Pushing Lines Helps: Efficient Universal Centralised Transformations for Programmable Matter[☆]

Abdullah Almethen*, Othon Michail, Igor Potapov

Department of Computer Science, University of Liverpool, UK

Abstract

In this work, we study a discrete system of entities residing on a two-dimensional square grid. Each entity is modelled as a node occupying a distinct cell of the grid. The set of all n nodes forms initially a connected shape A. Entities are equipped with a linear-strength pushing mechanism that can push a whole line of entities in parallel in a single time-step on one position in a given (one of the four possible) direction of a grid. A target connected shape B is also provided and the goal is to *transform* A into B via a sequence of line moves. Existing models based on local movement of individual nodes, such as rotating or sliding a single node, can be shown to be special cases of the present model, therefore their (inefficient, $\Theta(n^2)$-time) *universal transformations* carry over. Our main goal is to investigate whether the parallelism inherent in this new type of movement can be exploited for efficient, i.e., sub-quadratic worst-case, transformations. This paper provides several solutions for specific and universal centralised transformations in the context of the new model. In particular we first design $O(n \log n)$-time universal transformation without preserving the connectivity of original shape. Then we focus on transformations which preserve the connectivity of the shape throughout its course and develop an $O(n\sqrt{n})$-time transformation for the apparently hard instance of transforming a diagonal A into a straight line B.

Keywords:
programmable matter, transformation, reconfigurable robotics, shape formation, complexity, distributed algorithms

1. Introduction

As a result of recent advances in components such as micro-sensors, electromechanical actuators, and micro-controllers, a number of interesting systems are now within reach. A prominent type of such systems concerns collections of small robotic entities. Each individual robot is equipped with a number of actuation/sensing/communication/computation components that provide it with some autonomy; for instance, the ability to move locally and to communicate with neighbouring robots. Still, individual local dynamics are weak, and individual computations are restricted due to limited computational power, resources, and knowledge. What makes these systems interesting is the collective complexity of the population of devices. A number of fascinating recent developments in this direction have demonstrated the feasibility and potential of such collective robotic systems, where the scale can range from milli/micro [BG15, GKR10, KCL+12, RCN14, YSS+07] down to nano [DDL+09, Rot06].

This progress has motivated the parallel development of a theory of such systems. It has been already highlighted [MS18] that a formal theory (including modelling, algorithms, and computability/complexity) is

[☆]A preliminary version of the results in this paper has appeared in [AMP19a].

*Corresponding author (Telephone number: +44 (0)151 795 4275, Postal Address: Department of Computer Science, University of Liverpool, Ashton Street, Liverpool L69 3BX, UK).

Email addresses: A.Almethen@liverpool.ac.uk (Abdullah Almethen), Othon.Michail@liverpool.ac.uk (Othon Michail), Potapov@liverpool.ac.uk (Igor Potapov)

necessary for further progress in systems. This is because theory can accurately predict the most promising designs, suggest new ways to optimise them, by identifying the crucial parameters and the interplay between them, and provide with those (centralised or distributed) algorithmic solutions that are best suited for each given design and task, coupled with provable guarantees on their performance. As a result, a number of sub-areas of theoretical computer science have emerged such as mobile and reconfigurable robotics [ABD$^+$13, BKRT04, CFPS12, CKLWL09, DFSY15, DDG$^+$18, DGMRP06, DDG$^+$14, DGR$^+$15, DGR$^+$16, DLFS$^+$19, DLFP$^+$18, FPS12, KKM10, MSS19, SMO$^+$18, YS10, YUY16, YSS$^+$07], passively-mobile systems [AAD$^+$06, AAER07, MS16, MS18] including the theory of DNA self-assembly [Dot12, RW00, Win98, WCG$^+$13], and metamorphic systems [DP04, DSY04a, DSY04b, NGY00, WWA04]; connections are even evident with the theory of puzzles [BDF$^+$19, Dem01, HD05]. A latest ongoing effort is to join these theoretical forces and developments within the emerging area of "Algorithmic Foundations of Programmable Matter" [FRRS16]. *Programmable matter* refers to any type of matter that can *algorithmically* change its physical properties. "Algorithmically" means that the change (or *transformation*) is the result of executing an *underlying program*.

In this paper, we embark from the model studied in [DP04, DSY04a, DSY04b, MSS19], in which a number of spherical devices are given in the form of a (typically connected) shape A lying on a two-dimensional square grid, and the goal is to transform A into a desired target shape B via a sequence of valid movements of individual devices. In those papers, the considered mechanisms were the ability to rotate and slide a device over neighbouring devices (always through empty space). We here consider an alternative (linear-strength) mechanism, by which a line of one or more devices can translate by one position in a single time-step. As a first step towards understanding the power of the new mechanism, we restrict attention solely to centralised transformations and leave the distributed case as a direction for future research.

As our main goal is to determine whether the new move under consideration can *in principle* be exploited for sub-quadratic worst-case transformations, we naturally restrict our attention to centralised transformations. We first allow the transformations to break connectivity, and we manage to develop a universal transformation of $O(n \log n)$ worst-case running time. Then, we investigate the case in which the transformations must preserve connectivity and develop an $O(n\sqrt{n})$-time transformation for some specific pairs of connected shapes. Distributed transformations and connectivity-preserving universal transformations are left as interesting future research directions.

1.1. Our Approach

In [MSS19], it was proved that if the devices (called *nodes* from now on) are equipped only with a rotation mechanism, then the decision problem of transforming a connected shape A into a connected shape B is in $\mathbf{P}$, and a constructive characterisation of the (rich) class of pairs of shapes that are transformable to each other was given. In the case of combined availability of rotation and sliding, universality has been shown [DP04, MSS19], that is, any pair of connected shapes are transformable into each other. Still, in these and related models, where in any time step at most one node can move a single position in its local neighbourhood, it can be proved (see, for instance, [MSS19]) that there will be pairs of shapes that require $\Omega(n^2)$ steps to be transformed into each other. This follows directly from the inherent "distance" between the two shapes and the fact that this distance can be reduced by only a constant in every time step. An immediate question is then *"How can we come up with more efficient transformations?"*

Two main alternatives have been explored in the literature in an attempt to answer this question. One is to consider parallel time, meaning that the transformation algorithm can move more than one node (up to a linear number of nodes if possible) in a single time step. This is particularly natural and fair for distributed transformations, as it allows all nodes to have their chances to take a move in every given time-step. For example, such as transformations based on pipelining [DSY04b, MSS19], where essentially the shape transforms by moving nodes in parallel around its perimeter, can be shown to require $O(n)$ parallel time in the worst case and this technique has also been applied in systems (e.g., [RCN14]).

The other approach is to consider more powerful actuation mechanisms, that have the potential to reduce the inherent distance faster than a constant per sequential time-step. These are typically mechanisms where the local actuation has strength higher than a constant. This is different from the above parallel-time transformations, in which local actuation can only move a single node one position in its local neighbourhood

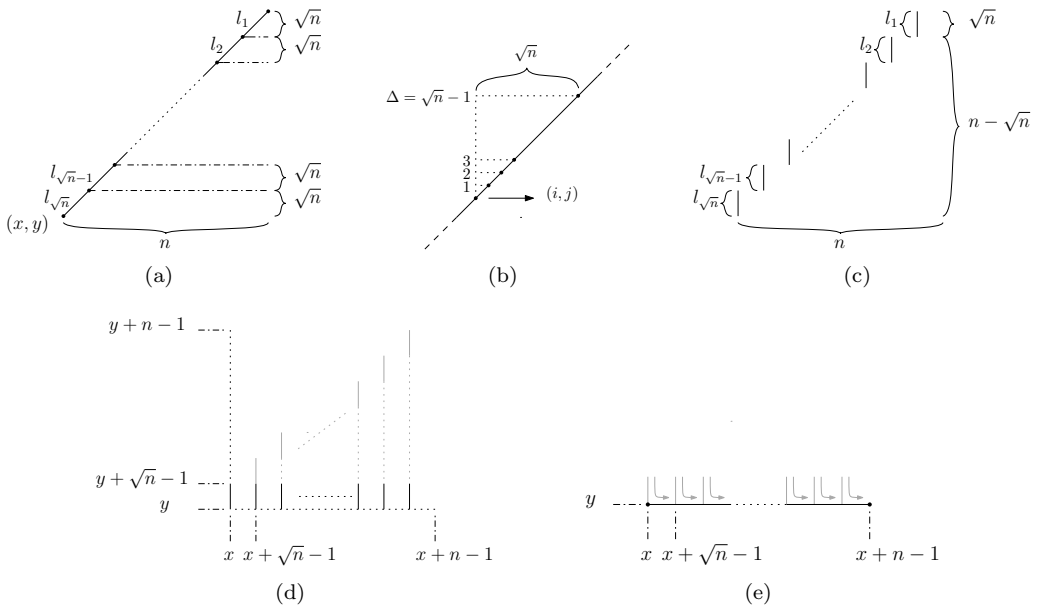


Figure 1: (a) Dividing the diagonal into $\sqrt{n}$ segments of length $\sqrt{n}$ each (integer $\sqrt{n}$ case). (b) A closer view of a single segment, where $1, 2, 3, \ldots, \sqrt{n}-1$ are the required distances for the nodes to form a line segment at the leftmost column (of the segment). (c) Each line segment is transformed into a line and transferred towards the bottommost row of the shape, ending up as in (d). (e) All line segments are turned into the bottommost row to form the target spanning line.

and the combined effect of many such moves at the same time is exploited. In contrast, in higher-strength mechanisms, it is a single actuation that has enough strength to move many nodes at the same time. Prominent examples in the literature are the linear-strength models of Aloupis *et al.* [ABD+13, ACD+08], in which nodes are equipped with extend/contract arms, each having the strength to extend/contract the whole shape as a result of applying such an operation to one of its neighbours, and of Woods *et al.* [WCG+13], in which a whole line of nodes can rotate around a single node (acting as a linear-strength rotating arm). The present paper follows this approach, by introducing and investigating a linear-strength model in which a node can push a line of consecutive nodes one position (towards an empty cell) in a single time-step.

