Distributed Transformations of Hamiltonian Shapes based on Line Moves

Abdullah Almethen, Othon Michail, and Igor Potapov

Department of Computer Science, University of Liverpool, Liverpool, UK
{A.Almethen, Othon.Michail, Potapov}@liverpool.ac.uk

Abstract. We consider a discrete system of n simple indistinguishable devices, called *agents*, forming a *connected* shape S_I on a two-dimensional square grid. Agents are equipped with a linear-strength mechanism, called a *line move*, by which an agent can push a whole line of consecutive agents in one of the four directions in a single time-step. We study the problem of transforming an initial shape S_I into a given target shape S_F via a finite sequence of line moves in a distributed model, where each agent can observe the states of nearby agents in a Moore neighbourhood. Our main contribution is the first distributed connectivity-preserving transformation that exploits line moves within a total of $O(n \log_2 n)$ moves, which is asymptotically equivalent to that of the best-known centralised transformations. The algorithm solves the *line formation problem* that allows agents to form a final straight line S_L, starting from any shape S_I, whose *associated graph* contains a Hamiltonian path.

Keywords: Line movement · Discrete transformations · Shape formation · Reconfigurable robotics · Programmable matter · Distributed algorithms

1 Introduction

The explosive growth of advanced technology over the last few decades has contributed significantly towards the development of a wide variety of distributed systems consisting of large collections of tiny robotic-units, known as *monads*. These monads are able to move and communicate with each other by being equipped with microcontrollers, actuators and sensors. However, each monad is severely restricted and has limited computational capabilities, such as a constant memory and lack of global knowledge. Further, monads are typically homogeneous, anonymous and indistinguishable from each other. Through a simple set of rules and local actions, they collectively act as a single unit and carry out several complex tasks, such as transformations and explorations.

In this context, scientists from different disciplines have made great efforts towards developing innovative, scalable and adaptive collective robotic systems.

The full version of the paper with all omitted details is available on arXiv at: http://arxiv.org/abs/2108.08953.

This vision has recently given rise to the area of programmable matter, first proposed by Toffoli and Margolus [35] in 1991, referring to any kind of materials that can algorithmically change their physical properties, such as shape, colour, density and conductivity through transformations executed by an underlying program. This newborn area has been of growing interest lately both from a theoretical and a practical viewpoint.

One can categorise programmable matter systems into *active* and *passive*. Entities in the passive systems have no control over their movements. Instead, they move via interactions with the environment based on their own structural characteristics. Prominent examples of research on passive systems appear in the areas of population protocols [7, 28, 29], DNA computing [1, 8] and tile self-assembly [15, 33, 37]. On the other hand, the active systems allow computational entities to act and control their movements in order to accomplish a given task, which is our primary focus in this work. The most popular examples of active systems include metamorphic systems [19, 30, 36], swarm/mobile robotics [10, 21, 31, 34, 39], modular self-reconfigurable robotics [5, 22, 40] and recent research on programmable matter [12, 13]. Moreover, those robotic systems have received an increasing attention from the the engineering research community, and hence many solutions and frameworks have been produced for milli/micro-scale [9, 23, 26] down to nanoscale systems [16, 32].

Shape transformations (sometimes called *pattern formation*) can be seen as one of the most essential goals for almost every system among the vast variety of robotic systems including programmable matter and swarm robotic systems. In this work, we focus on a system of a two-dimensional square grid containing a collection of entities typically connected to each other and forming an initial connected shape S_I. Each entity is equipped with a linear-strength mechanism that can push an entire line of consecutive entities one position in a single time-step in a given direction of a grid. The goal is to design an algorithm that can transform an initial shape S_I into a given target shape S_F through a chain of permissible moves and without losing the connectivity. That is, in each intermediate configuration we always want to guarantee that the graphs induced by the nodes occupied by the entities are connected. The connectivity-preservation is an important assumption for many practical applications, which usually require energy for data exchange as well as the implementation of various locomotion mechanisms.

1.1 Related Work

Many models of centralised or distributed coordination have been studied in the context of shape transformation problems. The assumed mechanisms in those models can significantly influence the efficiency and feasibility of shape transformations. For example, the authors of [2, 17–19, 27] consider mechanisms called sliding and rotation by which an agent can move and turn over neighbours through empty space. Under these models of individual movements, Dumitrescu and Pach [17] and Michail *et al.* [27] present universal transformations for any pair of connected shapes (S_I, S_F) of the same size to each other. By restricting to rotation only, the authors in [27] proved that the decision problem of transformability is in $\mathbf{P}$; however, with a constant number of extra seed nodes connectivity preserving transformation can be completed with $\Omega(n^2)$ moves [27].

The alternative less costly reconfiguration solutions can be designed by employing some parallelism, where multiple movements can occur at the same time, see theoretical studies in [11, 14] and more practical implementation in [34]. Moreover, it has been shown that there exists a universal transformation with rotation and sliding that converts any pair of connected shapes to each other within $O(n)$ parallel moves in the worst case [27]. Also fast reconfiguration might be achieved by exploiting actuation mechanisms, where a single agent is now equipped with more strength to move many entities in parallel in a single time-step. A prominent example is the linear-strength model of Aloupis *et al.* [5, 6], where an entity is equipped with arms giving it the ability to extend/extract a neighbour, a set of individuals or the whole configuration in a single operation. Another elegant approach by Woods *et al.* [38] studied another linear-strength mechanism by which an entity can drag a chain of entities parallel to one of the axes directions.

