Distrib. Comput.
DOI 10.1007/s00446-017-0309-z

Terminating distributed construction of shapes and patterns in a fair solution of automata

Othon Michail[1]

Received: 2 April 2016 / Accepted: 7 August 2017

Abstract In this work, we consider a *solution of automata* (or *nodes*) that move passively in a well-mixed solution without being capable of controlling their movement. Nodes can *cooperate* by interacting in pairs and every such interaction may result in an update of their local states. Additionally, the nodes may also choose to connect to each other in order to start forming some required structure. Such nodes can be thought of as *small programmable pieces of matter*, like tiny nanorobots or programmable molecules. The model that we introduce here is a more applied version of network constructors, imposing *physical* (or *geometric*) *constraints* on the connections that the nodes are allowed to form. Each node can connect to other nodes only via a very limited number of *local ports*. Connections are always made at *unit distance* and are *perpendicular to connections of neighboring ports*, which makes the model capable of forming *2D or 3D shapes*. We provide direct constructors for some basic shape construction problems, like *spanning line*, *spanning square*, and *self-replication*. We then develop *new techniques* for determining the computational and constructive capabilities of our model. One of the main novelties of our approach is that of exploiting the assumptions that the system is well-mixed and has a unique leader, in order to *give terminating protocols that are correct with high probability*. 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* with high probability. We then use this protocol as a subroutine in order to develop our *universal constructors*, establishing that *it is possible for the nodes to become self-organized with high probability into arbitrarily complex shapes while still detecting termination of the construction*.

Keywords Distributed network construction · Programmable matter · Shape formation · Well-mixed solution · Homogeneous population · Distributed protocol · Interacting automata · Fairness · Random schedule · Structure formation · Self-organization · Self-replication

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, which may include both cooperation and competition, and constant interaction 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 them.

Ulam's and von Neuman's Cellular Automata (cf., e.g., [40]), essentially a distributed grid network of automata,

Supported in part by the project "Foundations of Dynamic Distributed Computing Systems" (FOCUS) which is implemented under the "ARISTEIA" Action of the Operational Programme "Education and Lifelong Learning" and is co-funded by the European Union (European Social Fund) and Greek National Resources. A preliminary version of the results in this paper has appeared in [32].

✉ Othon Michail
Othon.Michail@liverpool.ac.uk

[1] Department of Computer Science, University of Liverpool, Ashton Street, Liverpool L69 3BX, UK

Published online: 23 August 2017

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. In the young area of DNA self-assembly and DNA computation (starting with the works of Adleman [6] and Winfree [44]), it has been already demonstrated that it is possible to (algorithmically) self-assemble DNA strands so that they carry out computations as they grow some structures. Recently, an interesting theoretical model was proposed, the *Nubot* model, for studying the complexity of self-assembled structures with active molecular components [43]. This model is "inspired by biology's fantastic ability to assemble biomolecules that form systems with complicated structure and dynamics, from molecular motors that walk on rigid tracks and proteins that dynamically alter the structure of the cell during mitosis, to embryonic development where large-scale complicated organisms efficiently grow from a single cell". Population Protocols of Angluin *et al.* [1] were originally motivated by highly dynamic networks of simple sensor nodes that cannot control their mobility. It was soon realized that their probabilistic version is formally equivalent to a restricted version of stochastic *chemical reaction networks* (CRNs), which model chemistry in a *well-mixed solution* (see, e.g., [41]). Moreover, the *Network Constructors* extension of population protocols [33], 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 [33] (but without any physical geometric considerations). Also recently a system was reported that demonstrates programmable self-assembly of complex 2-dimensional shapes with a thousand-robot swarm, called the *Kilobot* [38]. This was enabled by creating small, cheap, and simple "autonomous robots designed to operate in large groups and to cooperate through local interactions and by developing a collective algorithm for shape formation that is highly robust to the variability and error characteristic of large-scale decentralized systems".

1.1 Our approach

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, 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 may resemble the complex collective behavior of some 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 small 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, which motivates 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 [33], 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 [33]. In particular, we introduce some physical (or geometric) constraints on the connections that the processes are allowed to form. In the network constructors model of [33], 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. For example, even two nodes hidden in the middle of distinct dense components could interact and, additionally, there was no constraint on the number of active connections that a node could form (could be up to the order of the system). 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 geometric constraints. To capture this in our model, we assume that each device can connect to other devices only via a very limited (finite and independent of the size of the system) 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 2-dimensional or 3-dimensional 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 [33] 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*, allowing 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 enables us to improve eventual *stabilization* to eventual *termination* but, most importantly, because it enables us 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, [34] is the only work that has considered this issue but with totally different and more deterministic assumptions. Several other papers [1,2,33] have already exploited a uniform random interaction model, but in all cases it has been used to analyze the expected time to convergence of stabilizing protocols and not for maximizing the correctness probability of terminating protocols, as we do here.

In Sect. 2, we discuss further related literature. Section 3 formally defines the model under consideration and brings together all definitions and basic facts that are used throughout the paper. In Sect. 4, we provide direct (stabilizing) constructors for some basic shape construction problems. Section 5 introduces our technique for counting the size n of the system with high probability. The 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 Sect. 5 is then 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 Sect. 6. Finally, in Sect. 7 we conclude and give further research directions that are opened by our work.

2 Further related work

Population protocols Our model for shape construction is strongly inspired by the Population Protocol model [1] and the Mediated Population Protocol model [30]. In the former, connections do not have states. States on the connections were first introduced in the latter. The main difference to our model is that *in those models the focus was on the computation of functions of some input values and not on network construction*. Another important difference is that we allow the edges to choose between *only two possible states* which was not the case in [30]. Interestingly, when operating under a uniform random scheduler, population protocols are formally equivalent to a restricted version of stochastic *chemical reaction networks* (CRNs) which model chemistry in a *well-mixed solution* (see, e.g., [41]). "CRNs are widely used to describe information processing occurring in natural cellular regulatory networks, and with upcoming advances in synthetic biology, CRNs are a promising programming language for the design of artificial molecular control circuitry" [12,21]. 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* [3]. Semilinearity persists up to $o(\log \log n)$ local space but not more than this [13]. If, additionally, the connections between processes can hold a state from a finite domain (note that this is a stronger requirement than the on/off that the present work assumes)

then the computational power dramatically increases to the commutative subclass of **NSPACE**(n^2) [30]. Other important works include [25] which equipped the nodes of population protocols with unique identifiers (abbreviated "uids" or "ids" throughout) and [10] which introduced a (weak) notion of speed of the nodes that allowed the design of fast converging protocols with only weak requirements. For introductory texts see [8,31].

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. The research area of "algorithmic self-assembly" belongs to the field of "molecular computing". The latter was initiated by Adleman [6], who designed interacting DNA molecules to solve an instance of the Hamiltonian path problem. The model that has guided the study in algorithmic self-assembly is the Abstract Tile Assembly Model (aTAM) [39,44] and variations.

Recently, the Nubot model was proposed [43], which was another important influence for our work. That model aims at "motivating engineering of molecular structures that have complicated active dynamics of the kind seen in living biomolecular systems". It tries to "capture the interplay between molecular structure and dynamics". "Simple molecular components form assemblies that can grow" (exponentially fast, by successive doublings) "and shrink, and individual components undergo state changes and move relative to each other". The main result of [43] was that any computable shape of size $\leq n \times n$ can be built in time polylogarithmic in n, plus roughly the time needed to simulate a TM that computes whether or not a given pixel is in the final shape. One of the main differences between the Nubot model and our model is that in the former the nodes are equipped with an *active* actuation mechanism (see also [14] for another study of active self-assembly). This means that nodes (representing monomers there) are capable of firing transition rules that apart from changing their state can also change their relative position to neighboring nodes. This core characteristic brings the Nubot model closer to reconfigurable robotics (see, e.g., [5]) and active programmable matter (see, e.g., [15,36]) models. In contrast, reconfiguration in our model is *passive*, meaning that all mobility is controlled by the environment and the nodes can only "decide" whether to connect or disconnect whenever they are given the opportunity to interact.

Another type of self-assembly model that is close to the model studied in this paper, is the model of signal passing tiles [26,37]. Their main similarities are that signal-passing tiles are also passive and they can control connection and disconnection (via *glues*) as in our model. Still there are some important differences that set our model apart from the signal-passing tiles model. The most crucial one, is that in signal-passing tiles (and in the vast majority of algorithmic self-assembly models) there is an unlimited supply of tiles and any global parameter of the target configuration, such as its size n, must be somehow explicitly encoded in advance (as input), e.g., by assigning to each tile a number of glues that depends on n or, as in [14], by starting from an initial line of length $\log n$. In contrast, in our model n is always the number of nodes in the system, their number remaining unmodified throughout the execution, and, additionally, the nodes do not know n in advance and have to coordinate in order to compute it and become capable of constructing a sufficiently large shape (i.e., one that depends on the size of the system). Other important differences are the existence of various types of glues in signal-passing tile assembly and also temperature and strength parameters that determine stability of a configuration, whereas in our model stability only depends on the local states of nodes and their position in the configuration. See [20] for an introductory text to algorithmic self-assembly.

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 [7] that initiated the theoretical study of distributed computing systems up to now, the focus has been more on assuming a given communication topology and constructing a virtual network over it, e.g., a spanning tree for the purpose of fast dissemination of information. Moreover, these models assume most of the time unique identities, unbounded memories, and message-passing communication. Additionally, a process always communicates with its neighboring processes (see [9,29] for all the details). An exception is the area of geometric pattern formation by mobile robots (cf. [16,42] and references therein). A great difference, though, to our model is that in mobile robotics the computational entities have complete control over their mobility and thus over their future interactions. That is, the goal of a protocol is to result in a desired interaction pattern while in our model the goal of a protocol is to construct a structure while operating under a totally unpredictable interaction pattern.

