On Efficient Connectivity-Preserving Transformations in a Grid

Abdullah Almethen, Othon Michail, and Igor Potapov

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

Abstract. We consider a discrete system of n devices lying on a 2-dimensional square grid and forming an initial connected shape S_I. Each device is equipped with a linear-strength mechanism which enables it to move a whole line of consecutive devices in a single time-step. We study the problem of transforming S_I into a given connected target shape S_F of the same number of devices, via a finite sequence of *line moves*. Our focus is on designing *centralised* transformations aiming at *minimising the total number of moves* subject to the constraint of *preserving connectivity* of the shape throughout the course of the transformation. We first give very fast connectivity-preserving transformations for the case in which the *associated graphs* of S_I and S_F are isomorphic to a Hamiltonian line. In particular, our transformations make $O(n \log n)$ moves, which is asymptotically equal to the best known running time of connectivity-breaking transformations. Our most general result is then a connectivity-preserving *universal transformation* that can transform any initial connected shape S_I into any target connected shape S_F, through a sequence of $O(n\sqrt{n})$ moves.

Submitted to Track A. The full paper is included in a clearly marked Appendix, to be read at the discretion of the Program Committee.

Keywords: line movement, programmable matter, transformation, shape formation, reconfigurable robotics, time complexity

1 Introduction

Over the past few years, many fascinating systems have been developed, leveraging advanced technology in order to deploy large collections of tiny monads. Each monad is typically a highly restricted micro-robotic entity, equipped with a microcontroller and some actuation and sensing capabilities. Through its collaborative complexity, the collection of monads can carry out tasks which are well beyond the capabilities of individual monads. The vision is the development of materials that will be able to algorithmically change their physical properties, such as their shape, colour, conductivity and density, based on transformations executed by an underlying program. These efforts are currently shaping the research area of *programmable matter*, which has attracted much theoretical and practical interest. The implementation indicates whether the monads are operated centrally or through local decentralised control. In *centralised* systems, there is an external program which globally controls all monads with full knowledge of the entire system. On the other hand, *decentralised* systems provide each individual monad with enough autonomy to communicate with its neighbours and move locally. There are an impressive number of recent developments for collective robotic systems, demonstrating their potential and feasibility, starting from the scale of milli or micro [9, 26, 29, 36] down to nano size of individual monads [20, 34].

Recent research has highlighted the need for the development of an algorithmic theory of such systems. Michail and Spirakis [32] and Michail *et al.* [30] emphasised an apparent lack of a formal theoretical study of this prospective, including modelling, possibilities/limitations, algorithms and complexity. The development of a formal theory is a crucial step for further progress in those systems. Consequently, multiple theoretical computer science sub-fields have appeared, such as metamorphic systems [23, 33, 37], mobile robotics [11, 13, 14, 18, 40], reconfigurable robotics [4, 10, 15, 17, 42], passively-mobile systems [6, 7, 31, 32], DNA self-assembly [19, 35, 38, 39], and the latest emerging sub-area of "Algorithmic Foundations of Programmable Matter" [25].

Consider a system deployed on a two-dimensional square grid in which a collection of spherical devices are typically connected to each other, forming a shape S_I. By a finite number of valid individual moves, S_I can be transformed into a desired target shape S_F. In this prospective, a number of models are designed and introduced in the literature for such systems. For example, Dumitrescu and Pach [21], Dumitrescu *et al.* [22, 23] and Michail *et al.* [30] consider mechanisms where an individual device is capable to move over and turn around its neighbours through empty space. Transformations based on similar moves being assisted by small seeds, have also been considered in [1]. A new linear-strength mechanism was introduced by Almethen *et al.* in [3], where a whole line of consecutive devices can, in a single time-step, move by one position in a given direction.

Since any two shapes with an equal number of elements can be transformed into each other with line moves [3], the central question remains about understanding the bounds on reachability distances between different shapes (configurations) via line moves. Proving exact reachability bounds can influence the design and analysis of both centralised and distributed algorithms. Our hypothesis is that the reachability distances between any two shapes with n elements can be bounded by $O(n \log n)$, and the bound cannot be improved for a simple pair of shapes such as diagonal and horizontal lines.

In this paper, we embark from the *line-pushing* model of [3], which provided sub-quadratic centralised transformations that may, though, arbitrarily break connectivity of the shape during their course. As our main goal is to investigate the power of the line-pushing model, we focus solely on centralised transformations, as a first step. That is because distributed are model-dependent (e.g., knowledge, communication, etc.), while centralised show what is *in principle* possible. Moreover some of the ideas in centralised might prove useful for distributed and of course lower bounds also transfer to the distributed case. The only connectivity-preserving transformation in [3] was an $O(n\sqrt{n})$-time transformation for a single pair of shapes of order n, namely from a diagonal into a straight line. All transformations that we provide in the present study preserve connectivity of the shape during the transformation.

We first give very fast connectivity-preserving transformations for the case in which the *associated graphs* of S_I and S_F are isomorphic to a Hamiltonian line. In particular, our transformations make $O(n \log n)$ moves, which is asymptotically equal to the best known running time of connectivity-breaking transformations. Our most general result is then a connectivity-preserving *universal transformation* that can transform any initial connected shape S_I into any target connected shape S_F, through a sequence of $O(n\sqrt{n})$ moves.

1.1 Related Work

For the models of individual moves where only one node moves in a single time-step, [21, 30] show universality of transforming any pair of connected shapes (A, B) having the same number of devices (called *nodes* throughout this paper) to each other via sliding and rotation mechanisms. By allowing only rotation, [30] proves that the problem of deciding transformability is in **P**. It can be shown that in all models of constant-distance individual moves, $\Omega(n^2)$ moves are required to transform some pairs of connected shapes, due the inherent distance between them [30]. This motivates the study of alternative types of moves that are reasonable with respect to practical implementations and allow for sub-quadratic reconfiguration time in the worst case.

There are attempts in the literature to provide alternatives for more efficient reconfiguration. The first main approach is to explore parallel transformations, where multiple nodes move together in a single time-step. This is a natural step to tackle such a problem, especially in distributed systems where nodes can make independent decisions and move locally in parallel to other nodes. There are a number of theoretical studies on parallel and distributed transformations [15, 16, 18, 23, 30, 41] as well as practical implementations [36]. For example, it can be shown that a connected shape can transform into any other connected shape, by performing in the worst case $O(n)$ parallel moves around the perimeter of the shape [30].

The second approach aims to equip nodes in the system with a more powerful mechanism which enables them to reduce the inherent distance by a factor greater than a constant in a single time-step. There are a number of models in the literature in which individual nodes are equipped with strong actuation mechanisms, such as linear-strength mechanisms. Aloupis *et al.* [4, 5] provide a node with arms that are capable to extend and contract a neighbour, a subset of the nodes or even the whole shape as a consequence of such an operation.

Further, Woods *et al.* [39] proposed an alternative linear-strength mechanism, where a node has the ability to rotate a whole line of consecutive nodes.

Recently, the *line-pushing* model of [3] follows a similar approach in which a single node can move a whole line of consecutive nodes by simultaneously (i.e., in a single time-step) pushing them towards an empty position. The line-pushing model can simulate the rotation and sliding based transformations of [21, 30] with at most a 2-factor increase in their worst-case running time. This implies that all transformations established for individual nodes, transfer in the line-pushing model and their universality and reversibility properties still hold true. They achieved sub-quadratic time transformations, including an $O(n \log n)$-time universal transformation which does not preserve connectivity and a connectivity-preserving $O(n\sqrt{n})$-time transformation for the special case of transforming a diagonal into a straight line.

Another relevant line of research has considered a single moving robot that transforms an otherwise static shape by carrying its tiles one at a time [13, 24, 27]. Those models are partially centralised as a single robot (usually a finite automaton) controls the transformation, but, in contrast to our perspective, control in that case is local and lacking global information.

1.2 Our Contribution

In this work, we build upon the findings of [3] aiming to design very efficient and general transformations that are additionally able to keep the shape connected throughout their course. We first give an $O(n \log n)$-time transformation, called *Walk-Through-Path*, that works for all pairs of shapes (S_I, S_F) that have the same order and belong to the family of *Hamiltonian shapes*. A *Hamiltonian shape* is any connected shape S whose *associated graph* $G(S)$ is isomorphic to a Hamiltonian path (see also [28]). At the heart of our transformation is a recursive successive doubling technique, which starts from one endpoint of the Hamiltonian path and proceeds in $\log n$ phases (where n denotes the order of the input shape S_I, throughout this paper). In every phase i, it moves a terminal line L_i of length 2^i a distance 2^i higher on the Hamiltonian path through a *LineWalk* operation. This leaves a new terminal sub-path S_i of the Hamiltonian path, of length 2^i. Then the general procedure is recursively called on S_i to transform it into a straight line L'_i of length 2^i. Finally, the two straight lines L_i and L'_i which are perpendicular to each other are combined into a new straight line L_{i+1} of length 2^{i+1} and the next phase begins.

A core technical challenge in making the above transformation work is that Hamiltonian shapes do not necessarily provide free space for the *LineWalk* operation. Thus, moving a line has to take place through the remaining configuration of nodes while at the same time ensuring that it does not break their and its own connectivity, including keeping itself connected to the rest of the shape. We manage to overcome this by revealing a nice property of line moves, according to which a line L can *transparently* walk through *any* configuration S (independently of the latter's density) in a way that: (i) preserves connectivity of both L and S and (ii) as soon as L has gone through it, S has been restored to its original state, that is, all of its nodes are lying in their original positions. This property is formally proved in Proposition 1 (Section 2).

Finally, we develop a *universal transformation*, called *UC-Box*, that within $O(n\sqrt{n})$ moves transforms any pair of connected shapes of the same order to each other, while preserving connectivity throughout its course. Starting from the initial shape S_I, we first compute a spanning tree T of S_I. Then we enclose the shape into a square box of size n and divide it into sub-boxes of size $\sqrt{n}$, each of which contains at least one sub-tree of T. By moving lines in a way that does not break connectivity, we compress the nodes in a sub-box into an adjacent sub-box towards a parent sub-tree. By carefully repeating this we manage to arrive at a final configuration which is always a compressed square shape. The latter is a type of a *nice* shape (a family of connected shapes introduced in [3]), which can be transformed into a straight line in linear time. We provide an analysis of this strategy based on the number of *charging phases*, which turns out to be $\sqrt{n}$, each making at most n moves, for a total of $O(n\sqrt{n})$ moves.

Section 2 formally defines the model and the problems under consideration and proves a basic proposition which is a core technical tool in one of our transformations. Section 3 presents our $O(n \log n)$-time transformation for Hamiltonian shapes. Section 4 discusses our universal $O(n\sqrt{n})$-time transformation. Finally, in Section 5 we conclude and discuss interesting problems left open by our work.

2 Preliminaries

All transformations in this study operate on a two-dimensional square grid, in which each cell has a unique position of non-negative integer coordinates (x, y), where x represents columns and y denotes rows in the grid. A set of n nodes on the grid forms a shape S (of the order n), where every single node $u \in S$ occupies only one cell $cell(u) = (u_x, u_y)$. A node u can be indicated at any given time by the coordinates (u_x, u_y) of the unique cell that it occupies at that time. A node $v \in S$ is a *neighbour* of (or *adjacent* to) a node $u \in S$ if and only if their coordinates satisfy $u_x - 1 \leq v_x \leq u_x + 1$ and $u_y - 1 \leq v_y \leq u_y + 1$ (i.e., their cells are adjacent vertically, horizontally or diagonally). A graph $G(S) = (V, E)$ is *associated* with a shape S, where $u \in V$ iff u is a node of S and $(u, v) \in E$ iff u and v are neighbours in S. A shape S is connected iff $G(S)$ is a connected graph. We denote by $T(S)$ (or just T when clear from context) a spanning tree of $G(S)$. In what follows, n denotes the number of nodes in a shape under consideration, and all logarithms are to base 2.

In this paper, we exploit the linear-strength mechanism of the *line-pushing model* introduced in [3]. A line L is a sequence of nodes occupying consecutive cells in one direction of the grid, that is, either vertically or horizontally but not diagonally. A ***line move*** is an operation of moving all nodes of L together in a single time-step towards a position adjacent to one of L's endpoints, in a given direction d of the grid, $d \in \{up, down, right, left\}$. A line move may also be referred to as *step*, *move*, or *movement* in this paper. Throughout, the running time of transformations is measured in total number of line moves until completion. A *line move* is formally defined below.

Definition 1 (A permissible line move). *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 at $(x+k, y)$. The "down", "left", and "up" moves are defined symmetrically, by rotating the whole shape $90°$,$180°$, and $270°$ clockwise, respectively.*

We next define a family of shapes that are used in one of our transformation.

Definition 2 (Hamiltonian Shapes). *A shape S is called Hamiltonian iff $G(S) = (V, E)$ is isomorphic to a Hamiltonian path, i.e., a path starting from a node $u \in V$, visiting every node in V exactly once and ending at a node $v \in V$, where $v \neq u$. $\mathcal{H}$ denotes the family of all Hamiltonian shapes.*

The following proposition proves a basic property of line moves which will be a core technical tool in our transformation for Hamiltonian shapes.

Proposition 1 (Transparency of Line Moves). *Let S be any shape, $L \subseteq S$ any line and P any path of cells in the grid (under the vertical and horizontal neighbouring relation) starting from a position adjacent to one of L's endpoints. Let $C(P)$ denote the configuration of P defined by S. There is a way to move L along P, while satisfying all the following properties:*

1. No delay: *The number of steps is asymptotically equal to that of an optimum move of L along P in the case of $C(P)$ being empty (i.e., if no cells were occupied). That is, L is not delayed, independently of what $C(P)$ is.*
2. No effect: *After L's move along P, $C'(P) = C(P)$, i.e., the cell configuration has remained unchanged. Moreover, no occupied cell in $C(P)$ is ever emptied during L's move (but unoccupied cells may be temporarily occupied).*
3. No break: *S remains connected throughout L's move.*

Proof. Whenever L walks through an empty cell (x, y) of P, a node $u \in L$ fills in (x, y). If L pushes the node u of a non-empty cell of P, a node $v \in L$ takes its place. When L leaves a non-empty cell (x, y) that was originally occupied by node v, L restores (x, y) by leaving its endpoint $u \in L$ in (x, y). Finally, Figure 1 shows how to deal with the case in which L turns at a non-empty corner-cell (x, y) of P, which is only connected diagonally to a non-empty cell of S and is not adjacent to any cell occupied by L.

We now formally define all problems considered in this work.

HamiltonianConnected**.** Given a pair of connected Hamiltonian shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, transform S_I into S_F while preserving connectivity throughout the transformation.

DiagonalToLineConnected**.** A special case of HamiltonianConnected in which S_I is a diagonal and S_F is a straight line.

UniversalConnected**.** Given *any* pair of connected shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, transform S_I into S_F while preserving connectivity throughout the transformation.

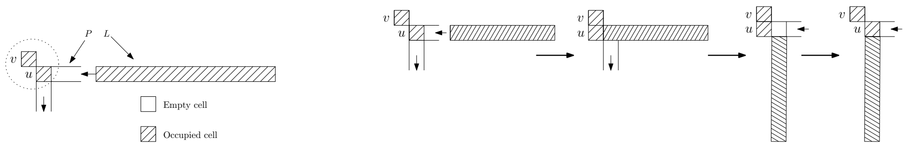
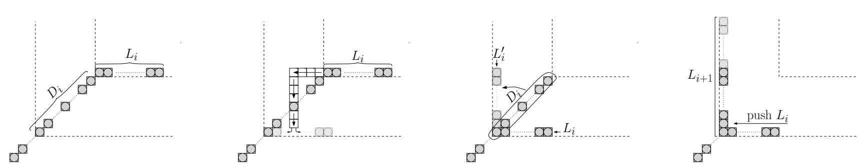


Fig. 1: A line L moving through a path P and arriving at a turning point of P. u occupies a corner cell of P and v occupies a cell of S and is only connected diagonally to u while not being adjacent to any cell occupied by L. L pushes u one position horizontally and turns all of its nodes vertically. Then u moves back to its original position in P. All other orientations are symmetric and follow by rotating the shape 90°, 180° or 270°.

3 An $O(n \log n)$-time Transformation for Hamiltonian Shapes

In this section, we present a strategy for HamiltonianConnected, called *Walk-Through-Path*. It transforms any pair of shapes $S_I, S_F \in \mathcal{H}$ of the same order to each other within $O(n \log n)$ moves while preserving connectivity of the shape throughout the transformation. Recall that $\mathcal{H}$ is the family of all Hamiltonian shapes. Our transformation starts from one endpoint of the Hamiltonian path of S_I and applies a recursive successive doubling technique to transform S_I into a straight line S_L in $O(n \log n)$ time. By replacing S_I with S_F in *Walk-Through-Path* and reversing the resulting transformation, one can then go from S_I to S_F in the same asymptotic time. We first demonstrate the core recursive technique of this strategy in a special case which is sufficiently sparse to allow local reconfigurations without the risk of affecting the connectivity of the rest of the shape. In this special case, S_I is a diagonal of any order and observe that $S_I, S_F \in \mathcal{H}$ holds for this case. We then generalise this recursive technique to work for any $S_I \in \mathcal{H}$ and add to it the necessary sub-procedures that can perform local reconfiguration in *any* area (independently of how dense it is), while ensuring that global connectivity is always preserved.