In terms of transformability, our model can easily simulate the combined rotation and sliding mechanisms of [DP04, MSS19] by restricting moves to lines of length 1 (i.e., individual nodes). It follows that this model is also capable of universal transformations, with a time complexity at most twice the worst-case of those models, i.e., again $O(n^2)$. Naturally, our focus is set on exploring ways to exploit the parallelism inherent in moving lines of larger length in order to speed-up transformations and, if possible, to come up with a more efficient in the worst case universal transformation. Further, as reversibility of moves is still valid in our model, we adopt the approach of transforming any given shape A into a spanning line L (vertical or horizontal). This is convenient, because if one shows that any shape A can transform fast into a line L, then any pair of shapes A and B can then be transformed fast to each other by first transforming fast A into L and then L into B by reversing the fast transformation of B into L.

Given this, our focus in the present study is to investigate whether the presented linear-strength mechanism can achieve faster transformations that transform any pair of connected shapes to each other (i.e., no requirement to preserve connectivity of the shape during the transformations). For example, take the diagonal worst-case shape D (e.g., Figure 1 (a)) and try to convert it into a straight line L. Observe that any transformation requires $\Theta(n^2)$ sequential individual movements to transform D into L. By exploiting linear-strength mechanism, let us now perform the following simple strategy; (1) divide D into several diagonal segments of length $\sqrt{n}$ each (see Figure 1 (a)), then (2) turn each segment into a straight line segment via individual moves, which takes linear time for each segment (Figure 1 (b) and (c)). Next, (3) transfer

every line segment all the way down to the bottom of D (Figure 1 (d)) in which each line segment pushes a maximum of n distance. Finally, (4) change the orientation of all line segments in linear time to form the target straight line L (Figure 1 (e)). This transformation takes a total of $O(n\sqrt{n})$ time-steps. Thus, in contrast to the aforementioned models, the new mechanism allows for sub-quadratic strategies, like the one just sketched. The complete technical description of this strategy can be found in the full report [AMP19b].

In this paper, we exploit the new mechanism in order to develop such sub-quadratic transformations. Our ultimate goal is *universal transformations*, meaning transformations that can transform any pair of connected shapes to each other. By allowing the transformations to break connectivity, we give such a universal transformation that takes $O(n \log n)$ time. We achieve this by enclosing the initial shape into a square bounding box and then subdividing the box into square sub-boxes of appropriate dimension. Then, we employ a successive doubling approach through phases of an increasing dimension of the sub-boxes, that is, through a new partitioning in each phase. Our ultimate theorem (followed by a constructive proof, providing the claimed transformation) states that: *"In this model, when connectivity needs not necessarily be preserved during the transformation, any pair of connected shapes A and B can be transformed to each other in sequential time $O(n \log n)$"*.

We then turn our attention to the case in which the transformations *cannot break connectivity*. We identify the diagonal shape D (which is considered connected in our model and is very similar to the staircase worst-case shape of [MSS19]) as a potential worst-case initial shape to be transformed into a line L. This intuition is supported by the $O(n^2)$ individual node distance between the two shapes and by the initial unavailability of long line moves: the transformation may move long lines whenever available, but has to pay first a number of individual and small line moves in order to construct longer lines. In this benchmark (special) case, the trivial lower and upper bounds $\Omega(n)$ and $O(n^2)$, respectively, hold. Moreover, observe that a sequential gathering of the nodes starting from the top right and collecting the nodes one after the other into a snake-like line of increasing length is still quadratic, because, essentially, for each sub-trip from one collection to the next, the line has to make a "turn", meaning to change both a row and a column, and in this model this costs a number of steps equal to the length of the line, that is, roughly, $1 + 2 + \ldots + (n-1) = \Theta(n^2)$ total time-steps.

We solve the problem of transforming D into L while *preserving connectivity* throughout the transformation (called DiagonalToLine problem), by developing an $O(n\sqrt{n})$-time transformation. This approach, called *DLC-Folding*, divides the diagonal into $\sqrt{n}$ segments of length $\sqrt{n}$ each and proceeds in $\sqrt{n}$ phases. In each phase, it folds the segments sequentially in a top-down order via two main operations, *turn* and *push*. In the final phase, the algorithm forms *a square shape*, which can be transformed fast into a line. Thus, this transformation (*folding*) preserves connectivity and takes total time $O(n\sqrt{n})$.

Table 1 summarises the running times of all the transformations (algorithms) developed in this paper.

Connectivity Preserving	Transformation	Problem	Running Time	Lower Bound
No	*U-Box-Doubling*	Universal	$O(n \log n)$	$\Omega(n)$
Yes	*DLC-Folding*	DiagonalToLine	$O(n\sqrt{n})$	$\Omega(n)$

Table 1: A summary of our transformations and their corresponding worst-case running times (the trivial lower bound is in all cases $\Omega(n)$). All problems are being formally defined in Section 2.2

Section 2 brings together all definitions and basic facts that are used throughout the paper. Section 3 presents our universal transformation in which connectivity is not necessarily preserved throughout transformations. In Section 4, we consider the case when connectivity must be preserved and study the problem of transforming a diagonal shape into a line. In Section 5 we conclude and discuss further research directions that are opened by our work. However, the complete discussion of all transformations and problems of this work can be found in the full report [AMP19b].

2. Preliminaries and Definitions

The transformations considered here run on a two-dimensional square grid. Each cell of the grid possesses a unique location addressed by non-negative coordinates (x, y), where x denotes columns and y indicates rows. A *shape* S is a set of n *nodes* on the grid, where each individual node $u \in S$ occupies a single cell $cell(u) = (x_u, y_u)$, therefore we may also refer to a node by the coordinates of the cell that it occupies at a given time. Two distinct nodes (x_1, y_1), (x_2, y_2) are *neighbours* (or *adjacent*) iff $x_2 - 1 \leq x_1 \leq x_2 + 1$ and $y_2 - 1 \leq y_1 \leq y_2 + 1$ (i.e., their cells are adjacent vertically, horizontally or diagonally). A shape S is *connected* iff the graph defined by S and the above neighbouring relation on S is connected. Throughout, n denotes the number of nodes in a shape under consideration, and all logarithms are to the base 2.

A line L is a sequence of consecutive non-empty cells occupied by nodes, $u \in S$, vertically or horizontally (not, e.g., diagonally). A ***line move*** is an operation by which all nodes of L move together in a single move, towards an empty *cell* adjacent to one of L's endpoints. A line move may also be referred to as a *step* (or *move* or *movement*) and time is discrete and measured in number of steps throughout. A move in this model is equivalent to choosing a node u and a direction $d \in \{up, down, left, right\}$ and moving u one position in direction d. This will additionally push by one position the whole line L of nodes in direction d, L (possibly empty) starting from a neighbour of u in d and ending at the first empty cell. More formally and in slightly different terms: A line $L = (x, y), (x+1, y), \ldots, (x+k-1, y)$ of length k, where $1 \leq k \leq n$, can push all its k nodes rightwards in a single move to positions $(x+1, y), (x+2, y), \ldots, (x+k, y)$ iff there exists an empty cell to the right of L at $(x + k, y)$. The "down", "left", and "up" moves are defined symmetrically, by rotating the whole system 90°, 180°, and 270° clockwise, respectively.

Definition 1 (A permissible line move)**.** *Let L be a line of k nodes, where $1 \leq k \leq n$. Then, L can move as follows (depending on its original orientation, i.e., horizontal or vertical):*

1. Horizontal. *Can push all k nodes rightwards in a single move from $(x, y), (x+1, y), \ldots, (x+k-1, y)$ to positions $(x+1, y), (x+2, y), \ldots, (x+k, y)$ iff there exists an empty cell to the right of L at $(x+k, y)$. Similarly, it can push all k nodes to the left to occupy $(x-1, y), (x, y), \ldots, (x+k, y)$, iff there exists an empty cell at $(x-1, y)$.*

2. Vertical. *Can push all k nodes upwards in a single move from $(x, y), (x, y+1), \ldots, (x, y+k-1)$ into $(x, y+1), (x, y+2), \ldots, (x, y+k)$, iff there exists an empty cell above L at $(x, y+k)$. Similarly, it can push all k nodes down to occupy $(x, y-1), (x, y), \ldots, (x, y+k)$, iff there exists an empty cell below L at $(x, y-1)$.*

We call this model, *line-pushing model*, in the rest of the paper. The following definitions from [MSS19] shall be useful for our study. Let us first agree that we colour black any cell occupied by a node (as in Figure 2).

Definition 2. *A hole H of a connected shape S is a set of empty cells enclosed by non-empty cells that are occupied by nodes $u \in S$, such that any simple path of infinite length that starts from an empty cell in the hole $h \in H$ and moves, only vertically and horizontally, must pass through a black cell of a node $u \in S$.*

Definition 3. *A compact shape is a connected shape that does not contain any holes.*

Definition 4. *The perimeter (border) of S is defined as a polygon of unit length line segments, which surrounds the minimum-area of the interior of S', such as the yellow line in Figure 2.*

Definition 5. *The surrounding layer of a connected shape S consists of all empty cells in the grid that share at least a line segment or a corner with the S's perimeter (e.g., the grey cells in Figure 2).*

Definition 6. *The external surface of a connected shape S consists of all non-empty cells in the grid that share at least a line segment or a corner with the S's perimeter (e.g., cells of black spherical nodes in Figure 2).*

The external surface of a connected shape S is connected (proved in [MSS19]).

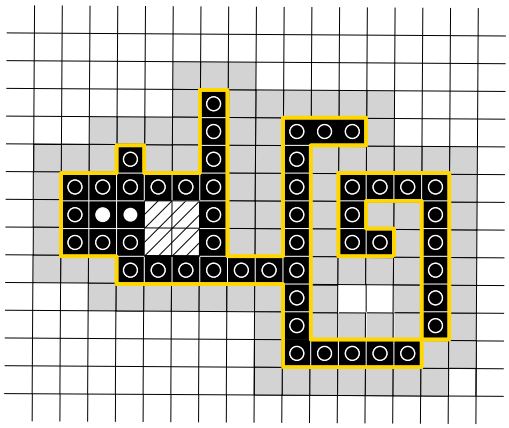


Figure 2: All nodes of S occupy the black cells, where black spherical nodes reside on the external surface of S. The surrounding layer's cells are coloured grey, while the yellow line depicts the perimeter of S. The dashed black cells define a hole of S.