A more recent study along this direction is shown in [4], and introduces the *line-pushing* model. In this model, an individual entity can push the whole line of consecutive entities one position in a given direction in a single time-step. As we shall explain, this model generalises some existing constant-strength models with a special focus on exploiting its parallel power for fast and more general transformations. Apart from the purely theoretical benefit of exploring fast reconfigurations, this model also provides a practical framework for more efficient reconfigurations in real systems. For example, self-organising robots could be reconfiguring into multiple shapes in order to pass through canals, bridges or corridors in a mine. In another domain, individual robots could be containers equipped with motors that can push an entire row to manage space in large warehouses. Another future application could be a system of very tiny particles injected into a human body and transforming into several shapes in order to efficiently traverse through the veins and capillaries and treat infected cells.

This model is capable of simulating some constant-strength models. For example, it can simulate the sliding and rotation model [17, 27] with an increase in the worst-case running time only by a factor of 2. This implies that all universality and reversibility properties of individual-move transformations still hold true in this model. Also, the model allows the diagonal connections on the grid. Several sub-quadratic time centralised transformations have been proposed, including an $O(n\sqrt{n})$-time universal transformation that preserves the connectivity of the shape during its course [3]. By allowing transformations to disconnect the shape during their course, there exists a centralised universal transformation that completes within $O(n \log n)$ time.

Another recent related set of models studied in [10, 20, 24] consider a single robot which moves over a static shape consisting of tiles and the goal is for the robot to transform the shape by carrying one tile at a time. In those systems, the single robot which controls and carries out the transformation is typically modelled as a finite automaton. Those models can be viewed as partially centralised as on one hand they have a unique controller but on the other hand that controller is operating locally and suffering from a lack of global information.

1.2 Our Contribution

In this work, our main objective is to give the first distributed transformations for programmable matter systems implementing the linear-strength mechanism of the model of line moves. All existing transformations for this model are centralised, thus, even though they reveal the underlying transformation complexities, they are not directly applicable to real programmable matter systems. Our goal is to develop distributed transformations that, if possible, will preserve all the good properties of the corresponding centralised solutions. These include the *move complexity* (i.e., the total number of line moves) of the transformations and their ability to preserve the connectivity of the shape throughout their course.

However, there are considerable technical challenges that one must deal with in order to develop such a distributed solution. As will become evident, the lack of global knowledge of the individual entities and the condition of preserving connectivity greatly complicate the transformation, even when restricted to special families of shapes. Timing is an essential issue as the line needs to know when to start and stop pushing. When moving or turning, all agents of the line must follow the same route, ensuring that no one is being pushed off. There is an additional difficulty due to the fact that agents do not automatically know whether they have been pushed (but it might be possible to infer this through communication and/or local observation).

Consider a discrete system of n simple indistinguishable devices, called *agents*, forming a connected shape S_I on a two-dimensional square grid. Agents act as finite-state automata (i.e., they have constant memory) that can observe the states of nearby agents in a Moore neighbourhood (i.e., the eight cells surrounding an agent on the square gird). They operate in synchronised Look-Compute-Move (LCM) cycles on the grid. All communication is local, and actuation is based on this local information as well as the agent's internal state.

Let us consider a very simple distributed transformation of a diagonal line shape S_D into a straight line S_L, $|S_D| = |S_L| = n$, in which all agents execute the same procedure in parallel synchronous rounds. In general, the diagonal appears to be a hard instance because any parallelism related to line moves that might potentially be exploited does not come for free. Initially, all agents are occupying the consecutive diagonal cells on the grid $(x_1, y_1), (x_1+1, y_1+1), \ldots, (x_1+n-1, y_1+n-1)$. In each round, an agent $p_i = (x, y)$ moves one step down if $(x-1, y-1)$ is occupied, otherwise it stays still in its current cell. After $O(n)$ rounds, all agents form S_L within a total number of $1+2+\ldots+n = O(n^2)$ moves, while preserving connectivity during the transformation (throughout, connectivity includes horizontal, vertical, and diagonal adjacency). See Figure 1.

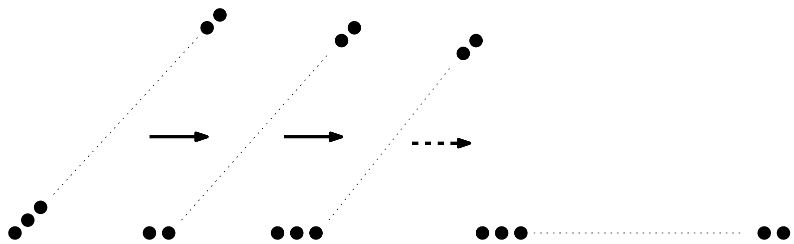


Fig. 1. A simulation of the simple procedure. From left to right, rounds $0, 1, 2, \ldots, n$.

The above transformation, even though time-optimal has a move complexity asymptotically equal to the worst-case single-move distance between S_I and S_F. This is because it always moves individual agents, thus not exploiting the inherent parallelism of line moves. Our goal, is to trade time for number of line moves in order to develop alternative distributed transformations which will complete within a sub-quadratic number of moves. Given that actuation is a major source of energy consumption in real programmable matter and robotic systems, moves minimisation is expected to contribute in the deployment and implementation of energy-efficient systems.

We already know that there is a centralised $O(n \log n)$-move connectivity-preserving transformation, working for a large family of connected shapes [3]. That centralised strategy transforms a pair of connected shapes (S_I, S_F) of the same order (i.e., the number of agents) to each other, when the associated graphs of both shapes contain a Hamiltonian path (see also Itai *et al.* [25] for rectilinear Hamiltonian paths), while preserving connectivity during the transformation. This approach initially forms a line from one endpoint of the Hamiltonian path, then flattens all agents along the path gradually via line moves, while successively doubling the line length in each round. After $O(n \log n)$ moves, it arrives at the final straight line S_L of length n, which can be then transformed into S_F by reversing the transformation of S_F into S_L, within the same asymptotic number of moves.