Programmable matter Programmable matter refers to any type of matter that can *algorithmically* change its physical properties (see, e.g., [24,35]). There is a recent growing interest in the theory and algorithms for programmable matter systems. The network constructors model [33] and its geometric variant studied in this paper, may be viewed as models for programmable matter operating in a dynamic environment. The *Amoebot* model, a programmable matter model inspired by the behavior of amoeba, was proposed in [15,18] (see also [17,19] for some more recent studies). Another very recent study considered spherical programmable matter modules that can rotate or slide relative to neighboring modules [36], trying to capture transformation mechanisms that are feasible by current technology. As already mentioned above, the core characteristic that distinguishes the present model is

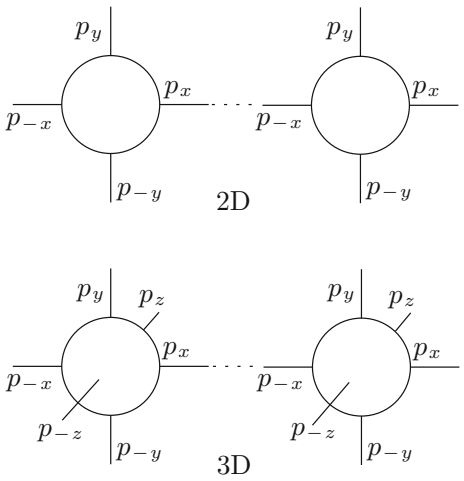

Fig. 1 The top figure depicts the 2D version of the model. Each node has four ports and consecutive ports are perpendicular to each other. Two nodes are interacting, the *left* one via its p_x port and the right one via its p_{-x} port. The interaction can occur because the distance between the nodes is unit and the corresponding ports are totally aligned (in a straight line). The bottom figure depicts the 3D version of the model. The only difference is an extra z dimension

the fact that all dynamicity is passive and the only actuation controlled by the program is the activation/deactivation of connections whenever some adversarially controlled conditions are met.

3 The model

The system consists of a population V of n distributed *processes* (finite-state machines), called *nodes* when clear from context. 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). Similarly, in the 3-dimensional (3D) case there are 6 ports p_y, p_z, p_x, p_{-y}, p_{-z}, and p_{-x} (see Fig. 1). Throughout this work, we denote by $\bar{j}$ the port "opposite" to port j, that is, if $j \equiv p_i$ then $\bar{j} \equiv p_{-i}$. Neighboring ports are perpendicular to each other, forming local axes. For example, in the 2-dimensional case, $u \perp r$, $r \perp d$, $d \perp l$, and $l \perp u$. 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. A node may be arbitrarily rotated so that, for example, its x local coordinate is aligned with the y global coordinate of the system or it is not aligned with any global coordinate. 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 the port of another node v. For example, it could be the case that, at some point during execution, the axis of the u port of w becomes aligned with the axis of the l port of another node v and the distance between them is one unit. Then w and v interact and, apart from updating their local states, they can also activate the connection between their corresponding ports. In a future pairwise interaction, they can again deactivate the connection if required.

Definition 1 A 2D (or 3D) 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 $P = \{u, r, d, l\}$ ($P = \{p_y, p_z, p_x, p_{-y}, p_{-z}, p_{-x}\}$, respectively, for the 3D case) is the set of *ports* and $\{0, 1\}$ is the set of *edge-states*. 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 $a \neq a'$ or $b \neq b'$ or $c \neq c'$ and *ineffective* otherwise. When we present the transition function of a protocol we only present the effective transitions.

Let $E = \{\{(v_1, p_1), (v_2, p_2)\} : v_1 \neq v_2 \in V$ and $p_1, p_2 \in P\}$ be the set of all unordered pairs of node-ports (cf. [33] for more details on unordered interactions).[1] 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 (i.e., of every 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 geometric 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 geometric 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

[1] To simplify notation, an unordered pair $\{a, b\}$ will typically be denoted by ab.

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." throughout) 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 $E_s = A(C) \cap \{(v_1, p_1)(v_2, p_2) : v_1 \neq v_2 \in V_s$ 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 (see, for an example, the black nodes and the connections between them in Fig. 7d on page 21). As already mentioned, our main focus will be on terminating protocols. In this case, we assume a set Q_{halt} such that $Q_{out} \subseteq Q_{halt} \subseteq Q$ and, for all $q_{halt} \in Q_{halt}$, every rule containing q_{halt} is ineffective. In contrast, in stabilizing protocols there is no Q_{halt} and states in Q_{out} may have effective interactions which we guarantee (by design) to cease eventually resulting in the stabilization of the final shape.

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 $V(R_G) \setminus V(G)$ 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, this process can be carried out by a protocol). 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 $\{0, 1\}$.[2] 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 j-th 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 (see the directed path in Fig. 7b on page 21). 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.[3] Moreover, in accordance to our definition of a

[2] Observe that we have intentionally restricted attention to *unary* languages as we want to focus on deterministic construction, in the sense that for any given "input-size" (here d) we want the population to always produce the same output shape.

[3] If the TM is not provided with the size of the square, then it can only compute uniform/symmetric shapes that are independent of n. Such a shape could for example be one that has every even pixel *on* and every odd pixel *off*. But such shapes rarely satisfy the connectivity condition. For example, it is not clear how to activate all the leftmost pixels of the square by a uniform TM, because such a TM should somehow guess that pixel $2\sqrt{n}-1$ should be accepted without knowing n and given that all pixels in $[1, 2\sqrt{n}-2]$ must be rejected. So, it seems more natural to consider TMs that apart from the pixel index are also provided with n or $\sqrt{n}$ (if the latter is more convenient) in binary. Now, it is straightforward how to resolve the acceptance of only the leftmost pixels of the square.

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.[4]

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 $G \in \mathcal{L}$[5] of order[6] 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.[7] Also, we say that $\mathcal{A}$ *constructs* $\mathcal{L}$ *with waste* $n - g(n)$.

4 Some basic constructions

We give in this section protocols for two very basic shape construction problems, the *spanning line* problem and the *spanning square* (or $\sqrt{n} \times \sqrt{n}$ *square*) problem. In both problems, for any number of nodes n, the n nodes must end up organized in a desired shape from a given shape-language. In the spanning line problem, the nodes must end up with an active line that is spanning and straight and in the spanning square problem the nodes must end up with an active square-grid spanning the population. These constructions not only serve as first expositions of the model in action, but are also very useful because they organize the nodes in a way that is convenient for TM simulations that exploit the whole distributed memory as a tape. Keep in mind that the protocols of this section are *stabilizing* (that is, eventually the output shape stops changing) and not terminating. Our technique that allows for terminating constructions will be introduced in Sect. 5.

4.1 Global line

We begin by presenting a protocol for the spanning line problem. Assume that there is initially a unique leader in state L_r (we typically use capital 'L' for the states of a leader to distinguish from the left port 'l') and all other nodes are in state q_0. A protocol that constructs a spanning line is described by the effective rules $(L_i, i), (q_0, j), 0 \to (q_1, L_{\bar{j}}, 1)$ for all $i, j \in \{u, r, d, l\}$, where $\bar{j}$ denotes the port opposite to port j. In words, initially the leader (i.e., L_r) waits to meet a q_0 via its right port. Assume that it meets port j of a q_0. Then the connection between them becomes activated and the leader takes the place of the q_0, leaving behind a q_1. Moreover, the new leader is now in state $L_{\bar{j}}$ indicating that it is now waiting to expand the line towards the port that is opposite to the one that is already active, which guarantees that a straight line will be formed.

We could even have a simplified version of the form $(L, r), (q_0, l), 0 \to (q_1, L, 1)$. This is a little slower, because now an effective interaction, and a resulting expansion of the line, only occurs when the r port of the leader meets the l port of a q_0.

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

We now give two protocols for the spanning square problem. We assume for simplicity that the square root of n is integer. We again assume that there is a pre-elected unique leader, which is initially in state L_u and all other nodes are in state q_0. The code of our first protocol for the spanning square problem is given in Protocol 1.

We now describe the idea that Protocol 1 implements. 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. This can only cause the attachment of a free q_0 from below, via rule $(L_d, d), (q_0, u), 0 \to (q_1, L_l, 1)$. When this occurs, the leader moves on the new node, updates its state to L_l, and tries to move to the left. This will occur by the attachment of another free node from the left this time, via rule $(L_l, l), (q_0, r), 0 \to (q_1, L_u, 1)$. When this occurs, the leader moves on the new node, updates its state to L_u, and tries to move up. But this time the up movement cannot succeed because the leader is below the bottom left corner of the square. Instead, the leader activates the connection with that corner, via rule $(L_u, u), (q_1, d), 0 \to (L_l, q_1, 1)$, and tries to move another step left. When it succeeds, it tries to move up again, which can now occur, via rule $(L_u, u), (q_0, d), 0 \to (q_1, L_r, 1)$, because the leader has moved outside the left boundary of the 2×2 square. In general, whenever the leader

Footnote 3 continued
The TM every time accepts the input-pixel i iff $i = 2k\sqrt{n} - 1$, for some $k \geq 1$, or $i = 2k\sqrt{n}$, for some $k \geq 0$. Observe that $2k\sqrt{n}$ can always be computed because the TM is also provided with $\sqrt{n}$ in its input.

[4] We should mention that part of the ideas related to the pixel-encoding and the TM operating on pixels have been inspired by similar constructions of Woods *et al.* [43].

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

[6] By "order" of a shape, we mean the number of nodes of the shape.

[7] By "greatest function" $g(n)$, we mean that for all functions $f(n)$ that satisfy the above properties and all n, it holds that $g(n) \geq f(n)$. Intuitively, $g(n)$ is a complete description of the guaranteed size of the shapes that $\mathcal{A}$ constructs; in practice, it is often sufficient to characterize $g(n)$ asymptotically.

Protocol 1 *Square*

$Q = \{L_u, L_r, L_d, L_l, q_0, q_1\}$, $L_0 = L_u$ (i.e., the initial leader-state is in this case L_u)
δ:

// an L_i-leader will move one step in the i direction,
// adding one node to the perimeter of the square;
// then it will try to change direction, clockwise
$(L_u, u), (q_0, d), 0 \to (q_1, L_r, 1)$
$(L_r, r), (q_0, l), 0 \to (q_1, L_d, 1)$
$(L_d, d), (q_0, u), 0 \to (q_1, L_l, 1)$
$(L_l, l), (q_0, r), 0 \to (q_1, L_u, 1)$
// changing of direction cannot succeed as long as the
// leader has not managed to go beyond the boundary
// of the square; the leader restores its previous direction
$(L_u, u), (q_1, d), 0 \to (L_l, q_1, 1)$
$(L_r, r), (q_1, l), 0 \to (L_u, q_1, 1)$
$(L_d, d), (q_1, u), 0 \to (L_r, q_1, 1)$
$(L_l, l), (q_1, r), 0 \to (L_d, q_1, 1)$

// All transitions that do not appear have no effect

is at the left (the up, right, and down cases are symmetric) of the already constructed square it tries to move to the right in order to walk above the square. If it does not succeed, it is because it has not yet moved over the upper boundary, so it activates the edge to the right, takes another step up and then tries again to move to the right. In this way, the leader always grows the square perimetrically and in the clockwise direction, i.e., following a spiral trajectory in the grid.

