On Efficient Connectivity-Preserving Transformations in a Grid[☆]

Abdullah Almethen[*], Othon Michail, Igor Potapov

Department of Computer Science, University of Liverpool, Liverpool, 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, called a *line move*. 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 contain a Hamiltonian path. 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.

Keywords:
Line movement, Discrete transformations, Shape formation, Reconfigurable robotics, Time complexity, Programmable matter

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 [2, 3, 4, 5] down to nano size of individual monads [6, 7].

Recent research has highlighted the need for the development of an algorithmic theory of such systems. An apparent lack of a formal theoretical study of this prospective, including modelling, possibilities/limitations,

[☆]A preliminary version of the results in this paper has appeared in [1].

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

Email addresses: `A.Almethen@liverpool.ac.uk` (Abdullah Almethen), `Othon.Michail@liverpool.ac.uk` (Othon Michail), `Potapov@liverpool.ac.uk` (Igor Potapov) algorithms and complexity has been emphasised in, e.g.,[8] and [9]. 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 [10, 11, 12], mobile robotics [13, 14, 15, 16, 17], reconfigurable robotics [18, 19, 20, 21, 22], passively-mobile systems [23, 24, 25, 8], DNA self-assembly [26, 27, 28, 29], and the latest emerging sub-area of "Algorithmic Foundations of Programmable Matter" [30].

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 [31], Dumitrescu *et al.* [32, 10] and Michail *et al.* [9] 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 [33].

A new linear-strength mechanism was introduced by Almethen *et al.* in [34], where a whole line of consecutive devices can, in a single time-step, move by one position in a given direction. That model comes as a natural generalisation of other existing models of reconfiguration with a particular focus on exploiting the power of parallelism for fast global reconfiguration. Apart from the pure theoretical interest of exploring fast transformations on a grid, this model also provides a practical framework for efficient reconfigurations of real systems. For example, this framework could be applied to reconfigurable robotic systems in which the individual devices are equipped with linear-strength locomotion mechanisms.

Since any two shapes with an equal number of elements can be transformed into each other with line moves [34], 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 [34], 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 [34] 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 contain a Hamiltonian path. 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, [31, 9] 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, [9] proves that the problem of deciding transformability is in $\mathbf{P}$. 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 [9]. 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 [20, 35, 16, 10, 9, 36] as well as practical implementations [5]. 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 [9].

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.* [18, 37] 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.* [29] 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 [34] 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 [31, 9] 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 [14, 38, 39]. 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 [34] 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 *HamiltonianToLine*, 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* $G(S)$ has a Hamiltonian path (see also [40]). 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 [34]), 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.

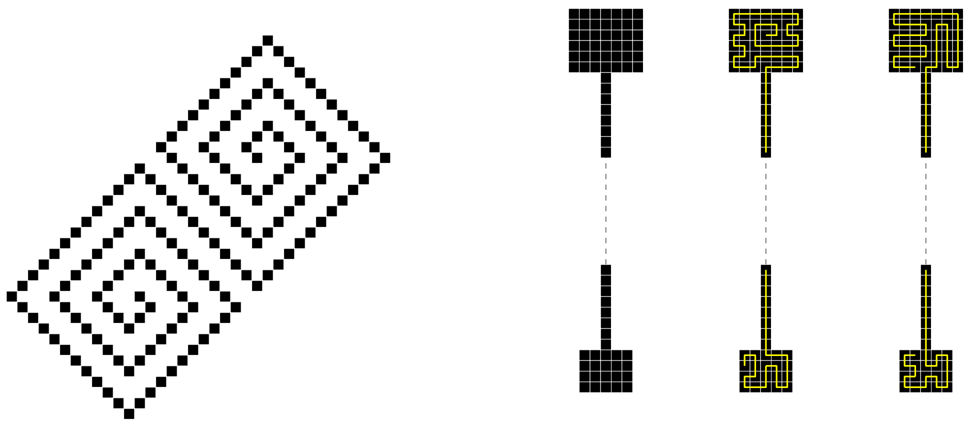
In this paper, we exploit the linear-strength mechanism of the *line-pushing model* introduced in [34]. A line L is a sequence of nodes occupying consecutive cells in one direction of the grid, that is, either vertically or horizontally but not diagonally. A ***line move*** is an operation of moving 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.*

A configuration of the system is defined as a mapping $C : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$, where $C(x, y) = 0$ if cell(x, y) is empty or $C(x, y) = 1$ if cell(x, y) is occupied by a node. Equivalently, a configuration can be defined as a set $\{(x, y) : x, y \in \mathbb{Z}$ and $C(x, y) = 1\}$. Let C_0 denote the initial configuration of the system. We say that C' is *directly reachable* from C and denoted $C \to C'$, if C can be transformed into C' in one line move. Moreover, C' is reachable from C, denoted $C \to^* C'$, if there is a sequence of configurations $C = C_1, C_2, \ldots, C_t = C'$ such that $C_i \to C_{i+1}$ holds for all $i \in \{1, 2, \ldots, t - 1\}$. We next define a family of shapes that are used in one of our transformations.

Definition 2 (Hamiltonian Shapes)**.** *A shape S is called Hamiltonian iff $G(S) = (V, E)$ contains 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, see Figure 1.*

We define a *rectangular path* P over the set of cells as $P = [c_1, c_2, c_3, \ldots, c_k]$, where $c_i, c_{i+1} \in \mathbb{Z} \times \mathbb{Z}$ are two cells adjacent to each other either vertically or horizontally, for all $i \in \{1, 2, \ldots, k - 1\}$. Given any *rectangular path* P, let C_P be the configuration of P, which is the subset of C (configuration of the system)


(a) A double-spiral shape

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

Figure 1: Examples of Hamiltonian shapes.

restricted to the cells of P. The following proposition proves a basic property of line moves which will be a core technical tool in our transformation for 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.

Proposition 1 (Transparency of Line Moves)**.** *Let S be any shape, $L \subseteq S$ any line and P a rectangular path starting from a position adjacent to one of L's endpoints. 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 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. Figure 2 shows an example of configuration C_P.

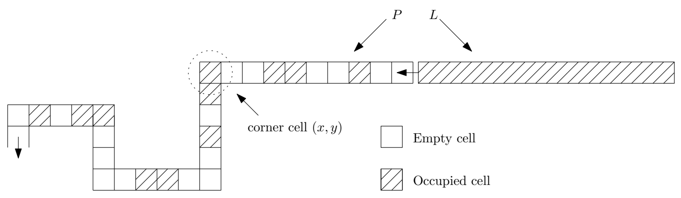
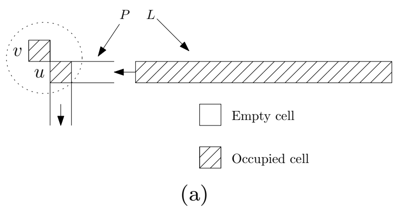


Figure 2: A path P of a given configuration C_P. A line L will pass along P.

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.