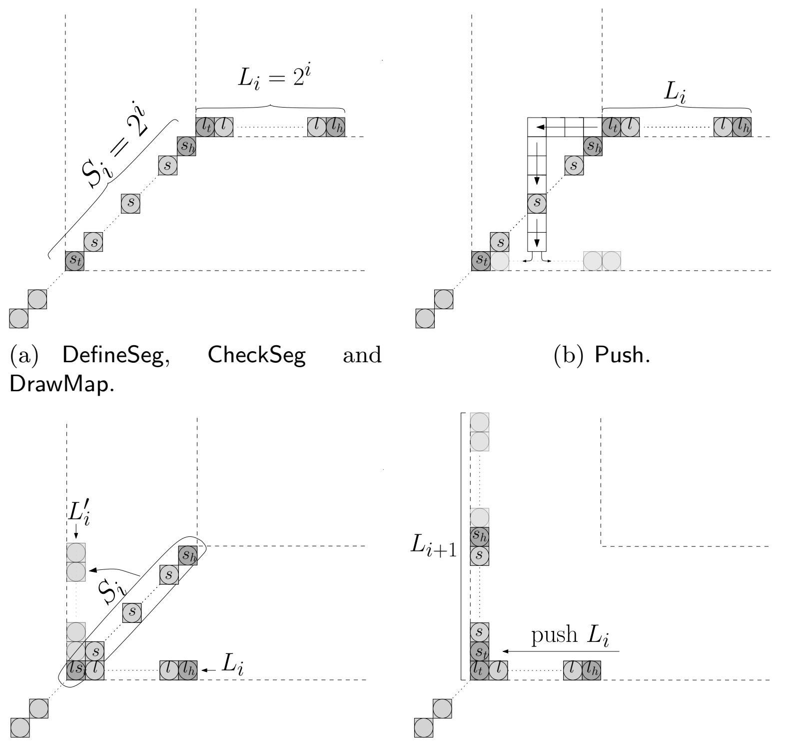
In this work, we introduce the first distributed transformation exploiting the linear-strength mechanism of the *line-pushing* model. It provides a solution to the line formation problem, that is, for any initial Hamiltonian shape S_I, form a final straight line S_L of the same order. It is essentially a distributed implementation of the centralised Hamiltonian transformation of [3]. We show that it preserves the asymptotic bound of $O(n \log n)$ line moves (which is still the best-known centralised bound), while keeping the whole shape connected throughout its course. This is the first step towards distributed transformations between any pair of Hamiltonian shapes. The inverse of this transformation (S_L into S_I) appears to be a much more complicated problem to solve as the agents need to somehow know an encoding of the shape to be constructed and that in contrast to the centralised case, reversibility does not apply in a straightforward way. Hence, the reverse of this transformation (S_L into S_I) is left as a future research direction.

We restrict attention to the class of Hamiltonian shapes. This class, apart from being a reasonable first step in the direction of distributed transformations in the given setting, might give insight to the future development of universal distributed transformations, i.e., distributed transformations working for any possible pair of initial and target shapes. This is because geometric shapes tend to have long simple paths. For example, the length of their longest path is provably at least $\sqrt{n}$. We here focus on developing efficient distributed transformations for the extreme case in which the longest path is a Hamiltonian path. However, one might be able to apply our Hamiltonian transformation to any pair of shapes, by, for example, running a different or similar transformation along branches of the longest path and then running our transformation on the longest path. We leave how to exploit the longest path in the general case (i.e., when initial and target shapes are not necessarily Hamiltonian) as an interesting open problem.

We assume that a pre-processing phase provides the Hamiltonian path, i.e., a global sense of direction is made available to the agents through a labelling of their local ports (e.g., each agent maintains two local ports incident to its predecessor and successor on the path). Similar assumptions exist in the literature of systems of complex shapes that contain a vast number of self-organising and limited entities. A prominent example is [34] in which the transformation relies on an initial central phase to gain some information about the number of entities in the system.

Now, we are ready to sketch a high-level description of the transformation. A Hamiltonian path P in the initial shape S_I starts with a head on one endpoint labelled l_h, which is leading the process and coordinating all the sub-procedures during the transformation. The transformation proceeds in $\log n$ phases, each consisting of six sub-phases (or sub-routines) and every sub-phase running for one or more synchronous rounds. Figure 2 gives an illustration of a phase of this transformation when applied on the diagonal line shape. Initially, the head l_h forms a trivial line of length 1. By the beginning of each phase i, $0 \leq i \leq \log n - 1$, there exists a line L_i starting from the head l_h and ending at a tail l_t with $2^i - 2$ internal agents labelled l in between. By the end of phase i, L_i will have doubled its length as follows.

First, it identifies the next 2^i agents on P. These agents are forming a segment S_i which can be in any configuration. To do that, the head emits a signal which is then forwarded by the agents along the line. Once the signal arrives at S_i, it will be used to re-label S_i so that it starts from a head in state s_h, has $2^i - 2$ internal agents in state s, and ends at a tail s_t; this completes the DefineSeg sub-phase. Then, l_h calls CheckSeg in order to check whether the line defined by S_i is in line or perpendicular to L_i. This can be easily achieved through a moving state initiated at L_i and checking for each agent of S_i its local directions relative to its neighbours. If the check returns true, then l_h starts a new round $i + 1$ and calls Merge to combine L_i and S_i into a new line L_{i+1} of length 2^{i+1}. Otherwise, l_h proceeds with the next sub-phase, DrawMap.



(a) DefineSeg, CheckSeg and DrawMap.

(b) Push.

(c) RecursiveCall.

(d) Merge.

Fig. 2. From [3], a snapshot of phase i of the Hamiltonian transformation on the shape of a diagonal line. Each occupied cell shows the current label state of an agent. Light grey cells show ending cells of the corresponding moves.