We next use turning marks to simplify and speed up the turning process. The unique leader begins in state L_d^2. Now, instead of always trying to turn, the leader turns only when it meets special marks left by the previous phase near the corners of the square. When it meets such a mark, the leader introduces the new corner and a new mark adjacent to that corner to be found during the next phase, and then makes a turn (see Fig. 2). A difference to the previous protocol is that now several of the nodes of the new perimeter may remain disconnected for a while from their internal neighbors (i.e., those belonging to the internal perimeter constructed in the previous phase). However, rules of the form $(q_1, i), (q_1, \bar{i}), 0 \to (q_1, q_1, 1)$ guarantee that these nodes eventually become connected. A disadvantage of this approach is that the structure may be less "rigid" than the previous one as long as several (q_1, q_1) connections are not yet established. The protocol is formally presented in Protocol 2.

A drawback of Protocol 2 is that the construction is never a true square but rather a square with four protruding turning marks (so, we here need $\sqrt{n-4}$ to be integer for a complete such construction). An alternative that circumvents this is the

Protocol 2 *Square2*

$Q = (\{L_i, L_i^1, L_i^2, L_i^3, L_i^4 : i \in \{u, r, d, l\}\} \setminus \{L_l^1\}) \cup \{L_{end}, q_0, q_1\}$,
$L_0 = L_d^2$
δ:

// adding the turning marks of the 2×2 square
$(L_d^2, d), (q_0, u), 0 \to (L_u^1, q_1, 1)$
$(L_l^2, l), (q_0, r), 0 \to (L_r^1, q_1, 1)$
$(L_u^2, u), (q_0, d), 0 \to (L_d^1, q_1, 1)$
$(L_r^2, r), (q_0, l), 0 \to (L_{end}, q_1, 1)$
// constructing the 2×2 square
$(L_u^1, u), (q_0, d), 0 \to (q_1, L_l^2, 1)$
$(L_r^1, r), (q_0, l), 0 \to (q_1, L_u^2, 1)$
$(L_d^1, d), (q_0, u), 0 \to (q_1, L_r^2, 1)$
// end of the present phase at the bottom right corner
$(L_{end}, d), (q_0, u), 0 \to (q_1, L_l, 1)$
// trying to grow the square perimetrically, L_i either
// suceeds and continues to be L_i or meets a turning
// mark and becomes L_i^3
$(L_l, l), (q_0, r), 0 \to (q_1, L_l, 1)$
$(L_l, l), (q_1, r), 0 \to (q_1, L_l^3, 1)$
$(L_u, u), (q_0, d), 0 \to (q_1, L_u, 1)$
$(L_u, u), (q_1, d), 0 \to (q_1, L_u^3, 1)$
$(L_r, r), (q_0, l), 0 \to (q_1, L_r, 1)$
$(L_r, r), (q_1, l), 0 \to (q_1, L_r^3, 1)$
$(L_d, d), (q_0, u), 0 \to (q_1, L_d, 1)$
$(L_d, d), (q_1, u), 0 \to (q_1, L_d^3, 1)$
// adding a new turning mark to the present corner
$(L_l^3, l), (q_0, r), 0 \to (q_1, L_d^4, 1)$
$(L_u^3, u), (q_0, d), 0 \to (q_1, L_l^4, 1)$
$(L_r^3, r), (q_0, l), 0 \to (q_1, L_u^4, 1)$
$(L_d^3, d), (q_0, u), 0 \to (q_1, L_r^4, 1)$

$(L_d^4, d), (q_0, u), 0 \to (L_u, q_1, 1)$
$(L_l^4, l), (q_0, r), 0 \to (L_r, q_1, 1)$
$(L_u^4, u), (q_0, d), 0 \to (L_d, q_1, 1)$
$(L_r^4, r), (q_0, l), 0 \to (L_{end}, q_1, 1)$
// activating missing internal edges of the square
$(q_1, i), (q_1, \bar{i}), 0 \to (q_1, q_1, 1)$, for all $i \in \{u, r, d, l\}$,
where $\bar{i}$ denotes the opposite port of i
$(L_u, r), (q_1, l), 0 \to (L_u, q_1, 1)$
$(L_r, d), (q_1, u), 0 \to (L_r, q_1, 1)$
$(L_d, l), (q_1, r), 0 \to (L_d, q_1, 1)$
$(L_l, u), (q_1, d), 0 \to (L_l, q_1, 1)$

following. The leader constructs the perimeter of the present phase by walking on the perimeter drawn by the previous

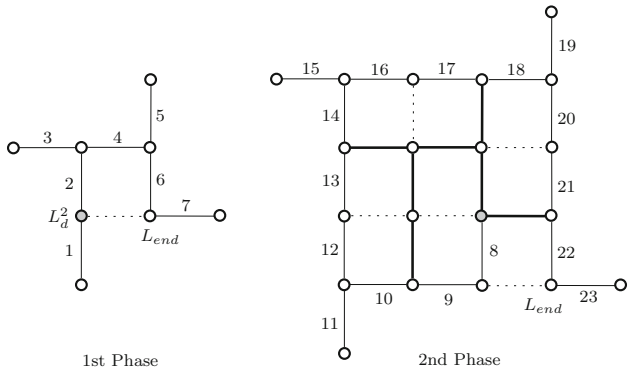

Fig. 2 The first two phases of Protocol 2. Gray nodes indicate the starting point of each phase. Edge labels indicate the order by which the square grew during the phase. The nodes labeled L_{end} are the points at which each of the phases ends. The unlabeled solid edges of Phase 2 indicate the shape that pre-existed from Phase 1. The nodes attached at "times" 1, 3, 5, 7 of Phase 1 and 11, 15, 19, 23 of Phase 2 are the turning marks that will be exploited for easier turning by the leader in the subsequent phase. Dotted edges are edges that have not be activated yet but will for sure be activated eventually resulting in a more "rigid" structure

phase. For example, while walking up the left border of the square it attaches nodes to the left of the border, thus constructing a new left border. In this way, the leader just needs to find a special state on the corner of the previous phase (or the absence of the corner) in order to determine that turning is required.

Finally, though the constructions in this section were based on a pre-elected unique leader, we should mention that this assumption is not necessary for solving the above problems. However, the protocols that do not require a leader are more complicated and do not serve as good first expositions of the model.

5 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. The importance of such protocols is further supported by the fact that we cannot guarantee anything much better than this. In particular, observe that if we require a population protocol to always terminate and additionally to always be correct, then we immediately obtain an impossibility result. It is easy to see this by imagining a system in which a unique leader interacts with the other nodes (there are no interactions between non-leaders and no connections are ever activated). Any fair execution s_1 of a protocol in a population of size n in which the leader outputs n and terminates can appear as an "unfair" prefix of a fair execution $s' = s_1 s_2$ on a population of size $n' > n$. This is a contradiction because in s' the leader must again terminate and output n even though $n' \neq n$. The main reason is that $|s_1|$ is finite and independent of n; it only depends on the maximum "depth" of a chain of rules of the protocol leading to termination. This implies that in s' the leader terminates before interacting with all other nodes.

In Sect. 5.1, we present a population protocol with a pre-elected 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. All probabilistic protocols that have appeared so far, like those in [1,2], are not terminating but stabilizing and the high probability arguments concern their time to convergence. Additionally, this protocol is crucial because all of our generic constructors, that are developed in Sect. 6, 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. enables us 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 [25,30,33], but not in [34] which was the first extension to allow for sequential composition based on some non-probabilistic assumptions). Then in Sect. 5.2 we comment on the possibility of dropping the unique leader assumption and leave this as an interesting open problem. Finally, in Sect. 5.3 we establish that if the nodes have unique ids then it is possible to solve the problem without a unique leader.

5.1 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 pre-elected leader node has unbounded local memory (of the order of n). In Sect. 6.1, we will adjust the protocol to make it work in our model, using constant memory on every node, including the leader.

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,

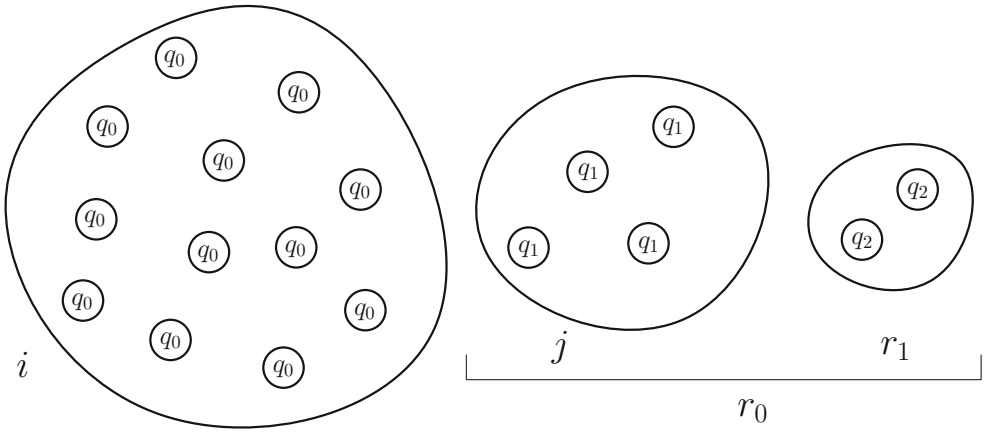

Fig. 3 A configuration of the system (excluding the leader). The number of q_0s remaining is denoted by i. The number of q_1s introduced so far is denoted by j. The value of the counter r_1 is equal to the number of q_1s encountered so far by the leader, which is in turn equal to the number of q_2s introduced. The value of the counter r_0 is equal to the number of q_0s encountered, which is equal to the number of q_1s and q_2s introduced

$0 \leq r_0, r_1 \leq n$. The rules that capture the core operations of the protocol are of the form

$$(l(r_0, r_1), q_0) \rightarrow (l(r_0 + 1, r_1), q_1), \text{ if } r_1 < r_0$$
$$(l(r_0, r_1), q_1) \rightarrow (l(r_0, r_1 + 1), q_2), \text{ if } r_1 < r_0 \text{ and}$$
$$(l(r_0, r_1), \cdot) \rightarrow (halt, \cdot) \text{ if } r_0 = r_1.$$

It is worth reminding that, for the time being, we have disregarded edge-states and, therefore, the rules of the protocol only specify how the states of the nodes are updated. 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 implemented in the protocol by having the leader convert b q_0s to q_1s as a preprocessing step).