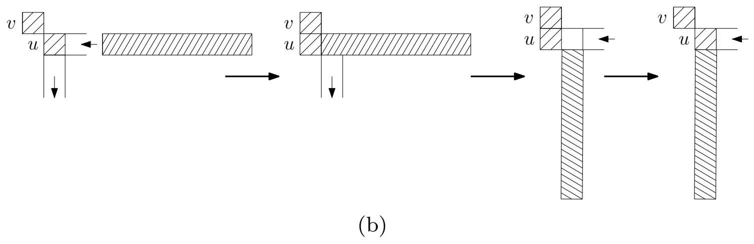


Figure 3: A line L moving through a path P and arriving at a turning point of P. u occupies a corner cell of P, and v occupies a cell of S and is only connected diagonally to u while not being adjacent to any cell occupied by L. L pushes u one position horizontally and turns all of its nodes vertically. 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.

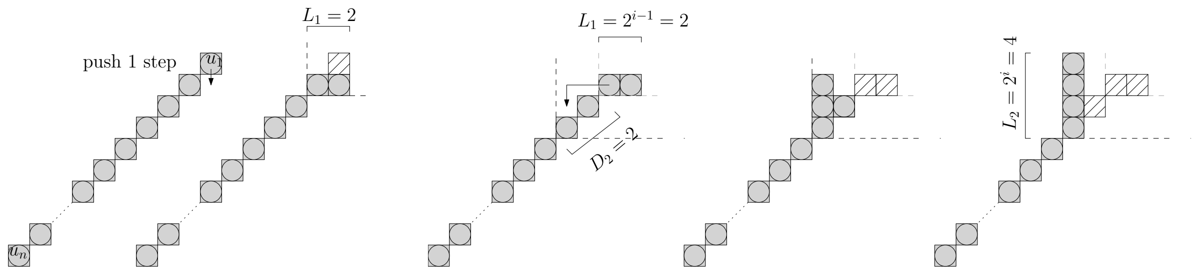
UniversalConnected. Given *any* pair of connected shapes (S_I, S_F) of the same order, where S_I is the initial shape and S_F the target shape, transform S_I into S_F while preserving connectivity throughout the transformation.

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

In this section, we present a strategy for HamiltonianConnected, called *HamiltonianToLine.* 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 *HamiltonianToLine* transformation and reversing the resulting transformation, one can then go from S_I to S_F in the same asymptotic time.

3.1. Transforming Diagonal shape into Line shape

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.

Figure 4: First and second phase of *HamiltonianToLine* transformation on the diagonal shape.

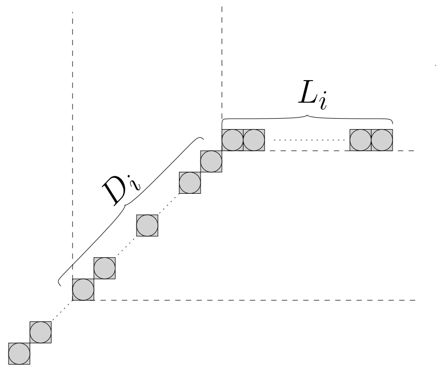
Let S_I be a diagonal of n nodes $u_n, u_{n-1}, \ldots, u_1$, occupying cells $(x, y), (x+1, y+1), \ldots, (x+n-1, y+n-1)$, respectively. Assume for simplicity of exposition that n is a power of 2; this can be dropped later. As argued above, it is sufficient to show how S_I can be transformed into a 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 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_2 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).

3.2. Transforming Hamiltonian shapes into Line shape

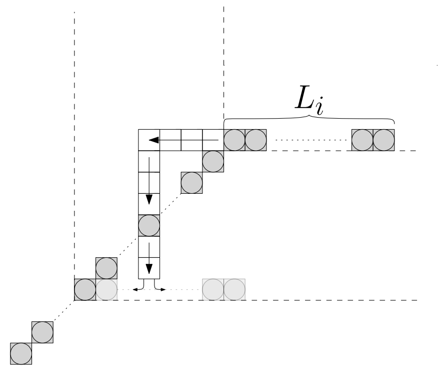
In any phase i, for all $1 \leq i \leq \log n$, a line L_i occupies 2^i consecutive cells in a terminal subset of S_I (see an example in Figure 5 (a)). L_i moves through a shortest path towards the far endpoint of the next diagonal segment D_i of length 2^i (Figure 5 (b)). Note that for general shapes, 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)). Observe that connectivity might be broken as L'_i moving up and L_i pushing left in Figure 5 (d); hence, this case can be resolved in many ways, such as Figure 3 in Proposition 1. 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.

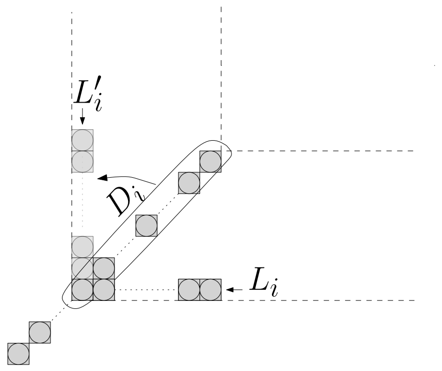
LineWalk. At the beginning of any phase i, there is a terminal straight line L_i of length 2^i containing the nodes $v_1, \ldots, v_{2^i}$, which is connected to an $S_i \subseteq S_I$, such that S_i consists of the 2^i subsequent nodes, that is $v_{2^i+1}, \ldots, v_{2^{i+1}}$. Observe that S_i is the 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



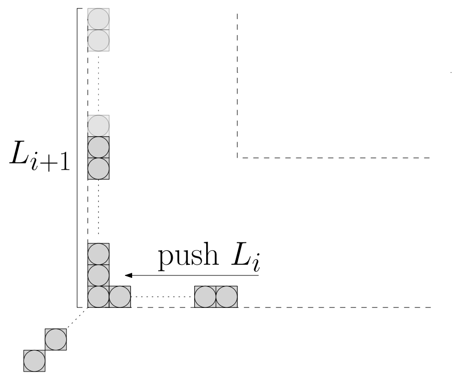
(a) A line L_i and a diagonal segment D_i both of length 2^i.



(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.