Proposition 1. *The surrounding layer of any connected shape S is itself a connected shape.*

Proof. The proof is an adaptation of Proposition 2 from [MSS19] to our model. Assume S is connected, then the perimeter of S is connected too; and hence, it forms a cycle. Each segment of the perimeter is contributed by two cells, belonging to the external surface and the surrounding layer (recall Definitions 4, 5 and 6). Now, if one walks on the perimeter (vertically or horizontally) or turns (left or right) clockwise or anticlockwise at any segment, one of the following cases will occur:

(i) Pass through two adjacent vertical or horizontal cells on the surrounding layer and the external surface of S.

(ii) Stay put at the same position (cell) on the external surface and move through three neighbouring cells connected perpendicularly on the surrounding layer of S.

(iii) Stay put at the same position (cell) on the surrounding layer and pass through three neighbouring cells connected perpendicularly on the external surface of S.

(iv) Stay put at the same position (cell) on the surrounding layer and:

  (a) Either pass through two neighbouring cells connected diagonally on the external surface of S.

  (b) Or pass through three neighbouring cells connected perpendicularly on the external surface of S.

Subsequently, all cases above preserve connectedness of the surrounding layer and the external surface of S. □

As mentioned early, we know that there are related settings in which any pair of connected shapes A and B of the same order ("order" of a shape S meaning the number of nodes of S throughout the paper) can be transformed to each other[1] while preserving the connectivity throughout the course of the transformation.[2] This, for example, has been proved for the case in which the available moves to the nodes are rotation and sliding [DP04, MSS19]. We now show that the model of [DP04, MSS19] is a special case of our model, implying all transformations established there (with their running time at most doubled, including universal transformations, are also valid transformations in the present model).

[1] We also use $A \to B$ to denote that shape A can be transformed to shape B.

[2] In this paper, whenever transforming into a target shape B, we allow any placement of B on the grid, i.e., any shape B' obtained from B through a sequence of rotations and translations.

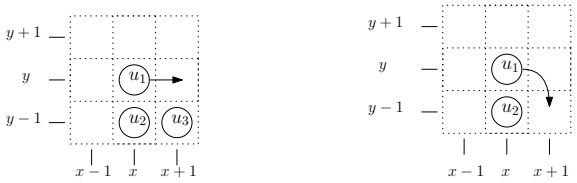


Figure 3: (a) An example of sliding u_1 over u_2 and u_3 to an empty cell to the left. (b) Rotate u_1 a 90° clockwise around u_2.

Proposition 2. *The rotation and sliding model of [DP04, MSS19] is a special case of the line-pushing model.*

Proof. We establish a technique to prove that our model is capable of simulating *rotation and sliding models* of a two-dimension square grid appeared in [DP04, MSS19]. First, the sliding operation is equivalent in all those models, that is, if a node u is located at a cell of the grid, $u = (x, y)$, then u can slide right to any empty cell at $(x + 1, y + 1)$ over a horizontal line of length 2, such as in Figure 3 (a). The "down", "left", and "up" moves are defined symmetrically, by rotating the whole system 90°, 180°, and 270° clockwise, respectively. Now, the presented model is capable of performing the same *sliding* rule in those models, i.e., push a line of length 1 vertically or horizontally, as explained in Definition 1. For rotation, all mentioned models perform a single operation to rotate a node $u_1 = (x, y)$ around another $u_2 = (x, y - 1)$ by a 90° clockwise iff there exits two empty cells at $(x + 1, y)$ and $(x + 1, y + 1)$, see Figure 3(b). Analogously, this holds for all possible rotations by again rotating the whole system 90°, 180°, and 270° clockwise, respectively. Still, the rotation mechanism is also adopted by this model following Definition 1, and actually it costs twice for a single rotation to take place, compared with others. Subsequently, it implies that all transformations established there (with its running time at most doubled), including universal transformations and preserving connectivity (recall Proposition 1), are also valid transformations in the present model. □

Consider a line L of k nodes occupying $(x, y), (x + 1, y), \ldots, (x + k - 1, y)$ and k consecutive empty cells starting horizontally from postilion $(x + k, y)$. Now, we aim to transfer all k nodes to the right to fill in all k empty cells. Any transformation of individual movements performs k per node, in a total of $\Theta(k^2)$ steps for all k nodes. The linear-strength mechanism of *the line-pushing model* achieves more efficient cost of at most $O(k)$ steps, by transferring all k nodes in parallel distance k to the right. Now, we are ready to show the following lemma which will be used several times in our transformations:

Lemma 1. *It is possible to turn a horizontal line of length k into a vertical line of length k or vice versa in $2k - 2$ steps.*

Proof. To simplify the argument, assume that a two-dimensional square grid contains only a straight line L_1 of k nodes at $(x, y), \ldots, (x, y + k - 1)$, where $k \geq 1$, and empty cells on $(x + 1, y), \ldots, (x + k - 1, y)$, as depicted in Figure 4. This assumption can be dropped easily when L_1 is a part of a connected shape S in the grid. Now, L_1 wants to change its direction, and without loss of generality, say that L_1 turns from vertical to horizontal, where L_1 occupies k consecutive rows and 1 column. Therefore, the first bottommost node, $u_1 \in L_1$, moves one position right to occupy an empty cell on $(x + 1, y)$, which consequently creates a new empty cell at (x, y) that was occupied by u_1. Then, by linear-strength pushing mechanism, the consecutive $k-1 \in L_1$ nodes push down one move altogether in parallel in a single-time-move towards the new empty cell (x, y) to occupy positions $(x, y), \ldots, (x, y + k - 2)$. Consequently, L_1 consists now of $k - 1$ nodes occupying $k - 1$ consecutive rows, as in Figure 4 (a), (b) and (c). Observe that u_1 take one time-step to move right, and all $k - 1$ nodes push down in one time-step. By repeating this at most $2k - 2$ steps, it shall completely turn L_1 to occupy a single row and k consecutive columns. Therefore, any straight line of k nodes changes its direction in a number of steps at most twice its length, $2k - 2 = O(k)$.

□

A property that typically facilitates the development of universal transformations, is reversibility of moves. To this end, we next show that line moves are reversible.

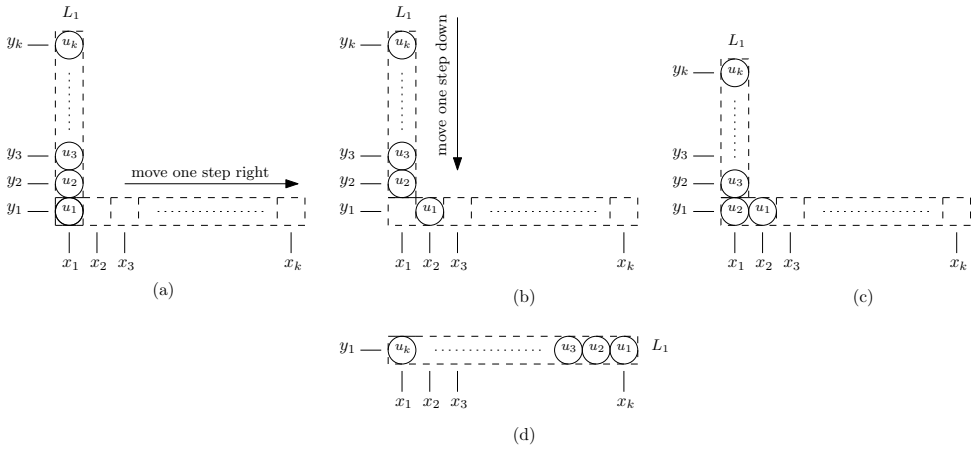


Figure 4: A line of k nodes changes orientation by two consecutive steps per node. (a) Move u_1 one step to the right. (b) and (c) All $k-1$ nodes push down altogether in a single step. In (d), the line has finally transformed from vertical to horizontal after $2k$ steps.

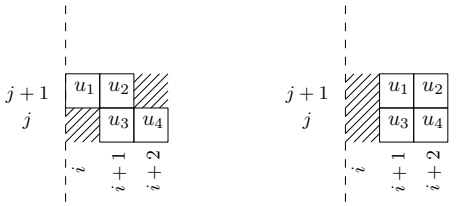


Figure 5: An example of a reversible line move.

Lemma 2 (Reversibility)**.** *Let (S_I, S_F) be a pair of connected shapes of the same number of nodes n. If $S_I \to S_F$ ("$\to$" denoting "can be transformed to via a sequence of line moves") then $S_F \to S_I$.*

Proof. First, we should prove that each single line move is reversible. Figure 5 left shows four nodes forming two vertical and one horizontal lines, $L_1 = \{u_1, u_2\}$, $L_2 = \{u_3, u_4\}$ and $L_3 = \{u_2, u_3\}$, respectively. Assume this configuration has no more space to the left, beyond the dashed line; therefore, L_1 moves to occupy the empty cell $(i+2, j+1)$. Now, all L_1 nodes are moving altogether to fill in positions $(i+1, j+1)$ and $(i+2, j+1)$, as depicted in Figure 5 right. Consequently, the previous line move creates another empty cell at $(i, j+1)$, which can be occupied reversibly by L_1. Since every line move is reversible, it implies that reversibility holds for any finite sequence of line moves.

□

2.1. Nice shapes

A family of shapes, denoted $\mathcal{NICE}$, is introduced in this study to act as efficient intermediate shapes of the transformation. A *nice* shape is, informally, any connected shape that contains a particular line called the *central line* (denoted L_C). Intuitively, one may think of L_C as a supporting (say horizontal) base of the shape, where each node u not on L_C must be connected to L_C through a vertical line.