(a) A line L_i and a diagonal segment D_i both of length 2^i.

(b) L_i moves through a shortest path towards the far endpoint of D_i.

(c) D_i recursively transforms into a line L'_i .

(d) A line L_{i+1} of length 2^{i+1} formed by combining L_i and L'_i.

Fig. 2: A snapshot of phase i of *Walk-Through-Path* applied on a diagonal. Light grey cells represent the ending positions of the corresponding moves depicted in each sub-figure.

Let S_I be a diagonal of n nodes $u_n, u_{n-1}, \dots, u_1$, occupying cells $(x, y), (x+1, y+1), \dots, (x+n-1, y+n-1)$, respectively. Assume for simplicity of exposition that n is a power of 2; this can be dropped later. As argued above, it is sufficient to show how S_I can be transformed into a straight line S_L. In phase $i = 0$, the top node u_1 moves one position to align with u_2 and form a line L_1 of length 2. In any phase i, for $1 \leq i \leq \log n$, a line L_i occupies 2^i consecutive cells in a terminal subset of S_I (Figure 2 (a)). L_i moves through a shortest path towards the far endpoint of the next diagonal segment D_i of length 2^i (Figure 2 (b)). Note that for general shapes, this move shall be replaced by a more general *LineWalk* operation (defined in the sequel). By a recursive call on D_i, D_i transforms into a line L'_i (Figure 2 (c)). Finally, the two perpendicular lines L_i and L'_i are combined in linear time into a straight line L_{i+1} of length 2^{i+1} (Figure 2 (d)). By the end of phase $\log n$, a straight line S_L of order n has been formed.

A core technical challenge in making the above transformation work in the general case, is that Hamiltonian shapes do not necessarily provide free space, thus, moving a line has to take place through the remaining configuration of nodes while at the same time ensuring that it does not break their and its own connectivity. In the more general *LineWalk* operation that we now describe, we manage to overcome this by exploiting *transparency* of line moves, according to which a line L can *transparently* walk through any configuration S (independently of the latter's density); see Proposition 1.

LineWalk. At the beginning of any phase i, there is a terminal straight line L_i of length 2^i containing the nodes $v_1, \dots, v_{2^i}$, which is connected to an $S_i \subseteq S_I$, such that S_i consists of the 2^i subsequent nodes, that is $v_{2^i+1}, \dots, v_{2^{i+1}}$. Observe that S_i is the next terminal sub-path of the remaining Hamiltonian path of S_I. We distinguish the following cases: (1) If L_i and S_i are already forming a straight line, then go to phase $i+1$. (2) If S_i is a line perpendicular to L_i, then combine them into a straight line by pushing L_i to extend S_i and go to phase $i+1$. Otherwise, (3) check if the (Manhattan) distance between v_{2^i} and $v_{2^{i+1}}$ is $\delta(v_{2^i}, v_{2^{i+1}}) \leq 2^i$, then L_i moves from $v_{2^i} = (x, y)$ vertically or horizontally towards either node (x, y') or (x', y) in which L_i turns and keeps moving to $v_{2^{i+1}} = (x', y')$ on the other side of S_I. If not, (4) L_i must first pass through a middle node of S_I at $v_{2^i+2^{i-1}} = (x'', y'')$, therefore L_i repeats (3) twice, from v_{2^i} to $v_{2^i+2^{i-1}}$ and then towards $v_{2^{i+1}}$.

Note that cases (3) and (4) ensure that L_i is not disconnected from the rest of the shape. Moreover, moving L_i must be performed in a way that respects transparency (Proposition 1), so that connectivity of the remaining shape is always preserved and its configuration is restored to its original state. These details can be found in the Appendix.

Algorithm 1, HamiltonianToLine, gives a general strategy to transform any Hamiltonian shape $S_I \in \mathcal{H}$ into a straight line in $O(n \log n)$ moves. In every phase i, it moves a terminal line L_i of length 2^i a distance 2^i higher on the Hamiltonian path through a *LineWalk* operation. This leaves a new terminal sub-path S_i of the Hamiltonian path, of length 2^i. Then the general procedure is recursively called on S_i to transform it into a straight line L'_i of length 2^i.

Finally, the two straight lines L_i and L'_i which are perpendicular to each other are combined into a new straight line L_{i+1} of length 2^{i+1} and the next phase begins. The output of HamiltonianToLine is a straight line S_L of order n.

Algorithm 1: HamiltonianToLine(S)

$S = (u_0, u_1, ..., u_{|S|-1})$ is a Hamiltonian shape
Initial conditions: $S \leftarrow S_I$ and $L_0 \leftarrow \{u_0\}$

for $i = 0, \ldots, \log|S|$ **do**
    LineWalk(L_i)
    $S_i \leftarrow$ select(2^i) `// select the next terminal subset of` 2^i `consecutive nodes of` S
    $L'_i \leftarrow$ HamiltonianToLine(S_i) `// recursive call on` S_i
    $L_{i+1} \leftarrow$ combine(L_i, L'_i) `// combines` L_i `and` L'_i `into a new straight line` L_{i+1}
end
Output: a straight line S_L

Lemma 1. *Given an initial Hamiltonian shape $S_I \in \mathcal{H}$ of order n,* HamiltonianToLine *transforms S_I into a straight line S_L in $O(n \log n)$ moves, without breaking connectivity during the transformation.*

Proof. It is not hard to see that the *LineWalk* operation does not break connectivity in cases (1) and (2) in any phase i. For case (3), *LineWalk* moves a line L_i of 2^i nodes, which are enough to fill a path of 2^i empty cells and stay connected. This holds also for case (4) by applying (3) twice. By a careful application of Proposition 1, it can be shown that the argument also holds true for any configuration of the path (and its surrounding cells) along which L_i moves. We now analyse the running time of HamiltonianToLine. By induction on the number of phases, *Walk-Through-Path* makes a total number of moves bounded by:

$$T = \sum_{i=1}^{\log n} T(i) = \sum_{i=1}^{\log n} 2^{i-1}(i-1) - 2^i = \sum_{i=1}^{\log n - 1} (i-2)2^i - 2^{\log n} \leq \sum_{i=1}^{\log n - 1} i \cdot 2^i - n$$
$$\leq \sum_{j=1}^{\log n} \sum_{i=j}^{\log n} 2^i - n \leq \sum_{j=1}^{\log n} n - n \leq n \log n - n = O(n \log n).$$

Finally, reversibility of line moves [3] and Lemma 1 together imply that:

Theorem 1. *For any pair of Hamiltonian shapes $S_I, S_F \in \mathcal{H}$ of the same order n, Walk-Through-Path transforms S_I into S_F (and S_F into S_I) in $O(n \log n)$ moves, while preserving connectivity of the shape during its course.*

4 An $O(n\sqrt{n})$-time Universal Transformation

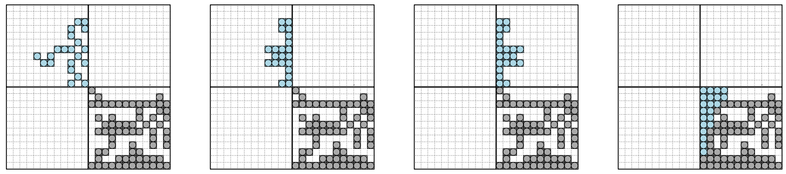
In this section, we develop a transformation that solves the UniversalConnected problem in $O(n\sqrt{n})$ moves. It is called *UC-Box* and transforms any pair of connected shapes (S_I, S_F) of the same order to each other, while preserving *connectivity* during its course.

Starting from the initial shape S_I of order n with an associated graph $G(S_I)$, compute a spanning tree T of $G(S_I)$. Then enclose the shape into an $n \times n$ square box and divide it into $\sqrt{n} \times \sqrt{n}$ square sub-boxes. Each occupied sub-box contains one or more maximal sub-trees of T. Each such sub-tree corresponds to a sub-shape of S_I, which from now on we call a *component*. Pick a leaf sub-tree T_l, let C_l be the component with which it is associated, and B_l their sub-box. Let also B_p be the sub-box adjacent to B_l containing the unique parent sub-tree T_p of T_l. Then compress all nodes of C_l into B_p through line moves, while keeping the nodes of C_p (the component of T_p) within B_p. Once compression is completed and C_p and C_l have been *combined* into a single component C'_p, compute a new sub-tree T'_p spanning $G(C'_p)$. Repeat until the whole shape is compressed into a $\sqrt{n} \times \sqrt{n}$ square. The latter belongs to the family of *nice* shapes (a family of connected shapes introduced in [3]) and can, thus, be transformed into a straight line in linear time.

Given that, the main technical challenges in making this strategy work universally is that a connected shape might have many different configurations inside the sub-boxes it occupies, while the shape needs to remain connected during the transformation. In the following, we describe the *compression* operation, which successfully tackles all of these issues by exploiting the linear strength of line moves.

Compress. Let $C_l \subseteq S_I$ be a leaf component containing nodes $v_1, \ldots, v_k$ inside a sub-box B_l of size $\sqrt{n} \times \sqrt{n}$, where $1 \leq k \leq n$, and $C_p \subseteq S_I$ the unique parent component of C_l occupying an adjacent sub-box B_p. If the direction of connectivity between B_l and B_p is vertical or horizontal, push all lines of C_l one move towards B_p sequentially one after the other, starting from the line furthest from B_p. Repeat the same procedure to first align all lines perpendicularly to the boundary between B_l and B_p (Figure 3(b)) and then to transfer them completely into B_p (e.g., Figure 3(c)). Hence, C_l and C_p are combined into C'_p, and the next round begins. The above steps are performed in a way which ensures that all lines (in C_l or C_p) being pushed by this operation do not exceed the boundary of B_p (e.g., Figure 3(d)). While C_l compresses vertically or horizontally, it may collide with a component $C_r \subseteq S_I$ inside B_l. In this case C_l stops compressing and combines with C_r into C'_r. Then the next round begins. If C_l compresses diagonally towards C_p (vertically then horizontally or vice versa) via an intermediate adjacent sub-box B_m and collides with $C_m \subseteq S_I$ inside B_m, then C_l completes compression into B_m and combines with C_m into C'_m. Figure 3 shows how to compress a leaf component into its parent component occupying a diagonal adjacent sub-box.

Algorithm 2, Compress, provides a universal procedure to transform an initial connected shape S_I of any order into a compressed square shape of the same order. It takes two arguments: S_I and the spanning tree T of the *associated graph* $G(S_I)$. In any round: Pick a leaf sub-tree of T_l corresponding to C_l inside a sub-box B_l. Compress C_l into an adjacent sub-box B_p towards its parent component C_p associated with parent sub-tree T_p. If C_l compressed with no collision,



(a) (b) (c) (d)

Fig. 3: A leaf component C_l in blue compressing from the top-left sub-box towards its parent component C_p in black inside a diagonal adjacent bottom-right sub-box. C_l compresses first horizontally towards an intermediate top-right sub-box, then vertically into the bottom-right. All other orientations are symmetric and follow by rotating the shape 90°, 180° or 270°.

perform combine(C_p, C_l) which combines C_l with C_p into one component C'_p. If C_l collides with another component C_r inside B_l, then perform combine(C_r, C_l) into C'_r. If not, as in the diagonal compression in which C_l collides with C_m in an intermediate sub-box B_m, then C_l compresses completely into B_m and performs combine(C_m, C_l) into C'_m. Once compression is completed, update(T) computes a new sub-tree and removes any cycles. The algorithm terminates when T matches a single component of n nodes compressed into a single sub-box.

Algorithm 2: Compress(S)

```
S = (u_1, u_2, ..., u_|S|) is a connected shape, T is a spanning tree of G(S) repeat
    C_l ← pick(T_l) // select a leaf component associated with a leaf
        sub-tree
    Compress(C_l) // start compressing the leaf component
    if C_l collides then
        C'_r ← combine(C_r, C_l) or C'_m ← combine(C_m, C_l) // as described
            in text
    else
        C'_p ← combine(C_p, C_l) // combine C_l with a parent component
    end
    update(T) // update sub-trees and remove cycles after
        compression
until the whole shape is compressed into a √n × √n square
Output: a square shape S_C
```

Lemma 2. *Any square box of size $\sqrt{n}$ can hold at most $2\sqrt{n}$ components.*

Proof. Observe that any component $C_l \subseteq S_I$ inside a sub-box B_l must be connected via a path to one of the $\sqrt{n}/2$ cells on one of the length-$\sqrt{n}$ boundaries of B_l, resulting in $2\sqrt{n}$ for the four boundaries. Hence, a square sub-box can contain at most $2\sqrt{n}$ disconnected components.

Lemma 3. *Starting from an initial connected shape S_I of order n divided into $\sqrt{n}$ square sub-boxes of size $\sqrt{n}$,* Compress *compresses a leaf component $C_l \subseteq S_I$ of $k \geq 1$ nodes, while preserving the global connectivity of the shape.*

The compression cost of this transformation could be very low taking only one move or being very high in some cases up to linear steps. To simplify the analysis, we divide the total cost of *UC-Box* into charging phases. We then manage to upper bound the cost of each charging phase independently of the order of compressions.

Lemma 4. Compress *compresses any connected shape S_I of order n into a $\sqrt{n} \times \sqrt{n}$ square shape, in $O(n\sqrt{n})$ steps without breaking connectivity.*

Hence, by Lemmas 3, 4 and reversibility of nice shapes (from [3]), we have:

Theorem 2. *For any pair of connected shapes (S_I, S_F) of the same order n,* UC-Box *transforms S_I into S_F (and S_F into S_I) in $O(n\sqrt{n})$ steps, while preserving connectivity during its course.*

5 Conclusions and Open Problems

We have presented efficient transformations for the line-pushing model introduced in [3]. Our first transformation works on the family of all Hamiltonian shapes and matches the running time of the best known $O(n \log n)$-time transformation while additionally managing to preserve connectivity throughout its course. We then gave the first universal connectivity preserving transformation for this model. Its running time is $O(n\sqrt{n})$ and works on any pair of connected shapes of the same order. This work opens a number of interesting problems and research directions. An immediate next goal is whether it is possible to develop an $O(n \log n)$-time universal connectivity-preserving transformation. If true, the existence of lower bound above linear is not known, then a natural question is whether a universal transformation can be achieved in $o(n \log n)$-time (even when connectivity can be broken) or whether there exists a general $\Omega(n \log n)$-time matching lower bound. As a first step, it might be easier to develop lower bounds for the connectivity-preserving case.

We establish $\Omega(n \log n)$ lower bounds for two restricted sets of transformations, which have been shown in our full report [2]. These are the first lower bounds for this model and are matching the best known $O(n \log n)$ upper bounds. For example, it can be shown that any such transformation has a labelled tree representation, and by restricting the consideration to the sub-set of those transformations in which every leaf-to-root path has length at most 2, this captures tranformations in which every node must reach its final destination through at most 1 meeting-hop and at most 2 hops in total. Interestingly, by disregarding the fact that our initial and target instances have specific geometric arrangements, it is known that computing a 2-HOPS MST in the Euclidean 2-dimensional space is a hard optimisation problem and the best known result is a PTAS by Arora *et al.* [8] (cf. also [12]). There are also a number of interesting variants of the present model. One is a centralised parallel version in which more than one line can be moved concurrently in a single time-step. Another, is a distributed version of the parallel model, in which the nodes operate autonomously through local control and under limited information.

