

Pushing Lines Helps: Efficient Universal Centralised Transformations for Programmable Matter

Abdullah Almethen

Department of Computer Science, University of Liverpool, UK
A.almethen@liverpool.ac.uk

Othon Michail

Department of Computer Science, University of Liverpool, UK
Othon.Michail@liverpool.ac.uk

Igor Potapov

Department of Computer Science, University of Liverpool, UK
Potapov@liverpool.ac.uk

Abstract

In this paper, we study a discrete system of entities residing on a two-dimensional square grid. Each entity is modelled as a node occupying a distinct cell of the grid. The set of all n nodes forms initially a connected shape A . Entities are equipped with a linear-strength pushing mechanism that can push a whole line of entities, from 1 to n , in parallel in a single time-step. A target connected shape B is also provided and the goal is to *transform* A into B via a sequence of line movements. Existing models based on local movement of individual nodes, such as rotating or sliding a single node, can be shown to be special cases of the present model, therefore their (inefficient, $\Theta(n^2)$) *universal transformations* carry over. Our main goal is to investigate whether the parallelism inherent in this new type of movement can be exploited for efficient, i.e., sub-quadratic worst-case, transformations. As a first step towards this, we restrict attention solely to centralised transformations and leave the distributed case as a direction for future research. Our results are positive. By focusing on the apparently hard instance of transforming a diagonal A into a straight line B , we first obtain transformations of time $O(n\sqrt{n})$ without and with preserving the connectivity of the shape throughout the transformation. Then, we further improve by providing two $O(n \log n)$ -time transformations for this problem. By building upon these ideas, we first manage to develop an $O(n\sqrt{n})$ -time universal transformation. Our main result is then an $O(n \log n)$ -time universal transformation. We leave as an interesting open problem a suspected $\Omega(n \log n)$ -time lower bound.

Keywords and phrases Line movement, programmable matter, transformation, shape formation, reconfigurable robotics, time complexity

1 Introduction

As a result of recent advances in components such as micro-sensors, electromechanical actuators, and micro-controllers, a number of interesting systems are now within reach. A prominent type of such systems concerns collections of small robotic entities. Each individual robot is equipped with a number of actuation/sensing/communication/computation components that provide it with some autonomy; for instance, the ability to move locally and to communicate with neighbouring robots. Still, individual local dynamics are uninteresting, and individual computations are restricted due to limited computational power, resources, and knowledge. What makes these systems interesting is the collective complexity of the population of devices. A number of fascinating recent developments in this direction have demonstrated the feasibility and potential of such collective robotic systems, where the scale can range from milli/micro [6, 26, 28, 36, 43] down to nano [20, 35].

This progress has motivated the parallel development of a theory of such systems. It



© Abdullah Almethen, Othon Michail and Igor Potapov;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

has been already highlighted [32] that a formal theory (including modelling, algorithms, and computability/complexity) is necessary for further progress in systems. This is because theory can accurately predict the most promising designs, suggest new ways to optimise them, by identifying the crucial parameters and the interplay between them, and provide with those (centralised or distributed) algorithmic solutions that are best suited for each given design and task, coupled with provable guarantees on their performance. As a result, a number of sub-areas of theoretical computer science have emerged such as mobile and reconfigurable robotics [1, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 25, 29, 30, 37, 41, 42, 43], passively-mobile systems [3, 4, 31, 32] including the theory of DNA self-assembly [19, 34, 39, 40], and metamorphic systems [21, 22, 23, 33, 38]; connections are even evident with the theory of puzzles [5, 13, 27]. A latest ongoing effort is to join these theoretical forces and developments within the emerging area of “Algorithmic Foundations of Programmable Matter” [24]. *Programmable matter* refers to any type of matter that can *algorithmically* change its physical properties. “Algorithmically” means that the change (or *transformation*) is the result of executing an *underlying program*.

In this paper, we embark from the model studied in [21, 22, 23, 30], in which a number of spherical devices are given in the form of a (typically connected) shape A lying on a two-dimensional square grid, and the goal is to transform A into a desired target shape B via a sequence of valid movements of individual devices. In those papers, the considered mechanisms were the ability to rotate and slide a device over neighbouring devices (always through empty space). We here consider an alternative (linear-strength) mechanism, by which a line of one or more devices can translate by one position in a single time-step.

As our main goal is to determine whether the new movement under consideration can *in principle* be exploited for sub-quadratic worst-case transformations, we naturally restrict our attention to centralised transformations. We generally allow the transformations to break connectivity, even though we also develop some connectivity-preserving transformations on the way. Our main result is a universal transformation of $O(n \log n)$ worst-case running time that is permitted to break connectivity. Distributed transformations and connectivity-preserving universal transformations are left as interesting future research directions.

1.1 Our Approach

In [30], it was proved that if the devices (called *nodes* from now on) are equipped only with a rotation mechanism, then the decision problem of transforming a connected shape A into a connected shape B is in \mathbf{P} , and a constructive characterisation of the (rich) class of pairs of shapes that are transformable to each other was given. In the case of combined availability of rotation and sliding, universality has been shown [21, 30], that is, any pair of connected shapes are transformable to each other. Still, in these and related models, where in any time step at most one node can move a single position in its local neighbourhood, it can be proved (see, for instance, [30]) that there will be pairs of shapes that require $\Omega(n^2)$ steps to be transformed to each other. This follows directly from the inherent “distance” between the two shapes and the fact that this distance can be reduced by only a constant in every time step. An immediate question is then “*How can we come up with more efficient transformations?*”

Two main alternatives have been explored in the literature in an attempt to answer this question. One is to consider parallel time, meaning that the transformation algorithm can move more than one node (up to a linear number of nodes if possible) in a single time step. This is particularly natural and fair for distributed transformations, as it allows all nodes to have their chances to take a movement in every given time-step. For example,

such transformations based on pipelining [23, 30], where essentially the shape transforms by moving nodes in parallel around its perimeter, can be shown to require $O(n)$ parallel time in the worst case and this technique has also been applied in systems (e.g., [36]).