So, in Counting-Upper-Bound 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 after an expected number of $O(n^2 \log n)$ interactions. Moreover, when this occurs, w.h.p. it holds that $r_0 \geq n/2$.*

Proof Recall that the scheduler is a uniform random one, which, in every step, selects independently and uniformly at random one of the $n(n-1)/2$ possible interactions. Recall also that the random variable i denotes the number of q_0s and j denotes the number of q_1s in the current configuration, where initially $i = n-b-1$ and $j = b$. Observe also that all the following hold: $j = r_0 - r_1$, $r_0 \geq r_1$, because every conversion of a q_1 to q_2 must have been first counted by r_0 as a conversion of a q_0 to q_1, $r_1 = (n-1) - (i+j)$, and $r_0 + r_1$ is equal to the number of effective interactions (see Fig. 3).

We focus only on the effective interactions (we also disregard the halting interaction), which are always interactions between l and q_0 or q_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} (see Fig. 4). This is a "difficult" random walk 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. That is, whenever the random walk returns to some position j of the line, its transition probabilities have changed (because every leaving and returning involves at least on step to the left, which decreases the sum). Observe also that, in our case, the duration of the random walk can be at most $n-b$, in the sense that if the

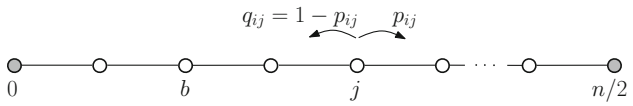

Fig. 4 A random walk modeling of the probabilistic process that the Counting-Upper-Bound protocol implements. A particle begins from position b. The position j of the particle corresponds to the difference between r_0 and r_1. Forward movement corresponds to an increment of r_0 and backward movement corresponds to an increment of r_1. Absorption at 0 corresponds to r_1 becoming equal to r_0 and thus to termination (and to failure if this occurs before $r_0 \geq n/2$ holds). Absorption at $n/2$ corresponds to r_0 becoming at least $n/2$ (before being absorbed at 0) and thus to success

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 like, for example, guaranteed winning in $(n/2)+r_1$ effective interactions, in which case we have winning in only $(n/2)+r_1$ effective interactions and j having become at most $(n/2)-r_1$). 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 $2r_1 \leq n \Rightarrow r_1 \leq n/2 \Rightarrow (n-1) - (i+j) \leq n/2 \Rightarrow i + j \geq (n/2) - 1$. 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 $p_{ij} \geq (n' - j)/n'$ and $q_{ij} \leq j/n'$ so that now the probabilities only depend on the position j. This new walk is the well-studied Ehrenfest random walk coming from the theory of brownian motion [22] and by results in [27] it is immediate to obtain that its recurrence time is exponential in n, thus, we do not expect the walk to return to 0 and fail in only n effective steps. In the sequel, we turn this into the desired high probability argument.[8]

We will reduce the Ehrenfest walk to one in which the probabilities do not depend on j. We first further restrict our walk, this time to the prefix $[0, b]$ of the line. In this part, it holds that $j \leq b$ which implies that $p_{ij} \geq (n' - b)/n'$ and $q_{ij} \leq b/n'$. Now we set $p_{ij} = p = (n' - b)/n'$ and $q_{ij} = q = b/n'$. Observe that this may only increase the probability of failure, so the probability of failure of the new walk is an upper bound on the probability of failure of our original walk. 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. So, we now have a r.w. with $b+1$ positions, where positions 0 and b are absorbing and 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 position 1)]. This is the probability of winning in the classical ruin problem analyzed, e.g., in [23] page 345. If we set $x = q'/p' = p/q = (n' - b)/b$ we have that:

$$\begin{aligned} \text{P[reach } b \text{ before 0]} &= 1 - \frac{x^b - x}{x^b - 1} = \frac{x - 1}{x^b - 1} \\ &\leq \frac{x}{x^b - 1} \approx \frac{1}{x^{b-1}} \\ &\approx \frac{1}{n^{b-1}}. \end{aligned}$$

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:

$$\begin{aligned} \text{P[fail at least once]} &\leq \sum_{m=1}^{n} \text{P[fail at repetition } m] \\ &\leq \sum_{m=1}^{n} \frac{1}{n^{b-1}} = \frac{n}{n^{b-1}} \\ &= \frac{1}{n^{b-2}}. \end{aligned}$$

[8] Imagine gas molecules that move about randomly in a container which is divided into two halves symmetrically by a partition. A hole is made in the partition to allow the exchange of molecules between the subcontainers. Suppose there are n molecules in the container. Think of the partitions as two urns, I and II, containing balls labeled 1 through n. Molecular motion can be modeled by choosing a number between 1 and n at random and moving the corresponding ball from the urn it is presently in to the other. This is a historically important physical model, known as the Ehrenfest model of diffusion, introduced in [22] in the early days of statistical mechanics to study thermodynamic equilibrium. So, the probability of failure of our counting protocol is asymptotically equivalent to the probability that urn I becomes empty in the first n steps assuming that it initially contains b balls. This walk has been studied by Kac in [27] who, among other things, proved that the mean recurrence time is $((R+k)!(R-k)!/(2R)!)2^{2R}$ ([27], page 386). If we set $k = -R$ so that the initial position is $R+k = 0$, then this evaluates to $2^{2R} = 2^{n/2}$, because $2R$ is the total length of the line. This shows that, even if we begin from position 0 instead of b, the recurrence time is expected to be huge and we do not expect the walk to return to 0 and fail in only n effective steps.

In summary, even if the protocol was restricted to disregard counter differences that are greater than b, still with probability at least $1-1/n^c$ (for constant $c=b-2$) the protocol has not terminated after at least n effective interactions, which in turn implies that the leader has counted at least half of the nodes.

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. [33]), thus the expected running time of Counting-Upper-Bound is $O(n^2 \log n)$ (interactions). □

Remark 1 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 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. (observe, e.g., that the last unvisited node requires an expected number of $\Theta(n^2)$ steps to be visited).

5.2 On dropping the leader assumption

An immediate question is whether the unique leader assumption of Theorem 1 can be dropped. Though we have not yet managed resolve this issue, we will describe a possible strategy for proving that this is not possible. In any case, we leave this as a challenging open problem.

The strategy would aim at showing that any protocol in which all nodes begin from the same state, may have some node terminate with (at least) constant probability, having participated in only a constant number of interactions. This would then imply that with constant probability the protocol terminates without having estimated any non-constant function of n. In the sequel, we describe this potential strategy in more detail.

Nodes again have a set of states Q and we also assume that they have unbounded private local memories. These memories are for internal purposes only and their contents are not communicated to other nodes. For example, a node u could maintain $|Q|$ counters, each counting the number of times the corresponding state has been encountered so far by u. We focus on protocols that always terminate (i.e for every $n \geq n_0$, for some finite n_0) and we want them to compute something w.h.p., e.g., the node that first terminates to know an upper bound on n w.h.p.. Let now $\mathcal{A}$ be a protocol as above. To establish the aforementioned impossibility, it would be sufficient to prove that, as n grows, there is (at least) a constant probability that some node terminates having interacted only a constant number of times.

First of all, observe that a protocol, apart from the usual transition function $\delta : Q \times Q \to Q \times Q$ that updates the communicating states, has also a function $\gamma : Q \times S \to S$ that updates the internal memory based on the encountered states. We focus on deterministic γ and, in this case, the internal state from S after k interactions only depends on the observed sequence Q^k of encountered states (because the initial state q_0 is always the same for all nodes). Every protocol $\mathcal{A}$ that always terminates, essentially defines a property $L_{\mathcal{A}} \subseteq Q^*$ consisting of those observed sequences that make a node terminate (the remaining sequences do not cause termination). Moreover, as the protocol does not know n, an $s_0 \in L_{\mathcal{A}}$ of minimum length has length that is independent of n (it could only be a function of $|Q|$). Observe that for every population size n, if s_0 is observed by some node u as a prefix of its interaction pattern (i.e., in its first $|s_0|$ interactions) then u terminates while having participated in only $|s_0|$ interactions, which is a constant number independent of n. So, for an impossibility of dropping the leader it suffices to prove that, for every $n \gg n_0$ and every such fixed s_0, there is (at least) a constant probability that some node observes s_0. This could be further broken down into proving the following set of arguments, provided that $n \gg n_0$:

1. With constant probability a configuration is reached, in which every state $q \in Q$ has multiplicity $\Theta(n)$ (that is, appears on $\Theta(n)$ distinct nodes).[9]
2. With constant probability the multiplicities of all states remain $\Theta(n)$ for $\Theta(n)$ steps.
3. While (2) holds, with constant probability one of the $\Theta(n)$ nodes, let it be u, whose state is q_0, interacts $|s_0|$ times.

If the above were true, then it would follow that u may observe s_0 with constant probability, in which case u will terminate having interacted only a constant (i.e., $|s_0|$) number of times. The reason for this is that in its ith interaction, for all $1 \leq i \leq |s_0|$, u observes the ith state of s_0, let it be q_i, with probability $(\#q_i \text{ in the population})/\Theta(n)$. As, by (2), the numerator is also $\Theta(n)$, for all $q_i \in Q$, the resulting probability is constant.

5.3 Counting without a leader but with UIDs

We now assume that nodes have unique ids from a universe $\mathcal{U}$ and that initially they 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 a unique leader,

[9] This seems to already follow from a result in [21], and, actually, not only with constant probability that we require here, but w.h.p..