References

1. Akitaya, H., Arkin, E., Damian, M., Demaine, E., Dujmovic, V., Flatland, R., Korman, M., Palop, B., Parada, I., van Renssen, A., Sacristán, V.: Universal reconfiguration of facet-connected modular robots by pivots: The O(1) musketeers. In: 27th Annual European Symposium on Algorithms, ESA. LIPIcs, vol. 144, pp. 3:1–3:14 (2019)
2. Almethen, A., Michail, O., Potapov, I.: On efficient connectivity-preserving transformations in a grid. CoRR **abs/2005.08351** (2020)
3. Almethen, A., Michail, O., Potapov, I.: Pushing lines helps: Efficient universal centralised transformations for programmable matter. Theoretical Computer Science **830-831**, 43 – 59 (2020)
4. 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)
5. 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)
6. 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 (March 2006)
7. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing **20**(4), 279–304 (November 2007)
8. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for euclidean k-medians and related problems. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 106–113 (1998)
9. Bourgeois, J., Goldstein, S.: Distributed intelligent MEMS: progresses and perspective. IEEE Systems Journal **9**(3), 1057–1068 (2015)
10. Butler, Z., Kotay, K., Rus, D., Tomita, K.: Generic decentralized control for lattice-based self-reconfigurable robots. The International Journal of Robotics Research **23**(9), 919–937 (2004)
11. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Distributed computing by mobile robots: Gathering. SIAM Journal on Computing **41**(4), 829–879 (2012)
12. Clementi, A., Di Ianni, M., Lauria, M., Monti, A., Rossi, G., Silvestri, R.: On the bounded-hop mst problem on random euclidean instances. Theoretical computer science **384**(2-3), 161–167 (2007)
13. Czyzowicz, J., Dereniowski, D., Pelc, A.: Building a nest by an automaton. In: Bender, M.A., Svensson, O., Herman, G. (eds.) 27th Annual European Symposium on Algorithms, ESA (2019)
14. Das, S., Flocchini, P., Santoro, N., Yamashita, M.: Forming sequences of geometric patterns with oblivious mobile robots. Distributed Computing **28**(2), 131–145 (April 2015)
15. 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)
16. 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)

17. 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)
18. Di Luna, G., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. Distributed Computing (Mar 2019)
19. Doty, D.: Theory of algorithmic self-assembly. Communications of the ACM **55**, 78–88 (2012)
20. 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)
21. Dumitrescu, A., Pach, J.: Pushing squares around. In: Proceedings of the twentieth annual symposium on Computational geometry. pp. 116–123. ACM (2004)
22. 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)
23. 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)
24. Fekete, S., Gmyr, R., Hugo, S., Keldenich, P., Scheffer, C., Schmidt, A.: Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata. CoRR **abs/1810.06360** (2018)
25. Fekete, S., Richa, A., Römer, K., Scheideler, C.: Algorithmic foundations of programmable matter (Dagstuhl Seminar 16271). In: Dagstuhl Reports. vol. 6 (2016), also in *ACM SIGACT News*, 48.2:87-94, 2017
26. 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)
27. 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)
28. Itai, A., Papadimitriou, C., Szwarcfiter, J.: Hamilton paths in grid graphs. SIAM Journal on Computing **11**(4), 676–686 (1982)
29. 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)
30. 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)
31. Michail, O., Spirakis, P.: Simple and efficient local codes for distributed stable network construction. Distributed Computing **29**(3), 207–237 (2016)
32. Michail, O., Spirakis, P.: Elements of the theory of dynamic networks. Commun. ACM **61**(2), 72–81 (2018)
33. 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)
34. Rothemund, P.: Folding dna to create nanoscale shapes and patterns. Nature **440**(7082), 297–302 (2006)
35. 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)

36. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014)
37. Walter, J., Welch, J., Amato, N.: Distributed reconfiguration of metamorphic robot chains. Distributed Computing **17**(2), 171–189 (2004)
38. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (June 1998)
39. 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)
40. Yamashita, M., Suzuki, I.: Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theoretical Computer Science **411**(26-28), 2433–2453 (2010)
41. Yamauchi, Y., Uehara, T., Yamashita, M.: 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. pp. 447–449. ACM (2016)
42. 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)

APPENDIX

On Efficient Connectivity-Preserving Transformations in a Grid

Abdullah Almethen, Othon Michail, and Igor Potapov

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

Abstract. We consider a discrete system of n devices lying on a 2-dimensional square grid and forming an initial connected shape S_I. Each device is equipped with a linear-strength mechanism which enables it to move a whole line of consecutive devices in a single time-step. We study the problem of transforming S_I into a given connected target shape S_F of the same number of devices, via a finite sequence of *line moves*. Our focus is on designing *centralised* transformations aiming at *minimising the total number of moves* subject to the constraint of *preserving connectivity* of the shape throughout the course of the transformation. We first give very fast connectivity-preserving transformations for the case in which the *associated graphs* of S_I and S_F are isomorphic to a Hamiltonian line. In particular, our transformations make $O(n \log n)$ moves, which is asymptotically equal to the best known running time of connectivity-breaking transformations. Our most general result is then a connectivity-preserving *universal transformation* that can transform any initial connected shape S_I into any target connected shape S_F, through a sequence of $O(n\sqrt{n})$ moves. Finally, we establish $\Omega(n \log n)$ lower bounds for two restricted sets of transformations. These are the first lower bounds for this model and are matching the best known $O(n \log n)$ upper bounds.

1 Introduction

Over the past few years, many fascinating systems have been developed, leveraging advanced technology in order to deploy large collections of tiny monads. Each monad is typically a highly restricted micro-robotic entity, equipped with a microcontroller and some actuation and sensing capabilities. Through its collaborative complexity, the collection of monads can carry out tasks which are well beyond the capabilities of individual monads. The vision is the development of materials that will be able to algorithmically change their physical properties, such as their shape, colour, conductivity and density, based on transformations executed by an underlying program. These efforts are currently shaping the research area of *programmable matter*, which has attracted much theoretical and practical interest.

The implementation indicates whether the monads are operated centrally or through local decentralised control. In *centralised* systems, there is an external program which globally controls all monads with full knowledge of the entire system. On the other hand, *decentralised* systems provide each individual monad with enough autonomy to communicate with its neighbours and move locally. There are an impressive number of recent developments for collective robotic systems, demonstrating their potential and feasibility, starting from the scale of milli or micro [8, 25, 30, 36] down to nano size of individual monads [19, 34].

Recent research has highlighted the need for the development of an algorithmic theory of such systems. Michail and Spirakis [32] and Michail *et al.* [31] emphasised an apparent lack of a formal theoretical study of this prospective, including modelling, possibilities/limitations, algorithms and complexity. The development of a formal theory is a crucial step for further progress in those systems. Consequently, multiple theoretical computer science sub-fields have appeared, such as metamorphic systems [22, 33, 37], mobile robotics [10, 12, 13, 17, 40], reconfigurable robotics [4, 9, 14, 16, 42], passively-mobile systems [6, **?**,**?**,32], DNA self-assembly [18, 35, 38, 39], and the latest emerging sub-area of "Algorithmic Foundations of Programmable Matter" [24].

Consider a system deployed on a two-dimensional square grid in which a collection of spherical devices are typically connected to each other, forming a shape S_I. By a finite number of valid individual moves, S_I can be transformed into a desired target shape S_F. In this prospective, a number of models are designed and introduced in the literature for such systems. For example, Dumitrescu and Pach [20], Dumitrescu *et al.* [21, 22] and Michail *et al.* [31] consider mechanisms where an individual device is capable to move over and turn around its neighbours through empty space. Transformations based on similar moves being assisted by small seeds, have also been considered in [1]. A new linear-strength mechanism was introduced by Almethen *et al.* in [3], where a whole line of consecutive devices can, in a single time-step, move by one position in a given direction.

In this paper, we embark from the *line-pushing* model of [3], which provided sub-quadratic centralised transformations that may, though, arbitrarily break connectivity of the shape during their course. The only connectivity-preserving transformation in [3] was an $O(n\sqrt{n})$-time transformation for a single pair of shapes of order n, namely from a diagonal into a straight line. All transformations that we provide in the present study preserve connectivity of the shape during the transformations. We first give very fast connectivity-preserving transformations for the case in which the *associated graphs* of S_I and S_F are isomorphic to a Hamiltonian line. In particular, our transformations make $O(n \log n)$ moves, which is asymptotically equal to the best known running time of connectivity-breaking transformation. Our most general result is then a connectivity-preserving *universal transformation* that can transform any initial connected shape S_I into any target connected shape S_F, through a sequence of $O(n\sqrt{n})$ moves. Finally, we establish $\Omega(n \log n)$ lower bounds for two restricted sets of transformations. These are the first lower bounds for this model and are matching the best known $O(n \log n)$ upper bounds.

1.1 Related Work

For the models of individual moves where only one node moves in a single time-step, [20, 31] show universality of transforming any pair of connected shapes (A, B) having the same number of devices (called *nodes* throughout this paper) to each other via sliding and rotation mechanisms. By allowing only rotation, [31] proves that the problem of deciding transformability is in **P**. It can be shown that in all models of constant-distance individual moves, $\Omega(n^2)$ moves are required to transform some pairs of connected shapes, due the inherent distance between them [31]. This motivates the study of alternative types of moves that are reasonable with respect to practical implementations and allow for sub-quadratic reconfiguration time in the worst case.

There are attempts in the literature to provide alternatives for more efficient reconfiguration. The first main approach is to explore parallel transformations, where multiple nodes move together in a single time-step. This is a natural step to tackle such a problem, especially in distributed systems where nodes can make independent decisions and move locally in parallel to other nodes. There are a number of theoretical studies on parallel and distributed transformations [14, 15, 17, 22, 31, 41] as well as practical implementations [36]. For example, it can be shown that a connected shape can transform into any other connected shape, by performing in the worst case $O(n)$ parallel moves around the perimeter of the shape [31].

The second approach aims to equip nodes in the system with a more powerful mechanism which enables them to reduce the inherent distance by a factor greater than a constant in a single time-step. There are a number of models in the literature in which individual nodes are equipped with strong actuation mechanisms, such as linear-strength mechanisms. Aloupis *et al.* [4, 5] provide a node with arms that are capable to extend and contract a neighbour, a subset of the nodes or even the whole shape as a consequence of such an operation. Further, Woods *et al.* [39] proposed an alternative linear-strength mechanism, where a node has the ability to rotate a whole line of consecutive nodes.

Recently, the *line-pushing* model of [3] follows a similar approach in which a single node can move a whole line of consecutive nodes by simultaneously (i.e., in a single time-step) pushing them towards an empty position. The line-pushing model can simulate the rotation and sliding based transformations of [20, 31] with at most a 2-factor increase in their worst-case running time. This implies that all transformations established for individual nodes, transfer in the line-pushing model and their universality and reversibility properties still hold true. They achieved sub-quadratic time transformations, including an $O(n \log n)$-time universal transformation which does not preserve connectivity and a connectivity-preserving $O(n\sqrt{n})$-time transformation for the special case of transforming a diagonal into a straight line.

Another relevant line of research has considered a single moving robot that transforms an otherwise static shape by carrying its tiles one at a time [12, 23, 26]. Those models are partially centralised as a single robot (usually a finite automaton) controls the transformation, but, in contrast to our perspective, control in that case is local and lacking global information.

1.2 Our Contribution

In this work, we build upon the findings of [3] aiming to design very efficient and general transformations that are additionally able to keep the shape connected throughout their course.

We first give an $O(n \log n)$-time transformation, called *Walk-Through-Path*, that works for all pairs of shapes (S_I, S_F) that have the same order and belong to the family of *Hamiltonian shapes*. A *Hamiltonian shape* is any connected shape S whose *associated graph* $G(S)$ is isomorphic to a Hamiltonian path (see also [29]). At the heart of our transformation is a recursive successive doubling technique, which starts from one endpoint of the Hamiltonian path and proceeds in $\log n$ phases (where n denotes the order of the input shape S_I, throughout this paper). In every phase i, it moves a terminal line L_i of length 2^i a distance 2^i higher on the Hamiltonian path through a *LineWalk* operation. This leaves a new terminal sub-path S_i of the Hamiltonian path, of length 2^i. Then the general procedure is recursively called on S_i to transform it into a straight line L'_i of length 2^i. Finally, the two straight lines L_i and L'_i which are perpendicular to each other are combined into a new straight line L_{i+1} of length 2^{i+1} and the next phase begins.

A core technical challenge in making the above transformation work is that Hamiltonian shapes do not necessarily provide free space for the *LineWalk* operation. Thus, moving a line has to take place through the remaining configuration of nodes while at the same time ensuring that it does not break their and its own connectivity, including keeping itself connected to the rest of the shape. We manage to overcome this by revealing a nice property of line moves, according to which a line L can *transparently* walk through *any* configuration S (independently of the latter's density) in a way that: (i) preserves connectivity of both L and S and (ii) as soon as L has gone through it, S has been restored to its original state, that is, all of its nodes are lying in their original positions. This property is formally proved in Proposition 1 (Section 2).

We next develop a *universal transformation*, called *UC-Box*, that within $O(n\sqrt{n})$ moves transforms any pair of connected shapes of the same order to each other, while preserving connectivity throughout its course. Starting from the initial shape S_I, we first compute a spanning tree T of S_I. Then we enclose the shape into a square box of size n and divide it into sub-boxes of size $\sqrt{n}$, each of which contains at least one sub-tree of T. By moving lines in a way that does not break connectivity, we compress the nodes in a sub-box into an adjacent sub-box towards a parent sub-tree. By carefully repeating this we manage to arrive at a final configuration which is always a compressed square shape. The latter is a type of a *nice* shape (a family of connected shapes introduced in

[3]), which can be transformed into a straight line in linear time. We provide an analysis of this strategy based on the number of *charging phases*, which turns out to be $\sqrt{n}$, each making at most n moves, for a total of $O(n\sqrt{n})$ moves.

Finally, we establish $\Omega(n \log n)$ lower bounds for two restricted sets of transformations. These are the first lower bounds for this model and are matching the best known $O(n \log n)$ upper bounds. The first set consists of transformations from an initial diagonal into a target straight line. If every node on the diagonal only meets through shortest paths with other nodes at their original positions and every such meeting results in an irreversible merging (i.e., nodes that merge cannot split in future steps), then it can be shown that any such transformation has a labelled tree representation. The nodes of the tree are the nodes of the shape, the edges represent mergings between the corresponding nodes at some point in the transformation and the labels of the edges represent the shortest path distances between the original positions of the corresponding nodes. Then the total cost of the transformation is equal to the sum of the labels plus the sum of the sizes of all sub-trees of the tree. The latter additive factors capture the cost of *turning* (i.e., changing the orientation of) a line of merged nodes, which is always equal to its length, and every meeting through a shortest path on the grid requires at least one turn.

We further restrict attention to the sub-set of those transformations in which every leaf-to-root path has length at most 2. This captures tranformations in which every node must reach its final destination (on the target straight line) through at most 1 meeting-hop and at most 2 hops in total. Interestingly, by disregarding the sub-tree additive costs and the fact that our initial and target instances have specific geometric arrangements, it is known that computing a 2-HOPS MST in the Euclidean 2-dimensional space is a hard optimisation problem and the best known result is a PTAS by Arora *et al.* [7] (cf. also [11]). By working on the tree representation, we show that any transformation in this set requires at least $\Omega(n \log n)$ moves. Our second lower bound is also $\Omega(n \log n)$ time, for an alternative set of one-way transformations.

Section 2 formally defines the model and the problems under consideration and proves a basic proposition which is a core technical tool in one of our transformations. Section 3 presents our $O(n \log n)$-time transformation for Hamiltonian shapes. Section 4 discusses our universal $O(n\sqrt{n})$-time transformation. In Section 5, our lower bounds are proved. Finally, in Section 6 we conclude and discuss interesting problems left open by our work.

2 Preliminaries

All transformations in this study operate on a two-dimensional square grid, in which each cell has a unique position of non-negative integer coordinates (x, y), where x represents columns and y denotes rows in the grid. A set of n nodes on the grid forms a shape S (of the order n), where every single node $u \in S$ occupies only one cell $cell(u) = (u_x, u_y)$. A node u can be indicated at any given time by the coordinates (u_x, u_y) of the unique cell that it occupies at that time.

A node $v \in S$ is a *neighbour* of (or *adjacent* to) a node $u \in S$ if and only if their coordinates satisfy $u_x - 1 \leq v_x \leq u_x + 1$ and $u_y - 1 \leq v_y \leq u_y + 1$ (i.e., their cells are adjacent vertically, horizontally or diagonally).

Definition 1. *A graph $G(S) = (V, E)$ is* associated *with a shape S, where $u \in V$ iff u is a node of S and $(u, v) \in E$ iff u and v are neighbours in S.*

A shape S is connected iff $G(S)$ is a connected graph. We denote by $T(S)$ (or just T when clear from context) a spanning tree of $G(S)$, and whenever we state that such a tree is given we make use of the fact that $T(S)$ can be computed in polynomial time.

Definition 2 (A tree). *A tree (T, r), or T whenever clear in the context, is rooted at a node $r \in V$, such that there is a unique path $\mathcal{P}$ from r to each node $v \in V$ denoted by $\mathcal{P}_T(r, v)$ on which the distance $\delta_T(r, v)$ is the number of edges between them. A node v is a successor of u iff $\mathcal{P}_T(r, v) \supset \mathcal{P}_T(r, u)$, and u is a parent of v iff $\delta_T(u, v) = 1$.*

The size of a tree, $size(T)$, denotes the number of all nodes in T, includes the root r and all its successors. In what follows, n denotes the number of nodes in a shape under consideration, and all logarithms are to base 2.

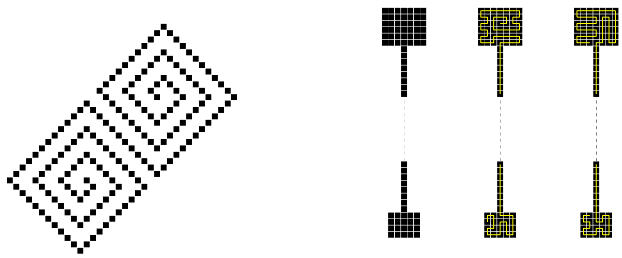
In this paper, we exploit the linear-strength mechanism of the *line-pushing model* introduced in [3]. A line L is a sequence of nodes occupying consecutive cells in one direction of the grid, that is, either vertically or horizontally but not diagonally. A ***line move*** is an operation of moving all nodes of L together in a single time-step towards a position adjacent to one of L's endpoints, in a given direction d of the grid, $d \in \{up, down, right, left\}$. A line move may also be referred to as *step*, *move*, or *movement* in this paper. Throughout, the running time of transformations is measured in total number of line moves until completion. A *line move* is formally defined below.

Definition 3 (A permissible line move). *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 $(x+k, y)$. The "down", "left", and "up" moves are defined symmetrically, by rotating the whole shape $90°$, $180°$, and$270°$ clockwise, respectively.*

We next define a family of shapes that are used in one of our transformations.

Definition 4 (Hamiltonian Shapes). *A shape S is called Hamiltonian iff $G(S) = (V, E)$ is isomorphic to a Hamiltonian path, i.e., a path starting from a node $u \in V$, visiting every node in V exactly once and ending at a node $v \in V$, where $v \neq u$. $\mathcal{H}$ denotes the family of all Hamiltonian shapes. Figure 1 shows some examples of Hamiltonian shapes.*

The following proposition proves a basic property of line moves which will be a core technical tool in one of our transformation for Hamiltonian shapes.