The other approach is to consider more powerful actuation mechanisms, that have the potential to reduce the inherent distance faster than a constant per sequential time-step. These are typically mechanisms where the local actuation has strength higher than a constant. This is different than the above parallel-time transformations, in which local actuation can only move a single node one position in its local neighbourhood and the combined effect of many such movements at the same time is exploited. In contrast, in higher-strength mechanisms, it is a single actuation that has enough strength to move many nodes at the same time. Prominent examples in the literature are the linear-strength models of Aloupis *et al.* [1, 2], in which nodes are equipped with extend/contract arms, each having the strength to extend/contract the whole shape as a result of applying such an operation to one of its neighbours and of Woods *et al.* [40], in which a whole line of nodes can rotate around a single node (acting as a linear-strength rotating arm). The present paper follows this approach, by introducing and investigating a linear-strength model in which a node can push a line of consecutive nodes one position (towards an empty cell) in a single time-step.

In terms of transformability, our model can easily simulate the combined rotation and sliding mechanisms of [21, 30] by restricting movements to lines of length 1 (i.e., individual nodes). It follows that this model is also capable of universal transformations, with a time complexity at most twice the worst-case of those models, i.e., again $O(n^2)$. Naturally, our focus is set on exploring ways to exploit the parallelism inherent in moving lines of larger length in order to speed-up transformations and, if possible, to come up with a more efficient in the worst case universal transformation.

As reversibility of movements is still valid for any line movement in our model, we adopt the approach of transforming any given shape A into a spanning line L (vertical or horizontal). This is convenient, because if one shows that any shape A can transform fast into a line L , then any pair of shapes A and B can then be transformed fast to each other by first transforming fast A into L and then L into B by reversing the fast transformation of B into L .

We start this investigation by identifying the diagonal shape D (which is considered connected in our model and is very similar to the staircase worst-case shape of [30]) as a potential worst-case initial shape to be transformed into a line L . This intuition is supported by the $O(n^2)$ individual node distance between the two shapes and by the initial unavailability of long lines: the transformation may move long lines whenever available, but has to pay first a number of movements of small lines in order to construct longer lines. In this benchmark (special) case, the trivial lower and upper bounds $\Omega(n)$ and $O(n^2)$, respectively, hold. Moreover, observe that a sequential gathering of the nodes starting from the top right and collecting the nodes one after the other into a snake-like line of increasing length is still quadratic, because, essentially, for each sub-trip from one collection to the next, the line has to make a “turn”, meaning to change both a row and a column, and in this model this costs a number of steps equal to the length of the line, that is, roughly, $1 + 2 + \dots + (n - 1) = \Theta(n^2)$ total time.

We first prove that by partitioning the diagonal into \sqrt{n} diagonal segments of length \sqrt{n} each, we can first transform each segment in time quadratic in its length into a straight line segment, then push all segments down to a “collection row” y_0 in time $O(n\sqrt{n})$ and finally re-orient all line segments to form a horizontal line in y_0 , paying a linear additive factor. Thus, this transformation takes total time $O(n\sqrt{n})$, which constitutes our first improvement

compared to the $\Omega(n^2)$ lower bound of [30]. We then take this algorithmic idea one step further, by developing two transformations building upon it, that can achieve the same time-bound while *preserving connectivity* throughout their course: one is based on *folding* segments and the other on *extending* them.

As the $O(\sqrt{n})$ length of uniform partitioning into segments is optimal for the above type of transformation, we turn our attention into different approaches, aiming at further reducing the running time of transformations. Allowing once more to break connectivity, we develop an alternative transformation based on *successive doubling*. The partitioning is again uniform for individual “phases”, but different phases have different partitioning length. The transformation starts from a minimal partitioning into $n/2$ lines of length 2, then matches them to the closest neighbours via shortest paths to obtain a partitioning into $n/4$ lines of length 4, and, continuing in the same way for $\log n$ phases, it maintains the invariant of having $n/2^i$ individual lines in each phase i , for $1 \leq i \leq \log n$. By proving that the cost of pairwise merging through shortest paths in each phase is linear in n , we obtain that this approach transforms the diagonal into a line in time $O(n \log n)$, thus yielding a substantial improvement. Observe that the problem of transforming the diagonal into a line seems to involve solving the same problem into smaller diagonal segments (in order to transform those into corresponding line segments). Then, one may naturally wonder whether a recursive approach could be applied in order to further reduce the running time. We provide a negative answer to this, for the special case of uniform recursion and at the same time obtain an alternative $O(n \log n)$ transformation for the diagonal-to-line problem.

Our final aim is on attempting to generalise the ideas developed for the above benchmark case in order to come up with *equally efficient universal transformations*. We successfully generalise both the $O(n\sqrt{n})$ and the $O(n \log n)$ approaches, obtaining universal transformations of worst-case running times $O(n\sqrt{n})$ and $O(n \log n)$, respectively. We achieve this by enclosing the initial shape into a square bounding box and then subdividing the box into square sub-boxes of appropriate dimension. For the $O(n\sqrt{n})$ bound, a single such partitioning into sub-boxes of dimension \sqrt{n} turns out to be sufficient. For the $O(n \log n)$ bound we again employ a successive doubling approach through phases of an increasing dimension of the sub-boxes, that is, through a new partitioning in each phase. Therefore, our ultimate theorem (followed by a constructive proof, providing the claimed transformation) states that: “*In this model, when connectivity need not necessarily be preserved during the transformation, any pair of connected shapes A and B can be transformed to each other in sequential time $O(n \log n)$* ”.

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

Section 2 brings together all definitions and basic facts that are used throughout the paper. In Section 3, we study the problem of transforming a diagonal shape into a line, without and with connectivity preservation. Section 4 presents our universal transformations. In Section 5, we conclude and discuss further research directions that are opened by our work.

