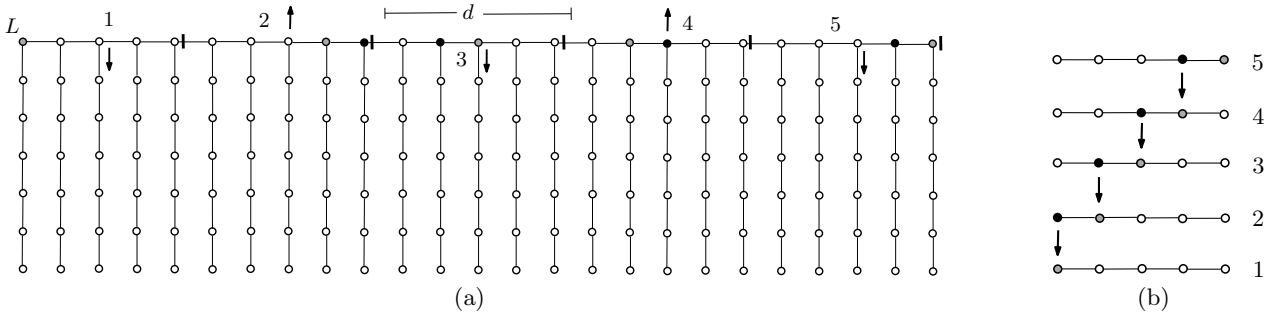

Figure 3: (a) d^2 lines of length $k-1$ each, are pendent below the d^2 pixels. The pixels are arranged linearly in dimension x and have been partitioned into equal segments of length d each (see the black delimiters). (b) The segments have been released in the solution, and now they have to gather together and form the square.

posed here) that take other real physical considerations into account. In this work, we have restricted attention on some geometrical constraints. Other properties of interest could be weight, mass, strength of bonds, rigid and elastic structure, collisions, and the interplay of these with the interaction pattern and the protocol. Moreover, in real applications mere shape construction will not be sufficient. Typically, we will desire to output a shape/structure that *optimizes some global property*, like energy and strength, or that achieves a *desired behavior in the given physical environment*.

Finally, it would be interesting to develop routines that can rapidly reconstruct broken parts. For example, imagine that a shape has stabilized but a part of it detaches, all the connections of the part become deactivated, and all its nodes become free. Can we detect and reconstruct the broken part efficiently (and without resetting the whole population and repeating the construction from scratch)? What knowledge about the whole shape should the nodes have in order to be able to reconstruct missing parts of it?

Acknowledgments: The author would like to thank D. Amaxilatis and M. Logaras for implementing (in Java) the counting protocol of Section 3.1 and experimentally verifying its correctness and also D. Doty for a few fruitful discussions on the same protocol at the very early stages of this work.

6. REFERENCES

[1] 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.

[2] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008.

[3] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, October 2007.

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

[5] P. Ehrenfest and T. Ehrenfest-Afanassjewa. Über zwei bekannte einwände gegen das boltzmannsche h-theorem. *Phys.Zeit.*, 8:311–314, 1907.

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

[7] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. Programmable matter. *Computer*, 38(6):99–101, 2005.

[8] 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.

[9] M. Kac. Random walk and the theory of brownian motion. *American Mathematical Monthly*, pages 369–391, 1947.

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

[11] 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.

[12] 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.

[13] O. Michail and P. G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing (JPDC)*, 81:1–10, 2015.

[14] P. W. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

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

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

[17] 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.

[18] M. Zakin. The next revolution in materials. *DARPA's 25th Systems and Technology Symposium (DARPATech)*, 2007.