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

(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: $S \leftarrow S_I$ 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 $S_L$
```

3.3. Correctness and runtime analysis

Now we are ready to show correctness of *HamiltonianToLine* transformation, which is capable of transforming any Hamiltonian shape $S \in \mathcal{H}$ into a line shape of the same order, while preserving connectivity during its course. Next, we show that the strategy takes a total of $O(n \log n)$ moves to complete the transformation.

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. $\square$

Lemma 2. *The operation of Line-Walk preserves the whole connectivity of the shape during phase i, where $1 \leq i \leq \log n$.*

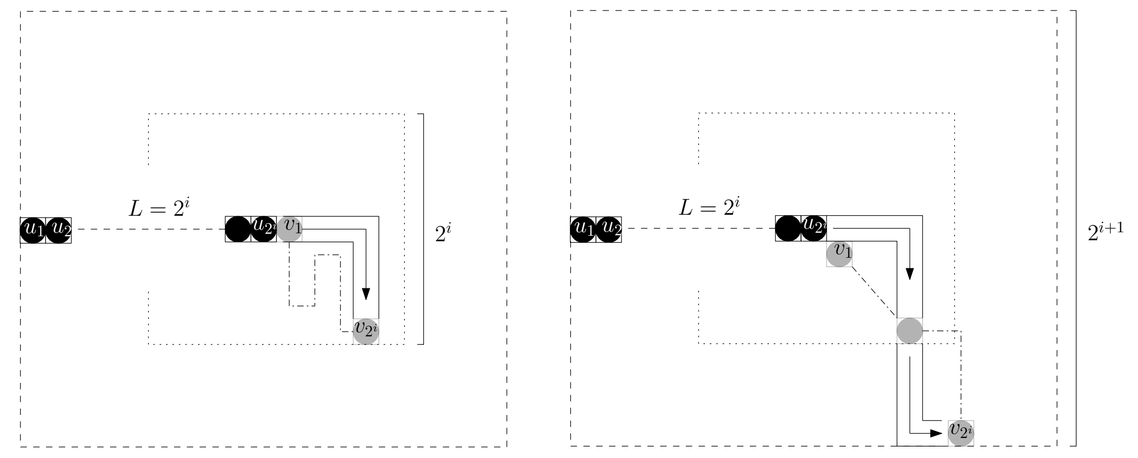
Proof. Let $S_I \in \mathcal{H}$ be a Hamiltonian shape of order n in phase i, which terminates at a straight line L_i of length 2^i nodes, starting from v_1 to v_{2^i}. During phase i, this transformation doubles the size of L_i by merging its nodes with the subsequent 2^i nodes on the Hamiltonian path 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° *clockwise*). 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 to form L_{i+1}, 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 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. Due to symmetry, assume L_i is horizontal and occupies $(x, y), (x + 1, y), \ldots, (x + 2^i, y)$ and S_i is the next terminal sub-path of the remaining Hamiltonian path. Then, the Manhattan distance between $v_{2^i} = u$ and $v_{2^{i+1}} = v$, $\delta(u, v) = |u_x - v_x| + |u_y - v_y|$, determines the path that the line L_i will go through in order to reach the far endpoint of S_i. There are two possible paths of a single move from u to v. The first path starts horizontally from cell (u_x, u_y) then turns at (v_x, u_y) continuing vertically towards (v_x, v_y), and the second one starts from (u_x, u_y) then turns at (u_x, v_y) continuing horizontally towards (v_x, v_y).

In case (3), the distance between v_{2^i} and $v_{2^{i+1}}$ is $\delta(v_{2^i}, v_{2^{i+1}}) \leq 2^i$, thus L_i moves horizontally from $v_{2^i} = (x, y)$ through (x', y) at which L_i changes its direction towards $v_{2^{i+1}} = (x', y')$. In a worst-case configuration, a path may consist of at least 2^i empty cells L_i goes through to reach the destination cell (x', y'). Recall that L_i already consists of 2^i nodes, which guarantees connectivity all the way until arriving at (x', y'). Once L_i has arrived there, it can safely change its direction to line up with $v_{2^{i+1}}$ and occupy the column x', while preserving connectivity. Further, any non-empty cells of the path are eventually restored due to the *transparency* of line moves shown in Proposition 1.

Finally, the same argument holds for (4) by applying (3) twice. Figure 6 shows an example of case (3) and (4). Thus, *Line-Walk* always keeps the whole shape connected during any phase i of the transformation. □



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

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

Figure 6: Two cases of *Line-Walk* operation.

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

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, while preserving connectivity during the course of the transformation.*

Proof. The bound $O(n)$ trivially holds for case (1) and (2). We then analyse a worst-case of (3) and (4) in which the transformation matches its maximum running time. Let T_i denote the total number of moves from phase 1 up to i for all $0 \leq i \leq \log n$. 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 paying a cost of at most $2^{i+2} - 4$ moves. There is an additive factor of 2 for the special-case when L_i is turning at a non-empty corner as depicted in Figure 3. Then the operation of *Line-Walk* takes total moves k'_i of at most:

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

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 k''_i of at most:

$$k''_i = 2(2^i - 1),$$

moves. Hence, the operation of *Line-Walk* and combination of L_i and L'_i requires at most k_i in phase i given by:

$$\begin{aligned} k_i &= k'_i + k''_i \\ &= 6(2^i - 1) + 2(2^i - 1) = 8(2^i - 1) \\ &\approx (2^i - 1) \\ &\leq O(2^i). \end{aligned}$$

Now, let T_{i-1} denote a total number of moves for a recursive call of HamiltonianToLine transformation on S_i (of 2^i nodes) to transform it into a straight line L'_i, then the transformation in phase i requires at most:

$$T_i = 2 \cdot T_{i-1} + k_i,$$

moves. Given that the first phase i costs $T_1 = 1$, we compute the recursion as follows:

$$\begin{aligned}
T_i &= 2 \cdot T_{i-1} + k_i = 2T_{i-1} + 2^i \\
&= 2(2T_{i-2} + 2^{i-1}) + 2^i = 4 \cdot T_{i-2} + 2\frac{2^i}{2} + 2^i = 2^2 \cdot T_{i-2} + 2 \cdot 2^i \\
&= 4(2 \cdot T_{i-3} + 2^{i-2}) + 2 \cdot 2^i = 8 \cdot T_{i-3} + 4\frac{2^i}{4} + 2 \cdot 2^i = 2^3 \cdot T_{i-3} + 3 \cdot 2^i \\
&= \dots \\
&= 2^i \cdot T_1 + i \cdot n \\
&= 2^i + i \cdot n.
\end{aligned}$$

Therefore, phase i of the transformation costs at most:

$$T_i \leq O(2^i + i \cdot n).$$

Thus in the final phase $i = \log n$, we conclude that HamiltonianToLine transformation performs a total cost $T_{\log n}$ of at most:

$$\begin{aligned}
T_{\log n} &\leq 2^{\log n} + n \log n \\
&= n + n \log n \\
&= O(n \log n).
\end{aligned}$$

□

By Lemmas 2 and 3, any Hamiltonian shape $S \in \mathcal{H}$ can be transformed into a straight line $S_L \in \mathcal{H}$ of the same order within $O(n \log n)$ moves, while preserving connectivity. By reversibility of line moves [34], any pair of Hamiltonian shapes $S_I, S_F \in \mathcal{H}$ of the same order can be transformed to each other by first transforming S_I into S_L and then reversing the transformation of S_F into S_L, within the same asymptotic time of $O(n \log n)$ moves. Thus we have arrived at the following theorem:

Theorem 1. *For any pair of Hamiltonian shapes $S_I, S_F \in \mathcal{H}$ of the same order n, S_I can be transformed into S_F (and S_F into S_I) in $O(n \log n)$ moves, while preserving connectivity of the shape during the course of the transformation.*

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 Compress 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 [34]) 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.

4.1. Universal transformation by compression approach

Let $C_l \subseteq S_I$ be a leaf component containing nodes $v_1, \dots, v_k$ 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 (for an example see Figure 7), 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 7 (d)) and then to transfer them completely into B_p (Figures 7(e)). 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 (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.

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 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.

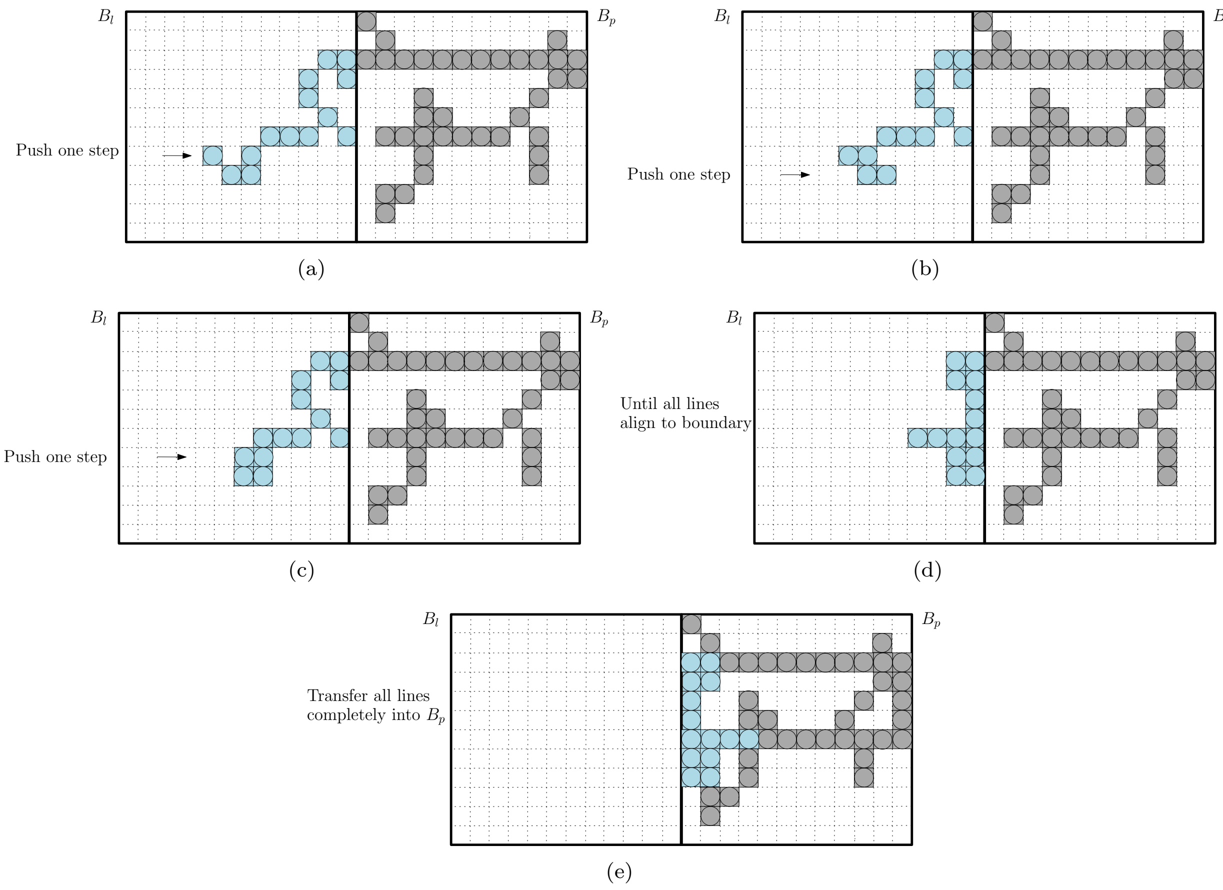

B_l
B_p
Push one step
Push one step
Push one step
Until all lines align to boundary
Transfer all lines completely into B_p
(a)
(b)
(c)
(d)
(e)

Figure 7: A leaf component C_l in blue compressing from the left sub-box B_l towards its parent component C_p in black inside a horizontal adjacent right sub-box B_p. C_l first pushes all lines to align all of them perpendicularly to the boundary between B_l and B_p then compresses into B_p. All other orientations are symmetric and follow by rotating the shape 90°, 180° or 270° clockwise.

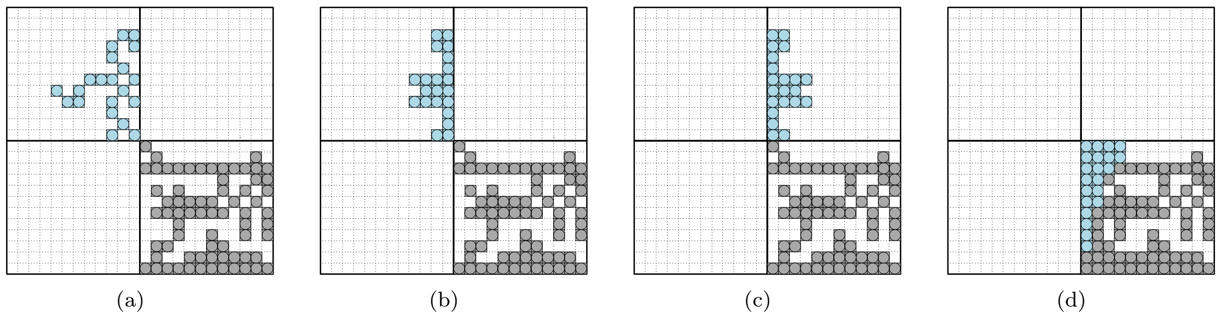


Figure 8: A leaf component C_l in blue compressing from the top-left sub-box towards its parent component C_p in black inside a diagonal adjacent bottom-right sub-box. C_l compresses first horizontally towards an intermediate top-right sub-box, then vertically into the bottom-right. All other orientations are symmetric and follow by rotating the shape 90°, 180° or 270° clockwise.

4.2. Correctness and runtime analysis

In this section, we prove correctness of Compress transformation. First, we show that the it is able to transform any pair of connected shapes (S_I, S_F) of the same order to each other, while preserving connectivity during its course. We then discuss its running time, which takes total cost of at most $O(n\sqrt{n})$ moves to compress any pair connected shapes having the same number of n nodes.

Given an initial connected shape S_I holding n nodes enclosed into an $n \times n$ square that is divided into $\sqrt{n} \times \sqrt{n}$ square sub-boxes, we provide the following definitions that are used in the rest of this section.

Definition 3 (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 $u, v \in S_I$, such that u occupies B_1 and v occupies B_2 where u and v are two adjacent neighbours connected vertically, horizontally or diagonally.*

Definition 4 (Connectivity of components)**.** *By the above partitioning, two connected components, C_1 and C_2, 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.*

Then we show that each sub-box holds at most $2\sqrt{n}$ components.

Lemma 4. *Any $\sqrt{n} \times \sqrt{n}$ square box can contain 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 $\sqrt{n}/2$ cells at a length-$\sqrt{n}$ boundary of B_l, resulting in $2\sqrt{n}$ for the four boundaries, see Figure 9. Hence, it can contain at most $2\sqrt{n}$ disconnected components. ☐