2 Preliminaries and Definitions

The transformations considered here run on a two-dimensional square grid. Each cell of the grid possesses a unique location addressed by non-negative coordinates (x, y) , where x denotes columns and y indicates rows. A *shape* S is a set of n nodes on the grid, where each individual node $u \in S$ occupies a single cell $cell(u) = (x_u, y_u)$, therefore we may also refer to a node by the coordinates of the cell that it occupies at a given time. Two distinct nodes

Transformation	Problem	Running Time	Lower Bound
<i>DL-Partitioning</i>	Diagonal	$O(n\sqrt{n})$	$\Omega(n)$
<i>DL-Doubling</i>	Diagonal	$O(n \log n)$	$\Omega(n)$
<i>DL-Recursion</i>	Diagonal	$O(n \log n)$	$\Omega(n)$
<i>DLC-Folding</i>	Diagonal Connected	$O(n\sqrt{n})$	$\Omega(n)$
<i>DLC-Extending</i>	Diagonal Connected	$O(n\sqrt{n})$	$\Omega(n)$
<i>U-Box-Partitioning</i>	Universal	$O(n\sqrt{n})$	$\Omega(n)$
<i>U-Box-Doubling</i>	Universal	$O(n \log n)$	$\Omega(n)$

■ **Table 1** A summary of our transformations and their corresponding worst-case running times (the trivial lower bound is in all cases $\Omega(n)$). The Diagonal, Diagonal Connected, and Universal problems correspond to the `DIAGONALTOLINE`, `DIAGONALTOLINECONNECTED`, and `UNIVERSALTRANSFORMATION` problems, respectively (being formally defined in Section 2.2).

$(x_1, y_1), (x_2, y_2)$ are *neighbours* (or *adjacent*) iff $x_2 - 1 \leq x_1 \leq x_2 + 1$ and $y_2 - 1 \leq y_1 \leq y_2 + 1$ (i.e., their cells are adjacent vertically, horizontally or diagonally). A shape S is *connected* iff the graph defined by S and the above neighbouring relation on S is connected. Throughout, n denotes the number of nodes in a shape under consideration.

A line, $L \subseteq S$, is defined by one or more consecutive nodes in a column or row. That is, $L = (x_0, y_0), (x_1, y_1), \dots, (x_k, y_k)$, for $0 \leq k \leq n$, $k \in \mathbb{Z}$, is a line iff $x_0 = x_1 = \dots = x_k$ and $|y_k - y_0| = k$, or $y_0 = y_1 = \dots = y_k$ and $|x_k - x_0| = k$. A *line move*, is an operation by which all nodes of a line L move together in a single step, towards an empty *cell* adjacent to one of L 's endpoints. A line move may also be referred to as *step* (or *move* or *movement*) and time is discrete and measured in number of steps throughout. A move in this model is equivalent to choosing a node u and a direction $d \in \{\text{up}, \text{down}, \text{left}, \text{right}\}$ and moving u one position in direction d . This will additionally push by one position the whole line L of nodes in direction d , L (possibly empty) starting from a neighbour of u in d and ending at the first empty cell. More formally and in slightly different terms: A line $L = (x_1, y), (x_2, y), \dots, (x_k, y)$ of length k , where $1 \leq k \leq n$, can push all k nodes rightwards in a single step to positions $(x_2, y), (x_3, y), \dots, (x_{k+1}, y)$ iff there exists an empty cell to the right of L at (x_{k+1}, y) . The “down”, “left”, and “up” movements are defined symmetrically, by rotating the whole system 90° , 180° , and 270° clockwise, respectively.

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

1. *Horizontal. Can push all k nodes rightwards in a single step from $(x_1, y), (x_2, y), \dots, (x_k, y)$ to positions $(x_2, y), (x_3, y), \dots, (x_{k+1}, y)$ iff there exists an empty cell to the right of L at (x_{k+1}, y) . Similarly, it can push all k nodes to the left to occupy $(x_0, y), (x_1, y), \dots, (x_{k-1}, y)$, iff there exists an empty cell at (x_0, y) .*
2. *Vertical. Can push all k nodes upwards in a single step from $(x, y_1), (x, y_2), \dots, (x, y_k)$ into $(x, y_2), (x, y_3), \dots, (x, y_{k+1})$, iff there exists an empty cell above L at (x, y_{k+1}) . Similarly, it can push all k nodes down to occupy $(x, y_0), (x, y_1), \dots, (x, y_{k-1})$, iff there exists an empty cell below L at (x, y_0) .*

The following definitions from [30] shall be useful for our study. Let us first agree that we colour black any cell occupied by a node (as in Figure 1).

► **Definition 2.** *A hole H is a set of empty cells that are unoccupied and enclosed by nodes $u \in S$, where $H = \{h_i \mid h_i = (x_i, y_i), (x_i, y_i) \in \mathbb{Z}^2, i \geq 0\}$, such that every infinite spanning*

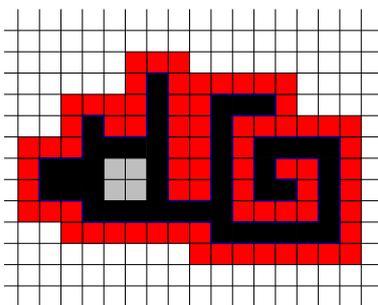
single path starting from the hole $h \in H$ and moving, only vertically and horizontally, certainly passes through a black cell of a node $u \in S$.

► **Definition 3.** A compact shape S' contains all cells of S and the hole H , such that S' has no holes, $S' = S \cup H$. For instance, the black and grey cells in Figure 1 are forming a connected compact shape S' .

► **Definition 4.** A perimeter (border) of S is defined as a polygon of unit length line segments, which surrounds the minimum-area of the interior of S' , such as the blue line in Figure 1.

► **Definition 5.** A surrounded layer of S is consisting of cells that are not occupied by nodes of S and contribute to the perimeter by at least one of its sides or corners (e.g the red cells in Figure 1). Normally, each cell in a 2D grid owns four line-segment sides and four corners.