(a) A double-spiral.

(b) A shape of two different Hamiltonian paths in yellow.

Fig. 1: Examples of Hamiltonian shapes.

Proposition 1 (Transparency of Line Moves). *Let S be any shape, $L \subseteq S$ any line and P any path of cells in the grid (under the vertical and horizontal neighbouring relation) starting from a position adjacent to one of L's endpoints. Let $C(P)$ denote the configuration of P defined by S. There is a way to move L along P, while satisfying all the following properties:*

1. No delay: *The number of steps is asymptotically equal to that of an optimum move of L along P in the case of $C(P)$ being empty (i.e., if no cells were occupied). That is, L is not delayed, independently of what $C(P)$ is.*
2. No effect: *After L's move along P, $C'(P) = C(P)$, i.e., the cell configuration has remained unchanged. Moreover, no occupied cell in $C(P)$ is ever emptied during L's move (but unoccupied cells may be temporarily occupied).*
3. No break: *S remains connected throughout L's move.*

Proof. Given $L \subseteq S$ and P, place additional nodes that occupy cells in P, possibly with gaps, in any configuration $C(P)$, see Figure 2 for example. Whenever L walks through an empty cell (x, y) of P, a node $u \in L$ fills in (x, y). If L pushes the node u of a non-empty cell of P, a node $v \in L$ takes its place. When L leaves a non-empty cell (x, y) that was originally occupied by node v, L restores (x, y) by leaving its endpoint $u \in L$ in (x, y).

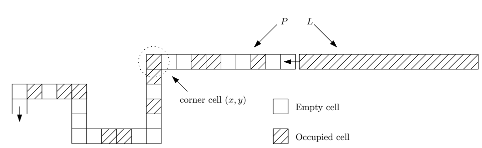


Fig. 2: A path P of a given configuration $C(P)$. A line L will pass along p.

Now assume that L turns at a non-empty corner cell (x, y) of P (say without loss of generality, from horizontal to vertical direction). Typically the node occupying the corner cell(x, y) moves vertically one step along P, and then L pushes one move to fill in the empty cell (x, y) by a node $u \in L$. Unless (x, y) is being only connected diagonally to a non-empty cell that is not a neighbour of any node $u \in L$. Figure 3 shows how to deal with the case in which L turns at a non-empty corner-cell (x, y) of P, which is only connected diagonally to a non-empty cell of S and is not adjacent to any cell occupied by L.

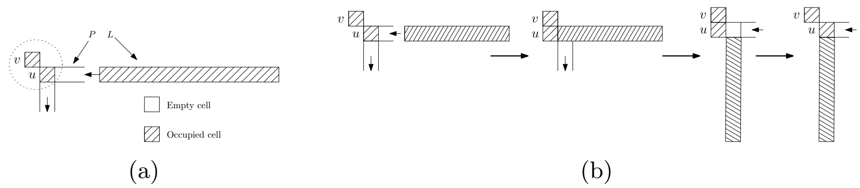


Fig. 3: A line L moving through a path P and arriving at a turning point of P. u occupies a corner cell of P and v occupies a cell of S and is only connected diagonally to u while not being adjacent to any cell occupied by L. L pushes u one position horizontally and turns all of its nodes vertically. Then u moves back to its original position in P. All other orientations are symmetric and follow by rotating the shape 90°, 180° or 270°.

Therefore, it always temporarily maintain global connectivity and restores all of those nodes to their original positions. Hence, L's move takes a number of moves to pass through any $C(P)$ equal to or even less than its optimum move in the case of empty $C(P)$. Therefore, L can *transparently* walk through *any* configuration S (independently of the latter's density) in a way that: (i) preserves connectivity of both L and S and (ii) as soon as L has gone through it, S has been restored to its original state, that is, all of its nodes are lying in their original positions.

We now formally define all problems considered in this work.

HamiltonianConnected. Given a pair of connected Hamiltonian shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, transform S_I into S_F while preserving connectivity throughout the transformation.

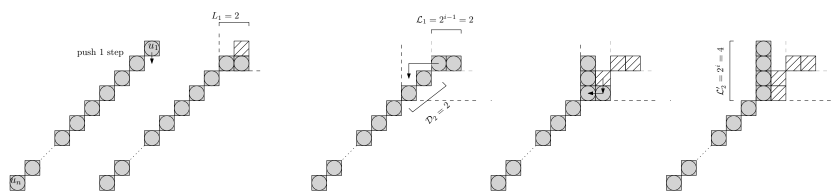
DiagonalToLineConnected. A special case of HamiltonianConnected in which S_I is a diagonal line and S_F is a straight line.

UniversalConnected. Given *any* pair of connected shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, transform S_I into S_F while preserving connectivity throughout the transformation.

3 $O(n \log n)$-time Transformations for Hamiltonian Shapes

In this section, we present a strategy for HamiltonianConnected, called *Walk-Through-Path*. It transforms any pair of shapes $S_I, S_F \in \mathcal{H}$ of the same order to each other within $O(n \log n)$ moves while preserving connectivity of the shape throughout the transformation. Recall that $\mathcal{H}$ is the family of all Hamiltonian shapes. Our transformation starts from one endpoint of the Hamiltonian path of S_I and applies a recursive successive doubling technique to transform S_I into a straight line S_L in $O(n \log n)$ time. By replacing S_I with S_F in *Walk-Through-Path* and reversing the resulting transformation, one can then go from S_I to S_F in the same asymptotic time.

We first demonstrate the core recursive technique of this strategy in a special case which is sufficiently sparse to allow local reconfigurations without the risk of affecting the connectivity of the rest of the shape. In this special case, S_I is a diagonal of any order and observe that $S_I, S_F \in \mathcal{H}$ holds for this case. We then generalise this recursive technique to work for any $S_I \in \mathcal{H}$ and add to it the necessary sub-procedures that can perform local reconfiguration in *any* area (independently of how dense it is), while ensuring that global connectivity is always preserved.

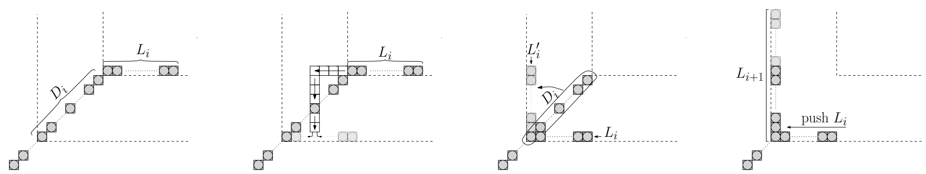


(a) First phase. (b) Second phase.

Fig. 4: First and second phase of *Walk-Through-Path* on the diagonal shape.

Let S_I be a diagonal of n nodes $u_n, u_{n-1}, \ldots, u_1$, occupying cells $(x, y), (x+1, y+1), \ldots, (x+n-1, y+n-1)$, respectively. Assume for simplicity of exposition that n is a power of 2; this can be dropped later. As argued above, it is sufficient to show how S_I can be transformed into a straight line S_L. In phase $i = 0$, the top node u_1 moves one position to align with u_2 and form a line L_1 of length 2, as depicted in Figure 4 (a). Next phase, L_1 moves two positions and turns to align with u_4, then repeat whatever done in phase $i = 0$ again on nodes u_3 and u_4 (where both form a diagonal segment D_1) to create a line L'_1, and then combine the two perpendicular line L_1 and L'_1 into a line L_2 of length 4, as shown in Figure 4 (b).

In any phase i, for $1 \le i \le \log n$, a line L_i occupies 2^i consecutive cells in a terminal subset of S_I (Figure 5 (a)). L_i moves through a shortest path towards the far endpoint of the next diagonal segment D_i of length 2^i (Figure 5 (b)). Note that for general shapes, this move shall be replaced by a more general *Line-Walk* operation (defined in the sequel). By a recursive call on D_i, D_i transforms into a line L_i' (Figure 5 (c)). Finally, the two perpendicular lines L_i and L_i' are combined in linear time into a straight line L_{i+1} of length 2^{i+1} (Figure 5 (d)). By the end of phase $\log n$, a straight line S_L of order n has been formed.



(a) A line L_i and a diagonal segment D_i both of length 2^i.

(b) L_i moves through a shortest path towards the far endpoint of D_i.

(c) D_i recursively transforms into a line L_i' .

(d) A line L_{i+1} of length 2^{i+1} formed by combining L_i and L_i'.

Fig. 5: A snapshot of phase i of *Walk-Through-Path* applied on a diagonal. Light grey cells represent the ending positions of the corresponding moves depicted in each sub-figure.

A core technical challenge in making the above transformation work in the general case, is that Hamiltonian shapes do not necessarily provide free space, thus, moving a line has to take place through the remaining configuration of nodes while at the same time ensuring that it does not break their and its own connectivity. In the more general *LineWalk* operation that we now describe, we manage to overcome this by exploiting *transparency* of line moves, according to which a line L can *transparently* walk through any configuration S (independently of the latter's density); see Proposition 1.

LineWalk. At the beginning of any phase i, there is a terminal straight line L_i of length 2^i containing the nodes $v_1, \ldots, v_{2^i}$, which is connected to an $S_i \subseteq S_I$, such that S_i consists of the 2^i subsequent nodes, that is $v_{2^i+1}, \ldots, v_{2^{i+1}}$. Observe that S_i is the next terminal sub-path of the remaining Hamiltonian path of S_I. We distinguish the following cases: (1) If L_i and S_i are already forming a straight line, then go to phase $i+1$. (2) If S_i is a line perpendicular to L_i, then combine them into a straight line by pushing L_i to extend S_i and go to phase $i+1$. Otherwise, (3) check if the (Manhattan) distance between v_{2^i} and $v_{2^{i+1}}$ is $\delta(v_{2^i}, v_{2^{i+1}}) \leq 2^i$, then L_i moves from $v_{2^i} = (x, y)$ vertically or horizontally towards either node (x, y') or (x', y) in which L_i turns and keeps moving to $v_{2^{i+1}} = (x', y')$ on the other side of S_I. If not, (4) L_i must first pass through a middle node of S_I at $v_{2^i+2^{i-1}} = (x'', y'')$, therefore L_i repeats (3) twice, from v_{2^i} to $v_{2^i+2^{i-1}}$ and then towards $v_{2^{i+1}}$.

Note that cases (3) and (4) ensure that L_i is not disconnected from the rest of the shape. Moreover, moving L_i must be performed in a way that respects transparency (Proposition 1), so that connectivity of the remaining shape is always preserved and its configuration is restored to its original state. These details are described later in this section.

Algorithm 1, HamiltonianToLine, gives a general strategy to transform any Hamiltonian shape $S_I \in \mathcal{H}$ into a straight line in $O(n \log n)$ moves. In every phase i, it moves a terminal line L_i of length 2^i a distance 2^i higher on the Hamiltonian path through a *LineWalk* operation. This leaves a new terminal sub-path S_i of the Hamiltonian path, of length 2^i. Then the general procedure is recursively called on S_i to transform it into a straight line L'_i of length 2^i. Finally, the two straight lines L_i and L'_i which are perpendicular to each other are combined into a new straight line L_{i+1} of length 2^{i+1} and the next phase begins. The output of HamiltonianToLine is a straight line S_L of order n.

Algorithm 1: HamiltonianToLine(S)

```
S = (u_0, u_1, ..., u_{|S|-1}) is a Hamiltonian shape
Initial conditions: S ← S_I and L_0 ← {u_0}

for i = 0, ..., log|S| do
    LineWalk(L_i)
    S_i ← select(2^i) // select the next terminal subset of 2^i
        consecutive nodes of S
    L'_i ← HamiltonianToLine(S_i) // recursive call on S_i
    L_{i+1} ← combine(L_i, L'_i) // combines L_i and L'_i into a new
        straight line L_{i+1}
end
Output: a straight line S_L
```

Now, we are ready to show correctness of *Walk-Through-Path* in the following lemmas.

Lemma 1. *Starting from an initial Hamiltonian shape $S_I \in \mathcal{H}$ of order n,* HamiltonianToLine *forms a straight line $S_L \in \mathcal{H}$ of length n.*

Proof. By the beginning of the final phase, the shape configuration consists of two parts, a straight line L of length $2^{\log n-1}$ and a shape S of $2^{\log n-1}$ nodes. During this phase, L performs a *LineWalk* operation, S transforms recursively into L' and then L combines with L' into a straight line S_L of length $2^{logn} = n$. Consequently, S_L shall occupy n consecutive cells on the grid, either vertically or horizontally.

Lemma 2. *The operation of Line-Walk preserves the whole connectivity of the shape during phase i.*

Proof. Let $S_I \in \mathcal{H}$ be a Hamiltonian shape of order n in phase i, which terminates at a straight line L_i of length 2^i nodes, starting from v_1 to v_{2^i}. During phase i, this transformation doubles the size of L_i by merging its nodes with the following 2^i nodes on the Hamiltonian path that are forming a shape S_i from v_{2^i+1} to $v_{2^{i+1}}$.

We now show case (1) and (2) of the *Line-Walk* operation on a horizontal L_i (*the other cases are symmetric by rotating the shape* 90°, 180° *or* 270°). In case 1, L_i and S_i are already forming a straight line L_{i+1} of length 2^{i+1}, hence the whole configuration of the shape left unchanged. In case (2), L_i and S_i are forming two perpendicular straight lines in which L_i can easily push into S_i and extend it by 2^i. As L_i pushes and S_i extends, they are replacing and restoring any occupied cell along their way through *any* configuration (independently of how density is) by exploiting *transparency* of line moves in Proposition 1. As a result, the *Line-Walk* operation preserves connectivity of L_i, S_i and the whole shape.

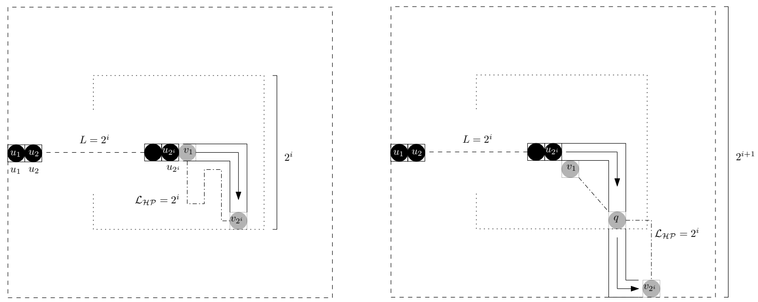
Now, let L_i and S_i be of the same configuration of case (3) or (4) described above, where L_i has a length of 2^i and S_i consists of 2^i nodes $v_{2^i+1}, \ldots, v_{2^{i+1}}$ that occupy multiple rows and columns. Assume that L_i is horizontal and occupies $(x, y), (x+1, y), \ldots, (x+2^i, y)$, this is sufficient as the other cases are symmetric if one rotates the whole shape 90°, 180° or 270°. Observe that S_i is the next terminal sub-path of the remaining Hamiltonian path. Consequently, the Manhattan distance between v_{2^i} and $v_{2^{i+1}}$ specifies the path that L_i will follow to meet and align with the far endpoint of S_i.

Recall that the minimum Manhattan (taxicab) distance of any path in a square grid, which starts from point u and ends at v, $\delta(u, v) = |u_x - v_x| + |u_y - v_y|$, will always have the same length and this transamination picks a path of minimum turns (aiming for low cost). Hence, there are two feasible L-shaped paths from u to v, each of which has one turn. The first path starts horizontally from point (u_x, u_y) towards (v_x, u_y) then turns vertically to (v_x, v_y), and the second one starts from (u_x, u_y) vertically to (u_x, v_y) then turns horizontally towards (v_x, v_y).

In case (3), the Manhattan distance between v_{2^i} and $v_{2^{i+1}}$ is $\delta(v_{2^i}, v_{2^{i+1}}) \leq 2^i$, then L_i moves horizontally from $v_{2^i} = (x, y)$ along (x', y) in which L_i changes its direction towards $v_{2^{i+1}} = (x', y')$. In a worst-case configuration, a path may consists of at 2^i empty cells L_i must pass to reach the destination cell (x', y'). Recall that L_i contains 2^i nodes, hence L_i shall arrive at (x', y'), occupy all 2^i cells and still connected. Once L_i arrived there, it can safely change its direction to line up with $v_{2^{i+1}}$ and occupy the column x', while being connected too. Moreover, assume the path along which L_i has moved contains non-empty cells, therefore all of them are restored by *transparency* of line moves shown in Proposition 1.

That is, as L_i moves along a path of non-empty cells within phase i, it pushes a node $u \notin L_i$ and replaces it by node $u \in L_i$. When L_i leaves this path during phase $i+1$, it rosters any non-empty cell occupied by a pre-existing node $u \notin L$. The same argument holds for (4) by applying (3) twice. Figure 6 demonstrates an example of case (3) and (4). As a result, The operation of *Line-Walk* keeps the whole shape connected during any phase i of this transformation.

As a result of Lemma 1 and 2, we obtain the following lemma:



(a) The case when $\delta(v_{2^i}, v_{2^{i+1}}) \leq 2^i$.

(b) The case when $\delta(v_{2^i}, v_{2^{i+1}}) > 2^i$, where node $v_{2^i+2^{i-1}}$ at the middle of S_i.