In DrawMap, l_h designates a route on the grid through which L_i pushes itself towards the tail s_t of S_i. It consists of two primitives: ComputeDistance and CollectArrows. In ComputeDistance, the line agents act as a distributed counter to compute the Manhattan distance between the tails of L_i and S_i. In CollectArrows, the local directions are gathered from S_i's agents and distributed into L_i's agents, which collectively draw the route map. Once this is done, L_i becomes ready to move and l_h can start the Push sub-phase. During pushing, l_h and l_t synchronise the movements of L_i's agents as follows: (1) l_h pushes while l_t is guiding the other line agents through the computed route and (2) both are coordinating any required swapping of states with agents that are not part of L_i but reside in L_i's trajectory. Once L_i has traversed the route completely, l_h calls RecursiveCall to apply the general procedure recursively on S_i in order to transform it into a line L'_i. Figure 3 shows a graphical illustration of the core recursion on the special case of a diagonal line shape. Finally, the agents of L_i and L'_i combine into a new straight line L_{i+1} of 2^{i+1} agents through the Merge sub-procedure. Then, the head l_h of L_{i+1} begins a new phase $i+1$.

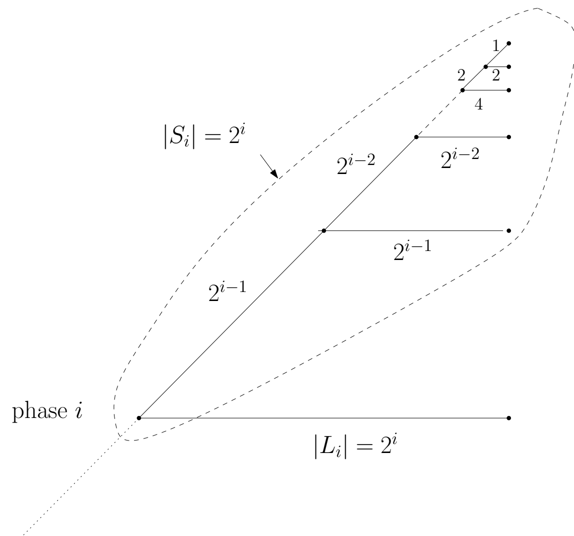


Fig. 3. A zoomed-in picture of the core recursive technique RecursiveCall in Figure 2(c).

Section 2 formally defines the model and the problem under consideration. Section 3 presents our distributed connectivity-preserving transformation that solves the line formation problem for Hamiltonian shapes, achieving a total of $O(n \log n)$ line moves.

2 Model

We consider a system consisting of n agents forming a connected shape S on a two-dimensional square grid in which each agent $p \in S$ occupies a unique cell $cell(p) = (x, y)$, where x indicates columns and y represents rows. Throughout, an agent shall also be referred to by its coordinates. Each cell (x, y) is surrounded by eight adjacent cells in each cardinal and ordinal direction, (*N*, *E*, *S*, *W*, *NE*, *NW*, *SE*, *SW*). At any time, a cell (x, y) can be in one of two states, either empty or occupied. An agent $p \in S$ is a *neighbour* of (or *adjacent* to) another agent $p' \in S$, if p' occupies one of the eight adjacent cells surrounding p, that is their coordinates satisfy $p'_x - 1 \leq p_x \leq p'_x + 1$ and $p'_y - 1 \leq p_y \leq p'_y + 1$. For any shape S, we associate a graph $G(S) = (V, E)$ defined as follows, where V represents agents of S and E contains all pairs of adjacent neighbours, i.e, $(p, p') \in E$ iff p and p' are neighbours in S. We say that a shape S is connected iff $G(S)$ is a connected graph. The *distance* between agents $p \in S$ and $p' \in S$ is defined as the Manhattan distance between their cells, $\Delta(p, p') = |p_x - p'_x| + |p_y - p'_y|$. A shape S is called *Hamiltonian shape* iff $G(S)$ contains a Hamiltonian path, i.e., a path starting from some $p \in S$, visiting every agent in S and ending at some $p' \in S$, where $p \neq p'$.

In this work, each agent is equipped with the linear-strength mechanism introduced in [4], called the *line pushing mechanism*. A line L consists of a sequence of k agents occupying consecutive cells on the grid, say w.l.o.g, $L = (x, y), (x+1, y), \ldots, (x+k-1, y)$, where $1 \leq k \leq n$. The agent $p \in L$ occupying (x, y) is capable of performing an operation of a **line move** by which it can push all agents of L one position rightwards to positions $(x+1, y), (x+2, y), \ldots, (x+k, y)$ in a single time-step. The *line moves* towards the "down", "left" and "up" directions are defined symmetrically by rotating the system 90°, 180° and 270° clockwise, respectively. From now on, this operation may be referred to as *move*, *movement* or *step*. We call the number of agents in S the *size* or *order* of the shape, and throughout this work all logarithms are to the base 2.

We assume that the agents share a sense of orientation through a consistent labelling of their local ports. Agents do not know the size of S in advance neither they have any other knowledge about S. Each agent has a constant memory (of size independent of n) and a local visibility mechanism by which it observes the states of its eight neighbouring cells simultaneously. The agents act as finite automata operating in synchronous rounds consisting of LCM steps. Thus, in every discrete round, an agent observes its own state and for each of its eight adjacent cells, checks whether it is occupied or not. For each of those occupied, it also observes the state of the agent occupying that cell. Then, the agent updates its state or leaves it unchanged and performs a *line move* in one direction $d \in \{$*up, down, right, left*$\}$ or stays still. A *configuration* C of the system is a mapping from $\mathbb{Z}^2_{\geq 0}$ to $\{0\} \cup Q$, where Q is the state space of agents. We define $S(C)$ as the shape of configuration C, i.e., the set of coordinates of the cells occupied in S. Given a configuration C, the LCM steps performed by all agents in the given round, yield a new configuration C' and the next round begins. If at least one move was performed, then we say that this round has transformed $S(C)$ to $S(C')$.