Definition 7 (Nice Shape)**.** *A nice shape $S \in \mathcal{NICE}$ is a compact connected shape (Definition 3), which contains a central line $L_C \subseteq S$, such that every node $u \in S \setminus L_C$ is connected to L_C via a line perpendicular to L_C (Figure 6 shows some examples of a nice shape and non nice shapes).*

By reversibility (Lemma 2), we provide the following proposition:

Proposition 3. *Let $(\mathcal{A}, \mathcal{B}) \in \mathcal{NICE}$ be a pair of nice shapes of the same order. Then $\mathcal{A} \to \mathcal{B}$ and $\mathcal{B} \to \mathcal{A}$ in $O(n)$ steps.*

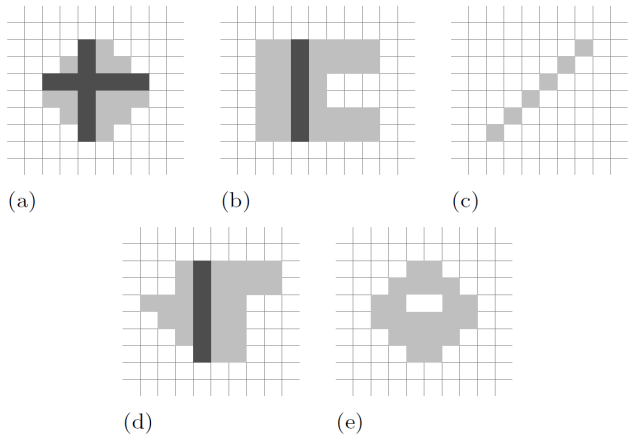


Figure 6: The central line L_C occupies black cells of *nice* shapes in (a), (b) and (d). In (c), the shape is not *nice* (due to the lack of L_C). (e) is also not *nice* because of the hole, white cells inside the shape, which prevents the formation of L_C.

Proof. Let $S_{\mathcal{NICE}}$ be a *nice shape* of order n that contains a central line of i nodes, for some $1 \leq i \leq n$, and assume without loss of generality, that L_C is horizontal and occupies row y_1. By Definition 7, there will be $n - i$ nodes, where all are connected to L_C via a line orthogonal to L_C. Similarly, we can say there are $L_1, L_2, \ldots, L_w$ vertical lines of total $n - i$ nodes, where $1 \leq w < n$, which are all perpendicularly connected to L_C. Now, by Lemma 1, such a vertical line would perform a number of steps twice its length to change its direction, occupy row y_1 and be an extension to L_C. Then, it follows that all those vertical lines requires a total cost of at most $2(n - i) = O(n)$ steps to eventually occupy n consecutive cells in row y_1 where a straight line of order n remains. By Lemma 2, we conclude that any pair of *nice shapes* are transformable to each other via a straight line in $O(n)$ steps.

□

2.2. Problem Definitions

We now formally define the problems to be considered in this paper.

DiagonalToLine**.** Given an initial connected diagonal line S_D and a target vertical or horizontal connected spanning line S_L of the same order, transform S_D into S_L, so that connectivity is preserved during the transformation.

UniversalTransformation**.** Give a general transformation, such that, for all pairs of shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, it will transform S_I into S_F, without necessarily preserving connectivity during its course.

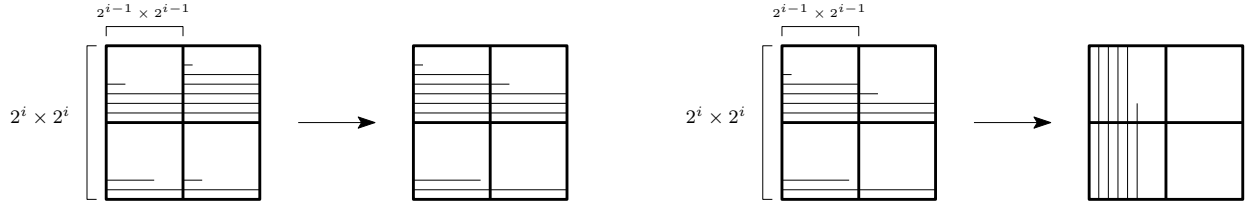
3. Breaking Connectivity: An $O(n \log n)$-time Universal Transformation

We now present our universal transformation, called *U-Box-Doubling*, that transforms any pair of connected shapes, of the same order, to each other in $O(n \log n)$ steps, *without preserving connectivity*. Given a connected shape S_I of order n, do the following. *Enclose* S_I into an arbitrary $n \times n$ *box*. For simplicity, we assume that n is a power of 2, but this assumption can be dropped. Proceed in $\log n$ phases as follows: In every phase i, where $1 \leq i \leq \log n$, partition the $n \times n$ box into $2^i \times 2^i$ sub-boxes, disjoint and completely covering the $n \times n$ box. Assume that from any phase $i - 1$, any $2^{i-1} \times 2^{i-1}$ sub-box is either empty or has its k, where $0 \leq k \leq 2^{i-1}$, bottommost rows completely occupied with nodes, possibly followed by a single

incomplete row on top of them containing l, where $1 \leq l < 2^{i-1}$, consecutive nodes that are left aligned on that row. This case holds trivially for phase 1 and inductively for every phase. That is, in odd phases, we assume that nodes occupy the leftmost columns of boxes in a symmetric way. Every $2^i \times 2^i$ sub-box (of phase i) consists of four $2^{i-1} \times 2^{i-1}$ sub-boxes from phase $i-1$, each of which is either empty or occupied as described above.

The operation of *Boundary-Filling* is to fill in empty cells at a boundary of the $2^i \times 2^i$ sub-box by nodes of lines that are aligned perpendicularly to that boundary. Due to symmetry, we only show the left boundary case. That is, start filling in empty cells from the leftmost column bottom-top and continuing to the right, by exploiting a linear procedure similar to that of Figure 4 (and of *nice* shapes). Without loss of generality, fill in the leftmost column until the row is exhausted or the column is being completely occupied, in this case, start filling in the next column to the right (see Figure 7 (b)). If an incomplete column remains in the top left $2^{i-1} \times 2^{i-1}$ sub-box, push the nodes in it to the bottom of that column, see Figure 8.

Consider the case where i is odd, thus, the nodes in the $2^{i-1} \times 2^{i-1}$ sub-boxes are bottom aligned. For every $2^{i-1} \times 2^{i-1}$ sub-box, move each line from the previous phase that resides in the sub-box to the left as many moves as required until that row contains a single line of consecutive nodes, starting from the left boundary of the sub-box, as shown in Figure 7 (a). Then, perform *Boundary-Filling* on the left boundary the $2^i \times 2^i$ sub-box, as in Figure 7 (b). The case of even i is symmetric, the only difference being that the arrangement guarantee from $i-1$ is left alignment on the columns of the $2^{i-1} \times 2^{i-1}$ sub-boxes and the result will be bottom alignment on the rows of the $2^i \times 2^i$ sub-boxes of the current phase. This completes the description of the transformation. We first prove *correctness*:



(a) Pushing left in each $2^{i-1} \times 2^{i-1}$ sub-box of the $2^i \times 2^i$ sub-box.

(b) The operation of *Boundary-Filling* on the left boarder of the $2^i \times 2^i$ sub-box.

Figure 7: An example of the transformations during phase i.

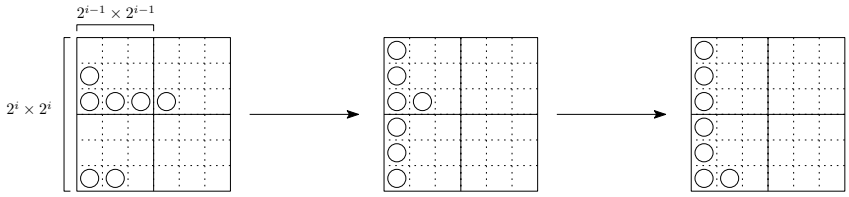


Figure 8: The operation of *Boundary-Filling* when an incomplete column remains.

Lemma 3. *Starting from any connected shape S_I of order n,* U-Box-Doubling *forms by the end of phase* $\log n$ *a line of length n.*

Proof. In phase $\log n$, the procedure partitions into a single box, which is the whole original $n \times n$ box. Independently of whether gathering will be on the leftmost column or on the bottommost row of the box, as all n nodes are contained in it, the outcome will be a single line of length n, vertical or horizontal, respectively. □

Now, we shall analyse the running time of *U-Box-Doubling*. To facilitate exposition, we break this down into a number of lemmas.

Lemma 4. *In every phase i, the "super-shape" formed by the occupied $2^i \times 2^i$ sub-boxes is connected.*

Proof. By induction on the phase number i. For the base of the induction, observe that for $i = 0$ it holds trivially because the initial S_I is a connected shape. Assuming that it holds for phase $i-1$, we shall now prove that it must also hold for phase i. By the inductive assumption, the occupied $2^{i-1} \times 2^{i-1}$ sub-boxes form a connected super-shape. Observe that, by the way the original $n \times n$ box is being repetitively partitioned, any box contains complete sub-boxes from previous phases, that is, no sub-box is ever split into more than one box of future phases. Additionally, observe that a sub-box is occupied iff any of its own sub-boxes (of any size) had ever been occupied, because nodes cannot be transferred between $2^i \times 2^i$ sub-boxes before phase $i+1$. Assume now, for the sake of contradiction, that the super-shape formed by $2^i \times 2^i$ sub-boxes is disconnected. This means that there exists a "cut" of unoccupied $2^i \times 2^i$ sub-boxes as in Figure 9. Replacing