Fig. 6: The two cases of applying *Line-Walk* operation on L.

Lemma 3. *Given an initial Hamiltonian shape $S_I \in \mathcal{H}$ of order n,* HamiltonianToLine *transforms S_I into a straight line S_L in $O(n \log n)$ moves, without breaking connectivity during the transformation.*

Now, we are ready to analyse the running time of HamiltonianToLine.

Lemma 4. *By the end of phase i, for all $0 \leq i \leq \log n$,* HamiltonianToLine *forms a straight line L of 2^i nodes in at most $O(n \log n)$ steps, without breaking connectivity of the whole shape.*

Proof. The bound $O(n)$ trivially holds for case (1) and (2), so we analyse a worst-case in which the transformation matches the maximum running time in every phase i, for all $1 \leq i \leq \log n$. In phase i, a straight line L_i of length 2^i traverses along a path of at most $2 \cdot (2^i - 2) = 2^{i+1} - 4$ cells in which L_i changes its direction twice by at most $2^{i+2} - 4$ moves. There is an additive factor of 2 for the special-case of turning L_i on a non-empty corner as in Figure 3. Then the operation of *Line-Walk* takes at total moves of at most:

$$(t_1)_i = (2^{i+1} - 4) + (2^{i+2} - 4) + 2 = 6(2^i - 1).$$

Next, a recursive call of the algorithm, HamiltonianToLine, on S_i of 2^i to transform it into a straight line L_i', requires the total sum given by:

$$(t_2)_i = \sum_{i=1}^{i-1} T(i-1).$$

By the end of phase i, L_i and L_i' combine together into a straight line L_{i+1} of length 2^{i+1}, in a total cost of at most:

$$(t_3)_i = 2(2^i - 1),$$

steps. Hence, HamiltonianToLine completes phase i in a total moves $T(i)$ of at most:

$$\begin{aligned}
T(i) &= (t_1)_i + (t_2)_i + (t_3)_i \\
&= 6(2^i - 1) + \Big(\sum_{i=1}^{i-1} T(i-1)\Big) + 2(2^i - 1) \\
&\approx 2(2^i) + \Big(\sum_{i=1}^{i-1} T(i-1)\Big)
\end{aligned}$$

Now, we compute the recursion of $(t_2)_i$ as follows:

$$\begin{aligned}
T(1) &= 2(2) \\
T(2) &= 2(2^2) + 2(2) = 2(2^2 + 2) \\
&. \\
&. \\
T(i-1) &= 2\Big(2^{i-1} + 2^{i-2} + 2(2^{i-3}) + 2^2(2^{i-4}) + \ldots + 2^{i-4}(2^2) + 2^{i-3}(2)\Big) \\
&< 2\Big(2^{i-1} + 2^{i-1} + 2^{i-1} + 2^{i-1} + \ldots + 2^{i-1} + 2^{i-1}\Big) \\
&= 2\big(2^{i-1}(i-1)\big).
\end{aligned}$$

Finally, in phase i, HamiltonianToLine takes a total moves $T(i)$ at most:

$$\begin{aligned}
T(i) &= (t_1)_i + (t_2)_i + (t_3)_i \\
&= 6(2^i - 1) + 2\big(2^{i-1}(i-1) + 2(2^i - 1) \\
&\le 2^{\log n - 1}(\log n - 1) - 2^{\log n} = \frac{n(\log n - 1)}{2} - n = \frac{n \log n - n}{2} - n \\
&= O(n \log n),
\end{aligned}$$

steps.

Lemma 5. *Given an initial Hamiltonian shape $S_I \in \mathcal{H}$ of order n,* HamiltonianToLine *transforms S_I into a straight line S_L in $O(n \log n)$ moves, without breaking connectivity during the transformation.*

Proof. By Lemma 4, we use induction to analyse the running time of this transformation. The base case is holds trivially for the first phase. Assume that it holds for phase i, and we prove this must hold also for phase $i+1$.

$$\begin{aligned}
T(i+1) &= (2^{(i+1)-1}\big((i+1) - 1\big) - 2^{i+1} = 2^i(i) - (2^i \cdot 2) = 2^i(i-2) \\
&\le 2^{\log n}(\log n - 2) = n \log n - 2n \\
&= O(n \log n).
\end{aligned}$$

The assumption is also true for phase i. Hence, HamiltonianToLine makes a total number of moves bounded by:

$$\begin{aligned} T &= \sum_{i=1}^{\log n} T(i) = \sum_{i=1}^{\log n} 2^{i-1}(i-1) - 2^i = \sum_{i=1}^{\log n - 1} (i-2)2^i - 2^{\log n} \\ &\leq \sum_{i=1}^{\log n - 1} i \cdot 2^i - n \leq \sum_{j=1}^{\log n} \sum_{i=j}^{\log n} 2^i - n \leq \sum_{j=1}^{\log n} n - n \leq n \log n - n \\ &\leq O(n \log n). \end{aligned}$$

Finally, reversibility of line moves [3], Lemmas 3 and 5 together imply that:

Theorem 1. *For any pair of Hamiltonian shapes $S_I, S_F \in \mathcal{H}$ of the same order n, Walk-Through-Path transforms S_I into S_F (and S_F into S_I) in $O(n \log n)$ moves, while preserving connectivity of the shape during its course.*

4 $O(n\sqrt{n})$-time Universal Transformation

In this section, we introduce a transformation that solves the UniversalConnected problem in $O(n\sqrt{n})$ moves. It is called *UC-Box* and transforms any pair of connected shapes (S_I, S_F) of the same order to each other, while preserving *connectivity* during its course.

Starting from the initial shape S_I of order n with an associated graph $G(S_I)$, compute a spanning tree T of $G(S_I)$. Then enclose the shape into an $n \times n$ square box and divide it into $\sqrt{n} \times \sqrt{n}$ square sub-boxes. Each occupied sub-box contains one or more maximal sub-trees of T. Each such sub-tree corresponds to a sub-shape of S_I, which from now on we call a *component*. Pick a leaf sub-tree T_l, let C_l be the component with which it is associated, and B_l their sub-box. Let also B_p be the sub-box adjacent to B_l containing the unique parent sub-tree T_p of T_l. Then compress all nodes of C_l into B_p through line moves, while keeping the nodes of C_p (the component of T_p) within B_p. Once compression is completed and C_p and C_l have been *combined* into a single component C'_p, compute a new sub-tree T'_p spanning $G(C'_p)$. Repeat until the whole shape is compressed into a $\sqrt{n} \times \sqrt{n}$ square. The latter belongs to the family of *nice* shapes (a family of connected shapes introduced in [3]) and can, thus, be transformed into a straight line in linear time.

Given that, the main technical challenges in making this strategy work universally is that a connected shape might have many different configurations inside the sub-boxes it occupies, while the shape needs to remain connected during the transformation. In the following, we describe the *compression* operation, which successfully tackles all of these issues by exploiting the linear strength of line moves.

Compress. Let $C_l \subseteq S_I$ be a leaf component containing nodes $v_1, \ldots, v_k$ inside a sub-box B_l of size $\sqrt{n} \times \sqrt{n}$, where $1 \leq k \leq n$, and $C_p \subseteq S_I$ the unique parent component of C_l occupying an adjacent sub-box B_p. If the direction of connectivity between B_l and B_p is vertical or horizontal, push all lines of C_l one move towards B_p sequentially one after the other, starting from the line furthest from B_p. Repeat the same procedure to first align all lines perpendicularly to the boundary between B_l and B_p (Figure 8(b)) and then to transfer them completely into B_p (e.g., Figure 8(c)). Hence, C_l and C_p are combined into C'_p, and the next round begins. The above steps are performed in a way which ensures that all lines (in C_l or C_p) which are being pushed by this operation do not exceed the boundary of B_p (e.g., Figure 8(d)). While C_l compresses vertically or horizontally, it may collide with a component $C_r \subseteq S_I$ inside B_l. In this case C_l stops compressing and combines with C_r into C'_r. Then the next round begins. If C_l compresses diagonally towards C_p (vertically then horizontally or vice versa) via an intermediate adjacent sub-box B_m and collides with $C_m \subseteq S_I$ inside B_m, then C_l completes compression into B_m and combines with C_m into C'_m. Figure 8 shows how to compress a leaf component into its parent component occupying a diagonal adjacent sub-box.

Examples 1 and 2 depict the compression in different directions. The formal description of *UC-Box* is illustrated in Algorithm 2.

Example 1: Let C_l and C_p be components occupying two horizontal sub-boxes, B_l and B_p, respectively. C_l transfers completely to join C_p in B_p, as in Figure 7. The vertical compression holds by rotating the system 90° clockwise or counter-clockwise.

Example 2: Let C_l and C_p be components occupying two sub-boxes, B_l and B_p that are connected diagonally, respectively. C_l transfers completely via an intermediate sub-box B_m, as shown in Figure 8.

Algorithm 2: COMPRESS(S)

$S = (u_1, u_2, ..., u_{|S|})$ is a connected shape, T is a spanning tree of $G(S)$

repeat
- $C_l \leftarrow$ pick(T_l) `// select a leaf component associated with a leaf sub-tree`
- Compress(C_l) `// start compressing the leaf component`
- **if** C_l *collides* **then**
  - $C'_r \leftarrow$ combine(C_r, C_l) or $C'_m \leftarrow$ combine(C_m, C_l) `// as described in text`
- **else**
  - $C'_p \leftarrow$ combine(C_p, C_l) `// combine` C_l `with a parent component`
- **end**
- update(T) `// update sub-trees and remove cycles after compression`

until *the whole shape is compressed into a* $\sqrt{n} \times \sqrt{n}$ *square*
Output: a square shape S_C

Algorithm 2, COMPRESS, provides a universal procedure to transform an initial connected shape S_I of any order into a compressed square shape of the same order. It takes two arguments: S_I and the spanning tree T of the *associated*

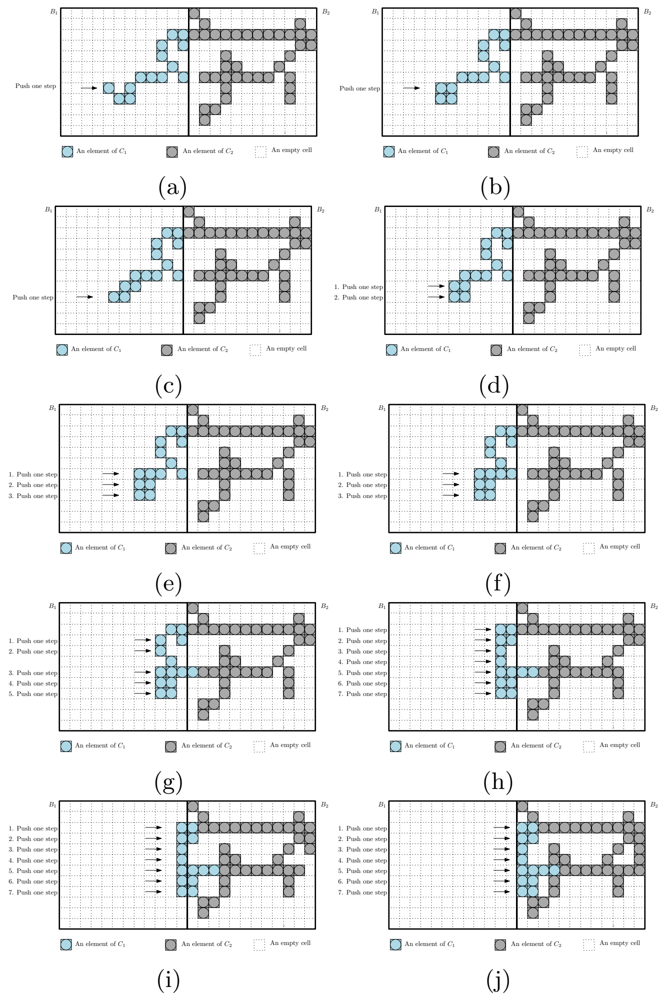
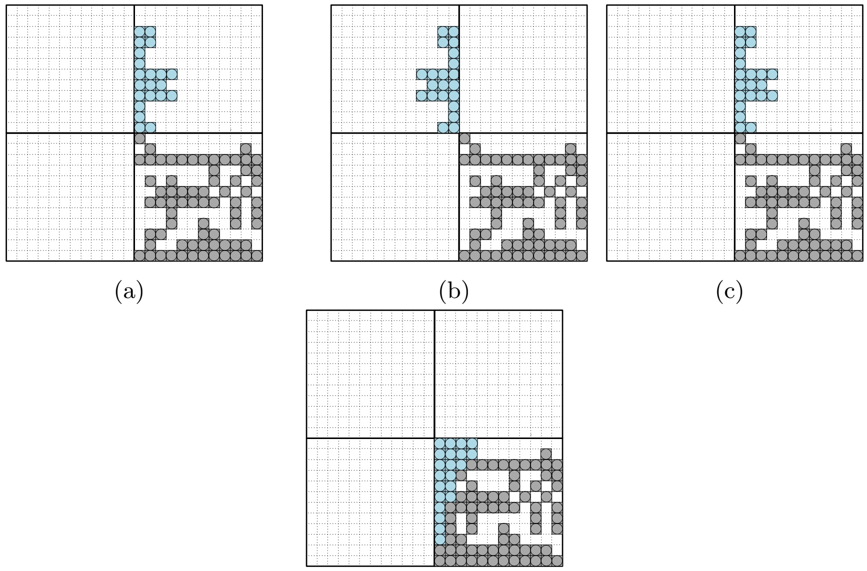


Fig. 7: Horizontal and vertical compression.

graph $G(S_I)$. In any round: Pick a leaf sub-tree of T_l corresponding to C_l inside a sub-box B_l. Compress C_l into an adjacent sub-box B_p towards its parent component C_p associated with parent sub-tree T_p. If C_l compressed with no collision, perform combine(C_p, C_l) which combines C_l with C_p into one component C'_p. If C_l collides with another component C_r inside B_l, then perform combine(C_r, C_l) into C'_r. If not, as in the diagonal compression in which C_l collides with C_m in an intermediate sub-box B_m, then C_l compresses completely into B_m and performs combine(C_m, C_l) into C'_m. Once compression is completed, update(T) computes a new sub-tree and removes any cycles. The algorithm terminates when T matches a single component of n nodes compressed into a single sub-box.



(a) (b) (c)

(d)

Fig. 8: Diagonal compression.

4.1 Correctness

In this section, we show that all properties of connectivity-preserving, transformability and universality hold in *UC-Box*, which is capable to transform any pair of connected shapes (S_I, S_F) of the same order n to each other, without breaking connectivity during its course.

Given a an initial connected shape S_I holding n nodes, then S_I can be always bounded by a square box of size $n \times n$, placed in a appropriate position to include all nodes in S_I. This box can be divided into at most $\sqrt{n}$ sub-boxes (proved in [3]), $B_1, B_2, \cdots, B_{\sqrt{n}}$, of size $\sqrt{n} \times \sqrt{n}$, each occupied sub-box may contain one or more sub-shapes (called components) of at least one node $u \in S_I$. As the shape S_I is *connected*, all occupied sub-boxes are *connected* too. This relation of connectivity can be defined as follows;

Definition 5 (Connectivity of sub-boxes). *By the above partitioning, two occupied sub-boxes, B_1 and B_2, are connected iff there are two distinct nodes $u_1, u_2 \in S_I$, such that u_1 occupies B_1 and u_2 occupies B_2 where u_1 and u_2 are two adjacent neighbours connected vertically, horizontally or diagonally.*

Next, we define connectivity between the components (sub-shapes).

Definition 6 (Connectivity of components). *By the above partitioning, two connected components, $C_1, C_2 \in S_1$ are connected iff there are two distinct elements $u \in C_1$ and $v \in C_2$, such that u and v are two adjacent neighbours connected vertically, horizontally or diagonally.*

Corollary 1. *Given the above partitioning dividing S_I into a number of components. Then, it holds that all components can be computed into a spanning tree T.*

In the following lemmas, we prove that any connected shape S_1 of n nodes can be compressed into a square box of dimension $\sqrt{n}$.

Lemma 6. *Any square box of size $\sqrt{n}$ can hold at most $2\sqrt{n}$ connected components.*

Proof. Assume S_I is a connected shape enclosed by a box of size n that is partitioned into $\sqrt{n}$ square sub-boxes of dimension $\sqrt{n}$. Then, a component $C \subseteq S_I$ of at least 1 node can occupy a sub-box, B. The component C must be connected to one of the four length-$\sqrt{n}$ boundaries of B. Assume for the sake of contradiction that C is not connected to any boundaries. This means that S_1 is disconnected and therefore $C \nsubseteq S_I$, which contracts our assumption. Observe that based in our setting, C can be connected via a path to any of the four length-$\sqrt{n}$ boundaries through at most $\sqrt{n}/2$ cells, as shown in Figure 9. Thus, one boundary can hold $\sqrt{n}/2$ distinct components, resulting in $2\sqrt{n}$ for the four boundaries. Therefore, the sub-box B can contain at most $2\sqrt{n}$ disconnected components.

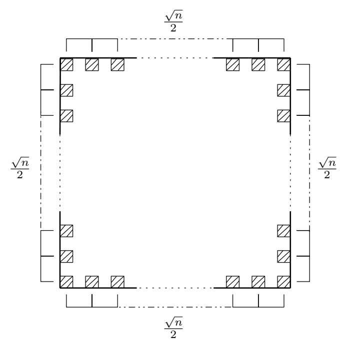


Fig. 9: A square box of four length-$\sqrt{n}$ boundaries, each can hold up to $2\sqrt{n}$ different components.