Throughout this work, we assume that the initial shape S_I is Hamiltonian and the final shape is a straight line S_L, where both S_I and S_L have the same order. We also assume that a pre-elected leader is provided at one endpoint of the Hamiltonian path of S_I. It is made available to the agents in the distributed way that each agent p_i knows the local port leading to its predecessor p_{i-1} and its successor p_{i+1}, for all $1 \leq i \leq n$.

An agent $p \in S$ is defined as a 5-tuple (X, M, Q, δ, O), where Q is a finite set of states, X is the input alphabet representing the states of the eight cells that surround an agent p on the square grid, so $|X| = |Q|^8$, $M = \{\uparrow, \downarrow, \rightarrow, \leftarrow, none\}$ is the output alphabet corresponding to the set of moves, a transition function $\delta : Q \times X \rightarrow Q \times M$ and the output function $O : \delta \times X \rightarrow M$.

We now formally define the problem considered in this work.

HamiltonianLine**.** Given any initial Hamiltonian shape S_I, the agents must form a final straight line S_L of the same order from S_I via line moves while preserving connectivity throughout the transformation.

3 The Distributed Hamiltonian Transformation

In this section, we develop a distributed algorithm exploiting line moves to form a straight line S_L from an initial connected shape S_I which is associated to a graph that contains a Hamiltonian path. As we will argue, this strategy performs $O(n \log n)$ moves, i.e., it is as efficient w.r.t. moves as the best-known centralised transformation [3], and completes within $O(n^2 \log n)$ rounds, while keeping the whole shape connected during its course.

We assume that through some pre-processing the Hamiltonian path P of the initial shape S_I has been made available to the n agents in a distributed way. P starts and ends at two agents, called the head p_1 and the tail p_n, respectively. The head p_1 is leading the process (as it can be used as a pre-elected unique leader) and is responsible for coordinating and initiating all procedures of this transformation. In order to simplify the exposition, we assume that n is a power of 2; this can be easily dropped later. The transformation proceeds in $\log n$ phases, each of which consists of six sub-phases (or sub-routines). Every sub-phase consist of one or more synchronous rounds. The transformation starts with a trivial line of length 1 at the head's endpoint, then it gradually flattens all agents along P gradually while successively doubling its length, until arriving at the final straight line S_L of length n.

A state $q \in Q$ of an agent p will be represented by a vector with seven components $(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$. The first component c_1 contains a label λ of the agent from a finite set of labels Λ, c_2 is the transmission state that holds a string of length at most three, where each symbol of the string can either be a special mark w from a finite set of marks W or an arrow direction $a \in A = \{\rightarrow, \leftarrow, \downarrow, \uparrow, \nwarrow, \nearrow, \swarrow, \searrow\}$ and c_3 will store a symbol from c_2's string, i.e., a special mark or an arrow. The local Hamiltonian direction $a \in A$ of an agent p indicating predecessor and successor is recorded in c_4, the counter state c_5 holds a bit from $\{0, 1\}$, c_6 stores an arrow $a \in A$ for map drawing (as will be explained later) and finally c_7 is holding a pushing direction $d \in M$. The "$\cdot$" mark indicates an empty component; a non-empty component is always denoted by its state. An agent p may be referred to by its label $\lambda \in \Lambda$ (i.e., by the state of its c_1 component) whenever clear from context.

By the beginning of phase i, $0 \leq i \leq \log n - 1$, there exists a terminal straight line L_i of 2^i active agents occupying a single row or column on the grid, starting with a head labelled l_h and ending at a tail labelled l_t, while internal agents have label l. All agents in the rest of the configuration are inactive and labelled k. During phase i, the head l_h leads the execution of six sub-phases, DefineSeg, CheckSeg, DrawMap, Push, RecursiveCall and Merge. For simplicity and due to space restrictions, we shall only mention the affected components of the state of the agents. The high-level idea of this strategy has already been provided in Section 1.2 and illustrated in Figure 2, therefore we can now immediately proceed with the detailed description of each sub-phase.

DefineSeg. The line head l_h transmits a special mark "Ⓗ" to go through all active agents in the Hamiltonian path P. It updates its transmission component

c_2 as follows: $\delta(l_h, \cdot, \cdot, a \in A, \cdot, \cdot, \cdot) = (l_h, \text{Ⓗ}, \cdot, a \in A, \cdot, \cdot, \cdot)$. This is propagated by active agents by always moving from a predecessor p_i to a successor p_{i+1}, until it arrives at the first inactive agent with label k, which then becomes active and the head of its segment by updating its label as $\delta(k, \text{Ⓗ}, \cdot, a \in A, \cdot, \cdot, \cdot) = (s_h, \cdot, \cdot, a \in A, \cdot, \cdot, \cdot)$. Similarly, once a line agent p_i passes "Ⓗ" to p_{i+1}, it also initiates and propagates its own mark "①" to activate a corresponding segment agent s. The line tail l_t emits "Ⓣ" to activate the segment tail s_t, which in turn bounces off a special end mark "$\otimes$" announcing the end of DefineSeg. By that time, the next segment S_i consisting of 2^i agents, starting from a head labelled s_h, ending at a tail s_t and having $2^i - 2$ internal agents with label s, has been defined. The "$\otimes$" mark is propagated back to the head l_h along the active agents, by always moving from p_{i+1} to p_i.