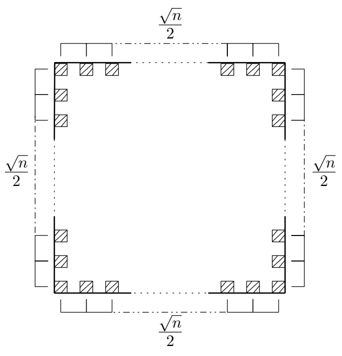


Figure 9: A square box of four length-$\sqrt{n}$ boundaries, each of $2\sqrt{n}$ cells.

Then, the following lemma proves that any connected shape S of n nodes can be compressed into a square box of dimension $\sqrt{n}$.

Lemma 5. *Let S 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. The number of cells inside any sub-box is $\sqrt{n} \times \sqrt{n} = n$, then it is sufficient to be filled by at most n nodes. ☐

Next, the following lemma shows that Compress transformation eventually forms a nice shape, belongs to a family of connected shapes introduced in [34] and can be transformed fast into a straight line in linear time.

Lemma 6. *Starting from an initial connected shape S_I of order n,* Compress *transformation eventually forms a nice shape $S_{\mathcal{NICE}}$ of order n.*

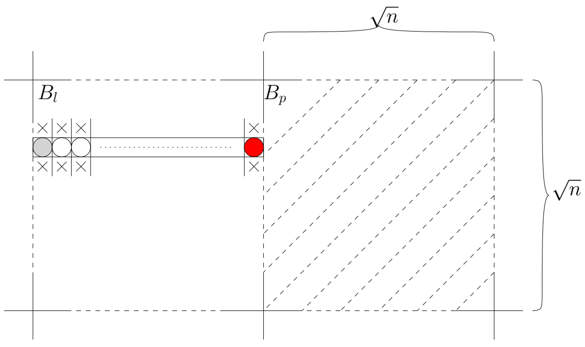
Proof. Regardless of which sub-box the shape will compress into, the resulting final shape will form a square of size $\sqrt{n}$, which satisfies all conditions of nice shapes. ☐

Given an initial connected shape S_I of n nodes with an associated graph $G(S_I)$, compute a spanning tree T of $G(S_I)$. Then enclose S_I 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 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. We then provide the following lemma:

Lemma 7. Compress *transformation compresses a leaf component* $C_l \subseteq S_I$ *of* $k \geq 1$ *nodes, while preserving the global connectivity of the shape.*

Proof. We shall discuss all possible cases of compressing all lines $k \in C_l$ from B_l towards B_p vertically, horizontally and diagonally, where $1 \leq k \leq \sqrt{n}$. Due to symmetry, we present only one direction as all other directions hold by rotating the shape 90°, 180° and 270° clockwise. Assume B_l is a left sub-box horizontally adjacent to right sub-box B_p. All k horizontal lines (rows) of C_l start to move towards B_p sequentially one after the other, starting from the furthest line from the boundary between B_l and B_p. Given a single line $l \in k$ of length i, $1 \leq i \leq \sqrt{n}$, push l horizontally from B_l into B_p follows one of these cases:

- **Case 1.** A line l of length $\sqrt{n}$ occupies cells $(x, y), \ldots, (x + \sqrt{n}, y)$ starting from the left to right boundary of B_l. Then l moves one step right to occupy $(x+1, y), \ldots, (x + \sqrt{n} + 1, y)$ in a way similar to a simple position permutation of l's nodes to their right neighbour positions. Regardless of the shape configuration, l consequently creates an empty cell at $(x + 1, y)$ and at the same time stays connected to any other nodes occupying cells $(x, y \pm 1), \ldots, (x + \sqrt{n}, y \pm 1)$. Therefore, connectivity is preserved in this case. See an example in Figure 10 (a) and (b).



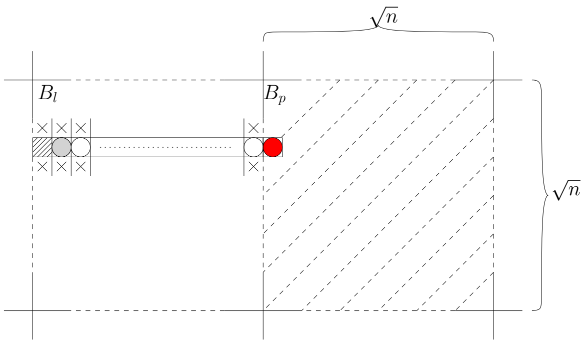


Figure 10: A line l of length $\sqrt{n}$ occupies the whole dimension of a sub-box in (a) is pushing one step right in (b).

- **Case 2.** A line l of length less than $\sqrt{n}$. It is similar of **Case 1** in which l moves one position right; thus connectivity is still preserved. See Figure 11.

- **Case 3.** Two horizontal lines l_1 and l_2, both of length less than $\sqrt{n}$, are occupying the same row separated by one empty cell. Say l_1 starts from the leftmost column x and ends at $x + i$, empty cell at $(x + i + 1, y)$ and l_2 occupies $(x + i + 2, y), \ldots, (x + \sqrt{n}, y)$. This is similar of **Case 2** in which l_1 pushes one step towards cell $(x + i + 1, y)$, then both lines combines into a single line creating a new empty cell at (x, y), see an example in Figure 12. Still, connectivity is preserved in this case.

As mentioned earlier, when a leaf component C_l occupying B_l compresses towards its parent C_p occupying B_p, we now show that no line exceeds the boundary of B_p while preserving connectivity.

- **Case 4.** A line l of length $i < \sqrt{n}$, starting from the left boundary and ending at cell $x + i$ of B_p, is adjacent to an empty cell to the right at $(x + i + 1, y)$. Once l is pushed one move right, it fills in that

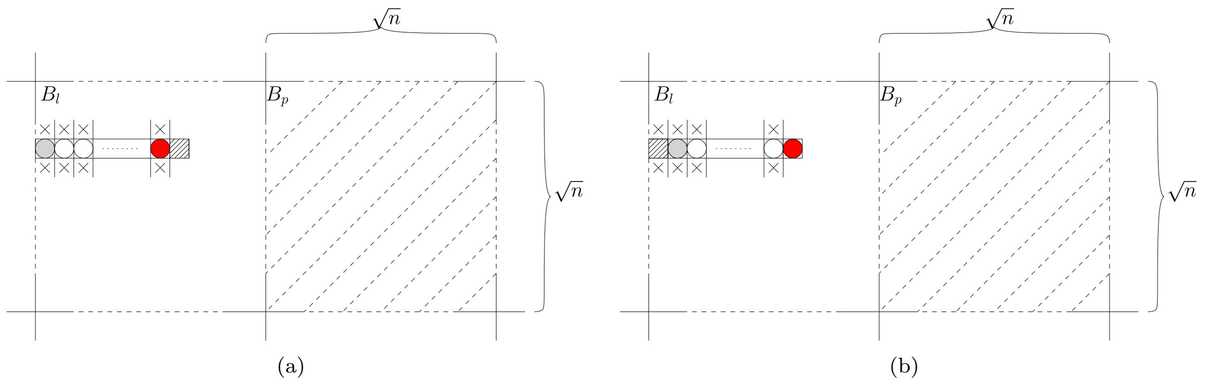


Figure 11: A line l of length $i < \sqrt{n}$ in (a) is pushing one step right in (b).

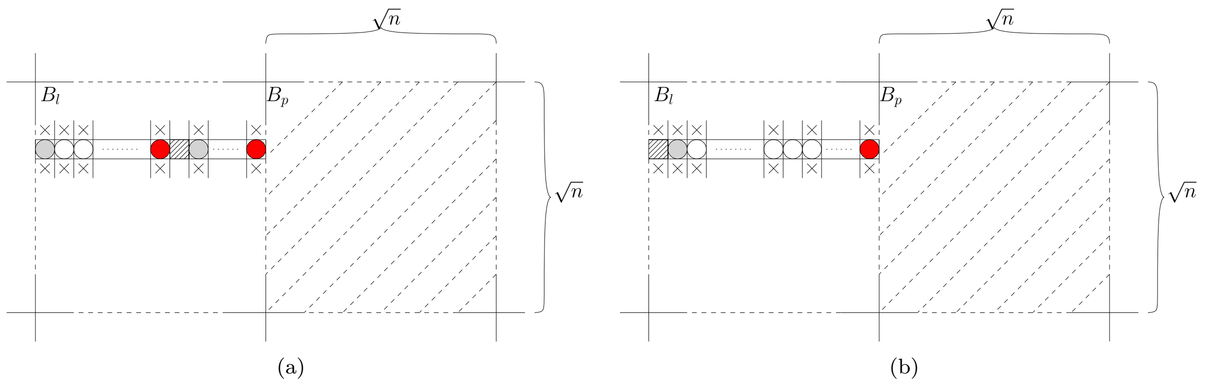


Figure 12: (a) Two lines occupy a row separated by one empty cell inside B_l, both of length less than $\sqrt{n}$. The left line is pushing one step right, then both combined into a single line in (b).