► **Definition 6.** The external surface of S is another shape W , which includes all cells occupied by nodes of S and adjacent to the perimeter vertically or horizontally. The external surface of S is also a connected shape itself, and that is proved in [30].



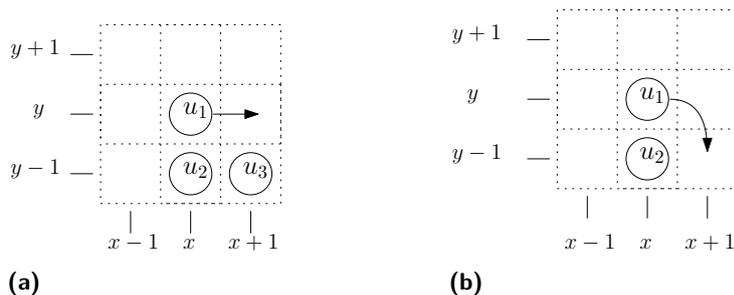
■ **Figure 1** All nodes of S occupy the black cells, while the grey cells define a hole. The blue line depicts the perimeter of S , and the surrounded layer is indicated by the red cells.

► **Proposition 7.** The surrounded layer of any connected shape S is always another connected shape.

Proof. It follows from [30]. Since S is connected, then the perimeter of S is connected too, and hence, forms a cycle. Each segment of the perimeter is contributed by two cells, belonging to the external surface and the surrounded layer. Now, if one walks on the perimeter (vertically or horizontally) or turns (left or right) clockwise or anticlockwise at any segment, one of the following cases will occur:

- (1) Pass through two adjacent vertical or horizontal cells on the surrounded layer and the external surface of S .
- (2) Stay put at the same position (cell) on the external surface and move through three neighbouring cells connected perpendicularly on the surrounded layer of S .
- (3) Stay put at the same position (cell) on the surrounded layer and pass through three neighbouring cells connected perpendicularly on the external surface of S .
- (4) Stay put at the same position (cell) on the surrounded layer and pass through two neighbouring cells connected diagonally on the external surface of S and one cell of the surrounded layer.

Subsequently, all cases above preserve connectedness of the surrounded layer and the external surface of S . ◀



■ **Figure 2** (a) An example of sliding u_1 over u_2 and u_3 to an empty cell to the left. (b) Rotate u_1 a 90° clockwise around u_2 .

As already mentioned, we know that there are related settings in which any pair of connected shapes A and B of the same order (“order” of a shape S meaning the number of nodes of S throughout the paper) can be transformed to each other¹ while preserving the connectivity throughout the course of the transformation.² This, for example, has been proved for the case in which the available movements to the nodes are rotation and sliding [21, 30]. We now show that the model of [21, 30] is a special case of our model, including universal transformations, are also valid transformations in the present model.

► **Proposition 8.** *The rotation and sliding model of [21, 30] is a special case of the present model.*

Proof. We establish a technique to prove that our model is capable of simulating *rotation and sliding models* of a two-dimension square grid appeared in [21, 30]. First, the sliding operation is equivalent in all those models, that is, if a node u is located at a cell of the grid, $u = (x, y)$, then u can slid right to any empty cell at $(x + 1, y + 1)$ over a horizontal line of length 2, such as in Figure 2 (a). The “down”, “left”, and “up” movements are defined symmetrically, by rotating the whole system 90° , 180° , and 270° clockwise, respectively. By Definition 1, our model is exploited *sliding* rule to push a line of 1 node into an adjacent empty cell, whether vertically or horizontally. For rotation, all mentioned models perform a single operation to rotate a node $u_1 = (x, y)$ around another $u_2 = (x, y - 1)$ by a 90° clockwise iff there exists two empty cells at $(x + 1, y)$ and $(x + 1, y + 1)$, see Figure 2(b). Analogously, this holds for all possible rotations by again rotating the whole system 90° , 180° , and 270° clockwise, respectively. Still, the rotation mechanism is also adopted by this model following Definition 1, and actually it costs twice for a single rotation to take place, compared with others. Subsequently, it implies that all transformations established there (with their running time at most doubled, including universal transformations, are also valid transformations in the present model). ◀

Consider a line L of k nodes occupying $(x_1, y), (x_2, y), \dots, (x_k, y)$ and empty cells at $(x_{k+1}, y), (x_{k+2}, y), \dots, (x_{k+o}, y)$ for all o , where $k = o \geq 1$. Now, we aim to transfer all k nodes to the right to fill in all k empty cells. As usual, any transformation of a single motion would move all k nodes in a total of $O(k^2)$ moves, since each k transfers a distance of $\Delta = k$. On the

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

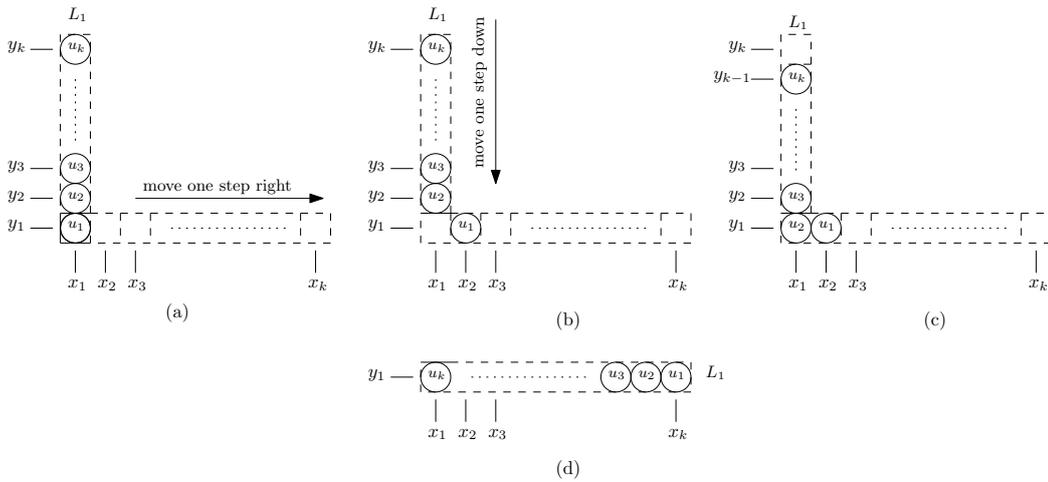
² In this paper, whenever transforming into a target shape B , we allow any placement of B on the grid, i.e., any shape B' obtained from B through a sequence of rotations and translations.