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 is to have the node u_{max} 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. However, as initially all nodes have to behave as if they were the maximum, we must also ensure that u_{max} is not affected and 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 whose id is greater than id_u or that has been marked by an id greater than id_u, u becomes deactivated and stops counting. This guarantees that only u_{max} will forever remain active. 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}. Protocol 3 puts all these together formally and Theorem 2 shows that this process correctly simulates the counting process of Theorem 1, thus providing w.h.p. an upper bound on n.

Theorem 2 *When a node u in Protocol 3 halts, w.h.p. it holds that $u = u_{max}$ and that $2 \cdot count1_u \geq n$.*

Proof We first show that u_{max} simulates the probabilistic process of the unique leader l of Theorem 1. Recall that in the protocol of Theorem 1, all other nodes are initially q_0 and when l meets a q_0 it makes it q_1 and when it meets a q_1 it makes it q_2, every time counting in the corresponding counter. First, observe that u_{max} is never deactivated, i.e., $active_{u_{max}} = 1$ forever, because it never interacts with a greater id nor with a node that belongs to a greater id than its own. It suffices to show that when u_{max} meets a node for the first time it marks it once (simulating a q_0 to q_1 conversion), when it meets a node for the second time it marks it twice (simulating a q_1 to q_2 conversion), and that no other node can ever alter the nodes marked by u_{max}. When u_{max} interacts with a node v for the first time, then either $belongs_v = \perp$ or $\perp \neq belongs_v < max\_id$. So, in this case it marks v once by setting $marked_v \leftarrow 1$, $belongs_v \leftarrow max\_id$, and records this by increasing $count1_{u_{max}}$ by one. From now on, no other active node $w \neq u_{max}$ can ever affect the state of v, because for every such w it holds that $id_w < belongs_v = max\_id$ and the only effect in this case is the deactivation of w. The second time that u_{max} interacts with v, it still holds that $belongs_v = id_{u_{max}} (= max\_id)$ and $marked_v = 1$, and u_{max} marks v for a second time by setting $marked_v \leftarrow 2$ and records this by incrementing $count2_{u_{max}}$ by one. Again, v still belongs to max_id and no other node can ever affect its state. We conclude that if we were only interested in u_{max}'s output then, by Theorem 1, this would w.h.p. be an upper bound on n.

However, observe that not only u_{max} but also the other nodes execute a similar process and it could be the case that one of them terminates early (and before u_{max}) giving as output a wrong count. We now show that this is not the case. Take any node w with $id_w < max\_id$. Observe that if there were no nodes with id greater than id_w then w would simply execute the process described for u_{max}. However, in the presence such nodes, some nodes may be marked by a greater id before w counts them and others may be marked after w has counted them once but before counting them twice. Still, we shall show that none of these increases the probability of early termination of w (where termination of w occurs when $count1_w = count2_w$ first becomes satisfied). Consider the partition of $V \setminus \{w\}$ into the sets $S_{w,0}$, $S_{w,1}$, and $S_{w,2}$ of nodes which w has not marked yet, has marked once, and has marked twice, respectively. The counting process of w can only be affected when a node v in any of these sets is marked by an id greater than id_w: (i) If $v \in S_{w,2}$ then w has already counted v both in $count1_w$ and $count2_w$, therefore marking v does not affect w's counting at all. (ii) If $v \in S_{w,1}$ then v is a node that has been counted in $count1_w$ but not yet in $count2_w$. Marking v in this case does not speed up termination, as it only decreases the probability of $count2_w$ to increase (and recall that $count2_w$ is always trying to catch up $count1_w$). (iii) If $v \in S_{w,0}$ then marking v indeed slows down the rate of grow of $count1_w$ b, because a node that could contribute to $count1_w$ and has not been counted yet is no longer available. However, notice that every such v will from now on forever satisfy $belongs_v > id_w$, because $belongs_v$ can only increase, therefore every interaction of w with such a v will result in the deactivation of w. This implies that the "success" events of w (those corresponding to a $count1_w$ increment) have now been partitioned into $count1_w$ increment events and w deactivation events. So, if w ever fails to increment $count1_w$ due to an interference of

Protocol 3 Counting with UIDs

Require: Every node u has a unique id id_u and maintains a $(belongs, marked)$ pair, a $(count1, count2)$ pair, and a variable $active$, where $belongs \in \mathcal{U} \cup \{\perp\}$ initially $\perp$, $marked \in \{0, 1, 2\}$ initially 0, $count1, count2 \in \mathbb{N}_{\geq 0}$ initially $count1 = count2 = 0$ and $active \in \{0, 1\}$ initially 1. All nodes know a predetermined constant $b > 0$. The following is the code for every interaction of u, v with $id_u > id_v$.

1: **if** $active_v = 1$ **then**
2: $active_v \leftarrow 0$
3: **end if**
4: **if** $active_u = 1$ **then**
5: **if** $belongs_v = \perp$ or $\perp \neq belongs_v < id_u$ **then**
6: $belongs_v \leftarrow id_u$
7: $marked_v \leftarrow 1$
8: $count1_u \leftarrow count1_u + 1$
9: **end if**
10: **if** $\perp \neq belongs_v > id_u$ **then**
11: $active_u \leftarrow 0$
12: **end if**
13: **if** $belongs_v = id_u$ and $marked_v = 1$ and $count1_u \geq b$ **then**
14: $marked_v \leftarrow 2$
15: $count2_u \leftarrow count2_u + 1$
16: **if** $count1_u = count2_u$ **then**
17: u halts and outputs $2 \cdot count1_u$
18: **end if**
19: **end if**
20: **end if**

some u with $id_u > id_w$ on some $v \in S_{w,0}$, the effect is the deactivation of w, which prevents w from continuing with unfavorable probabilities. In other words, the only event that could negatively affect w's counting, deactivates w and, in this case, w cannot terminate early any more. □

6 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 Sect. 3), 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 Sect. 6.1 by adapting the Counting-Upper-Bound protocol of Sect. 5 to work in our model. The result is, again w.h.p., a line of length $\Theta(\log n)$ with a unique leader, containing n in binary. Then, in Sect. 6.2 the leader exploits its knowledge of n to construct a $\sqrt{n} \times \sqrt{n}$ square. In the sequel (Sect. 6.3), 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, in a systematic way, 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.

6.1 Storing the count on a line

We begin by adapting the Counting-Upper-Bound protocol of Theorem 1 to work in our model. In particular, the obtained protocol does not require the leader to have large local memory. Instead, it stores the r_0 and r_1 counters distributedly throughout the execution and when the protocol terminates the final correct count is stored 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 binary, in a distributed tape that it controls. The ith cell of the tape has three components, one storing the ith bit of r_0, the other storing the ith bit of r_1, and the third one storing the ith bit of an r_2 counter that will be discussed in the sequel. 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.[10] In the sequel, together with the other operations of the protocol, we also describe how the leader handles such overflows.

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 (in fact it is improved).

First of all, 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 for sure 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 $r_0 \geq 1$ (to see this, simply rewrite it as $r_0/2 \geq \lfloor \lg r_0 \rfloor$). 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. □

6.2 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 we did in Sect. 4.2, we again assume for simplicity that $\sqrt{n}$ is integer. Observe that now the leader has n stored in its line. The present construction exploits this knowledge and this makes it essentially different than the constructions of Sect. 4.2. Moreover, knowledge of n allows the protocol to terminate after constructing the square and to know that the square has been successfully constructed, a fact that was not the case in the stabilizing constructions of Sect. 4.2. The following protocol assumes that the guarantee of Lemma 1 is provided somehow and based on this assumption we will show that it works correctly in every execution (this is in contrast to the high probability argument of Lemma 1). This means that given the guarantee, the protocol that constructs the square is always correct. Of course, if we take the composition of Counting-on-a-Line that provides the guarantee and the protocol that constructs the square based on the guarantee, the resulting protocol is again correct w.h.p., however we still allow the possibility that some other deterministic (even centralized) preprocessing provides the required guarantee.

Square-knowing-n protocol The initial leader L first computes $\sqrt{n}$ on its line by any plausible algorithm (observe that the available space for computing the square root is expo-

[10] 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.

nential in the binary representation of n, which is the input to the algorithm, because, if needed, the leader can expand its line up to length n). In principle, it is not necessary to use additional space, because the leader can execute one after the other the multiplications $1 \cdot 1, 2 \cdot 2, 3 \cdot 3, \ldots$ in binary until the result becomes equal to n. Each of these operations can be executed in the initial $\log n$ space of the line of the leader. The time needed, though exponential in the binary representation of n, is still linear in the population size n. Now that the leader also knows $\sqrt{n}$, it 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 and 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 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 Figs. 5 and 6 for illustrations.

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

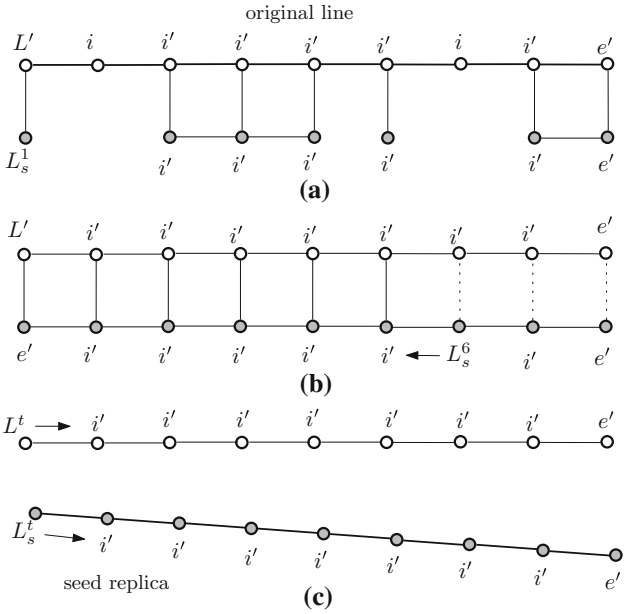

Fig. 5 **a** Several free nodes have already been attached to the original line. Some of them have already activated some horizontal connections forming some segments of the replica. **b** The leader (L') of the original line remains blocked while the leader (L_s^6) of the replica has detected that the replica is ready for detachment. It has already detached the three rightmost nodes and keeps moving to the left until it reaches the left endpoint and detaches the whole replica. **c** The seed replica has been released in the solution. The leader (L^t) of the original line has waken up and is restoring the nodes of its line to their original states. When it finishes (that is, when it will have traversed the whole line and have returned to the left endpoint), it will go to state L_{start} to start the square formation process. Similarly, the leader (L_s^t) of the seed replica is setting the nodes of its line to their normal i and e states, so that they start accepting the attachment of other nodes in order to create non-seed replicas

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 4, one of the possible protocols for the replication process of the original leader's line that creates the seed. The other replication processes, i.e., of L_s to L_r and of L_r to L_r, are almost identical to this one. Without loss of generality we assume that the original leader's line has state L on its left endpoint, e on its right endpoint, and every other internal node of the line is in state i. All other (free) nodes are in state q_0.