Lemma 1. *DefineSeg correctly activates all agents of S_i in $O(n)$ rounds.*

CheckSeg. Once l_h observes "$\otimes$", it propagates its own local direction stored in component $c_4 = a \in A$ by updating $c_2 \leftarrow c_4$. Then, all active agents on the path forward a from p_i to p_{i+1} via their transmission components. Whenever a p_i having a local direction $c_4 = a' \in A$ observes $a' \neq a$, it combines a with its local direction a' and changes its transmission component to $c_2 \leftarrow aa'$. After that, if a p'_i having $c_4 = a'' \in A$ observes $a'' \neq a'$, it updates its transmission component into a negative mark, $c_2 \leftarrow \neg$. All signals are to be reflected by the segment tail s_t back to l_h, which acts accordingly as follows: (1) starts the next sub-phase DrawMap if it observes "$\neg$", (2) calls Merge to combine the two perpendicular lines if it observes aa' or (3) begins a new phase $i + 1$ if it receives back its local direction a.

Lemma 2. *CheckSeg correctly checks the configuration of S_i in $O(n)$ rounds.*

DrawMap. This sub-phase computes the Manhattan distance $\Delta(l_t, s_t)$ between the line tail l_t and the segment tail s_t, by exploiting ComputeDistance in which the line agents implement a distributed binary counter. First, the head l_h broadcasts "Ⓒ" to all active agents, asking them to commence the calculation of the distance. Once a segment agent p_i observes "Ⓒ", it emits one increment mark "$\oplus$" if its local direction is cardinal or two sequential increment marks if it is diagonal. The "$\oplus$" mark is forwarded from p_i to p_{i-1} back to the head l_h. Correspondingly, the line agents are arranged to collectively act as a distributed binary counter, which increases by 1 bit per increment mark, starting from the least significant at l_t. When a line agent observes the last "$\oplus$" mark, it sends a special mark "①" if $\Delta(l_t, s_t) \leq |L_i|$ or "②" if $\Delta(l_t, s_t) > |L_i|$ back to l_h. As soon as l_h receives "①" or "②", it calls CollectArrows to draw a route that can be either heading directly to s_t or passing through the middle of S_i towards s_t. In CollectArrows, l_h emits "$\leftrightarrows$" to announce the collection of local directions (arrows) from S_i. When "$\leftrightarrows$" arrives at a segment agent, it then propagates its local direction stored in c_4 back towards l_h. Then, the line agents distribute and rearrange S_i's local directions via several primitives, such as cancelling out pairs of opposite directions, priority collection and pipelined transmission. Finally, the remaining arrows cooperatively draw a route map for L_i, see a demonstration in Figure 4. The following lemma shows that this procedure calculates $\Delta(l_t, s_t)$ in linear time.

Lemma 3. *ComputeDistance requires $O(|L_i|)$ rounds to compute $\Delta(l_t, s_t)$.*

Lemma 4. *CollectArrows completes within $O(|L_i|)$ rounds.*

By Lemmas 3 and 4, we conclude that:

Lemma 5. *DrawMap draws a map within $O(|L_i|)$ rounds.*

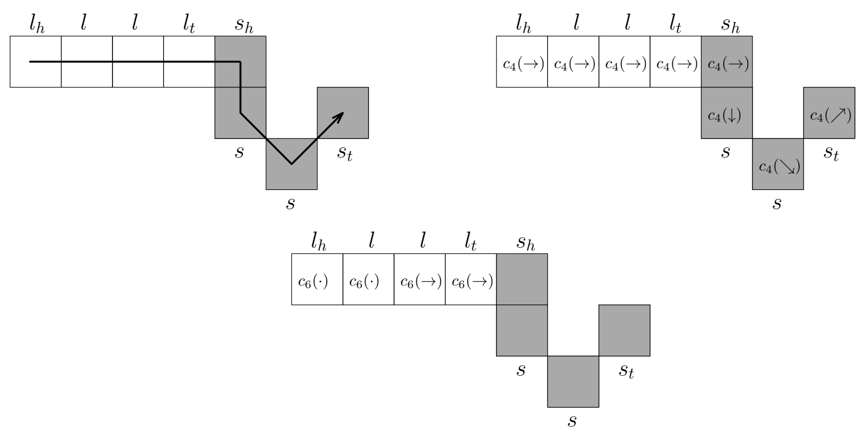


Fig. 4. Drawing a map: from top-left a path across occupied cells and corresponding local arrows stored on state c_4 in top-tight, where the diagonal directions, "$\searrow$" and "$\nearrow$", are interpreted locally as, "$\downarrow\rightarrow$" and "$\uparrow\rightarrow$". The bottom shows a route map drawn locally on state c_6 of each line agent.

Push. After some communication, l_h observes that L_i is ready to move and can start Push now. It synchronises with l_t to guide line agents during pushing. To achieve this, it propagates fast "p1" and slow "p2" marks along the line, "p1" is transmitted every round and "p2" is three rounds slower. The "p1" mark reflects at l_t and meets "p2" at a middle agent p_i, which in turn propagates two pushing signals "P" in either directions, one towards l_h and the other heading to l_t. This synchronisation liaises l_h with l_t throughout the pushing process, which starts immediately after "P" reaches both ends of the line at the same time. Recall the route map has been drawn starting from l_t, and hence, l_t moves simultaneously with l_h according to a local map direction $\hat{a} \in A$ stored in its map component c_6. Through this synchronisation, l_t checks the next cell (x, y)

that L_i pushes towards and tells l_h, whether it is empty or occupied by an agent $p \notin L_i$ in the rest of the configuration. If (x, y) is empty, then l_h pushes L_i one step towards (x, y), and all line agents shift their map arrows in c_6 forwardly towards l_t. If (x, y) is occupied by $p \notin L_i$, then l_t swaps states with p and tells l_h to push one step. Similarly, in each round of pushing a line agent p_i swaps states with p until the line completely traverses the drawn route map and restores it to its original state. Figure 5 shows an example of pushing L_i through a route of empty and occupied cells. In this way, the line agents can transparently push through a route of any configuration and leave it unchanged (see Appendix for more details). Once L_i has traversed completely through the route and lined up with s_t, then RecursiveCall begins. Below, we show that under this model there is a way to sync a Hamiltonian path of n agents in which all can preform concurrent actions in linear time.