Lemma 7. *Let S_I be a connected shape of order n occupies $\sqrt{n}$ sub-boxes of size $\sqrt{n} \times \sqrt{n}$ each. Then, it is always possible to compress all n nodes into a single sub-box.*

Proof. It is sufficient to show that the number of cells inside any sub-box, $\sqrt{n} \times \sqrt{n} = n$ is enough to be filled by at most n nodes.

Now, we show that transformation *UC-Box* form a nice shape by the end of the final phase.

Lemma 8. *Starting from any connected shape S_I of order n,* Compress *forms a nice shape of order n.*

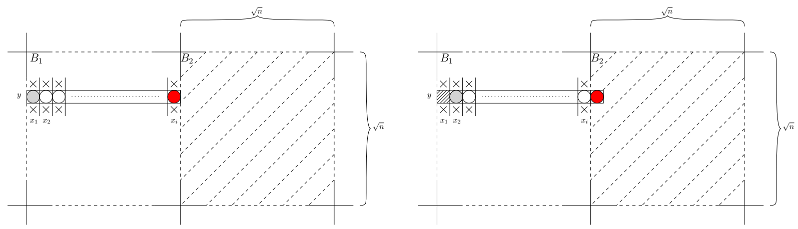
Proof. The strategy will eventually compress all components of $n \in S_I$ nodes into a $\sqrt{n} \times \sqrt{n}$ square sub-box. Regardless on which sub-box the final compressing is, the resulting final shape will be a compressed square of size $\sqrt{n}$, which is a nice shape.

Lemma 9. *Starting from an initial connected shape S_I of order n divided into $\sqrt{n}$ square sub-boxes of size $\sqrt{n}$,* Compress *compresses a leaf component $C_l \subseteq S_I$ of $k \geq 1$ nodes, while preserving the global connectivity of the shape.*

Proof. Given an initial connected shape S_I of order n enclosed into a box of length n, which is divided into $\sqrt{n}$ sub-boxes of size $\sqrt{n} \times \sqrt{n}$, each occupied sub-box contains at least one component of a total C, for all $1 \leq C \leq n$. By Corollary 1, S_I is computed into a spanning tree $T = (V, E)$ of its associated graph $G(S_I)$, where V represents components C inside the sub-boxes and E is the *neighbouring relation of connectivity* between those sub-boxes (see Definitions 5 and 6). Say that a component $C_l \in C$, occupies a sub-box B_l and represented by a leaf $v \in V$, compresses into a parent component C_p occupies an adjacent sub-box B_p and corresponds to a parent $u \in V$. We shall discuss all possible cases of moving all $k \in C_l$ lines from B_l towards B_p vertically, horizontally and diagonally, for all $1 \leq k \leq \sqrt{n}$. Due to symmetry, we only present all transformations in one direction, which holds for all other directions by rotating the shape 90°, 180°, and 270°.

Assume a left B_l and right sub-box B_p are connected horizontally. Then, all horizontal lines (rows) $k \in C_l$ push a single move right towards B_p sequentially one after the other, starting from the furthest line from the boundary between B_l and B_p. A single line $l \in k$ of length i, $1 \leq i \leq \sqrt{n}$, can occupy a row in B_l in one of the following cases:

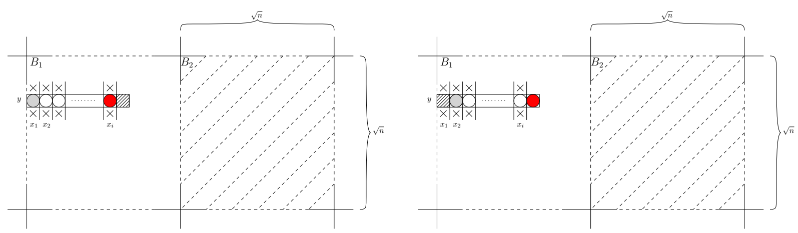
- **Case 1.** The line l of length $\sqrt{n}$ starts from the left and finishes at the right boundary of B_l. Regardless of the current configuration, l pushes one move the right from $(x, y), \ldots, (x + \sqrt{n}, y)$ to $(x + 1, y), \ldots, (x + \sqrt{n} + 1, y)$ and decreases its length by 1. This move is just like simple position permutations of the l's elements to their right neighbours positions. As a result, l stays connected to any nodes at cells $(x, y \pm 1), \ldots, (x + \sqrt{n}, y \pm 1)$, creates an empty cell at $(x + 1, y)$ and dose not break *connectivity* of all other lines in S_I. See an example in Figure 10 (a) and (b).
- **Case 2.** Similar of **Case 1** but with a line l of length less than $\sqrt{n}$. l pushes one move the right, and the length of l dose not decrease in this case. Therefore, the whole connectivity of the shape is not effected. See Figure 11.
- **Case 3.** Similar of **Case 2** in which there is two horizontal lines, l_1 and l_2, where l_1 starts from the leftmost column x and ends at $x + i$ of B_l, and l_2 occupies $(x + i + 2, y), \ldots, (x + \sqrt{n}, y)$. Now, l_1 pushes one move to fill the empty cell $(x + i + 1, y)$, a new empty cell has been created at (x, y) and



(a) A line l of length $\sqrt{n}$ occupyis a whole row in B_l.

(b) l pushes one move towards B_l.

Fig. 10: Case 1. A line l of length $\sqrt{n}$ of a leaf component that occupies the whole dimension of a sub-box.

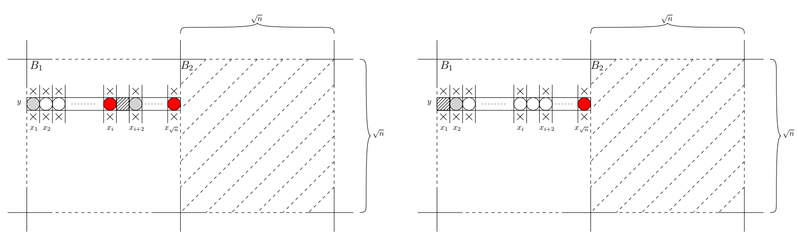


(a) A line l of length $i < \sqrt{n}$

(b) l pushes one move towards B_1.

Fig. 11: Case 2. A line l of length $i < \sqrt{n}$ of a child component.

then both lines combines into a single line in of length $\sqrt{n} - 1$, as in Figure 12. Still, this move dose not violate *connectivity* of the whole shape.



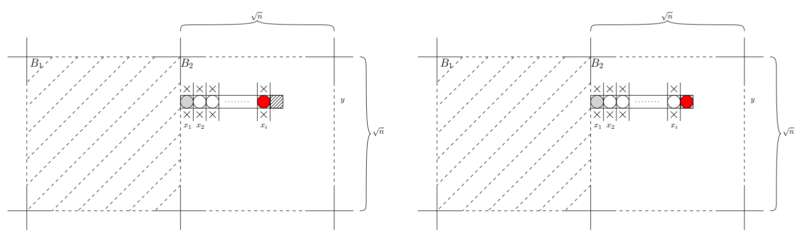
(a) Two horizontal lines occupy row y, both of lengths less than $\sqrt{n}$.

(b) l pushes one move towards B_l.

Fig. 12: Case 3. Two lines of a child component occupy a row, both of lengths less than $\sqrt{n}$.

As mentioned earlier, when a component $C_l \in B_l$ moves to merge with its parent $C_p \in B_p$, no line exceeds the four boundary of B_p. This shall preserves connectivity as well, and the following cases show how COMPRESS accomplishes this task, if l occupies a row in B_p.

- **Case 4.** The line l of length $i < \sqrt{n}$ starts from the leftmost column x and ends at $x+i$, where there is an empty cell to the right at $(x+i+1, y)$. Once l is pushed a single move to the right, l fills in that empty cell and occupies positions $(x+1, y), \ldots, (x+i+1, y)$. Therefore, the length of l increases by 1, while the connectivity is preserved. See an example of this move in Figure 13.



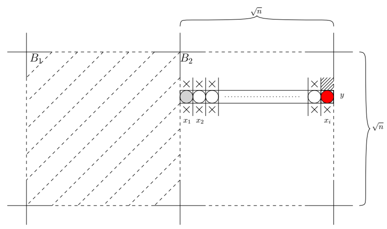
(a) The line l starts from a boundary between (B_l, B_p) and ends at $(x+i, y)$, where $i < \sqrt{n}$.

(b) l is moved one positon right to occupy the empty cell to its right.

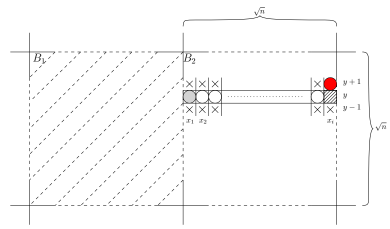
Fig. 13: Case 4. A line l of length $i < \sqrt{n}$ in a parent component.

- **Case 5.** The line l of length $\sqrt{n}$ starts from the left and finishes at the right boundary of B_p. Once l is pushed towards the right, it turns to fill empty cells at the right boundary of B_p, starting from the rightmost column to the left. The line l needs two moves per node to change its orientation. Figures 14 and 15 depicts two different examples of filling a boundary. Hence, this case preserves *connectivity* of the whole shape.

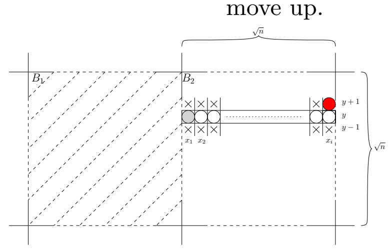
Finally, in all above cases, l pushes one move towards the right without breaking *connectivity* of S_I. As an immediate observation: whenever a line $l \subset S_I$ inside a sub-box of dimension $\sqrt{n}$, for all $1 \leq l \leq \sqrt{n}$, that starts (*perpendicularly*) from a boundary *pushes* one move towards the opposite boundary between (B_l, B_p), the global *connectivity* of the whole shape is preserved. Further, this holds also for all l lines that are pushing one move from B_l towards B_p, sequentially one after another at any order, starting from the furthest-to-nearest line from that boundary between B_l and B_p. Therefore, this must hold for a finite number of line moves a leaf C_l requires to merge with its parent C_p in B_p.



(a) A line l of length $i = \sqrt{n}$ occupyis a whole row in B_p.

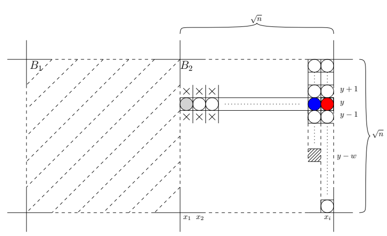


(b) l starts to fill in an empty cell $(x+i, y+1)$ at the boundary of B_p by pushing the node at $(x+i, y)$ one move up.

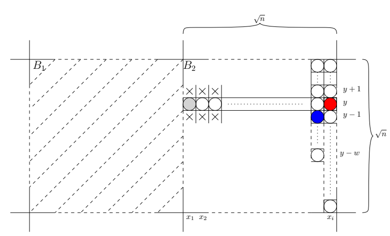


(c) l pushes one move right.

Fig. 14: Case 5 - Example 1. A line l of length $\sqrt{n}$ of a parent component occupies the whole dimension of a sub-box., where there is empty cell at the rightmost column.



(a) A line l of length $i = \sqrt{n}$ occupyis a whole row in B_p.



(b) l turns to fill in the first empty cell at the right boundary of B_p, cell $(x+i-1, y+w)$. Then, l pushes one move right to occupy the new empty cell $(x+i-1, y)$.

Fig. 15: Case 5 - Example 2. A line l of length $\sqrt{n}$ of a parent component occupies the whole dimension of a sub-box., where there is no empty cell at the rightmost column. In this case, l fills in an empty cell at the column $x+i-1$ of B_p.

4.2 Running Time

Now, we are ready to analyse the time complexity of Compress. The following lemmas provide a rough upper bound for all possible shape configurations. Given a uniform partitioning of any initial connected shape S_1 of order n, let us first show the total steps required to compress a leaf component C_l in a sub-box B_l into a parent C_p occupying an adjacent sub-box B_p, in a worst-case.

Lemma 10. *Given a pair of components C_l, C_p of k_l and k_p nodes, $1 \leq k_l + k_p \leq n$, occupying adjacent sub-boxes B_l, B_p of size $\sqrt{n}$ each, receptively. Then, C_l requires at most $O(n)$ steps to move from B_l and compress into C_p in B_p, without breaking connectivity.*

Proof. Assume B_l, B_p are connected diagonally (see Definition 5), where a component C_l occupies $\sqrt{n}$ lines in B_l and C_p consists of $\sqrt{n}$ lines in B_p as well. C_l pushes from B_i via an intermediate sub-box B_m towards B_p. Then, the $\sqrt{n}$ lines of C_l moves a distance of at most $\sqrt{n}$ to cross the boundary between B_l and B_p, in a total of at most n moves to completely occupy B_m. Again, C_l takes additional n to move into B_p and join C_p. Moreover, assume that C_l requires additional $2n$ steps to fill in a boundary at B_p. Therefore and by Lemma 9, C_l compresses into C_p in a total of at most:

$$\begin{aligned} t &= n + n + 2n = 4n \\ &= O(n), \end{aligned}$$

moves, while preserving connectivity of the shape during transformations.

The compression cost of this transformation could be very low taking only one move or being very high in some cases up to linear steps. To simplify the analysis, we divide the total cost of *UC-Box* into charging phases. We then manage to upper bound the cost of each charging phase independently of the sequential order of compressions.

Lemma 11. Compress *compresses any connected shape S_I of order n into a $\sqrt{n} \times \sqrt{n}$ square shape, in $O(n\sqrt{n})$ steps without breaking connectivity.*

Proof. Let us compute a spanning tree $T = (V, E)$ of the associated graph $G(S_I)$, where nodes V correspond to elements linked by edges E representing *relation connectivity* between them. Recall that the partitioning process of S_I into small $\sqrt{n} \times \sqrt{n}$ sub-boxes shall provide at most $O(\sqrt{n})$ occupied sub-boxes (proved in [3]). Observe that each component inside theses occupied sub-boxes matches a subtree in T. In each charging phase, the strategy compresses a single or multiple components of at most $O(\sqrt{n})$ nodes distance $O(\sqrt{n})$, which incurring a total cost of at most $O(n)$ (the worst-case is analysed in Lemma 10). Once this computed, a single or multiple subtrees of $\sqrt{n}$ nodes are removed form T. By repeating the same argument for at most $O(\sqrt{n})$ charging phases, then we arrive at the case where all nodes are removed from T, which means that all components have been compressed into a single sub-box in a total cost at most $O(n\sqrt{n})$ moves, while the whole connectivity of the shape is not broken (consult Lemma 9).

Similar to Lemma 11 but of different perspective, assume that S_I is hidden of which we cannot see the actual configuration. Colour black all the $O(\sqrt{n})$ occupied sub-boxes by S_I. Each black sub-box consists of n cells in a total of $n\sqrt{n}$ cells for all black occupied sub-boxes. Given that, in each charging phase the strategy moves $\sqrt{n}$ lines $\sqrt{n}$ distance of a total cost at most $O(n)$ moves to compress all nodes inside a black sub-box. This might happen in any order throughout the transformation. As the cost $O(n)$ is mostly sufficient to compress all nodes inside a single black sub-box and by Lemma 9, a total of at most $O(\sqrt{n})$ charging phases are fairly enough to compress all components inside the $O(\sqrt{n})$ occupied black sub-boxes, in a maximum total cost $O(n\sqrt{n})$ moves, while preserving connectivity during the transformations.

There are a number of connected shapes which can be divided, by some partitionings, into n connected components. This kind of dividing brings a wort-case complexity in which Compress meets its maximum cost, due to several reasons. First, it splits the shape into the maximum possible number of components n. Moreover, the diameter of the shape is spread over the largest space to cover n (rows or columns). Unlike other dense connected shapes of shorter diameters, outspread shapes are harder to compress due to the lack of long lines and the additional cost required for individuals and short lines. The following lemma shows that there are a finite number of specific shapes that has n components produced by some artificial partitionings.

Lemma 12. *There are a finite number of initial shapes denoted $\mathscr{S}_{\mathscr{I}}$ that can be divided into n components by some uniform partitionings. It holds that* Compress *compresses any instance $A \in \mathscr{S}_{\mathscr{I}}$ into a single square sub-box in a total of $O(n\sqrt{n})$ steps, while preserving connectivity during its course.*

Proof. Given $A \in \mathscr{S}_{\mathscr{I}}$ of n nodes with a particular partitioning positioned to divide A into n connected components. See partitioning examples of a zigzag line in Figure 16 and diagonal zigzag in Figure 17. By Lemma 6, a sub-box can have at most $2\sqrt{n}$ components, and with a given partitioning, A can occupy at most $2\sqrt{n}/n = 2\sqrt{n} = O(\sqrt{n})$ sub-boxes. As A is connected, each occupied sub-box contains at most $\sqrt{n}/2$ components of size 1 each.