We additionally show that, in principle, the lines do not need a leader in order to successfully self-replicate. We give such a protocol which is "more parallel" and has a much more concise description than the previous one. We now 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 code is presented in Protocol 5. 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

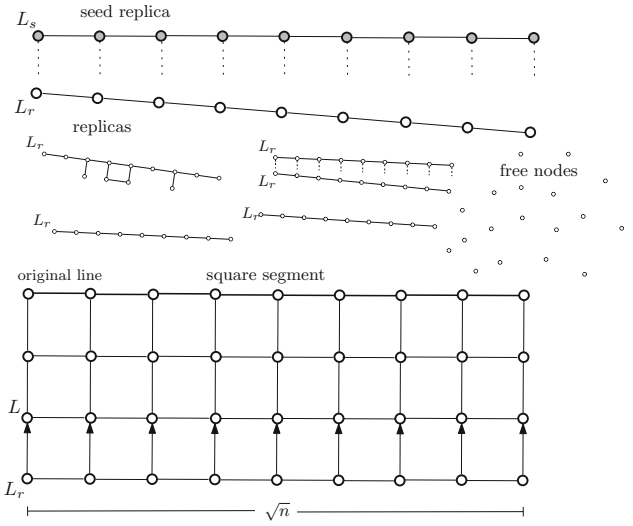

Fig. 6 The seed at the top has created another replica which has just been released in the solution. Below it, some additional replicas appear. One of them is in the middle of a replication that has not completed yet. There are also several nodes that are still free. At the bottom appears the square segment that has been constructed so far. The original line of the L-leader is the one at the top of the rectangle. The other rows below it have been formed by replicas that have been attached to the segment in previous steps. The L-leader keeps waiting at the bottom left corner for new replicas to arrive. One such has just arrived and will be attached to the segment

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. In fact, 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.

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.*

Proof From Lemma 1, when the leader in Counting-on-a-Line protocol 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. Then the leader computes $\sqrt{n}$ on its line and expands its line to make its length $\sqrt{n}$. Next the leader creates the seed replica by executing the routine described in Protocol 4. The seed replica keeps creating new self-replicating replicas. All these replications are performed by a routine essentially equivalent to Protocol 4. Every replica is a line of length $\sqrt{n}$ and will be eventually attached to the square-segment to form another row of the square. First observe that the seed may only be attached to the square,

Protocol 4 *Line-Replication*

$Q = \{L, L', L_s^j, L_s^t, L_s^{t'}, L_s^{t''}, L^t, L^{t'}, L^{t''}, L_{start}, i, i', e, e'\}$, $j \in \{1, 2, \ldots, 7\}$
δ:

// attaching nodes below the original line
$(L, d), (q_0, u), 0 \to (L', L_s^1, 1)$
$(i, d), (q_0, u), 0 \to (i', i', 1)$
$(e, d), (q_0, u), 0 \to (e', e', 1)$
// connecting attached nodes with each other
// horizonally to form the replica line
$(i', r), (i', l), 0 \to (i', i', 1)$
$(i', r), (e', l), 0 \to (i', e', 1)$
// the leader of the replica starts moving along its line
// activating any missing connections on the way
$(L_s^1, r), (i', l), 0 \to (e', L_s^2, 1)$
$(L_s^2, r), (i', l), \cdot \to (i', L_s^2, 1)$
// once it reaches the right endpoint it starts to detach
// the replica from the original line by dectivating one
// after the other all vertical connections while moving
// to the left
$(L_s^2, r), (e', l), \cdot \to (i', L_s^3, 1)$
$(L_s^3, u), (e', d), 1 \to (L_s^4, e', 0)$
$(i', r), (L_s^4, l), 1 \to (L_s^5, e', 1)$
$(L_s^5, u), (i', d), 1 \to (L_s^6, i', 0)$
$(i', r), (L_s^6, l), 1 \to (L_s^5, i', 1)$
// once it reaches the left endpoint it deactivates the
// last remaining vertical connection and the replica is
// separated from the original line
$(e', r), (L_s^6, l), 1 \to (L_s^7, i', 1)$
$(L_s^7, u), (L', d), 1 \to (L_s^t, L^t, 0)$
// the leaders of the two lines restore the local states
// of all nodes to their default values to enable further
// replications
$(x^t, r), (i', l), 1 \to (e', x^{t'}, 1), x \in \{L, L_s\}$
$(x^{t'}, r), (i', l), 1 \to (i', x^{t'}, 1), x \in \{L, L_s\}$
$(x^{t'}, r), (e', l), 1 \to (x^{t''}, e, 1), x \in \{L, L_s\}$
$(i', r), (x^{t''}, l), 1 \to (x^{t''}, i, 1), x \in \{L, L_s\}$
$(e', r), (L_s^{t''}, l), 1 \to (L_s, i, 1)$
$(e', r), (L^{t''}, l), 1 \to (L_{start}, i, 1)$

Protocol 5 *No-Leader-Line-Replication*

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

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

when the square has already obtained $\sqrt{n} - 1$ rows. This implies that replications do not cease before the square has been successfully constructed. Additionally, any non-seed replica r can be attached to the square-segment (whenever the l leader is in the state of waiting for new attachments) independently of whether r is in the middle of an incomplete replication. The reason is that attachment occurs via the up ports of r while replication takes place via the down ports of r. If this occurs, then the nodes of the incomplete replication are simply released as free nodes. So, assume that there are k nodes that are either free or part of an incomplete replication. We only have to prove that as long as $k \geq \sqrt{n}$ then eventually another replica has to be formed. If not, then for an infinite number of steps it holds that $k \geq \sqrt{n}$. Moreover, every non-seed replica in a finite number of steps becomes attached to the square-segment and releases any nodes of an incomplete replication. Thus, in a finite number of steps, every one of the $k \geq \sqrt{n}$ nodes is either free or part of an incomplete replication of the seed. Clearly, given that the seed does not cease self-replication and given that there are enough nodes to fill the $\sqrt{n}$ replication positions of the seed, in a finite number of steps (due to fairness) all these positions should have been filled and a replica should have been created. Thus, the assumption that no further replication occurs violates the fairness condition. □

6.3 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 (see Fig. 7b). The tape will be used to simulate a TM M of the form described in the Sect. 3. 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.[11] 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. Alternatively, the leader can detect termination by exploiting the fact that the last pixel to be examined is the one corresponding to the upper left or right corner of the square (depending on whether $\sqrt{n}$ is even or odd), which can be detected.

When the above procedure ends, the leader starts walking the tape in the opposite direction until it reaches the bottom left corner. In the way, it passes a *release* signal to every node it goes through. A node enters the release phase exactly when the leader departs from that node, apart from the bottom left corner which enters the release phase when the leader arrives. When two nodes that are both in the release phase interact, if at least one of them is *off* and their connection is

[11] When we refer to the *tape*, we mean the line produced by traversing the square in a zig-zag way beginning from the bottom left node, as described above. So the "leftmost", here, corresponds to the leftmost nodes of the line, e.g., the left part of the bottom row of the square, and should not be confused with the nodes on the leftmost column of the square.