empty cell and occupies positions $(x+1, y), \dots, (x+i+1, y)$ with length $i+1$. This is similar of Case 2 in Figure 11.

- **Case 5.** A line l of length $\sqrt{n}$ starts from the left to right boundary of B_p. Once l is pushed towards the right, it turns to fill in empty cells at the right boundary of B_p, starting from the rightmost column to the left. Figures 13 and 14 depicts two different examples of filling B_p boundary. This case also preserves connectivity of the whole shape.

In all above cases, the horizontal line l pushes one move towards the right while preserving connectivity of S_I. As an immediate observation: whenever a line $l \subset S_I$ inside a sub-box B_l can push one move towards the boundary between (B_l, B_p) while the global connectivity of the whole shape is preserved. This holds also for all k lines that are moving one step from B_l towards B_p, sequentially one after another at any order, starting from the furthest-to-nearest line from B_p. Thus, this must hold for any finite number of line moves required to compress a leaf component C_l towards its parent C_p. □

Now, let us analyse the running time of this transformation. The compression cost could be very low taking only one move or very high up to linear moves in some cases. To simplify the analysis, we divide the total cost into charging phases. We then manage to upper bound the cost of each charging phase independently of the sequential order of compression. Below we provide a rough upper bound for all possible shape configurations.

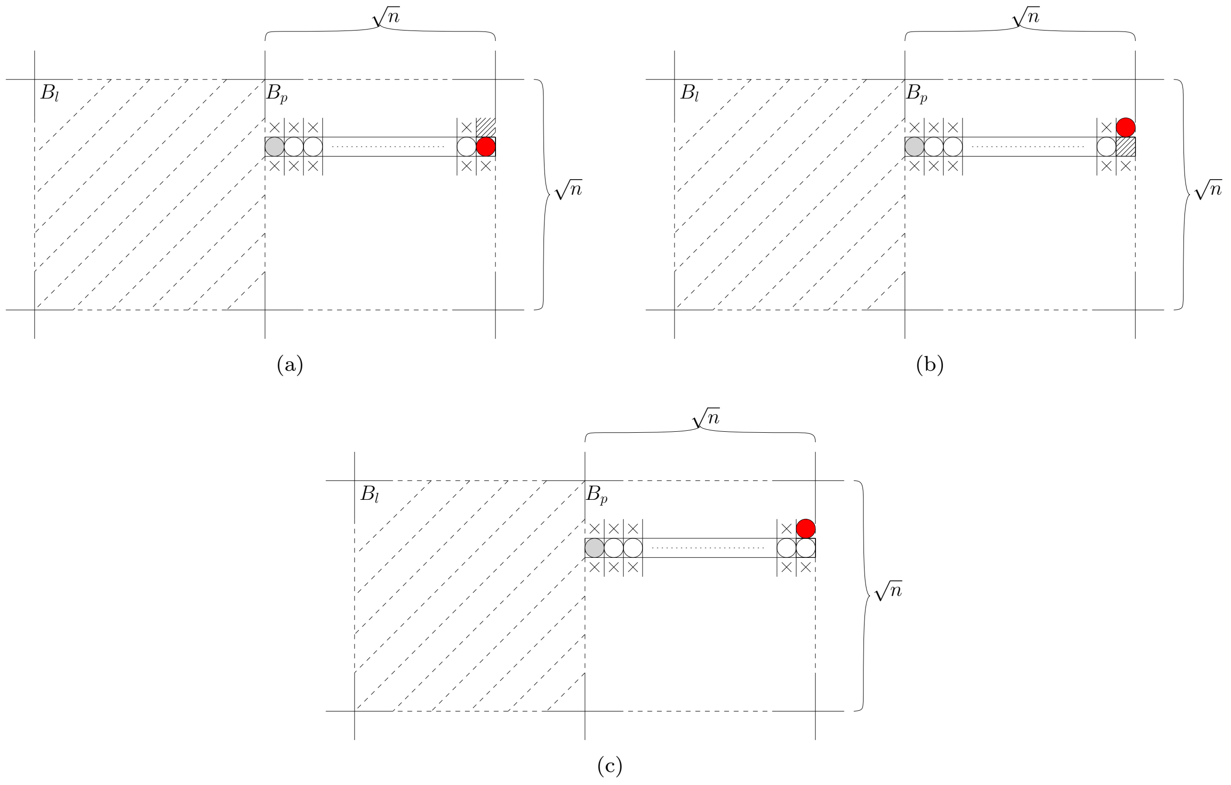


Figure 13: (a) A line l of length $\sqrt{n}$ occupies a whole dimension of a parent sub-box B_p where there is an empty cell above its right end node. As l pushing right, the end node moves up towards that empty cell (b), then in (c) l moves one step right.

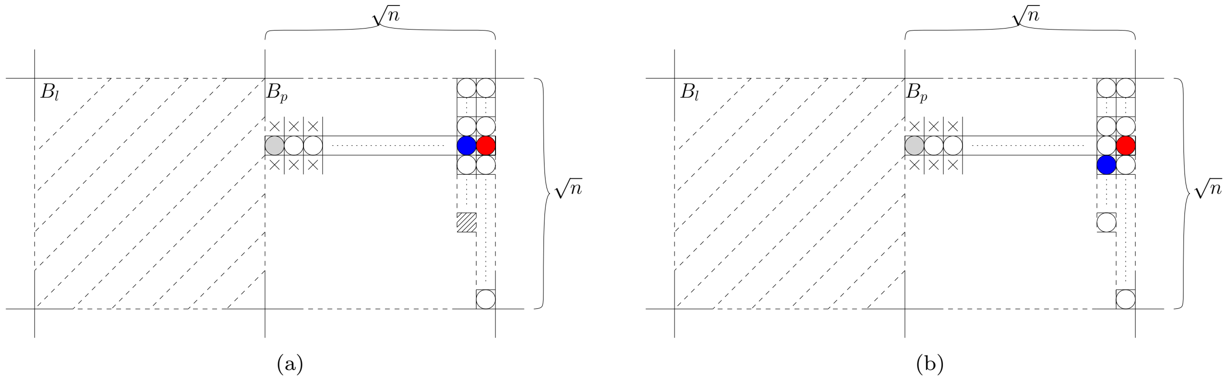


Figure 14: (a) A line l of length $\sqrt{n}$ occupies a whole dimension of B_p in which the rightmost column is fully occupied. In this case, l fills in the nearest empty cell at the second column from the right (b).

Given a uniform partitioning of any initial connected shape S_1 of order n, we first show the total cost required to compress a leaf component C_l inside B_l into a parent C_p occupying an adjacent sub-box B_p.

Lemma 8. *Given a pair of components C_l and C_p of k_l and k_p nodes, $1 \leq k_l + k_p \leq n$, occupying two adjacent sub-boxes B_l and B_p of size $\sqrt{n} \times \sqrt{n}$ each, receptively. Then, C_l requires at most $O(n)$ moves to compress into C_p inside B_p, while preserving connectivity.*

Proof. Assume a worst-case configuration when B_l and B_p are connected diagonally. In this case, the compression needs to go through an intermediate sub-box B_m first then carry on towards B_p. Each line $k_l \in C_l$ moves at most distance $\sqrt{n}$ to cross the boundary between B_l and B_p; hence, all k_l lines need to pay at most n moves to completely transfer into B_m and another n moves to compress into B_p. Further, we give an extra $2n$ moves for filling in a boundary at B_p (as depicted in Figure 14). Thus, the transformation pays at most t moves to compress C_l into C_p; that is,

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

□

Given some partitionings, there is a family of connected shapes that can be divided into n connected components deployed on $\sqrt{n}$ sub-boxes. This family seems to achieve a worst-case complexity where Compress transformation 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, which is the distance between the two furthest nodes of the shape, is in a maximum length of n. Unlike other dense connected shapes of shorter diameters, the compression cost of these shapes can be very high due to the lack of long lines which requires additional work for individuals and short lines. We provide this observation which closely follows Corollary 1 from [34].

Observation 1. *Given a connected shape S of order n enclosed in a square box of dimension n that is divided by some uniform partitioning into $\sqrt{n}$ sub-boxes of dimension $\sqrt{n}$, then there are a finite number of connected shapes denoted $\mathcal{C}$, such that an instance $S \in \mathcal{C}$ can have n components occupying the $\sqrt{n}$ sub-boxes.*

Figure 15 and 16 show partitioning examples of two shapes belonging to $\mathcal{C}$. Since $S \in \mathcal{C}$ is connected, each occupied sub-box contains at most $O(\sqrt{n})$ connected components of size 1 each.

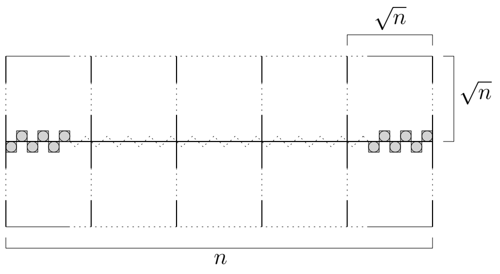


Figure 15: A zigzag line with a partitioning positioned to cross the middle through every two nodes.