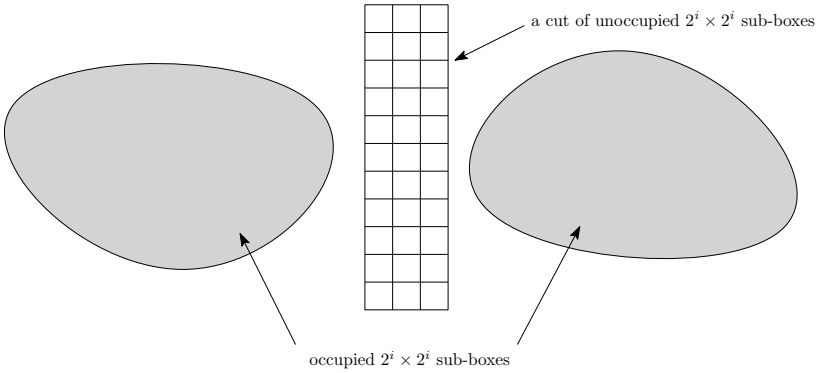


Figure 9: An example of a "cut" of unoccupied $2^i \times 2^i$ sub-boxes.

everything by $2^{i-1} \times 2^{i-1}$ sub-boxes, yields that this must also be a cut of $2^{i-1} \times 2^{i-1}$ sub-boxes, because a node cannot have transferred between $2^i \times 2^i$ sub-boxes before phase $i+1$. But this contradicts the assumption that $2^{i-1} \times 2^{i-1}$ sub-boxes form a connected super-shape. Therefore, it must hold that the $2^i \times 2^i$ sub-boxes super-shape must have been connected. □

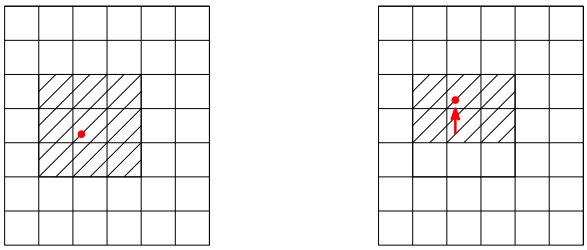
Next, we give an upper bound on the number of occupied sub-boxes in a phase i.

Lemma 5. *Given that* U-Box-Doubling *starts from a connected shape S_I of order n, the number of occupied sub-boxes in any phase i is $O(\frac{n}{2^i})$.*

Proof. First, observe that a $2^i \times 2^i$ sub-box of phase i is occupied in that phase iff S_I was originally going through that sub-box. This follows from the fact that nodes are not transferred by this transformation between $2^i \times 2^i$ sub-boxes before phase $i+1$. Therefore, the $2^i \times 2^i$ sub-boxes occupied in (any) phase i are exactly the $2^i \times 2^i$ sub-boxes that the original shape S_I would have occupied, thus, it is sufficient to upper bound the number of $2^i \times 2^i$ sub-boxes that a connected shape of order n can occupy. Or equivalently, we shall lower bound the number N_k of nodes needed to occupy k sub-boxes.

In order to simplify the argument, whenever S_I occupies another unoccupied sub-box, we will award it a constant number of additional occupations for free and only calculate the additional distance (in nodes) that the shape has to cover in order to reach another unoccupied sub-box. In particular, pick any node of S_I and consider as freely occupied that sub-box and the 8 sub-boxes surrounding it, as depicted in Figure 10 (a). Giving sub-boxes for free can only help the shape, therefore, any lower bound established including the free sub-boxes will also hold for shapes that do not have them (thus, for the original problem). Given that free boxes are surrounding the current node, in order for S_I to occupy another sub-box, at least one surrounding $2^i \times 2^i$ sub-box must be exited. This requires covering a distance of at least 2^i, through a connected path of nodes. Once this happens, S_I has just crossed the boundary between an occupied sub-box and an unoccupied sub-box. Subsequently, it has just crossed the boundary between an occupied sub-box and an unoccupied sub-box. Then, by giving it for free at most 5 more unoccupied sub-boxes, S_I has to pay

another 2^i nodes to occupy another unoccupied sub-box; see Figure 10 (b). We then continue applying this 5-for-free strategy until all n nodes have been used.



(a) (b)

Figure 10: (a) A node of shape S_I in red and the occupied sub-boxes that we give for free to the shape. (b) The shape just exited the sub-box with arrow entering an unoccupied sub-box. By giving the 5 horizontally dashed sub-boxes for free, a distance of at least 2^i has to be travelled in order to reach another unoccupied sub-box.

To sum up, the shape has been given 8 sub-boxes for free, and then for every sub-box covered it has to pay 2^i and gets 5 sub-boxes. Thus, to occupy $k = 8 + l \cdot 5$ sub-boxes, at least $l \cdot 2^i$ nodes are needed, that is,

$$N_k \geq l \cdot 2^i. \tag{1}$$

But, that leads to

$$k = 8 + l \cdot 5 \Rightarrow l = \frac{k-8}{5}. \tag{2}$$

Thus, from (1) and (2):

$$N_k \geq \frac{k-8}{5} \cdot 2^i. \tag{3}$$

But shape S_I has order n, which means that the number of nodes available is upper bounded by n, i.e., $N_k \leq n$, which gives:

$$\begin{gathered}\frac{k-8}{5} \cdot 2^i \leq N_k \leq n \Rightarrow \\ \frac{k-8}{5} \cdot 2^i \leq n \Rightarrow \frac{k-8}{5} \leq \frac{n}{2^i} \Rightarrow \\ k \leq 5\left(\frac{n}{2^i}\right) + 8.\end{gathered}$$

We conclude that the number of $2^i \times 2^i$ sub-boxes that can be occupied by a connected shape S_I, and, thus, also the number of $2^i \times 2^i$ sub-boxes that are occupied by *U-Box-Doubling* in phase i, is at most $5(n/2^i) + 8 = O(n/2^i)$. □

As a corollary of this, we obtain:

Corollary 1. *Given a uniform partitioning of $n \times n$ square box containing a connected shape S_I of order n into $d \times d$ sub-boxes, it holds that S_I can occupy at most $O(\frac{n}{d})$ sub-boxes.*

We are now ready to analyse the running time of *U-Box-Doubling*.

Lemma 6. *Starting from any connected shape of n nodes,* U-Box-Doubling *performs $O(n \log n)$ steps during its course.*

Proof. We prove this by showing that in every phase i, $1 \leq i \leq \log n$, the transformation performs at most a linear number of steps. We partition the occupied $2^i \times 2^i$ sub-boxes into two disjoint sets, B_1 and B_0, where sub-boxes in B_1 have at least 1 *complete line* (from the previous phase), i.e., a line of length 2^{i-1}, and sub-boxes in B_0 have 1 to 4 *incomplete lines*, i.e., lines of length between 1 and $2^{i-1}-1$. For B_1, we have that $|B_1| \leq n/2^{i-1}$. In each phase, we may have horizontal or vertical lines that need to be aligned to the left or bottom boundary of their $2^i \times 2^i$ sub-box, respectively, depending on the parity of i. As the two cases are symmetric, without loss of generality we only show horizontal lines which are moving to their left. Hence, for every complete line, we pay at most 2^{i-1} to transfer it left. As there are at most $n/2^{i-1}$ such complete lines in phase i, the total cost for this is at most $2^i \cdot (n/2^{i-1}) = n$.

Each sub-box in B_1 may also have at most 4 incomplete lines from the previous phase, as in Figure 7 left, where at most two of them may have to pay a maximum of 2^{i-1} to be transferred left (as the other two are already aligned). As there are at most $n/2^{i-1}$ sub-boxes in B_1, the total cost for this is at most $2 \cdot 2^{i-1} \cdot (n/2^{i-1}) = 2n$. Therefore, the total cost for pushing all lines towards the required border in B_1 sub-boxes is at most:

$$n + 2n = 3n. \tag{4}$$

For B_0, we have (by Lemma 5) that the total number of occupied sub-boxes in phase i is at most $5(n/2^i)+8$, therefore, $|B_0| \leq 5(n/2^i)+8$ (taking into account also the worst case where every occupied sub-box may be of type B_0). There is again a maximum of 2 incomplete lines per such sub-box that need to be transferred a distance of at most 2^{i-1}, therefore, the total cost for this to happen in every B_0 sub-box is at most:

$$2 \cdot 2^{i-1}\left(5 \cdot \frac{n}{2^i} + 8\right) = 5n + 8 \cdot 2^i \leq 13n. \tag{5}$$

By paying the above costs, all occupied sub-boxes have their lines aligned to the left, and the final task of the transformation for this phase is to apply a linear procedure in order to fill in the left boundary of the $2^i \times 2^i$ sub-box. This procedure costs at most $2k$ for every k nodes aligned as above, in a total of at most $(2n-2)$ steps (see Lemma 1). As there is an additional cost of 2^{i-1} for an incomplete line to be transferred into the bottom left, as shown in Figure 8, the total cost for this phase is at most:

$$2n - 2 + (2^{i-1}) \leq 3n. \tag{6}$$

This completes the operation of *U-Box-Doubling* for phase i. Putting (4), (5), and (6) together, we obtain that the total cost T_i, in steps, for phase i is,

$$\begin{aligned} T_i &\leq 3n + 13n + 3n \\ &= 19n. \end{aligned}$$

As there is a total of $\log n$ phases, we conclude that the total cost T of the transformation is,

$$\begin{aligned} T &\leq 19n \cdot \log n \\ &= O(n \log n). \end{aligned}$$

□

Finally, together Lemma 3, Lemma 6, and reversibility (Lemma 2) imply that:

Theorem 1. *For any pair of connected shapes S_I and S_F of the same order n, transformation* U-Box-Doubling *can be used to transform S_I into S_F (and S_F into S_I) in $O(n \log n)$ steps.*

4. Preserving Connectivity: Transforming the Diagonal into a Line through Folding in $O(n\sqrt{n})$ time

In this section, we study the case in which the transformation *cannot break connectivity* during its course. We identify the diagonal connected shape S_D of order n as an initial potential worst-case shape to be transformed into a straight line S_L. Our goal is then to transform S_D into S_L *while preserving connectivity* during transformations, i.e., solve the DiagonalToLine problem. We do this, because this problem, as mentioned before in Section 1.1, seems to capture the worst-case complexity of transformations in this model.