contrary, the k nodes move altogether in parallel $\Delta = k$ distance to occupy the empty cells in a total of $O(k)$ steps, by exploiting the linear-strength of this model. Now, we are ready to show a more beneficial property in the following lemma:

► **Lemma 9.** *The minimum number of line moves by which a line of length k , $1 \leq k \leq n$, can completely change its orientation³, is $2k - 2$.*

Proof. Assume a 2D square grid contains only a line L_1 of k nodes at $(x_1, y_1), \dots, (x_1, y_k)$ for all k , where $k \geq 1$, and empty cells on $(x_2, y_1), \dots, (x_k, y_1)$, as depicted in Figure 3. Now, the first bottommost node, $u_1 \in L_1$, moves one step right to occupy an empty cell on (x_1, y_1) . Consequently, a new empty cell is created at (x_1, y_1) , and the length of L_1 decreases by $k - 1$. By the linear-strength, L_1 pushes all $k - 1$ nodes down altogether in parallel in a single-time-move to occupy $(x_1, y_1), \dots, (x_1, y_{k-1})$. Hence, two line steps have been performed to move u_1 and push down all $k - 1$ nodes. Therefore, it implies that each of $k - 1 \in L_1$ requires two steps to transfer into the bottommost row, y_1 . Thus, we conclude that any line of length (k nodes) requires at most twice of its length, $2k - 2$, to change its orientation, which is bounded above by $O(k)$.

For the matching lower bound consider w.l.o.g. a horizontal line. A complete change of orientation requires the nodes of the line to occupy k consecutive rows (from just one originally). The bound follows by observing that in any two consecutive steps, at most one new row can become occupied. ◀

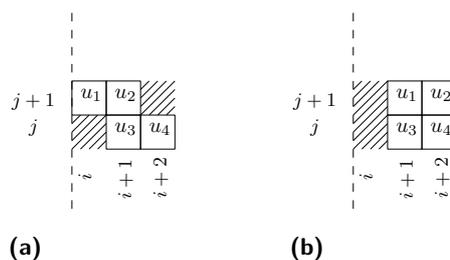


■ **Figure 3** A line of k nodes changes orientation by two consecutive steps per node. (a) Move u_1 one step to the right. (b) and (c) All $k - 1$ nodes push down altogether in single step. In (d), the line has finally transformed from vertical to horizontal after $2k$ steps.

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

► **Lemma 10 (Reversibility).** *Let (S_I, S_F) be a pair of connected shapes of the same number of nodes n . If $S_I \rightarrow S_F$ (“ \rightarrow ” denoting “can be transformed via a sequence of line movements”) then $S_F \rightarrow S_I$.*

³ From vertical to horizontal and vice versa.



■ **Figure 4** An example of a reversible line step.

Proof. First, we should prove that each single line step is reversible. Figure 4 (a) shows a simple connected shape of four nodes forming two horizontal and one vertical lines at $L_1 = \{u_1, u_2\}$, $L_2 = \{u_3, u_4\}$ and $L_3 = \{u_2, u_3\}$, respectively. Assume this configuration has no more space to the left, beyond the dashed line; therefore, L_1 moves one step to occupy the empty cell $(i + 2, j + 1)$. Now, all L_1 nodes are moving altogether to fill in positions (x_{i+1}, y_{j+1}) and (x_{i+2}, y_{j+1}) , as depicted in Figure 4 (b). Consequently, the previous line step creates another empty cell at (x_i, y_{j+1}) , which then gives the ability to L_1 to move back reversibly to fill in this new empty cell on the left. Thus, we conclude that any single line step is reversible, which implies that any finite sequence of line steps are reversible too.

Still, reversibility is valid for any line movement in our model by transforming any given shape S_I into a line S_L (vertical or horizontal). This is sufficient because if any shape S_I transforms into a line S_L , then any pair of shapes S_I and S_F can then be transformed to each other, via an intermediate connected shape S_L , by first transforming S_I into S_L and then S_L to S_F , by reversing the transformation of S_F to S_L . ◀

2.1 Nice shapes

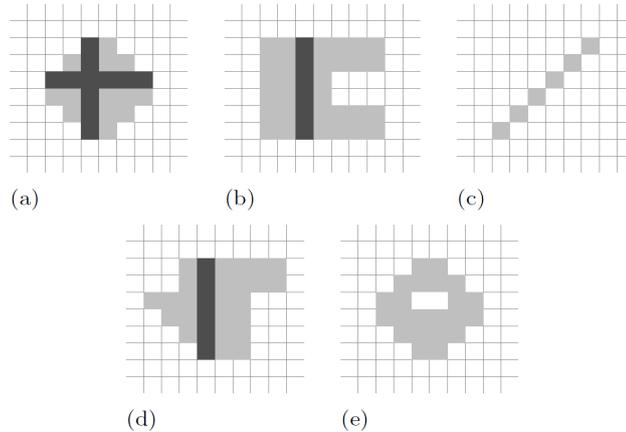
A family of shapes, denoted *NICE*, is introduced in this study to probably act as efficient intermediate shapes of the transformation. A *nice* shape is, informally, any connected shape that contains a particular line called the *central line* (denoted L_C). Intuitively, one may think of L_C as a supporting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

► **Definition 11** (Nice Shape). *A connected shape $S \in NICE$ if there exists a central line $L_C \subseteq S$, such that every node $u \in S \setminus L_C$ is connected to L_C via a line perpendicular to L_C (Figure 5 shows some examples of a nice shape and non nice shapes).*