Then, we aim to upper bound the cost that any connected shape S pays at most to compress based on the number of occupied sub-boxes, apart from of the sequential order of the transformation.

Lemma 9. Compress *transformation compresses any connected shape S of order n into a $\sqrt{n} \times \sqrt{n}$ square shape, in $O(n\sqrt{n})$ moves while preserving connectivity during its course.*

Proof. We analyse the compression cost of these shapes based a worst-case scenario. To simplify the analysis, we divide the total cost T into $\sqrt{n}$ charging phases $t_1, \ldots t_{\sqrt{n}}$, where each phase corresponds to an occupied sub-box. Then we upper bound the cost in each phase independently of the compression order. In any charging phase t_i, for all $1 \leq i \leq \sqrt{n}$, the strategy compresses at most $O(\sqrt{n})$ lines distance of $O(\sqrt{n})$, incurring a cost of n moves, while preserving connectivity. In a worst case (see Lemma 8), the compression may go through diagonal sub-boxes occurring at most $2n$ moves and pay an additional cost of $2n$ moves for boundary rearrangements. Thus, the cost phase t_i is bounded by:

$$t_i = 4n,$$

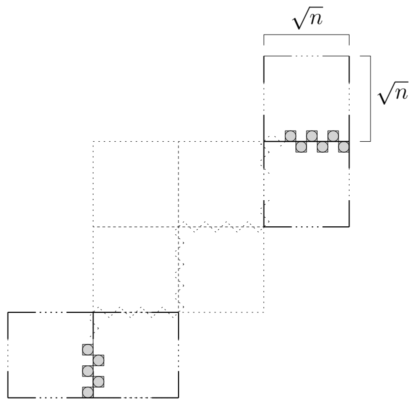


Figure 16: A diagonal zigzag line with a partitioning positioned to cross the middle through every two nodes.

moves, which is mostly sufficient for an occupied sub-box to be emptied of its lines during the transformation. Then, paying all $\sqrt{n}$ charging phases are sufficient for all $\sqrt{n}$ occupied sub-boxes to be emptied of all lines inside them over the transformation, in a maximum total cost T:

$$\begin{aligned} T &= 4n \cdot \sqrt{n} = 4(n\sqrt{n}) \\ &\leq O(n\sqrt{n}), \end{aligned}$$

moves. □

By Lemmas 7 and 9, any connected shape S of order n can be transformed into a nice shape $S_{\mathcal{NICE}}$ of the same order n within $O(n\sqrt{n})$ moves while preserving connectivity. By reversibility, any pair of connected shapes S_I and S_F of the same order can be transformed to each other within the same asymptotic time of $O(n\sqrt{n})$ moves by first transforming S_I into $S_{\mathcal{NICE}}$ and then reversing the transformation from S_F into $S_{\mathcal{NICE}}$. Finally, we have arrived at the following theorem:

Theorem 2. *For any pair of connected shapes S_I, S_F of the same order n, S_I can be transformed into S_F (and S_F into S_I) in $O(n\sqrt{n})$ moves, while preserving connectivity during the course of the transformation.*

5. Conclusions and Open Problems

We have presented efficient transformations for the line-pushing model introduced in [34]. 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. 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.

Further, we establish $\Omega(n \log n)$ lower bounds for two restricted sets of transformations, which have been shown in our full report [41]. 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 transformations 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.* [42] (cf. also [43]). Our second lower bound is also $\Omega(n \log n)$ time, for an alternative set of one-way transformations in which all nodes move towards only a one direction and do not spilt once they combined during the transformation.

These are the first lower bounds, under restrictions, for this model and are matching the best known $O(n \log n)$ upper bounds. 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.

References

[1] A. Almethen, O. Michail, I. Potapov, On efficient connectivity-preserving transformations in a grid, in: Algorithms for Sensor Systems - 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS, Vol. 12503, 2020, pp. 76–91.

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

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

[4] A. Knaian, K. Cheung, M. Lobovsky, A. Oines, P. Schmidt-Neilsen, N. 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, IEEE, 2012, pp. 1447–1453.

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

[6] S. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf, W. Shih, Self-assembly of dna into nanoscale three-dimensional shapes, Nature 459 (7245) (2009) 414.

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

[8] O. Michail, P. Spirakis, Elements of the theory of dynamic networks, Commun. ACM 61 (2) (2018) 72–81.

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

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

[11] A. Nguyen, L. Guibas, M. Yim, Controlled module density helps reconfiguration planning, in: Proc. of 4th International Workshop on Algorithmic Foundations of Robotics, 2000, pp. 23–36.