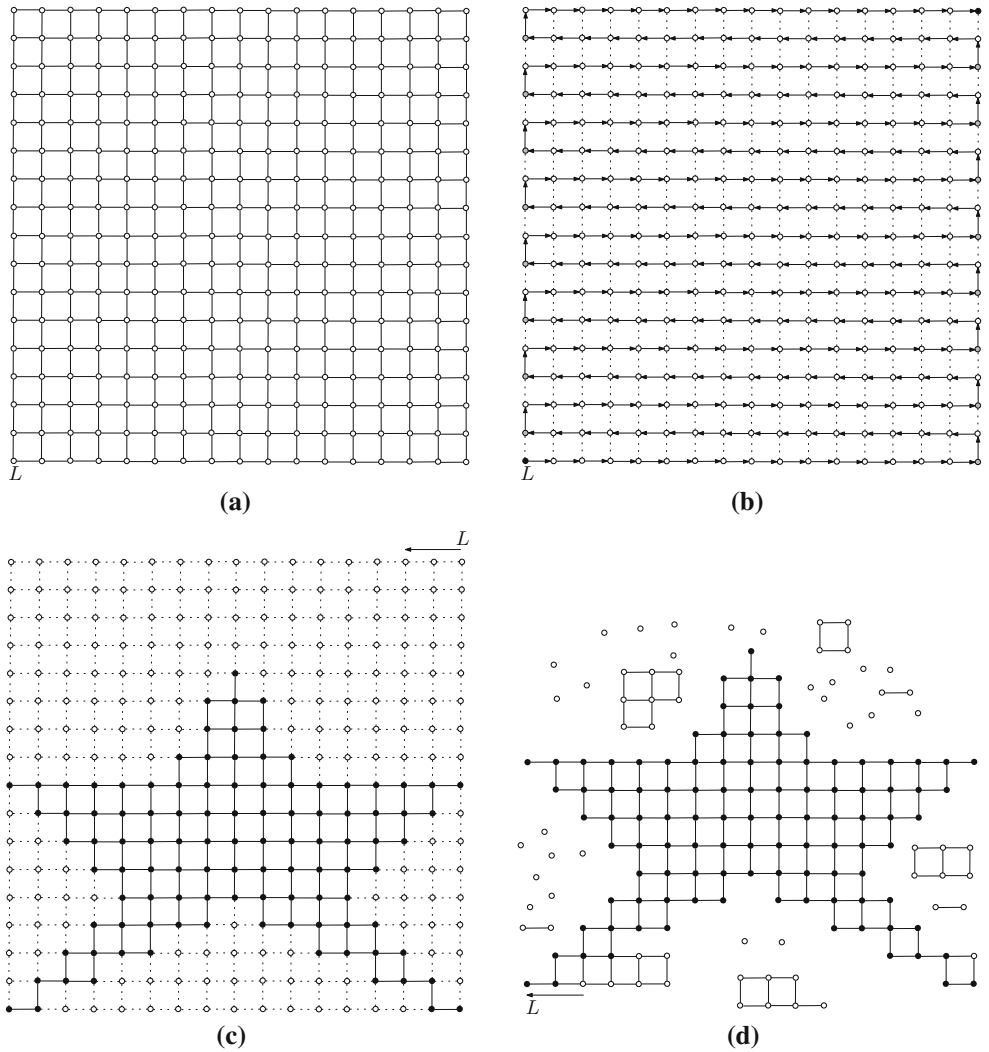

Fig. 7 **a** The $\sqrt{n} \times \sqrt{n}$ square has just been constructed. **b** The virtual tape on the square. The arrows show the direction in which the tape is traversed from left to right (opposite arrows for the opposite direction are not shown). The two endpoints of the tape are marked as black here and the turning points are marked as gray. These facilitate the leader to detect and choose the right action, e.g., turn left twice (equivalently, follow the up port and then the left port) when it arrives at the bottom right corner and wishes to continue on the second row. The indices of the pixels that the procedure assumes follow the order of the tape, that is, the first position of the tape corresponds to pixel 0, the second to pixel 1,..., the last position of the tape to pixel $n-1$. **c** The shape, which looks like a star, has been formed on the square. It consists of the pixels that the TM M decided to be *on*, which are colored black here. All other white pixels are the *off* pixels. The simulations have completed and the leader has just reached the upper right corner and now it starts releasing the shape. To improve visibility, the edges that will eventually be deactivated appear as dotted here. **d** Releasing is almost complete. The leader has reached the bottom left corner and has updated all nodes to the release phase. Any connection involving at least one *off* node (i.e., a white one) will be eventually deactivated

active, they deactivate the connection. Clearly, the only nodes that will remain connected in the solution are the *on* nodes forming the desired connected 2-dimensional shape that M computes. If we additionally require the leader to know when all deactivations have completed and terminate, then we can either (i) have the leader deactivate them itself while moving backwards, also ensuring that it does not remain on a node that will be released, or (ii) have the leader repeatedly explore the final connected shape until it detects that all potential deactivations have occurred.

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

Theorem 3 *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 $(d-1)d = O(d^2) = O(n)$.*

Proof We have to show that for every $n = d^2$, when the protocol is executed on d^2 nodes constructs G_d. From Lemma 2, we have a subroutine that terminates having w.h.p. constructed a $d \times d$ square with a unique leader on the bottom left node. Next, the leader can easily organize the square into a tape of length d^2 that has d stored in binary in its leftmost cells. Moreover, $\mathcal{L}$ is computable, so, by Definition 3, there is a TM M that when executed on the pixels of a $d \times d$ square constructs S_d. The protocol simulates M on the pixels of such a $d \times d$ square thus the result is S_d, which is an *on*/*off* labeled $d \times d$ square the *on* pixels of which form G_d. To perform the simulation, the protocol just feeds M with $(i, d) = (0, d), (1, d), \ldots, (d^2 - 1, d)$, one at a time, simulates M on input (i, d) in space $\Theta(d^2)$, marks the corresponding pixel as *on* or *off* according to M's decision, and moves on to the next input. When $i = d^2$, the square contains G_d and the leader releases G_d by one of the terminating approaches described above and terminates. Observe that, given the guarantees of Lemma 2, the procedure described here is always correct. So, the probability of failure of the whole protocol is just the probability of failure of the initial counting subroutine, thus the protocol succeeds w.h.p.. Finally, the waste is always equal to the number of pixels of the $d \times d$ square that are not part of G_d. Observe now that the waste can never be more than $(d-1)d$, because if it was at least $(d-1)d+1 = d^2-d+1$, then the size of G_d (i.e., the useful space) would be at most $d^2 - (d^2 - d + 1) = d - 1$. But then, connectivity of G_d implies that $max\_dim_{G_d} \leq d - 1$, which contradicts the assumption that $max\_dim_{G_d} = d$. Thus, the worst possible waste is indeed $(d-1)d = O(d^2) = O(n)$. Notice that here the waste of the protocol is equal to the waste of the simulated TM: the protocol just provides the maximum square that fits in the population and the TM determines which nodes will be part of the final shape and which will be thrown away as waste. □

Remark 2 It is worth mentioning that 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, for example the TM corresponding to the Kolmogorov complexity of the shape (which is in turn the Kolmogorov complexity of the desired binary pixel sequence $(s_0, s_1, \ldots, s_{n-1})$). However, in this work we consider systems in which n *is not known in advance*, so the natural approach is to preprogram the nodes with a TM that can work for all n. The protocol must first compute n (w.h.p.) and then simulate the TM on input n to construct a shape of the appropriate size. For example, it could be a TM constructing a star, as in Fig. 7c, such that the size of the star grows as n grows.

Remark 3 The above results can be immediately modified to refer to *patterns* instead of shapes. In fact, observe that the $\sqrt{n} \times \sqrt{n}$ square that has been labeled by *off* and *on* by the TM is already such a (computed) 0/1 pattern. The generic idea to extend this is to keep the same constructor as above and simulate TMs that for every pixel output a color from a set of colors $\mathcal{C}$. Then the resulting square with its nodes labeled from $\mathcal{C}$ is the desired computed pattern and no releasing is required in this case.

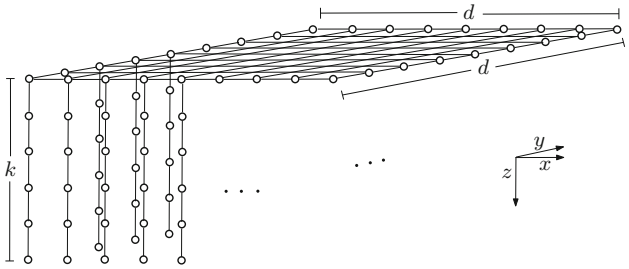

Fig. 8 The constructed $d \times d$ square lies in dimensions x and y. We can think as its "bottom left" corner, its leftmost node in the figure. Every internal intersection point of the square is also a node, but we have not drawn these nodes here to improve visibility. "Below" it, in dimension z, are the d^2 lines of length k each. The protocol executes a distinct simulation of the TM on each of these lines. In particular, on the line attached to pixel i, for all $0 \leq i \leq d^2 - 1$, the protocol simulates the TM on input (i, d)

6.4 Parallelizing the simulations

We now present two approaches for parallelizing the simulations, instead of executing them sequentially one after the other.

6.4.1 Approach 1

One approach is to construct a 2D shape in 3D space, by using the 3D version of our model (that is, the one with 6 ports). The idea is to construct a square as before and have each node in the square to grow its own line in the third dimension to carry out the TM simulation for that node (see Fig. 8).

6.4.2 Approach 2

We now show how to achieve a similar parallelism while avoiding the use of a third dimension. Now the unique leader that knows n, instead of constructing a square, constructs a spanning line of length d^2, say in the x dimension. This line corresponds to a linear expansion of the pixels of the $d \times d$ square of the previous construction. Moreover, the leader creates a seed of length $k-1$ as before, to partition the rest of the nodes into lines of length $k-1$, this time in the y dimension. Each such line will be attached below one of the nodes of the x-line. As before, when all y-lines have been attached, the leader initializes their memories with (i, d), where i is the index of the corresponding pixel (the index of each pixel is now its distance from the left endpoint of the x-line, begin-

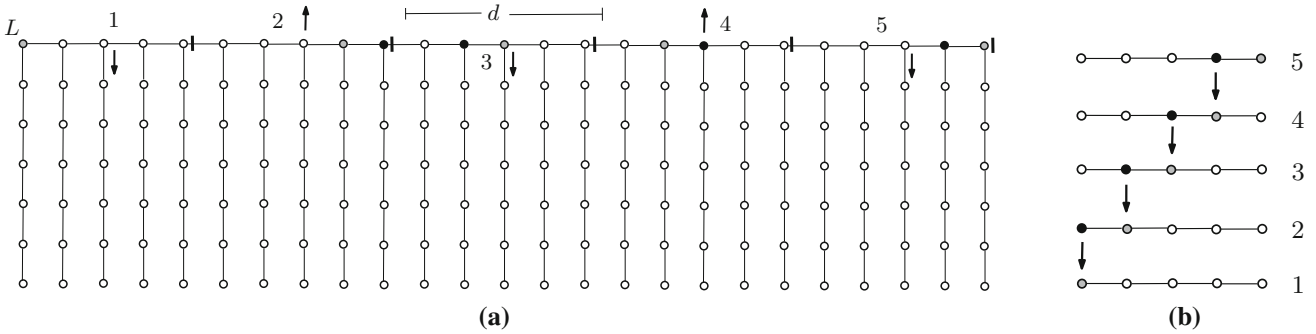

Fig. 9 **a** As in Fig. 8, d^2 lines of length $k-1$ each, are hanging below the d^2 pixels. The difference now is that the pixels have been arranged linearly in dimension x. So, the whole construction is now 2-dimensional. The pixels have been partitioned into equal segments of length d each (see the black vertical delimiters). The numbers represent the indices of the segments counted from left to right. The arrows leaving above or below the segments, indicate which side of the segment should look "downwards" in the square that will be constructed. For example, segment 1 can remain as it is, while segment 2 has to be rotated so that its upper side attaches to the upper side of segment 1. Every segment has been marked by a black and a gray node placed at an appropriate position. **b** The segments have been released in the solution, and now they have to gather together in order to form the square. Each segment knows the correct orientation, i.e., whether it should use its up or down ports, and also it can detect its predecessor row by exploiting the marking. In particular, it attaches to a row if its black mark is above the gray mark of the other row when their orientation is correct and their endpoints are totally aligned

ning from 0 and ending at $d^2 - 1$). 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 auxiliary memories (i.e., the y-lines) and then partitions the x-line into consecutive segments of length d by placing appropriate marks on the boundary nodes (see Fig. 9a). 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 (rows being counted bottom-up). Observe that, in the way the pixels have been indexed, segment 2 should match with its upper side the upper side of segment 1 (that is, segment 2 should rotate 180°), segment 3 should match with its lower side the lower side of segment 2, and so on. In general, if i is even, segment i should match with its upper side to the upper side of segment $i-1$ and, if i is odd, segment i should match with its lower side the 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, it marks nodes i and $i-1$ of the segment counting from left to right (for segment 1 it only marks the leftmost node), and, if i is even, it marks nodes i and $i-1$ of the segment counting from right to left. In this manner, given that segments respect the correct orientation and provided that attachment is only performed when their endpoints match, every segment i uniquely matches to segment $i-1$ because the first mark of i is uniquely aligned with the second mark of $i-1$ (see Fig. 9b). Then the leader releases all segments, one after the other, and it remains on the last segment. The segments are free to move in the solution until they meet 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 position corresponding to its index counting in a zig zag fashion as in the previous sections). The leader periodically walks on its component to detect when it has become equal to the desired square. When this occurs, it initiates as before the releasing phase to isolate the final connected shape consisting of the *on* pixels.