Consider an S_D of n nodes occupying $(x, y), (x+1, y+1), \ldots, (x+n-1, y+n-1)$, such as the diagonal line of 25 nodes depicted in Figure 12. Observe that the diagonal comprises some special properties which cannot be found in other connected shapes. First, this staircase shape of stairs of length 1 has a special property in which each single node occupies a unique row and a unique column. Hence, it is much harder than the staircase worst-case shape of [MSS19] of stairs of length 2, where most rows and columns contain two nodes. Further, this particular diagonal shape consists of n lines of length 1, which is the maximum possible number of lines a connected shape can have. Below, we give an $O(n\sqrt{n})$-time transformation to transform S_D into S_L by exploiting the line-pushing mechanism, while preserving connectivity of the shape throughout transformations.

This strategy, called *DLC-Folding*, divides S_D into $\sqrt{n}$ segments each of length $\sqrt{n}$ and proceeds in $\sqrt{n}$ phases. Informally, *DLC-Folding* performs two different operations, *turn* and *push* to fold the diagonal segments in every phase as follows: *turn* diagonals into straight lines, *push* the lines $\sqrt{n}$ distance towards the following diagonal segment and then inversely *turn* the lines again into diagonals. Figure 11 shows the transformations of the first phase that is folding the top segment of S_D. The strategy keeps folding segments above each other, until arriving at the bottom segment. By the end of the final phase, *DLC-Folding* forms a *nice shape*, which can then be trivially transformed into a line (see Proposition 3). Figures 12, 13, 14, 15, and 16 demonstrate the above transformations on a diagonal of 25 nodes.

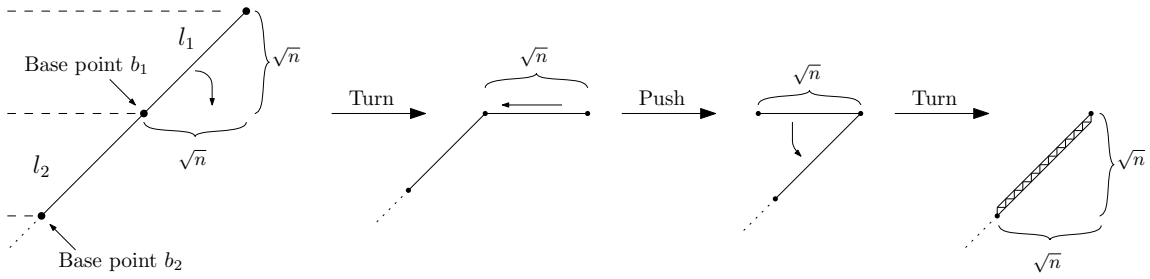


Figure 11: Folding the top segment of the diagonal line.

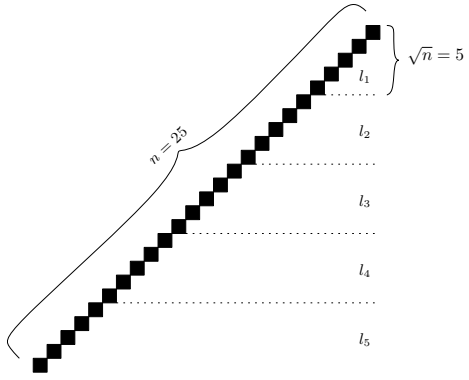


Figure 12: A diagonal line of 25 nodes.

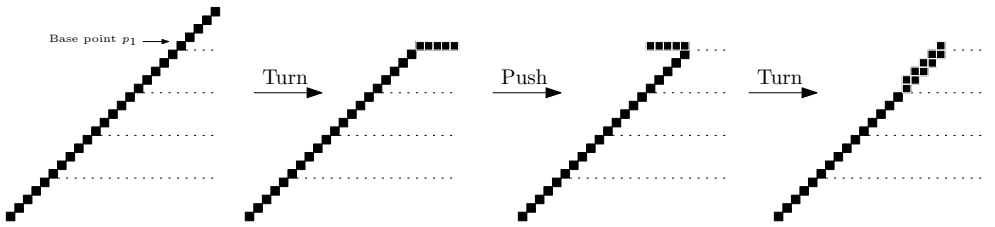


Figure 13: The first phase of folding.

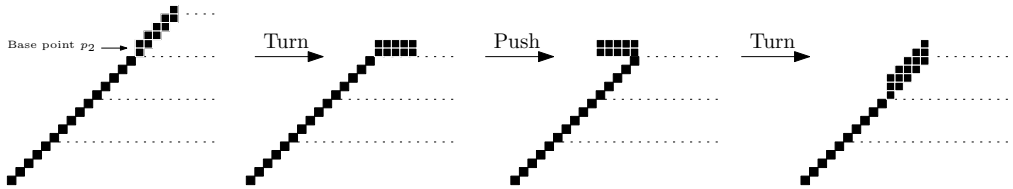


Figure 14: The second phase of folding.

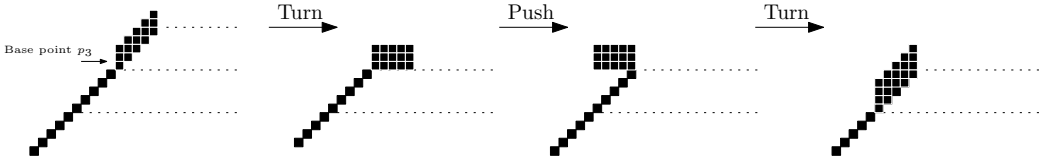


Figure 15: The third phase of folding.

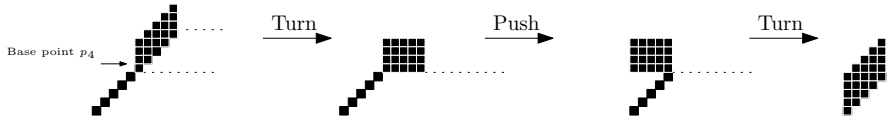


Figure 16: The fourth phase of folding. Notice the resulting connected shape in the far right represents a *nice* shape.

More formally, let S_D be a diagonal of n nodes occupying $(x, y), (x+1, y+1), \ldots, (x+n-1, y+n-1)$, such that (x, y) is the left bottom point of S_D. Then, S_D is divided into $\sqrt{n}$ segments, $l_1, l_2, \ldots, \sqrt{n}$, each of which has a length of $\sqrt{n}$, where l_1 and $l_{\sqrt{n}}$ are the top and bottom segments of S_D, respectively. Each segment l_k, $1 \leq k \leq \sqrt{n}$, consists of $\sqrt{n}$ nodes occupying $(i, j), (i+1, j+1), \ldots, (i+\sqrt{n}-1, j+\sqrt{n}-1)$, where $i = x + h_k$ and $j = y + h_k$, for $h_k = n - k\sqrt{n}$. Here, l_k has a base point $b_k = (i, j)$, which is the bottommost node of l_k.

In the first phase, the top segment l_1 folds around b_1 as follows *(due to symmetry, it is sufficient to demonstrate one orientation)*; (1) *Turn* all $\sqrt{n}$ nodes into the bottommost row of l_1 (brute-force line formation). Consider that the l_1 nodes change their positions from $(i, j), (i+1, j+1), \ldots, (i+\sqrt{n}-1, j+\sqrt{n}-1)$ into $(i, j), (i+1, j), \ldots, (i+\sqrt{n}-1, j)$. By the end of this operation, a horizontal line segment of length $\sqrt{n}$ is formed, and the base point b_1 keeps place at (i, j). Then, (2) *Push* the line segment l_1 a distance of $\sqrt{n}$ towards the leftmost column y of S_D. All $\sqrt{n}$ nodes of l_1 transfer altogether into $(i-\sqrt{n}, j), (i+1-\sqrt{n}, j), \ldots, (i+\sqrt{n}-1-\sqrt{n}, j)$. Finally, perform an inverse operation of (1), which converts the line segment l_1 into diagonal again to align above the next following diagonal segment l_2, by transferring them into positions $(i-\sqrt{n}, j-\sqrt{n}+1), (i+1-\sqrt{n}, j-\sqrt{n}+2), \ldots, (i+\sqrt{n}-1-\sqrt{n}, j)$, (except the base point b_1 which stays still in place, at $(i-\sqrt{n}, j)$). By the end of this phase, two parallel diagonal segments l_1 and l_2 have been created, as in Figure 11.

Then, a new *connected* shape is formed, which can be defined as follows:

Definition 8. *A Ladle is a connected shape of order n consisting of two parts, $\mathcal{D}$ and $\mathcal{S}$. For a given phase k, where $2 \leq k \leq \lceil \sqrt{n} \rceil$, we have $Ladle_k = \mathcal{D}_k + \mathcal{S}_k$, where both parts are connected via a base point $b_k = (i, j)$, such that:*

- $\mathcal{D}_k$, is a diagonal containing $n - k\sqrt{n} + 1$ nodes occupying $(x, y), (x + 1, y + 1), \ldots, (i, j)$, such that (x, y) are the left bottom point of $Ladle_k$, where $\sqrt{n} < i = j < n - \sqrt{n} + 1$. $\mathcal{D}_k$ is connected to $\mathcal{S}_k$ via (i, j).

- $\mathcal{S}_k$, is parallelogram consists of k parallel diagonal segments of size $k\sqrt{n}$ nodes formed $\sqrt{n}$ lines. $\mathcal{S}_k$ is connected to $\mathcal{D}_k$ via its bottommost node at (i, j), as depicted in Figure 17.

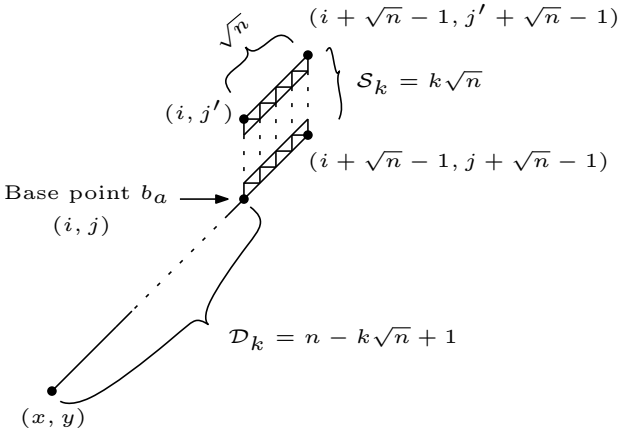


Figure 17: A *Ladle* shape in phase k, where $j\prime = j + k - 1$.