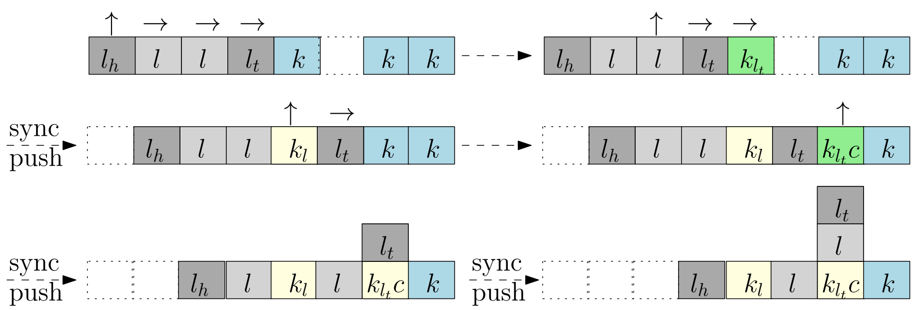


Fig. 5. A line L_i of agents inside grey cells (of labels l_t, l, and l_t), with map directions above, pushing and turning through empty and non-empty cells in blue (of label k).

Lemma 6 (Agents synchronisation). *Let P denote a a Hamiltonian path of n agents on the square grid, starting from a head p_1 and ending at a tail p_n, where $p_1 \neq p_n$. Then, all agents of P can be synchronised in at most $O(n)$ rounds.*

Let R denote a rectangular path consisting of a set of cells $R = [c_1, \ldots, c_{|R|}]$ on $\mathbb{Z}^2$, where c_i and c_{i+1} are two cells adjacent vertically or horizontally, for all $1 \leq i \leq |R| - 1$. Let C be a system configuration, C_R denotes the configuration of R where $C_R \subset C$ defined by $[c_1, \ldots, c_{|R|}]$. Then, we give the following lemma:

Lemma 7. *Let L_i denote a terminal straight line and R be a rectangular path of any configuration C_R, starting from a cell adjacent to the tail of L_i, where $R \leq 2|L_i| - 1$. Then, there exists a distributed way to push L_i along R without breaking connectivity.*

In the following lemma, we provide the complexity of Push on the number of line moves and the communication rounds.

Lemma 8. *A straight line L_i traverses a route R of any configuration C_R, taking at most $O(|L_i|)$ line moves in $O(|L_i| \cdot |R|)$ rounds.*

RecursiveCall. When a segment tail s_t swaps states with l_h, it accordingly acts as follows: (1) propagates a special mark transmitted along all segment agents towards the head s_h, (2) deactivates itself by updating label to $c_1 \leftarrow k$, (3) resets all of its components, except local direction in c_4. Similarly, once a segment agent p_i observes this special mark, it propagates it to its successor p_{i+1}, deactivates itself, and keeps its local direction in c_4 while resetting all other components. When the segment head s_h notices this special mark, it changes to a line head state ($c_1 \leftarrow l_h$) and then recursively repeats the whole transformation from round 1 to $i-1$. Figure 3 presents a graphical illustration of RecursiveCall applied on a diagonal line shape.

Merge. This sub-phase begins once RecursiveCall has transformed S_i into a straight line L'_i, with the tail of L'_i occupying a cell adjacent to the head l_h of L_i. First, Merge calls CheckSeg to check whether L'_i is in line or perpendicular to L_i. If the latter is true (that is both L_i and L'_i are perpendicular to each other), then l_h calls Push to move L_i towards L'_i and form a new line L_{i+1}. Otherwise, they swap states and elect one head l_h and tail l_t of L_{i+1}. Thus, all agents require linear cost of communications and movements during this sub-phase:

Lemma 9. *An execution of Merge requires at most $O(|L_i|)$ line moves and $O(|L_i|)$ rounds of communication.*

Overall, given a Hamiltonian path of individuals with limited knowledge in an initial connected shape S_I. Then, the following lemma states that S_I can be transformed into a straight line S_L through a series of line moves that match the optimal centralised transformation and satisfy the connectivity-preserving condition.

Lemma 10. *Given an initial Hamiltonian shape S_I of n agents, this strategy transforms S_I into a straight line S_L of the same order in $O(n \log_2 n)$ moves and $O(n^2 \log_2 n)$ rounds, while preserving connectivity during transformation.*

Thus, we can finally provide the following theorem:

Theorem 1. *The above distributed transformation solves* HamiltonianLine *within $O(n \log_2 n)$ line moves and $O(n^2 \log_2 n)$ rounds.*

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(11), 1021–1024 (1994)