Remark 4 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 in [33], but should be carefully rewritten.

7 Conclusions and further research

There are several interesting open problems related to the findings of this work. A possible refinement of the model could be a distinction between the *speed of the scheduler* and the *internal operation speed of a component*. For example, a connected component will operate in synchronous rounds, where in each round a node observes its neighborhood and its own state and updates its state based on what it sees. Nodes can of course update also the state of their local connections and we may assume that a connection is formed/dropped if both nodes agree on the decision (another possibility is to allow a link change state if at least one of the nodes say so). This distinction between two different "times", though ignored so far in the literature, is very natural because a connected component should operate at a different speed than

it takes for the scheduler to bring two nodes (e.g., of different components, or an isolated node and a node of some component) into contact.

It would be also interesting to consider for the first time a *hybrid model* combining *active* mobility (that is, mobility controlled by the protocol) and *passive* mobility (that is, mobility controlled by the environment as in this paper). For example, it could be a combination of the Nubot model and the model presented in this work. Another very intriguing problem is to give a proof, or strong experimental evidence, of whether the unique leader assumption is necessary for solving counting w.h.p. (see Sect. 5.2). If true, it would imply that there is no analogue of Theorem 1 if all processes are identical. 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 Sect. 5.1 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.

It would be also important to develop models (e.g., variations of the one proposed here) that take other real physical considerations into account. In this work, we have restricted attention on some geometric 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*. The latter also indicates that the construction and the environment that the construction inhabits cannot be studied in isolation. Instead, the two will constantly affect each other, the optimal output will highly depend on the optimality that the environment allows and also the environment may highly and continuously affect the construction process. The capability of the environment to affect the construction process suggests many robustness issues. Imagine an environment that can at any given time break an active link with some (small) probability (a similar question was also posed to the author during his talk at PODC '14 by some attendee, who the author would like to thank). Under such a perpetual setback no construction can ever stabilize. However, we may still be able to have a construction that constantly exists in the population by evolving and self-replicating.

In the same spirit, 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 the beginning)? What knowledge about the whole shape should the nodes have to be able to reconstruct missing parts of it? Finally, it would be interesting to study in depth the shape self-replication problem in our model in 2 and 3 dimensions (possibly by adjusting known techniques on replication [4,11,26,28]). Some of our preliminary results, show that replication of 2D shapes in two dimensions is possible by "squaring" the original shape, then copying one column at a time, and shifting the copy of the column to the right to create a replica to the right of the original shape. It is worth mentioning that several related problems have been studied in the literature of algorithmic self-assembly, so future work in distributed network/shape construction will greatly benefit from taking into account those developments, trying to adjust them where not directly applicable, and from highlighting the differences and similarities between the various related models originating from different research areas.

Acknowledgements The author would like to thank Dimitrios Amaxilatis and Marios Logaras for implementing (in Java) the probabilistic counting protocol of Sect. 5.1 and experimentally verifying its correctness and also David Doty for a few fruitful discussions on the same protocol at the very early stages of this work. Finally, the author would like to thank the anonymous reviewers of this article and of its preliminary version. Their thorough reading and comments have helped the author to improve his work substantially.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**, 235–253 (2006)
2. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distrib. Comput. **21**, 183–199 (2008)
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**, 279–304 (2007)
4. Abel, Z., Benbernou, N., Damian, M., Demaine, E.D., Demaine, M.L., Flatland, R., Kominers, S.D., Schweller, R.: Shape replication through self-assembly and RNase enzymes. In: Proceedings

of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1045–1064. SIAM (2010)
5. Aloupis, G., Benbernou, N., Damian, M., Demaine, E.D., Flatland, R., Iacono, J., Wuhrer, S.: Efficient reconfiguration of lattice-based modular robots. Comput. Geom. **46**, 917–928 (2013)
6. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**, 1021–1024 (1994)
7. Angluin, D.: Local and global properties in networks of processors. In: Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC), pp. 82–93. ACM (1980)
8. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Garbinato, B., Miranda, H., Rodrigues, L. (eds.) Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer, Berlin (2009)
9. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics, vol. 19. Wiley, New York (2004)
10. Beauquier, J., Burman, J., Clement, J., Kutten, S.: On utilizing speed in networks of mobile agents. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), pp. 305–314. ACM (2010)
11. Chalk, C., Demaine, E.D., Demaine, M.L., Martinez, E., Schweller, R., Vega, L., Wylie, T.: Universal shape replicators via self-assembly with attractive and repulsive forces. In: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 225–238. SIAM (2017)
12. Chen, H.-L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Nat. Comput. **13**, 517–534 (2014)
13. Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A., Spirakis, P.G.: Passively mobile communicating machines that use restricted space. Theor. Comput. Sci. **412**, 6469–6483 (2011)
14. Chen, M., Xin, D., Woods, D.: Parallel computation using active self-assembly. Nat. Comput. **14**, 225–250 (2015)
15. Derakhshandeh, Z., Dolev, S., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Brief announcement: amoebot—a new model for programmable matter. In: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 220–222. ACM (2014)
16. Das, S., Flocchini, P., Santoro, N., Yamashita, M.: Forming sequences of geometric patterns with oblivious mobile robots. Distrib. Comput. **28**, 131–145 (2015)
17. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal shape formation for programmable matter. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 289–299. ACM (2016)
18. Dolev, S., Gmyr, R., Richa, A.W., Scheideler, C.: Ameba-inspired self-organizing particle systems. arXiv preprint arXiv:1307.4259 (2013)
19. Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. arXiv preprint arXiv:1705.03538 (2017)
20. Doty, D.: Theory of algorithmic self-assembly. Commun. ACM **55**, 78–88 (2012)
21. Doty, D.: Timing in chemical reaction networks. In: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 772–784 (2014)
22. Ehrenfest, P., Ehrenfest-Afanassjewa, T.: Über zwei bekannte einwände gegen das boltzmannsche h-theorem. Phys. Zeit. **8**, 311–314 (1907)
23. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1, 3rd edn. Wiley, New York (1968). (Revised Printing)
24. Goldstein, S.C., Campbell, J.D., Mowry, T.C.: Programmable matter. Computer **38**, 99–101 (2005)
25. Guerraoui, R., Ruppert, E.: Names trump malice: tiny mobile agents can tolerate byzantine failures. In: 36th International Colloquium on Automata, Languages and Programming (ICALP), Volume 5556 of LNCS, pp. 484–495. Springer (2009)
26. Hendricks, J., Patitz, M.J., Rogers, T.A.: Replication of arbitrary hole-free shapes via self-assembly with signal-passing tiles. In: International Conference on Unconventional Computation and Natural Computation, pp. 202–214. Springer (2015)
27. Kac, M.: Random walk and the theory of brownian motion. Am. Math. Mon. **54**(7), 369–391 (1947)
28. Keenan, A., Schweller, R., Zhong, X.: Exponential replication of patterns in the signal tile assembly model. Nat. Comput. **14**, 265–278 (2015)
29. Lynch, N.A.: Distributed Algorithms, 1st edn. Morgan Kaufmann Inc, San Francisco (1996)
30. Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theor. Comput. Sci. **412**, 2434–2450 (2011)
31. Michail, O., Chatzigiannakis, I., Spirakis, P.G.: New models for population protocols. In: Lynch, N.A. (ed.) Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, San Rafael (2011)
32. Michail, O.: Terminating distributed construction of shapes and patterns in a fair solution of automata. In: Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC), pp. 37–46. ACM (2015)
33. Michail, O., Spirakis, P.G.: Simple and efficient local codes for distributed stable network construction. Dist. Comput. **29**(3), 207–237 (2016). doi:10.1007/s00446-015-0257-4v
34. Michail, O., Spirakis, P.G.: Terminating population protocols via some minimal global knowledge assumptions. J. Parallel Distrib. Comput. (JPDC) **81**, 1–10 (2015)
35. Michail, O., Spirakis, P.G.: Elements of the theory of dynamic networks. Commun. ACM (2017). https://livrepository.liverpool.ac.uk/3006836/ **(To appear)**
36. Michail, O., Skretas, G., Spirakis, P.G.: On the transformation capability of feasible mechanisms for programmable matter. In: Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP), pp. 136:1–136:15 (2017)
37. Padilla, J.E., Patitz, M.J., Schweller, R.T., Seeman, N.C., Summers, S.M., Zhong, X.: Asynchronous signal passing for tile self-assembly: fuel efficient computation and efficient assembly of shapes. Int. J. Found. Comput. Sci. **25**, 459–488 (2014)
38. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**, 795–799 (2014)
39. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), pp. 459–468 (2000)
40. Schiff, J.L.: Cellular Automata: A Discrete View of the World, vol. 45. Wiley, New York (2011)
41. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**, 615–633 (2008)
42. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: formation of geometric patterns. SIAM J. Comput. **28**, 1347–1363 (1999)
43. Woods, D., Chen, H.-L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, pp. 353–354. ACM (2013). Full version: arXiv preprint arXiv:1301.2626
44. Winfree, E.: Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology, June (1998)