The following lemmas prove *correctness* of *DLC-Folding*:

Lemma 7. *Let S_D be a diagonal of order n partitioned into $\lceil \sqrt{n} \rceil$ segments $l_1, l_2, ..., l_{\sqrt{n}}$. DLC-Folding converts S_D into a Ladle by the end of the first phase.*

Proof. Consider a diagonal S_D of n nodes as defined above, which partitioned into $\sqrt{n}$ segments of length $\sqrt{n}$ each. Due to symmetry, it is sufficient to show the implementation on the top segment of S_D. Given that, we will obtain a connected shape consists of two parts, (1) a diagonal part occupies $(x, y), (x + 1, y + 2), \ldots, (x + n - \sqrt{n} - 1, y + n - \sqrt{n} - 1)$ and (2) two parallel diagonal segments of $2\sqrt{n}$ nodes. Both are connected via the base point $(x + n - \sqrt{n} - 1, y + n - \sqrt{n} - 1)$, which is the topmost node of the diagonal part and the bottommost of the two parallel diagonal segments. As a result, the new shape constructed by the end of the first phase meets all conditions mentioned in Definition 8, therefore, it is a *Ladle*. □

Lemma 8. *Consider a Ladle of n nodes in phase k, where $1 < k \leq \sqrt{n}$. Then, in phase $k+1$, DLC-Folding increases the size of $\mathcal{S}_k$ by $\sqrt{n}$ and decreases the length of $\mathcal{D}_k$ by $\sqrt{n}$.*

Proof. The size of the *Ladle* $= |n|$ must be the same each phase and all time. In phase k, a $Ladle_k$ consists of two parts, $\mathcal{D}_k = |n - k\sqrt{n} + 1|$ and $\mathcal{S}_k = |k\sqrt{n}|$ connected via a common node (i, j) (*see Definition 8*). Now, fold the $\mathcal{S}_k$ part that contains k segments of length $\sqrt{n}$ aligned diagonally on top of each other. Without loss of generality, move all $\sqrt{n}$ vertical lines downwards to the bottommost row i of $\mathcal{S}_k$, which shall form k horizontal lines by completely filling in the k bottom rows of $\mathcal{S}_k$. Hence, those horizontal lines create a rectangle, as depicted in Figure 18 (a). Then, push the k horizontal lines $\sqrt{n}$ distance towards the left, see Figure 18 (b). Lastly, the strategy *turns* these lines inversely above the next diagonal segment *(notice that every vertical line is moved except the rightmost one)*, see Figure 18 (c). By the end of phase $k + 1$, a new *Ladle* has been created, which is consisting of $\mathcal{D}_{k+1} = |n - (k-1)\sqrt{n} + 1|$ and $\mathcal{S}_{k+1} = |(k+1)\sqrt{n}|$ connected via the common b_{k+1} base point at $(i - \sqrt{n}, j - \sqrt{n})$. Therefore, we conclude that in phase $k + 1$, the size

of $\mathcal{S}_{k+1}$ increased by $\sqrt{n}$ nodes, while the length of $\mathcal{D}_{k+1}$ decreased by $\sqrt{n}$. Thus, this holds trivially and inductively for any phase k, where $1 < k \leq \sqrt{n}$. □

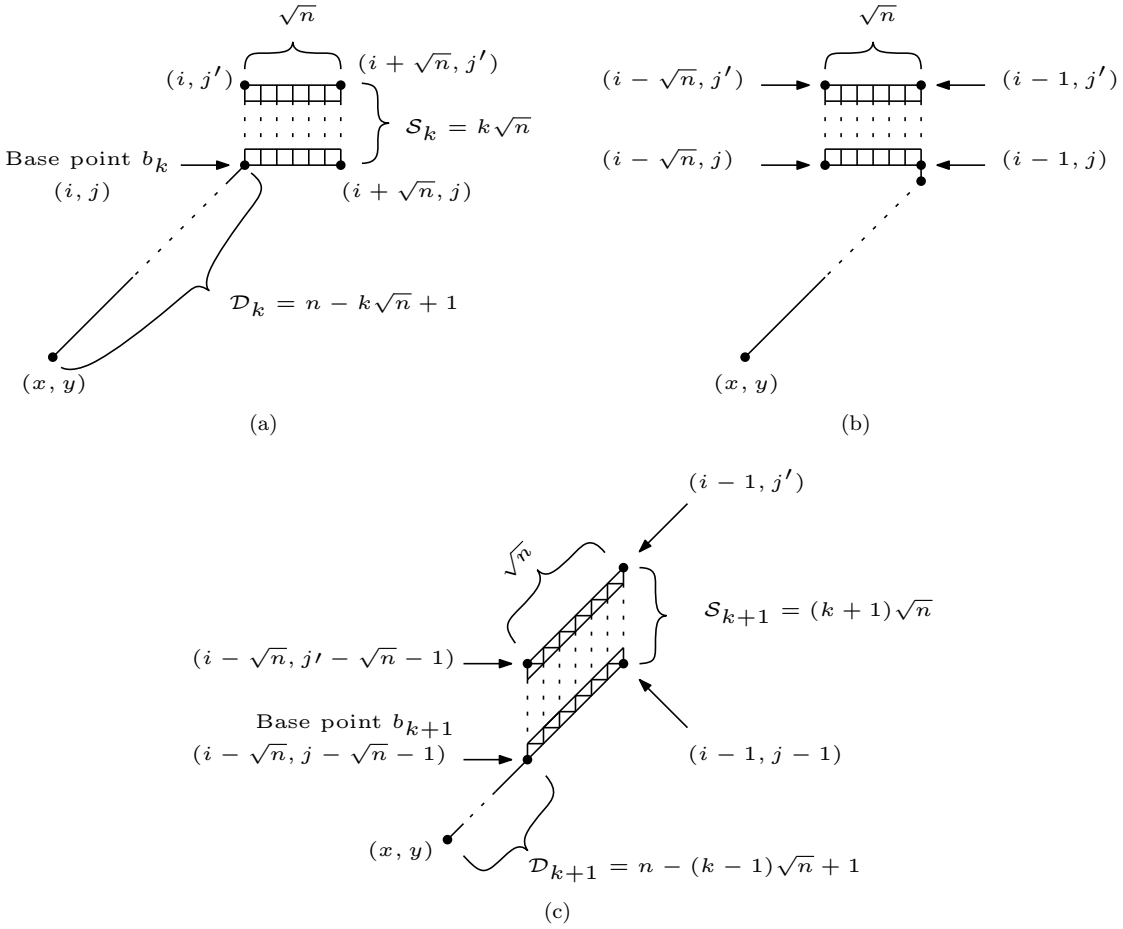


Figure 18: Folding a $Ladle_k$ over phase k, , where $j\prime = j + k - 1$, see Lemma 8 for further explanation.

Next, we prove that *DLC-Folding* transforms S_D into a *nice* shape in $\sqrt{n}$ phases.

Lemma 9. *Given a diagonal S_D of order n partitioned into $\sqrt{n}$ segments, DLC-Folding converts S_D into a nice shape in $\sqrt{n}$ phases.*

Proof. By Lemma 7, S_D converts to a $Ladle_2$ which consists of two parts $\mathcal{D}_2 = |n-2\sqrt{n}+1|$ and $\mathcal{S}_2 = |2\sqrt{n}|$. Then, by Lemma 8, through the final phase $k = \sqrt{n}$, the diagonal part of the *Ladle* will be exhausted $\mathcal{D}_{\sqrt{n}} = \phi$, whilst the parallelogram part acquires all n nodes, $\mathcal{S}_{\sqrt{n}} = |n|$, by folding all segments diagonally over each other. By the end of the final phase, the resulting new shape of $\sqrt{n}$ vertical (horizontal) lines is a *nice shape*. □

Now, we are ready to analyse the running time of *DLC-Folding* that preserves connectivity over its course.

Lemma 10. *Given a diagonal S_D of order n partitioned into $\sqrt{n}$ segments, DLC-Folding folds the topmost (bottommost) segment in $O(n)$ steps.*

Proof. In the first phase, the top (bottom) segment of S_D of length $\sqrt{n}$ *turns* into a *line segment* by a brute-force line formation (similar of Figure 1 (b)) that is trivially computed by:

$$1 + 2 + ... + (\sqrt{n} - 1) = \frac{\sqrt{n}(\sqrt{n} - 1)}{2} = \frac{n - \sqrt{n}}{2} = O(n).$$

After that, *DLC-Folding* performs *push* operation on that *line segment*in a total cost of $O(\sqrt{n})$ steps and *turn* it again inversely into diagonal with the same cost of $(n - \sqrt{n})/2 = O(n)$. Therefore, the first segment folds in a total cost of at most:

$$\begin{aligned} t_1 &= \frac{n - \sqrt{n}}{2} + \sqrt{n} + \frac{n - \sqrt{n}}{2} = n - \sqrt{n} + \sqrt{n} = n \\ &= O(n). \end{aligned}$$

□

Lemma 11. *By the end of phase k, for all* $1 < k \leq \sqrt{n}$*, DLC-Folding folds* $Ladle_k$ *in* $O(n)$ *steps.*

Proof. In phase k, we have $Ladle_k$ consists of two parts, $\mathcal{D}_k = |n - k\sqrt{n} + 1|$ and $\mathcal{S}_k = |k\sqrt{n}|$. *DLC-Folding* *turns* all $\sqrt{n}$ lines of $\mathcal{S}_k$ in a total run of steps at most:

$$1 + 2 + ... + (\sqrt{n} - 1) = \frac{\sqrt{n}(\sqrt{n} - 1)}{2} = \frac{n - \sqrt{n}}{2} = O(n). \tag{7}$$

Now, the $\sqrt{n}$ lines have moved and formed another k horizontal lines in a different orientation. *DLC-Folding* pushes those k lines a distance of $\sqrt{n}$ in a total of at most:

$$k\sqrt{n} = O(n), \tag{8}$$

steps.