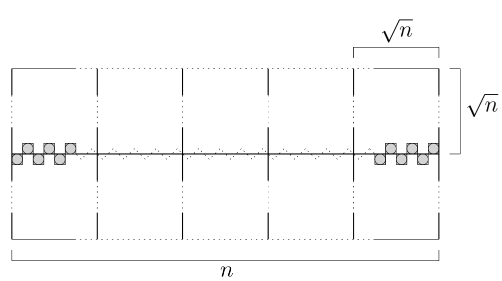


Fig. 16: A zigzag line with a partitioning positioned to cross the middle through every two nodes of $A \in \mathscr{S}_{\mathscr{I}}$.

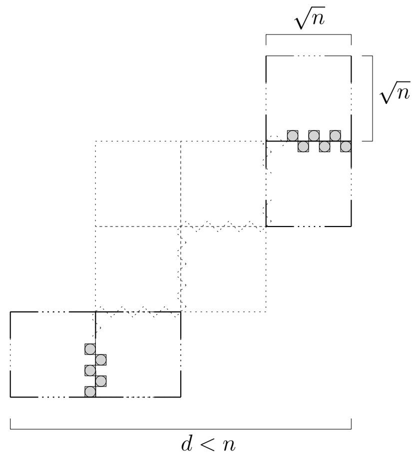


Fig. 17: A diagonal zigzag line with a partitioning positioned to cross the middle through every two nodes in $A \in \mathscr{S}_{\mathscr{I}}$ of dimension $d < n$.

We investigate how the current strategy behaves in the worst scenario. In any given charging phase i, for all $1 \leq i \leq \sqrt{n}$, Compress compresses $\sqrt{n}$ lines of a single of multiple components to their parent by moving them $\sqrt{n}$ distance in a total of $O(n)$ steps, with preserving connectivity. More, the compression may be via two diagonal sub-boxes occurring at most $2 \cdot i\sqrt{n}$. Additional cost is also given for rearrangements of at most $2 \cdot \sqrt{n}/2 = \sqrt{n}$ moves. Therefore, the charging phase i takes a total moves t_1 of at most:

$$
\begin{aligned}
t &= \sum_{i=1}^{\sqrt{n}} i + (2 \cdot i\sqrt{n}) + \sqrt{n} = \frac{\sqrt{n}(\sqrt{n}+1)}{2} + (2 \cdot i\sqrt{n}) + \sqrt{n} = \frac{n+\sqrt{n}}{2} + (2 \cdot i\sqrt{n}) + \sqrt{n} \\
&= \frac{n + \sqrt{n} + (4 \cdot i\sqrt{n}) + 2\sqrt{n}}{2} = \frac{n + 3\sqrt{n} + (4 \cdot i\sqrt{n})}{2} = \frac{5n + 3\sqrt{n}}{2} \\
&= O(n).
\end{aligned}
$$

For the upper bound, we will assign the cost t for each of the $2\sqrt{n}$ occupied sub-boxes in those particular shapes of Figures 16 and 17. Hence, the total running time T in moves is as follows:

$$
\begin{aligned}
T &= t \cdot 2\sqrt{n} \\
&= \frac{5n + 3\sqrt{n}}{2} \cdot 2\sqrt{n} = \frac{10n\sqrt{n} + 6n}{2} = 5n\sqrt{n} + 3n \\
&= O(n\sqrt{n}).
\end{aligned}
$$

By Lemma 9, Compress compresses any shape $A \in \mathscr{S}_{\mathscr{I}}$ of n nodes with a particular partitioning that dividing A into n components in at most $O(n\sqrt{n})$

steps with preserving the whole connectivity of the shape during the transformations.

By Lemma 8, the resulting compressed square shape of COMPRESS is a nice shape. Hence, Lemma 9, Lemma 11, and reversibility of nice shapes (from [3]), we therefore have:

Theorem 2. *For any pair of connected shapes (S_I, S_F) of the same order n,* UC-Box *transforms S_I into S_F (and S_F into S_I) in $O(n\sqrt{n})$ steps, while preserving connectivity during its course.*

5 Lower Bounds

In this section, we discuss a necessary minimum cost to transform the diagonal of order n into a line exploiting the parallelism of line moves in a two dimensional grid. The *Input* is a diagonal shape S_D of n nodes occupying (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$, and the *output* is a straight line S_L of n nodes occupying n consecutive cells at a column y_i or row x_i, for all $1 \leq i \leq n$. Observe that S_D matches the maximum number of steps a transformation takes to transform it into S_L, due to the inherent distance between these two pairs of shapes.

Given a complete graph $G = (V, E)$ in which V is a set of nodes in S_D and E is non-negative edge weights (Manhattan distance between nodes). Then, a simplification of this problem is collect all nodes on S_D at the bottom-most node. That is, every node in S_D must perform one or more hops through other nodes and end up at the bottom-most node. When going through a node the two or more nodes can continue traveling together and exploit parallelism.

Any such solution to the problem forms a spanning tree $T \subseteq G$, where every leaf to root path corresponds to the hops of a specific node until it reached the end. The cost of each subtree is: $c(T)$ is the total sum of the distances of its edges plus the cost of nodes $c(V)$. Every edge $E(u, v)$ has a cost equal to the distance of moving u to v, where each node has a cost of paying for each internal node of a subtree the number of nodes in its subtree. The latter cost is due to not being able to exploit parallelism whenever turning, and any hop requires another turn. The cost due to distances is just:

$$c(E) = \sum_{e \in E(T)} cost(e), \tag{1}$$

The cost of internal nodes is equal to:

$$c(V) = \sum_{i=1}^{d(T)} i \cdot v \in d(T)_i, \tag{2}$$

Where $d(T)$ is the depth of tree T and $d(T)_i$ is the number of nodes at level i. The total cost in number of moves given by such a tree T is the sum of 1 and 2:

$$c(T) = c(E) + c(V). \tag{3}$$

Now, the two sums seem to give some trade-off. If the depth is very small, then the cost due to distances seems to increase (e.g., if all nodes travel into one hop, they all pay their distances and the cost is quadratic). This approach is similar to any sequential transformation of individual movements which pays a cost of $\Theta(n^2)$ to transform S_D into S_L. The summation of the total individual distances is, $\Sigma\Delta = 0 + 1 + 2 + \ldots + (n-1) = \Theta(n^2)$, independently of whether connectivity is preserved or not during transformations. This is because of the inherent individual distance between S_D and S_L. On the other hand, the tree T of very large depths looks as a spanning line where a lot of parallelism must be exploited. The distance in this case would cost only $c(E) = n - 1$. While the sum of turns at each node becomes quadratic, $c(V) = n^2$. Therefore, we observe that more balanced trees of logarithmic depth, such as binary trees, manage to balance both sums and give total cost $n \log n$. Due to the trade-off, it does not seem easy to lower bound in the general case. Further, it does not seem easy to lower bound the edges-sum even by some parameters depending on the depth (so that both sums will be using similar parameters). It might not even be related to that parameter. Therefore, we tried to further simplify the problem by restricting the solutions to extremely limited depths. Below, we have successfully managed to establish some special-case lower bounds for this problem.

It can be easily seen that no uniform strategy can achieve better bound than the $O(n \log n)$-time strategy of [3], by simply increasing the number of lines that are merging in every phase to decrease the number of phases. Hence, we have the following proposition.

Proposition 2. *Any strategy represented by a balanced tree performs $\Omega(n \log n)$ moves.*

Proof. Observe that such a strategy is essentially trying to increase the degrees of the nodes of a balanced tree and decrease its depth. For example, take any merging parameter $k \geq 2$. Notice that the $O(n \log n)$-time transformation of [2] (called *DL-Doubling*) has k = 2, as it is merging pairs of lines and get $\log n$ phases. So, in every phase i we are going to partition the L lines into L/k groups of k consecutive lines each and merge the lines within each group into a single line.

First , in phase 1, $L = n$, and we are partitioning into n/k groups. For each group we are paying at least k^2 asymptotically to merge the lines in it. Therefore for phase 1 we pay $(n/k)k^2 = nk$ (this is similar also to the $O(n\sqrt{n})$-time transformation in [3], but there it only did it once and gathered all the lines to the bottom and not in any further phases). Then in phase 2 $L = n/k$, we are partitioning into $L/k = n/k^2$ groups. Each group is paying at least k^3 asymptotically, because the distance between consecutive lines has now increased to k (roughly). Thus gives again cost at least nk. This should hold for the other phases.

Now, Observe that this strategy gives $\log_k n = \log n \log k$ phases. If each is paying nk, then the total cost is $(nk)(\log n \log k) = n \log n(k \log k)$, which for all $k \geq 2$ is at least $2n \log n = \Omega(n \log n)$. This would be helpful because it excludes any attempts to get a better than the $O(n \log n)$-time transformation by simply playing with the degrees of the tree in a uniform way (which in turn decreases its depth and thus the number of phases).

5.1 An $\Omega(n \log n)$ Lower Bound for The 2-HOP Tree

We start to study a special case lower bound for all solutions that represented by a tree T of a minimum depth. Assume any such solution moves all nodes in only one-way via shortest paths towards the target node. As a node joins other nodes, they do not split after that during the transformation. Let $d(T)$ denotes the depth of the tree. For $d(T) = 1$, the tree becomes a star, and the total cost is quadratic in this case, due to the summation of individual distances $c(E) = 0 + 1 + 2 + \ldots + (n-1) = \Theta(n^2)$.

Then, we investigate the tree T of depth at most 2, $d(T) = 2$. Observe that for any node in the tree we are paying "*asymptotically*" at least the square of number of children that it has. The reason is that at most 2 of its children can be nodes at distance 1, then at most 2 can be nodes of distance 2, at most 2 of distance i in general due to the neighbouring properties of the diagonal. Thus, it gives a total cost for any such tree which is similar to $c(T) = \sum_i d(u_i)^2$, where $d(u_i)$ is the number of children of u_i. That is, the squares of the degrees of all internal nodes, excluding their parent (the root u_i). When taking into account all nodes, this gives a graph-theoretic measure related to chemical compounds, known as the *Zagreb index* [28, 27]. A great amount of bounds have been established for it, but none of which could be directly used in our case.

Let T of k nodes be a tree of depth 1, as shown in Figure 18. Then, the total asymptotic cost of the tree $c(T)$ is at least:

$$c(T) \geq \sum_{i=0}^{k} d(u_i)^2, \tag{4}$$

Where $d(u_i)$ is the degree of node u_i.

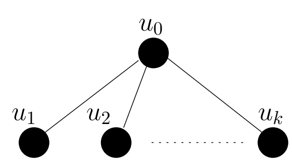


Fig. 18: A tree T of k nodes.

Given the tree T of k we show the first case of a minimum total cost T must pay if it has a node $u_i \in T$ with degree at least $n \log n$;

Lemma 13. *If $\exists\ d(u_i) \geq \sqrt{n \log n}$, then $c(T) \geq n \log n$, for all $0 \leq i \leq k$.*

Proof. The proof is straightforward. Consider the tree T of k nodes in Figure 18. If $k \geq \sqrt{n \log n}$, then the tree shall have a minimum total cost of $c(T) \geq d(u_0)^2 = k^2 = (\sqrt{n \log n})^2 = n \log n$. In general, if there exists a node $u_i \in T$, for all $0 \leq i \leq k$, such that $d(u_i) \geq \sqrt{n \log n}$, then the total cost of the tree must be at least $c(T) \geq n \log n$.

Now, let us assume that all nodes in the tree have degrees less than $n \log n$. Thus, we show the lower bound of Lemma 13 holds in this case.

Lemma 14. *Let $d(u_i) < \sqrt{n \log n}$ $\forall i$, where $0 \leq i \leq d(u_0) = k$. Then, $c(T) > n \log n$.*

Proof. Given a tree T of n nodes that has depth of 2, and a subtree $T' \subseteq T$ of k nodes as in Figure 18. So, let $d(u_i) < \sqrt{n \log n}$, for all $0 \leq i \leq d(u_0) = k$. Assume without loss of generality that the nodes $u_i \in T'$, for all $1 \leq i \leq d(u_0) = k$, are ordered in non-increasing degrees from left to right (increasing order i), that is, $d(u_1) \geq d(u_2) \geq \ldots \geq d(u_k)$. Hence, there are $n-(k+1) \in T$ nodes remaining to be assigned. As $d(u_1) \in T'$ is the maximum, it must hold that, $d(u_1) \geq \frac{n-(k+1)}{k}$, thus $\frac{n-(k+1)}{k} \leq d(u_1) < \sqrt{n \log n}$.

Next, there are $n-(k+1)-d(u_1) \in T$ nodes need to be allocated. As $d(u_2) \in T'$ is the maximum among the rest, it must hold that $d(u_2) \geq \frac{n-(k+1)-d(u_1)}{k-1}$, thus $\frac{n-(k+1)-d(u_1)}{k-1} \leq d(u_2) < \sqrt{n \log n}$. In general, if a node $d(u_i) \in T'$ is the maximum, then the following must hold that,

$$d(u_i) \geq \frac{n - \left(\sum_{j=0}^{i-1} d(u_i)\right) - 1}{k - (i-1)}, \tag{5}$$

Thus,

$$\frac{n - \left(\sum_{j=0}^{i-1} d(u_i)\right) - 1}{k - (i-1)} \leq d(u_i) < \sqrt{n \log n}, \tag{6}$$

Now, plug $i = 1$ and $k = d(u_0)$ in (5) yields,

$$d(u_1) \geq \frac{n - d(u_0) - 1}{d(u_0)} > \frac{n - \sqrt{n \log n} - 1}{\sqrt{n \log n}}, \tag{7}$$

When $i = 2$, we will get,

$$d(u_2) \geq \frac{n - (d(u_0) + d(u_1)) - 1}{d(u_0)} > \frac{n - 2\sqrt{n \log n} - 1}{\sqrt{n \log n} - 1}, \tag{8}$$

For all $1 \leq i \leq d(u_0) = k$, we shall obtain,

$$d(u_i) \geq \frac{n - \left(\sum_{j=0}^{i-1} d(u_i)\right) - 1}{d(u_0) - (i-1)} > \frac{n - i\sqrt{n \log n} - 1}{\sqrt{n \log n} - (i-1)}, \tag{9}$$

Then, we plug (9) into (4) of Observation **??**, which implies,

$$c(T) > \sum_{i=0}^{d(u_0)} \left[\frac{n - i\sqrt{n \log n} - 1}{\sqrt{n \log n} - (i-1)} \right]^2 > (\sqrt{n \log n})^{-1} \sum_{i=0}^{d(u_0)} (n - i\sqrt{n \log n} - 1)^2 \tag{10}$$

$$\simeq (\sqrt{n \log n})^{-1} \sum_{i=0}^{d(u_0)} (n - i\sqrt{n \log n})^2 \tag{11}$$

$$= (\sqrt{n \log n})^{-1} \sum_{i=0}^{d(u_0)} (n^2 + i^2 \cdot n \log n - 2i \cdot n\sqrt{n \log n}) \tag{12}$$

$$= (\sqrt{n \log n})^{-1} \left[d(u_0) \cdot n^2 + \sum_{i=0}^{d(u_0)} n(i^2 \log n - 2i\sqrt{n \log n}) \right] \tag{13}$$

$$= (\sqrt{n \log n})^{-1} \left[d(u_0) \cdot n^2 + n \sum_{i=0}^{d(u_0)} (i^2 \log n - 2i\sqrt{n \log n}) \right]. \tag{14}$$

We need to bound the summation of (14):

$$\sum_{i=0}^{d(u_0)} n(i^2 \log n - 2i\sqrt{n \log n}) = \sum_{i=0}^{d(u_0)} i^2 \log n - \sum_{i=0}^{d(u_0)} 2i\sqrt{n \log n} \tag{15}$$

$$= \left(\log n \sum_{i=0}^{d(u_0)} i^2 \right) - \left(2\sqrt{n \log n} \sum_{i=0}^{d(u_0)} i \right) \tag{16}$$

$$= \log n \left(\frac{d(u_0)^3}{3} + \frac{d(u_0)^2}{2} + \frac{d(u_0)}{6} \right) \tag{17}$$

$$- 2\sqrt{n \log n} \cdot \frac{d(u_0)(d(u_0 + 1))}{2} \tag{18}$$

$$\simeq \log n \left(d(u_0)^3 + d(u_0)^2 + d(u_0) \right) \tag{19}$$

$$- 2\sqrt{n \log n} \cdot d(u_0)^2. \tag{20}$$

Now, plug (20) into (14), then it will give a total cost of the tree $c(T)$ that asymptotically bounded on:

$$c(T) > (\sqrt{n \log n})^{-1} \left(n^2 \cdot d(u_0) + n \log n \cdot d(u_0)^3 - \sqrt{n \log n} \cdot d(u_0)^2 \right) \tag{21}$$

$$= \frac{n^2 \cdot d(u_0) + n \log n \cdot d(u_0)^3}{\sqrt{n \log n}} - d(u_0)^2 \tag{22}$$

$$> \frac{n^2 \cdot d(u_0) + n \log n \cdot d(u_0)^3}{\sqrt{n \log n}} - n \log n \tag{23}$$

Finally, since $d(u_0) > 1$, it implies that,

$$c(T) > \frac{n^2}{\sqrt{n \log n}} - n\log n = \frac{n^2 \cdot n \log n}{\sqrt{n \log n} \cdot \log n} - n \log n = \frac{n^2}{\sqrt{n \log n}} - n \log n \tag{24}$$

$$= \frac{n \cdot n}{\sqrt{n}\sqrt{\log n}} - n \log n = \frac{n \cdot \sqrt{n}}{\sqrt{\log n}} - n \log n = \frac{n \log n \cdot \sqrt{n}}{\log n \cdot \sqrt{\log n}} - n \log n \tag{25}$$

$$> \frac{n \log n \cdot \sqrt{n}}{\log^2 n} - n \log n = \left(\frac{\sqrt{n}}{\log^2 n} - 1 \right) n \log n = \tag{26}$$

$$= \Omega(n \log n). \tag{27}$$

As a result, both Lemmas 13 and 14 show that the total cost of any spanning tree $c(T)$ of n nodes and depth at most $d(T) \leq 2$ is always bounded by $\Omega(n \log n)$.