2. Akitaya, H.A., Arkin, E.M., Damian, M., Demaine, E.D., Dujmović, V., Flatland, R., Korman, M., Palop, B., Parada, I., van Renssen, A., et al.: Universal reconfiguration of facet-connected modular robots by pivots: the o (1) musketeers. Algorithmica **83**(5), 1316–1351 (2021)
3. Almethen, A., Michail, O., Potapov, I.: On efficient connectivity-preserving transformations in a grid. In: Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS. vol. 12503, pp. 76–91 (2020)
4. Almethen, A., Michail, O., Potapov, I.: Pushing lines helps: Efficient universal centralised transformations for programmable matter. Theoretical Computer Science **830-831**, 43 – 59 (2020)
5. Aloupis, G., Benbernou, N., Damian, M., Demaine, E., Flatland, R., Iacono, J., Wuhrer, S.: Efficient reconfiguration of lattice-based modular robots. Computational geometry **46**(8), 917–928 (2013)
6. Aloupis, G., Collette, S., Demaine, E., Langerman, S., Sacristán, V., Wuhrer, S.: Reconfiguration of cube-style modular robots using O(logn) parallel moves. In: International Symposium on Algorithms and Computation. pp. 342–353. Springer (2008)
7. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing **18**(4), 235–253 (2006)
8. Boneh, D., Dunworth, C., Lipton, R.J., Sgall, J.: On the computational power of dna. Discrete Applied Mathematics **71**(1-3), 79–94 (1996)
9. Bourgeois, J., Goldstein, S.: Distributed intelligent MEMS: progresses and perspective. IEEE Systems Journal **9**(3), 1057–1068 (2015)
10. Czyzowicz, J., Dereniowski, D., Pelc, A.: Building a nest by an automaton. Algorithmica **83**(1), 144–176 (2021)
11. Daymude, J., Derakhshandeh, Z., Gmyr, R., Porter, A., Richa, A., Scheideler, C., Strothmann, T.: On the runtime of universal coating for programmable matter. Natural Computing **17**(1), 81–96 (2018)
12. Derakhshandeh, Z., Gmyr, R., Porter, A., Richa, A., Scheideler, C., Strothmann, T.: On the runtime of universal coating for programmable matter. In: International Conference on DNA-Based Computers. pp. 148–164. Springer (2016)
13. Derakhshandeh, Z., Gmyr, R., Richa, A., Scheideler, C., Strothmann, T.: Universal shape formation for programmable matter. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures. pp. 289–299. ACM (2016)
14. Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. Distributed Computing (Mar 2019)
15. Doty, D.: Theory of algorithmic self-assembly. Communications of the ACM **55**, 78–88 (2012)
16. Douglas, S., Dietz, H., Liedl, T., Högberg, B., Graf, F., Shih, W.: Self-assembly of dna into nanoscale three-dimensional shapes. Nature **459**(7245), 414 (2009)
17. Dumitrescu, A., Pach, J.: Pushing squares around. In: Proceedings of the twentieth annual symposium on Computational geometry. pp. 116–123. ACM (2004)
18. Dumitrescu, A., Suzuki, I., Yamashita, M.: Formations for fast locomotion of metamorphic robotic systems. The International Journal of Robotics Research **23**(6), 583–593 (2004)
19. Dumitrescu, A., Suzuki, I., Yamashita, M.: Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. IEEE Transactions on Robotics and Automation **20**(3), 409–418 (2004)

20. Fekete, S.P., Gmyr, R., Hugo, S., Keldenich, P., Scheffer, C., Schmidt, A.: Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata. Algorithmica **83**(1), 387–412 (2021)
21. Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by oblivious mobile robots. Synthesis Lectures on Distributed Computing Theory **3**(2), 1–185 (2012)
22. Fukuda, T.: Self organizing robots based on cell structures-cebot. In: Proc. IEEE Int. Workshop on Intelligent Robots and Systems (IROS'88). pp. 145–150 (1988)
23. Gilpin, K., Knaian, A., Rus, D.: Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on. pp. 2485–2492. IEEE (2010)
24. Gmyr, R., Hinnenthal, K., Kostitsyna, I., Kuhn, F., Rudolph, D., Scheideler, C., Strothmann, T.: Forming tile shapes with simple robots. Natural Computing pp. 1–16 (2019)
25. Itai, A., Papadimitriou, C., Szwarcfiter, J.: Hamilton paths in grid graphs. SIAM Journal on Computing **11**(4), 676–686 (1982)
26. Knaian, A., Cheung, K., Lobovsky, M., Oines, A., Schmidt-Neilsen, P., Gershenfeld, N.: The milli-motein: A self-folding chain of programmable matter with a one centimeter module pitch. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1447–1453. IEEE (2012)
27. Michail, O., Skretas, G., Spirakis, P.: On the transformation capability of feasible mechanisms for programmable matter. Journal of Computer and System Sciences **102**, 18–39 (2019)
28. Michail, O., Spirakis, P.: Simple and efficient local codes for distributed stable network construction. Distributed Computing **29**(3), 207–237 (2016)
29. Michail, O., Spirakis, P.: Elements of the theory of dynamic networks. Commun. ACM **61**(2), 72–81 (2018)
30. Nguyen, A., Guibas, L., Yim, M.: Controlled module density helps reconfiguration planning. In: Proc. of 4th International Workshop on Algorithmic Foundations of Robotics. pp. 23–36 (2000)
31. Prakash, V.P., Patvardhan, C., Srivastav, A.: Effective heuristics for the bi-objective euclidean bounded diameter minimum spanning tree problem. In: International Conference on Next Generation Computing Technologies. pp. 580–589. Springer (2017)
32. Rothemund, P.: Folding dna to create nanoscale shapes and patterns. Nature **440**(7082), 297–302 (2006)
33. Rothemund, P., 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. ACM (2000)
34. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014)
35. Toffoli, T., Margolus, N.: Programmable matter: concepts and realization. Physica. D, Nonlinear phenomena **47**(1-2), 263–272 (1991)
36. Walter, J., Welch, J., Amato, N.: Distributed reconfiguration of metamorphic robot chains. Distributed Computing **17**(2), 171–189 (2004)
37. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (1998)
38. Woods, D., Chen, H., 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)

39. Yamashita, M., Suzuki, I.: Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theoretical Computer Science **411**(26-28), 2433–2453 (2010)
40. Yim, M., Shen, W., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirikjian, G.: Modular self-reconfigurable robot systems [grand challenges of robotics]. IEEE Robotics & Automation Magazine **14**(1), 43–52 (2007)