Then, the $\sqrt{n}$ lines *turn* diagonally above the following segment incurring the same cost of (7) by at most:

$$\frac{n - \sqrt{n}}{2} = O(n). \tag{9}$$

With this, the total cost of phase k is given by summing (7), (8), and (9):

$$\begin{aligned} t_k &= \frac{n - \sqrt{n}}{2} + k\sqrt{n} + \frac{n - \sqrt{n}}{2} = n - \sqrt{n} + k\sqrt{n} \\ &= O(n). \end{aligned} \tag{10}$$

Finally, this trivially holds from phase 2 and inductively for every phase k, for all $1 < k \leq \sqrt{n}$. □

Altogether, Proposition 3 and Lemmas 10 and 11, the running time of *DLC-Folding* is,

Theorem 2. *Given an initial connected diagonal of n nodes, DLC-Folding solves the* DiagonalToLine *problem in* $O(n\sqrt{n})$ *steps.*

Proof. By Lemma 10, *DLC-Folding* creates a *Ladle* in a total cost of:

$$T_1 = \frac{n - \sqrt{n}}{2}. \tag{11}$$

Now, by Lemma 11, the total running time for all k phases, $1 < k \leq \sqrt{n}$, is given as follows:

$$\begin{aligned}
T_2 &= \sum_{i=1}^{\sqrt{n}-1} n - \sqrt{n} + i\sqrt{n} = n\sqrt{n} - 2n - \sqrt{n} + \sum_{i=1}^{\sqrt{n}-1} i\sqrt{n} \\
&= n\sqrt{n} - 2n - \sqrt{n} + \sqrt{n} \sum_{i=1}^{\sqrt{n}-1} i = n\sqrt{n} - 2n - \sqrt{n} + n\left(\frac{\sqrt{n}-1}{2}\right) \\
&= n\sqrt{n} - 2n - \sqrt{n} + \left(\frac{n\sqrt{n}-n}{2}\right) = \frac{n\sqrt{n} - 5n - 2\sqrt{n}}{2} \\
&= O(n\sqrt{n}). \qquad (12)
\end{aligned}$$

By summing the cost of the first phase in (11) with (12):

$$\begin{aligned}
T_3 &= T_1 + T_2 \\
&= \frac{n-\sqrt{n}}{2} + \frac{n\sqrt{n} - 5n - 2\sqrt{n}}{2} = \frac{2n - 2\sqrt{n} + 2n\sqrt{n} - 10n - 4\sqrt{n}}{4} \\
&= \frac{2n\sqrt{n} - 8n - 6\sqrt{n}}{4} = \frac{n\sqrt{n} - 4n - 3\sqrt{n}}{2} \\
&= O(n\sqrt{n}). \qquad (13)
\end{aligned}$$

Finally, the resulting shape of *DLC-Folding* is a *nice* shape, which can be transformed into a line S_L in $O(n)$ steps *(see Proposition 3)*, therefore, the total cost T required to transform S_D into S_L, is bounded by:

$$\begin{aligned}
T &= T_3 + O(n) \\
&= O(n\sqrt{n}) + O(n) \\
&= O(n\sqrt{n}).
\end{aligned}$$

□

In the full report [AMP19b], we provide another $O(n\sqrt{n})$-time transformation, called DLC-Extending, which also preserves connectivity of the shape during the transformation through different strategy.

5. Conclusions

In this work, we studied a new linear-strength model of line moves. The nodes can now move in parallel by translating a line of any length by one position in a single time-step. This model, having the model of [DP04, MSS19] as a special case, adopts all its transformability results (including universal transformations). Then, our focus naturally turned to investigating if pushing lines can help achieve a substantial gain in performance (compared to the $\Theta(n^2)$-time of those models). Even though it can be immediately observed that there are instances in which this is the case (e.g., initial shapes in which there are many long lines, thus, much initial parallelism to be exploited), it was not obvious that this holds also for the worst case. We have successfully developed an $O(n \log n)$-time universal transformation that can transform any pair of connected shapes to each other. Further, by identifying the diagonal as a potentially worst-case shape (essentially, because in it any parallelism to be exploited does not come for free), we provide an $O(n\sqrt{n})$-time transformation that preserves connectivity when transforming the diagonal into a line.

There is a number of interesting problems that are opened by this work. The obvious first target (and apparently intriguing) is to answer whether there is an $o(n \log n)$-time transformation (e.g., linear) or whether there is an $\Omega(n \log n)$-time lower bound matching our best transformations. We suspect the latter, but do not have enough evidence to support or prove it. Moreover, we didn't consider parallel time in this paper. If more

than one line can move in parallel in a time-step, then are there variants of our transformations (or alternative ones) that further reduce the running time? In other words, are there parallelisable transformations in this model? In particular, it would be interesting to investigate whether the present model permits an $O(\log n)$ parallel time (universal) transformation, i.e., matching the best transformation in the model of Aloupis *et al.* [ACD+08]. It would also be worth studying in more depth the case in which connectivity has to be preserved during the transformations. In the relevant literature, a number of alternative types of grids have been considered, like triangular (e.g., in [DDG+14]) and hexagonal (e.g., in [WWA04]), and it would be interesting to investigate how our results translate there. Finally, an immediate next goal is to attempt to develop distributed versions of the transformations provided here.

6. Acknowledgements

We would like to thank the anonymous reviewers of this work and its preliminary versions for their thorough comments and suggestions, which have helped us improve our work substantially.

References

[AAD+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18[4]:235–253, March 2006.

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

[ABD+13] G. Aloupis, N. Benbernou, M. Damian, E. D. Demaine, R. Flatland, J. Iacono, and S. Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Computational geometry*, 46[8]:917–928, 2013.

[ACD+08] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer. Reconfiguration of cube-style modular robots using O(logn) parallel moves. In *International Symposium on Algorithms and Computation*, pages 342–353. Springer, 2008.

[AMP19a] A. Almethen, O. Michail, and I. Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. In *Algorithms for Sensor Systems - 15th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS*, 2019.

[AMP19b] A. Almethen, O. Michail, and I. Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *CoRR*, abs/1904.12777, 2019.

[BDF+19] A. T. Becker, E. D. Demaine, S. P. Fekete, J. Lonsford, and R. Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, 18[1]:181–201, 2019.

[BG15] J. Bourgeois and S. C. Goldstein. Distributed intelligent MEMS: progresses and perspective. *IEEE Systems Journal*, 9[3]:1057–1068, 2015.

[BKRT04] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *The International Journal of Robotics Research*, 23[9]:919–937, 2004.

[CFPS12] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41[4]:829–879, 2012.

[CKLWL09] A. Cornejo, F. Kuhn, R. Ley-Wild, and N. Lynch. Keeping mobile robot swarms connected. In *Proceedings of the 23rd international conference on Distributed computing*, DISC'09, pages 496–511, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[DDG+18] J. J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17[1]:81–96, 2018.

[DDL+09] S. M. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf, and W. M. Shih. Self-assembly of dna into nanoscale three-dimensional shapes. *Nature*, 459[7245]:414, 2009.

[Dem01] E. D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.

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

[DGMRP06] X. Défago, M. Gradinariu, S. Messika, and P. Raipin-Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2006.

[DGR+15] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, page 21. ACM, 2015.

[DGR+16] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–299. ACM, 2016.

[DLFP+18] G. A. Di Luna, P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Line recovery by programmable particles. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ICDCN '18, pages 4:1–4:10, New York, NY, USA, 2018. ACM.

[DLFS+19] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distributed Computing*, Mar 2019.

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

[DP04] A. Dumitrescu and J. Pach. Pushing squares around. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 116–123. ACM, 2004.

[DSY04a] A. Dumitrescu, I. Suzuki, and M. Yamashita. Formations for fast locomotion of metamorphic robotic systems. *The International Journal of Robotics Research*, 23[6]:583–593, 2004.

[DSY04b] A. Dumitrescu, I. Suzuki, and M. Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20[3]:409–418, 2004.

[FPS12] P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3[2]:1–185, 2012.

[FRRS16] S. Fekete, A. W. Richa, K. Römer, and C. Scheideler. Algorithmic foundations of programmable matter (Dagstuhl Seminar 16271). In *Dagstuhl Reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. Also in *ACM SIGACT News*, 48.2:87-94, 2017.

[GKR10] K. Gilpin, A. Knaian, and D. Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2485–2492. IEEE, 2010.

[HD05] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343[1-2]:72–96, 2005.

[KCL+12] A. N. Knaian, K. C. Cheung, M. B. Lobovsky, A. J. Oines, P. Schmidt-Neilsen, and N. A. Gershenfeld. 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*, pages 1447–1453. IEEE, 2012.

[KKM10] E. Kranakis, D. Krizanc, and E. Markou. The mobile agent rendezvous problem in the ring. *Synthesis Lectures on Distributed Computing Theory*, 1[1]:1–122, 2010.

[MS16] O. Michail and P. G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29[3]:207–237, 2016.

[MS18] O. Michail and P. G. Spirakis. Elements of the theory of dynamic networks. *Commun. ACM*, 61[2]:72–81, 2018.

[MSS19] O. Michail, G. Skretas, and P. G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019.

[NGY00] A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, pages 23–36, 2000.

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

[Rot06] P. W. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440[7082]:297–302, 2006.

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

[SMO+18] M. Shibata, T. Mega, F. Ooshita, H. Kakugawa, and T. Masuzawa. Uniform deployment of mobile agents in asynchronous rings. *Journal of Parallel and Distributed Computing*, 119:92–106, 2018.

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

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

[WWA04] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17[2]:171–189, 2004.

[YS10] M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411[26-28]:2433–2453, 2010.

[YSS+07] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14[1]:43–52, 2007.

[YUY16] Y. Yamauchi, T. Uehara, and M. Yamashita. Brief announcement: pattern formation problem for synchronous mobile robots in the three dimensional euclidean space. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 447–449. ACM, 2016.