Theorem 3. *Any 2-HOP spanning tree $c(T)$ of n nodes and depth at most $d(T) \leq 2$ has a total cost $c(T)$ of $\Omega(n \log n)$.*

5.2 A conditional $\Omega(n \log n)$ Lower Bound - One way transformation

Now we present another special case lower bound for transformations that are exploiting line moves. Again, our techniques is based on one-way assumption in which all nodes move in one direction via shortest paths towards the target node (e.g., from top to bottommost node in the diagonal). Whenever a node joins other nodes, they continue travelling together and do not split thereafter.

Let S_D be a diagonal connected shape occupies of order n nodes (lines of length 1) on positions (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$. The argument starts by deciding a potential target position of the final straight line, S_L. Assume a potential placement of S_L horizontally on the bottommost row y_1 or vertically at the leftmost column x_1 of the shape. With this, and without loss of generality, we therefore assume that lines only move down and leftwards. This is convenient as they always push a minimum distance towards the target potential placement, i.e, in our assumption at row y_1 or column x_1 of S_L.

Enclose each individual node of S_D into a square box of dimension $d = 1$ to have a total of n boxes, see the black squares boxes in Figure 19. Then, double the dimension of the square boxes to surround every two nodes in a total of $n/2$ boxes of $d = 2$, such as the red squares boxes in Figure 19. Repeat doubling dimensions each of different colours $\log n$ times, until arriving at 1 square box of $d = n$, which contains all nodes of S_D. Assume that n is a power of 2, therefore, the total number of all square boxes shall be exactly $n+n/2+n/4+\ldots+1 = 2n-1$ boxes, where there are n boxes of $d = 1$, $n/2$ boxes of $d = 2$, $\ldots$ and 1 box of $d = n$.

Now, observe that such a transformation at any order during its course, must pay at least n steps to push n nodes out from their black boxes of dimension 1. Likewise, when a line l_1 of 1 node occupying a cell (x, y) (e.g, Figure 20 (a)) is

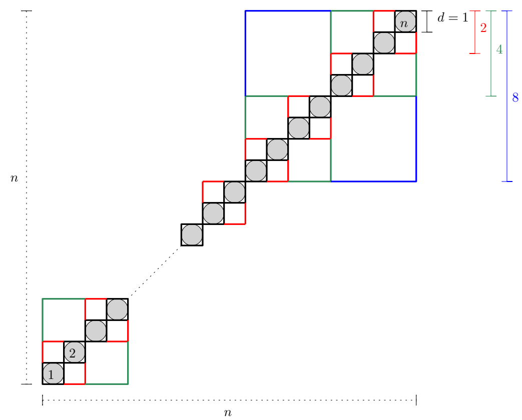


Fig. 19: An initial diagonal shape S_D on positions (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$ enclosed into $\log n$ boxes of dimensions $1, 2, 4, \ldots, n$.

pushed one step to cross the boundary of its black box of dimension 1, no one will be pushed for free to move from any box of any size. The same argument follows, when a line l_2 of 2 nodes at $(x-1, y)$ and $(x-1, y-1)$ pushes 2 steps, say to the left, and leaves its red box of size 2×2, then no line is pushed to leave their red box of dimension 2 for free, see an example in Figure 20 (b). More formally, by this observation, any transformation exploiting linear-strength pushing mechanism requires at least $d \cdot n/d$ steps, where the dimension $d = 2^k$ for all $0 \leq k \leq \log n$, to evacuate all lines from n/d boxes of dimension d, without pushing any other lines for free in any arrangements during its course.

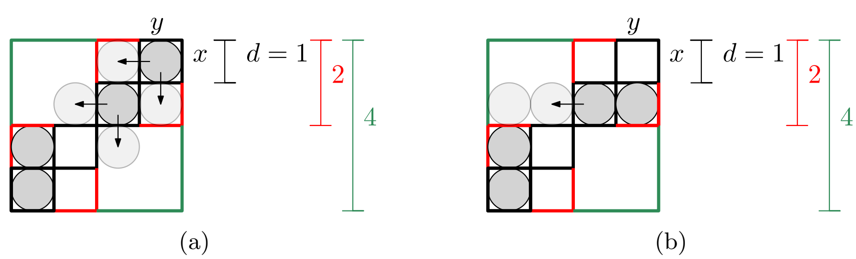


Fig. 20: Artificial boxes of dimensions 1, 2 and 4 enclosing nodes of S_D.

There is another case might happen during transformations of any strategy based on line moves. Consider a square box of dimension $d = 2^k$, for all $0 \leq k \leq \log n$, consists of four sub-boxes of dimension 2^{k-1} each, say without loss of generality, a blue box of size 8×8 holds four green sub-boxes of length-4 dimension, as depicted in Figure 21. Here, one can say that the line of length 4 in the top-right corner pushes 4 steps towards the left, which consequently moves the one in top-left for no cost. In such case, we should not forget the cost of forming the two length-4 line is prepaid previously. That is, the strategy already paid a cost of forming them initially from sub-boxes of dimensions 1 and 2. Further, one of the length-4 line is incurred the transformation a cost of 4 steps at least, to change its direction completely to occupy new 4 consecutive columns, that is, to line up vertically with the other length-4 line. Recall that any line of length 2^k occupying a box of $d = 2^k$ and crosses a boundary of that box vertically or horizontally, no line is pushed originally for free. This holds for any initial sub-lines of lengths less than 2^k. With this, we can then

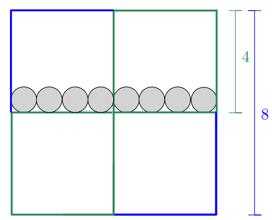


Fig. 21: An example of the special case when nodes push others for free (on the same row of column).

calculate the total minimum steps that must be paid to evacuate all lines (of various lengths) from the $2n-1$ square boxes. In each box, any strategy has to pay a minimum number of steps equals to the box's dimension $d = 2^k$, for all $0 \leq k \leq \log n$, at any order during transformations. Thus the total minimum steps will be $(1 \cdot n) + (2 \cdot n/2) + (4 \cdot n/4) + \ldots + (n \cdot 1) = n + n + \ldots + n$. Now, since we have $\log n$ different dimensions, we obtain a total of $n \log n$ minimum number of steps. Hence, any transformation exploits linear-strength pushing mechanism asks for at least $\Omega(n \log n)$ steps to form all n nodes at the potential placement and transform S_D into S_L.

Then, we try to apply a recursive transformation to check whether this will yield a better lower bound. That is, let S_L be an initial straight line of n nodes (say horizontal) which occupies the bottommost row y_1 and S_D is a target diagonal of order n (lines of length 1) occupies positions (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$. By reversibility, the pair (S_D, S_L) are transformable to each other, such that if $S_D \rightarrow S_L$ ("$\rightarrow$" means "is transformed to") then $S_L \rightarrow S_D$ via a sequence of line moves. Then the cost of $S_D \rightarrow S_L$ is equivalent to $S_L \rightarrow S_D$.

We define two independent sets, $S^1_{n/2}$ and $S^2_{n/2}$, each of which contains arbitrary $n/2$ nodes during configurations, such that $S^1_{n/2} \neq S^2_{n/2}$ and $S^1_{n/2} \cap S^2_{n/2} = \phi$, see Figure 22. Given a transformation A, then pick any $n/2$ nodes randomly chosen from S_L. At any time, A must pay a cost of at least $n/2$ for these specific $n/2$ nodes to cross a boundary of the $S^2_{n/2}$ box and get inside it, through the two shaded areas. This cost is based of the minimum distance any group of $n/2$ nodes have to pay, in order to reach their final positions inside the $S^2_{n/2}$ box.

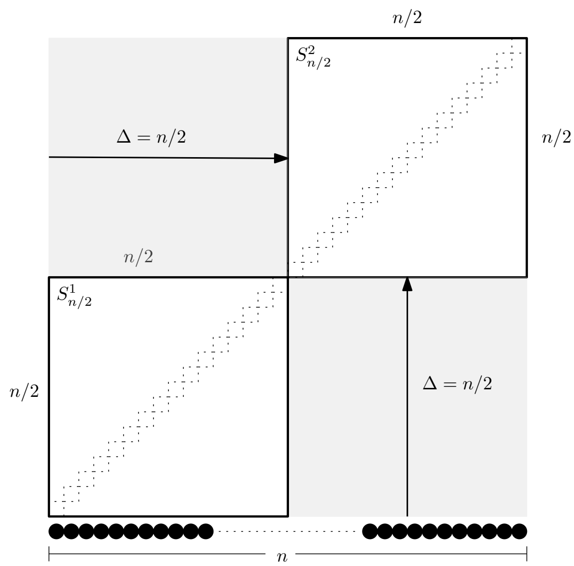


Fig. 22: All nodes are initially placed on the bottom. Two independent sets $S^1_{n/2}$ and $S^2_{n/2}$ are defined.

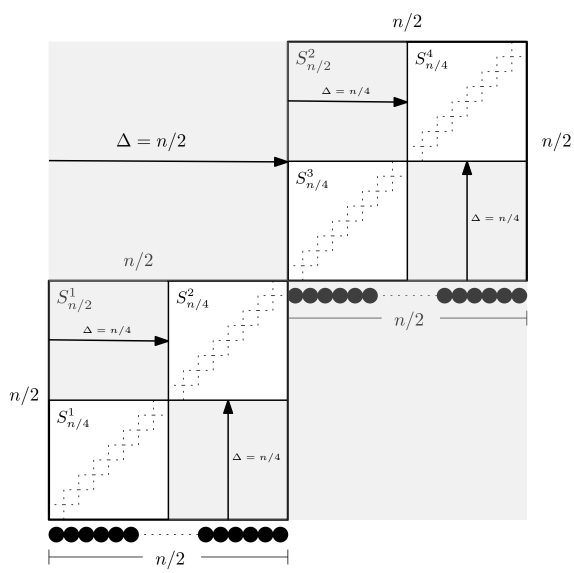


Fig. 23: The two sets $S^1_{n/2}$ and $S^2_{n/2}$ are divided into four independent sets $S^1_{n/4}, S^2_{n/4} \subset S^1_{n/2}$ and $S^3_{n/4}, S^4_{n/4} \subset S^2_{n/2}$.

Similarly, we split the two boxes into half, by defining four independent sets $S^1_{n/4}, S^2_{n/4} \subset S^1_{n/2}$ and $S^3_{n/4}, S^4_{n/4} \subset S^2_{n/2}$. At any time, A chooses two random group of nodes, each of size $n/4$. Then, A has to pay a cost of at least $2 \cdot n/4$ for these nodes to cross boundaries and get inside the $S^2_{n/4}$ and $S^4_{n/4}$ boxes. See Figure 23. Repeat the same argument for the rest of $\log n$ charging phases, until each nodes occupies its final target positions in the diagonal S_D. As every node in each independent set will eventually reach its final position, then it will be contained into $\log n$ boxes. Therefore, the total amortized cost will be $(1 \cdot n/2) + (2 \cdot n/4) + (4. \cdot n/8) + \ldots + (n/2 \cdot 1) = n \log n/2$. As a result, we state that:

Theorem 4. *Any transformation strategy exploiting line moves requires $\Theta(n \log n)$ steps to transform the diagonal into a line.*

6 Conclusions and Open Problems

We have presented efficient transformations for the line-pushing model introduced in [3] and some first lower bounds for restricted sets of transformations. Our first transformation works on the family of all Hamiltonian shapes and matches the running time of the best known transformations ($O(n \log n)$) while additionally managing to preserve connectivity throughout its course. We then gave the first universal connectivity preserving transformation for this model. Its running time is $O(n\sqrt{n})$ and works on any pair of connected shapes of the same order. Our $\Omega(n \log n)$ lower bounds match the best known upper bounds, still they are valid only for restricted sets of transformations.

This work opens a number of interesting problems and research directions. An immediate next goal is whether it is possible to develop an $O(n \log n)$-time universal connectivity-preserving transformation. If true, then a natural question is whether a universal transformation can be achieved in $o(n \log n)$-time (even when connectivity can be broken) or whether there exists a general $\Omega(n \log n)$-time matching lower bound. As a first step, it might be easier to develop lower bounds for the connectivity-preserving case. There are also a number of interesting variants of the present model. One is a centralised parallel version in which more than one line can be moved concurrently in a single time-step. Another, is a distributed version of the parallel model, in which the nodes operate autonomously through local control and under limited information.

References

1. Hugo Akitaya, Esther Arkin, Mirela Damian, Erik Demaine, Vida Dujmovic, Robin Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The O(1) musketeers. In *27th Annual European Symposium on Algorithms, ESA*, volume 144 of *LIPIcs*, pages 3:1–3:14, 2019.
2. Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *CoRR*, abs/1904.12777, 2019.

3. Abdullah Almethen, Othon Michail, and Igor Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science*, 830-831:43 – 59, 2020.
4. Greg Aloupis, Nadia Benbernou, Mirela Damian, Erik Demaine, Robin Flatland, John Iacono, and Stefanie Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Computational geometry*, 46(8):917–928, 2013.
5. Greg Aloupis, Sébastien Collette, Erik Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using O(logn) parallel moves. In *International Symposium on Algorithms and Computation*, pages 342–353. Springer, 2008.
6. Dana Angluin, James Aspnes, Zoë Diamadi, Michael Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006.
7. Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, 1998.
8. Julien Bourgeois and Seth Goldstein. Distributed intelligent MEMS: progresses and perspective. *IEEE Systems Journal*, 9(3):1057–1068, 2015.
9. Zack Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *The International Journal of Robotics Research*, 23(9):919–937, 2004.
10. Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.
11. Andrea EF Clementi, Miriam Di Ianni, Massimo Lauria, Angelo Monti, Gianluca Rossi, and Riccardo Silvestri. On the bounded-hop mst problem on random euclidean instances. *Theoretical computer science*, 384(2-3):161–167, 2007.
12. Jurek Czyzowicz, Dariusz Dereniowski, and Andrzej Pelc. Building a nest by an automaton. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA*, 2019.
13. Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, April 2015.
14. Joshua Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.
15. Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. In *International Conference on DNA-Based Computers*, pages 148–164. Springer, 2016.
16. Zahra Derakhshandeh, Robert Gmyr, Andréa Richa, Christian Scheideler, and Thim 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.
17. Giuseppe A. Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, Mar 2019.
18. David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55:78–88, 2012.

19. Shawn Douglas, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William Shih. Self-assembly of dna into nanoscale three-dimensional shapes. *Nature*, 459(7245):414, 2009.
20. Adrian Dumitrescu and János Pach. Pushing squares around. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 116–123. ACM, 2004.
21. Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for fast locomotion of metamorphic robotic systems. *The International Journal of Robotics Research*, 23(6):583–593, 2004.
22. Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration. *IEEE Transactions on Robotics and Automation*, 20(3):409–418, 2004.
23. Sándor Fekete, Robert Gmyr, Sabrina Hugo, Phillip Keldenich, Christian Scheffer, and Arne Schmidt. Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata. *CoRR*, abs/1810.06360, 2018.
24. Sándor Fekete, Andréa Richa, Kay Römer, and Christian Scheideler. Algorithmic foundations of programmable matter (Dagstuhl Seminar 16271). In *Dagstuhl Reports*, volume 6, 2016. Also in *ACM SIGACT News*, 48.2:87-94, 2017.
25. Kyle Gilpin, Ara Knaian, and Daniela 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.
26. Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, Christian Scheideler, and Thim Strothmann. Forming tile shapes with simple robots. *Natural Computing*, pages 1–16, 2019.
27. Ivan Gutman, B Ruscić, Nenad Trinajstić, and Charles F Wilcox Jr. Graph theory and molecular orbitals. xii. acyclic polyenes. *The Journal of Chemical Physics*, 62(9):3399–3405, 1975.
28. Ivan Gutman and Nenad Trinajstić. Graph theory and molecular orbitals. total φ-electron energy of alternant hydrocarbons. *Chemical Physics Letters*, 17(4):535–538, 1972.
29. Alon Itai, Christos Papadimitriou, and Jayme Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
30. Ara Knaian, Kenneth Cheung, Maxim Lobovsky, Asa Oines, Peter Schmidt-Neilsen, and Neil 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.
31. Othon Michail, George Skretas, and Paul Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019.
32. Othon Michail and Paul Spirakis. Elements of the theory of dynamic networks. *Commun. ACM*, 61(2):72–81, 2018.
33. An Nguyen, Leonidas Guibas, and Mark Yim. Controlled module density helps reconfiguration planning. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, pages 23–36, 2000.
34. Paul Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.
35. Paul Rothemund and Erik 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.

36. Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.
37. Jennifer Walter, Jennifer Welch, and Nancy Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.
38. Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.
39. Damien Woods, H Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng 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.
40. Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010.
41. Yukiko Yamauchi, Taichi Uehara, and Masafumi 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.
42. Mark Yim, WeiMin Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.