[12] J. Walter, J. Welch, N. Amato, Distributed reconfiguration of metamorphic robot chains, Distributed Computing 17 (2) (2004) 171–189.

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

[14] J. Czyzowicz, D. Dereniowski, A. Pelc, Building a nest by an automaton, in: M. A. Bender, O. Svensson, G. Herman (Eds.), 27th Annual European Symposium on Algorithms, ESA, 2019.

[15] S. Das, P. Flocchini, N. Santoro, M. Yamashita, Forming sequences of geometric patterns with oblivious mobile robots, Distributed Computing 28 (2) (2015) 131–145.

[16] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, Y. Yamauchi, Shape formation by programmable particles, Distributed Computing.

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

[18] G. Aloupis, N. Benbernou, M. Damian, E. Demaine, R. Flatland, J. Iacono, S. Wuhrer, Efficient reconfiguration of lattice-based modular robots, Computational geometry 46 (8) (2013) 917–928.

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

[20] J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A. Richa, C. Scheideler, T. Strothmann, On the runtime of universal coating for programmable matter, Natural Computing 17 (1) (2018) 81–96.

[21] Z. Derakhshandeh, R. Gmyr, A. Richa, C. Scheideler, T. Strothmann, Universal shape formation for programmable matter, in: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2016, pp. 289–299.

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

[23] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distributed Computing 18 (4) (2006) 235–253.

[24] D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, The computational power of population protocols, Distributed Computing 20 (4) (2007) 279–304.

[25] O. Michail, P. Spirakis, Simple and efficient local codes for distributed stable network construction, Distributed Computing 29 (3) (2016) 207–237.

[26] D. Doty, Theory of algorithmic self-assembly, Communications of the ACM 55 (2012) 78–88.

[27] P. Rothemund, E. Winfree, The program-size complexity of self-assembled squares, in: Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC), ACM, 2000, pp. 459–468.

[28] E. Winfree, Algorithmic self-assembly of dna, Ph.D. thesis, California Institute of Technology (June 1998).

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

[30] S. Fekete, A. Richa, K. Römer, C. Scheideler, Algorithmic foundations of programmable matter (Dagstuhl Seminar 16271), in: Dagstuhl Reports, Vol. 6, 2016, also in *ACM SIGACT News*, 48.2:87-94, 2017.

[31] A. Dumitrescu, J. Pach, Pushing squares around, in: Proceedings of the twentieth annual symposium on Computational geometry, ACM, 2004, pp. 116–123.

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

[33] H. Akitaya, E. Arkin, M. Damian, E. Demaine, V. Dujmovic, R. Flatland, M. Korman, B. Palop, I. Parada, A. van Renssen, V. Sacristán, Universal reconfiguration of facet-connected modular robots by pivots: The O(1) musketeers, in: 27th Annual European Symposium on Algorithms, ESA, Vol. 144 of LIPIcs, 2019, pp. 3:1–3:14.

[34] A. Almethen, O. Michail, I. Potapov, Pushing lines helps: Efficient universal centralised transformations for programmable matter, Theoretical Computer Science 830-831 (2020) 43 – 59.

[35] Z. Derakhshandeh, R. Gmyr, A. Porter, A. Richa, C. Scheideler, T. Strothmann, On the runtime of universal coating for programmable matter, in: International Conference on DNA-Based Computers, Springer, 2016, pp. 148–164.

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

[37] G. Aloupis, S. Collette, E. Demaine, S. Langerman, V. Sacristán, S. Wuhrer, Reconfiguration of cube-style modular robots using O(logn) parallel moves, in: International Symposium on Algorithms and Computation, Springer, 2008, pp. 342–353.

[38] S. Fekete, R. Gmyr, S. Hugo, P. Keldenich, C. Scheffer, A. Schmidt, Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata, CoRR abs/1810.06360.

[39] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, T. Strothmann, Forming tile shapes with simple robots, Natural Computing (2019) 1–16.

[40] A. Itai, C. Papadimitriou, J. Szwarcfiter, Hamilton paths in grid graphs, SIAM Journal on Computing 11 (4) (1982) 676–686.

[41] A. Almethen, O. Michail, I. Potapov, On efficient connectivity-preserving transformations in a grid, CoRR abs/2005.08351. `arXiv:abs/2005.08351`.

[42] S. Arora, P. Raghavan, S. Rao, Approximation schemes for euclidean k-medians and related problems, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 106–113.

[43] A. E. Clementi, M. Di Ianni, M. Lauria, A. Monti, G. Rossi, R. Silvestri, On the bounded-hop mst problem on random euclidean instances, Theoretical computer science 384 (2-3) (2007) 161–167.