By reversibility (Lemma 10), a *nice* shape is exploited as an intermediate shape that simplifies the universal transformations. The following proposition shows that:

► **Proposition 12.** *Let S_{Nice} be a nice shape and S_L a straight line, both of the same order n . Then $S_{Nice} \rightarrow S_L$ (and $S_L \rightarrow S_{Nice}$) in $O(n)$ steps.*

Proof. By Definition 11, any S_{Nice} of n nodes must have a central line L_C of k nodes, $1 \leq k \leq n$, where other $n - k$ nodes are connecting to L_C via a line. By Lemma 9, each of the $n - k$ nodes requires at most two line steps to be included into L_C in a maximum total of $2(n - k)$ steps for all $n - k$. Then, S_{Nice} requires at most $O(2n - 2k)$ steps to transform into S_L . As a consequence, the connected pair of shapes (S_{Nice}, S_L) of the same order are transformable to each other by Lemma 10, in $O(n)$ steps. ◀



■ **Figure 5** The central line L_C occupies black cells of *nice* shapes in (a), (b) and (d). In (c), the shape is not *nice* (due to the lack of L_C). (e) is also not *nice* because of the hole, white cells inside the shape, which prevents the formation of L_C .

2.2 Problem Definitions

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

DIAGONALTO LINE. Given an initial connected diagonal line S_D and a target vertical or horizontal connected spanning line S_L of the same order, transform S_D into S_L , without necessarily preserving the connectivity during the transformation.

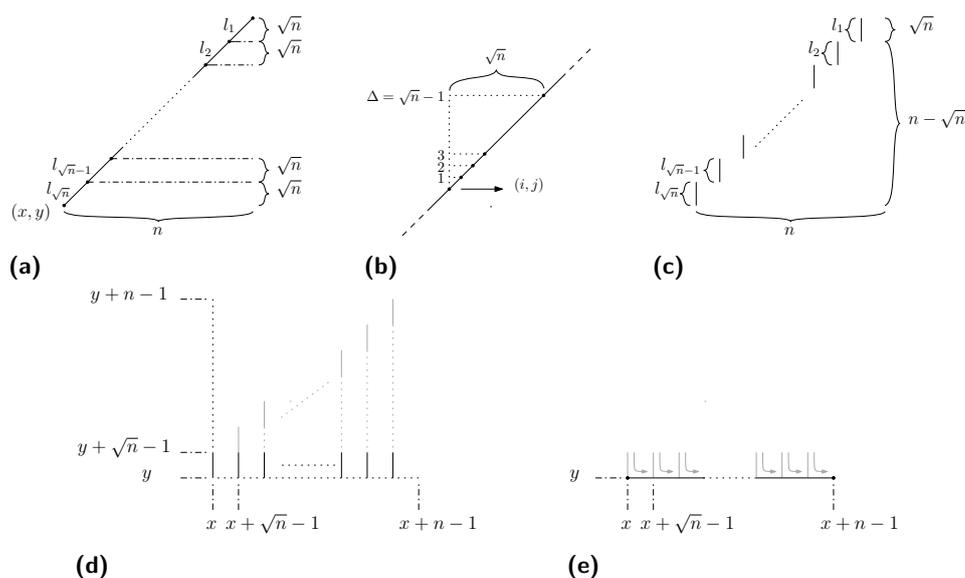
DIAGONALTO LINECONNECTED. Restricted version of **DIAGONALTO LINE** in which connectivity must be preserved during the transformation.

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

3 Transforming the Diagonal into a Line

We begin our study from the case in which the initial shape is a diagonal line S_D of order n . Our goal throughout the section is to transform S_D into a spanning line S_L , i.e., solve the **DIAGONALTO LINE** and/or **DIAGONALTO LINECONNECTED** problems. We do this, because these problems seem to capture the worst-case complexity of transformations in this model.

Consider a S_D of n nodes occupying $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, such as the diagonal line of 6 nodes depicted in Figure 5 (c). Observe that the diagonal comprises some special properties which cannot be found in other connected shapes, that is, each single node of S_D enjoys a unique x - and y - axis, in other words $n = \#rows = \#columns$. Below, we give three $O(n\sqrt{n})$ -time strategies to transform the diagonal into a line. In Section 3.1, the algorithm allows the shape to break its connectivity throughout transformation, whilst connectedness is preserved in Sections 3.2 and 3.3.



■ **Figure 6** (a) Dividing the diagonal into \sqrt{n} segments of length \sqrt{n} each (integer \sqrt{n} case). (b) A closer view of a single segment, where $1, 2, 3, \dots, \sqrt{n} - 1$ are the required distances for the nodes to form a line segment at the leftmost column (of the segment). (c) Each line segment is transformed into a line and transferred towards the bottommost row of the shape, ending up as in (d). (e) All line segments are turned into the bottommost row to form the target spanning line.

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

We start from `DIAGONALTO LINE` (i.e., no requirement to preserve connectivity). Our strategy is as follows. We divide the diagonal into several segments, as in Figure 6 (a). Then in each segment, we perform a trivial (inefficient, but enough for our purposes) line formation by moving each node independently to the leftmost column in that segment (Figure 6 (b)). Therefore, all segments are transformed into lines (Figure 6 (c)). Then, we transfer each line segment all the way down to the bottommost row of the diagonal S_D (Figure 6 (d)). Finally, we change the orientation of all line segments to form the target spanning line (Figure 6 (e)).

More formally, let S_D be a diagonal, occupying $(x, y), (x+1, y+1), \dots, (x+n-1, y+n-1)$, such that x and y are the leftmost column and the bottommost row of S_D , respectively. S_D is divided into $\lceil \sqrt{n} \rceil$ segments, $l_1, l_2, \dots, l_{\lceil \sqrt{n} \rceil}$, each of length $\lfloor \sqrt{n} \rfloor$, apart possibly from a single smaller one. Figure 6 (a) illustrates the case of integer \sqrt{n} and in what follows, w.l.o.g., we present this case for simplicity. This strategy (called *DL-Partitioning*) consists of three phases:

- **Phase 1:** Transforms each diagonal segment $l_1, l_2, \dots, l_{\sqrt{n}}$ into a line segment. Notice that segment l_k , $1 \leq k \leq \sqrt{n}$, contains \sqrt{n} nodes occupying positions $(x+h_k, y+h_k), (x+h_k+1, y+h_k+1), \dots, (x+h_k+\sqrt{n}-1, y+h_k+\sqrt{n}-1)$, for $h_k = n - k\sqrt{n}$; see Figure 6 (b). Each of these nodes moves independently to the leftmost column of l_k , namely column $x+h_k$, and the new positions of the nodes become $(x+h_k, y+h_k), (x+h_k, y+h_k+1), \dots, (x+h_k, y+h_k+\sqrt{n}-1)$. By the end of Phase 1, \sqrt{n} vertical line segments have been created (Figure 6 (c)).
- **Phase 2:** Transfers all \sqrt{n} line segments from Phase 1 down to the bottommost row y of the diagonal S_D . Observe that line segment l_k has to move distance h_k (see Figure 6 (d)).

- **Phase 3:** Turns all \sqrt{n} line segments into the bottommost row y (Figure 6 (e)). In particular, line l_k will be occupying positions $(x + h_k, y), (x + h_k + 1, y), \dots, (x + h_k + \sqrt{n} - 1, y)$.

Now, we are ready to analyse the running time of all phases of *DL-Partitioning*.

► **Theorem 13.** *Given an initial diagonal of n nodes, DL-Partitioning solves the DIAGONAL-TOLINE problem in $O(n\sqrt{n})$ steps.*

Proof. From the above reasoning, the proof follows by analysing all phases of *DL-Partitioning*. In the first phase, the cost of the trivial line formation is the run of all distances for $\lceil\sqrt{n}\rceil$ nodes to be gathered at the leftmost column of a single segment l_k for all k , where $1 \leq k \leq \sqrt{n}$. Observe that in Figure 6 (b), the $\lceil\sqrt{n}\rceil$ nodes of l_k have to move distances of $\Delta = 0, 1, 2, 3, \dots, (\sqrt{n} - 1)$. Therefore, the total run of all distances in l_k (*except the bottommost node which stays still in place*), is:

$$\begin{aligned}
 t_1 &= 1 + 2 + \dots + (\sqrt{n} - 1) = \sum_{i=1}^{\sqrt{n}-1} i \\
 &= \frac{\sqrt{n}(\sqrt{n} - 1)}{2} = \frac{n - \sqrt{n}}{2} \\
 &= O(n),
 \end{aligned} \tag{1}$$

From (1), the total run T_1 for all \sqrt{n} segments is given by:

$$\begin{aligned}
 T_1 &= t_1 \cdot \lceil\sqrt{n}\rceil \\
 &= \frac{n - \sqrt{n}}{2} \cdot \lceil\sqrt{n}\rceil = \frac{n\sqrt{n} - n}{2} = \frac{n(\sqrt{n} - 1)}{2} \\
 &= O(n\sqrt{n}).
 \end{aligned} \tag{2}$$

Next, in phase 2, all \sqrt{n} line segments transfer down into the bottommost of the diagonal except the one already there. As mentioned above, any line segment l_k has to transfer a distance of $n - k\sqrt{n}$ to reach the y bottommost row in a total T_2 as follows:

$$\begin{aligned}
 T_2 &= \sum_{k=1}^{\sqrt{n}-1} (n - k\sqrt{n}) = (n\sqrt{n} - n) - \sum_{k=1}^{\sqrt{n}-1} k\sqrt{n} \\
 &= (n\sqrt{n} - n) - \sqrt{n} \sum_{k=1}^{\sqrt{n}-1} k = (n\sqrt{n} - n) - \sqrt{n} \left(\frac{\sqrt{n}(\sqrt{n} - 1)}{2} \right) \\
 &= (n\sqrt{n} - n) - n \left(\frac{\sqrt{n} - 1}{2} \right) = \frac{n(\sqrt{n} - 1)}{2} \\
 &= O(n\sqrt{n}).
 \end{aligned} \tag{3}$$

The last phase is basically to turn (*re-orientate*) every line segments of \sqrt{n} nodes into the y bottommost row of length n . Now, each line segment l_k contributes a single node at the bottommost row y , therefore, we have \sqrt{n} nodes are sitting on row y , and the other $n - \sqrt{n}$ nodes are waiting to be pushed (*turned*) into the bottommost row of length $\Delta = n$, in order to form the goal spanning line. By Lemma 32 and 9, the transformation starts *re-orientating* each l_k towards the bottommost row y , until it is being completely filled in by n nodes of all segments. Recall that turning each node of the $n - \sqrt{n}$ into the bottommost row y costs two

line steps, therefore, the total T_3 for all $n - \sqrt{n}$ is:

$$\begin{aligned} T_3 &= 2 \cdot (n - \sqrt{n}) = 2n - 2\sqrt{n} = 2(n - \sqrt{n}) \\ &= O(n). \end{aligned} \tag{4}$$

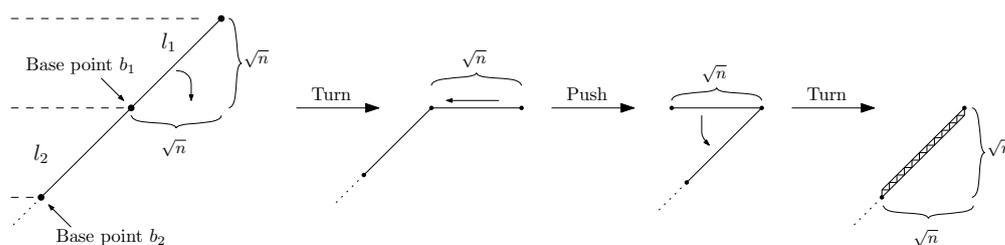
Altogether, the sum of (2), (3) and (4) gives the total cost T , in steps, for all phases,

$$\begin{aligned} T &= T_1 + T_2 + T_3 \\ &= \frac{n(\sqrt{n} - 1)}{2} + \frac{n(\sqrt{n} - 1)}{2} + 2(n - \sqrt{n}) \\ &= n(\sqrt{n} - 1) + 2n - 2\sqrt{n} = n\sqrt{n} + n - 2\sqrt{n} = n(\sqrt{n} + 1) - 2\sqrt{n} \\ &= O(n\sqrt{n}). \end{aligned}$$

◀

3.2 Preserving Connectivity through Folding

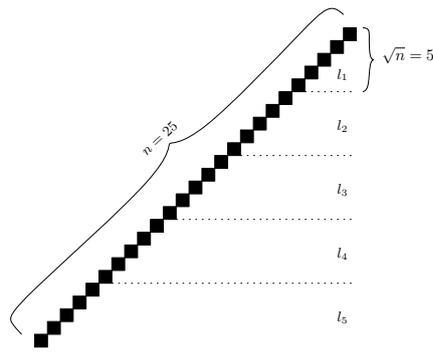
In this section, our purpose is to transform the diagonal S_D into a line S_L by exploiting the line mechanism, with preserving connectivity of the shape throughout transformations. This strategy, called *DLC-Folding*, is as follows; we partition S_D into several segments of the same lengths, as we did previously in *DL-Partitioning* (see Figure 6 (a)). Then, in each segment, we perform three operations: *turn*, *push* and *turn*. From the topmost segment of S_D , form a *line* segment by a trivial brute-force line formation, that is, move each node to the bottommost row in that segment. After that, push the *line* segment towards the leftmost column of S_D by a distance of $\Delta = \sqrt{n}$. Lastly, we fold this *line* segment diagonally to align above the next following diagonal segment, therefore two parallel diagonal segments are formed, as illustrated in Figure 7. The strategy keeps folding segments above each other in this way, until finishing at the bottommost segment of S_D . In the end, S_D transforms into a *nice shape*, which is trivially converted into a line. Figures 8 to 12 demonstrate the above transformations on a diagonal of 25 nodes.



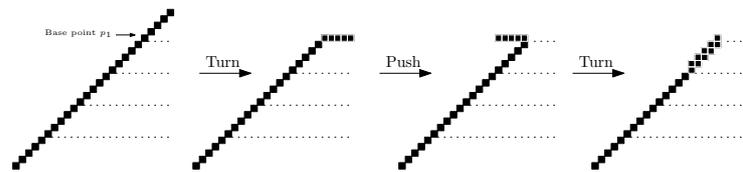
■ **Figure 7** Three operations (*turn*, *push* and *turn*) for folding the topmost segment of a diagonal line.

3.2.1 Formal Description

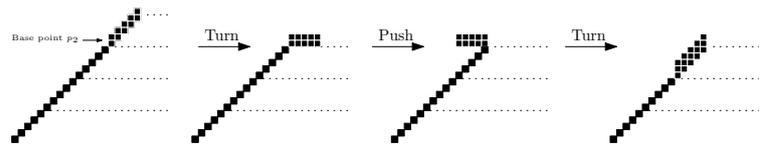
Let S_D be a diagonal of n nodes occupying $(x, y), (x + 1, y + 1), \dots, (x + n - 1, y + n - 1)$, such that x and y are the leftmost column and the bottommost row of S_D , respectively. S_D is then divided into \sqrt{n} segments, $l_1, l_2, \dots, l_{\sqrt{n}}$, each of which has a length of \sqrt{n} , where l_1 and $l_{\sqrt{n}}$ are the topmost and bottommost segments of S_D , respectively. Observe that segment l_k , $1 \leq k \leq \sqrt{n}$, consists of \sqrt{n} nodes occupying $(i, j), (i + 1, j + 1), \dots, (i + \sqrt{n} - 1, j + \sqrt{n} - 1)$,



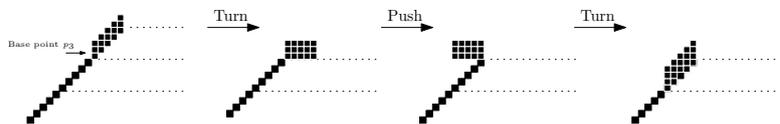
■ **Figure 8** A diagonal line of 25 nodes.



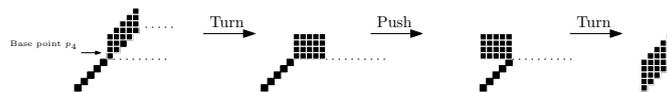
■ **Figure 9** *turn, push* and *turn* of the first phase.



■ **Figure 10** *turn, push* and *turn* of the second phase.



■ **Figure 11** *turn, push* and *turn* of the third phase.



■ **Figure 12** *turn, push* and *turn* of the fourth phase. Notice the resulting figure in the far right represents a *nice shape*.

where $i = x + h_k$ and $j = y + h_k$, for $h_k = n - k\sqrt{n}$. Here, l_k has a base point $b_k = (i, j)$, which is the bottommost node of l_k . Moreover, *DLC-Folding* completes its course in k phases, in each phase k , we apply three operations (*turn, push* and *turn*) to fold the corresponding segment(s) around the base point b_k . The first segment l_1 folds around b_1 in the first phase as follows (*due to symmetry, it is sufficient to demonstrate one orientation*):

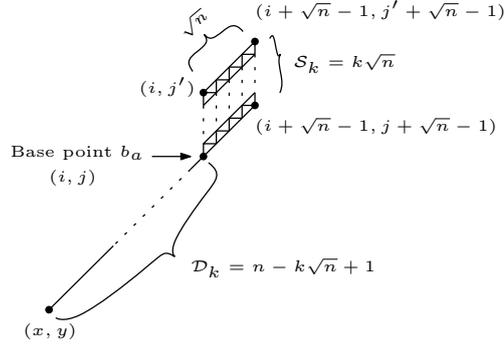
- (a) *turn*. Moves all \sqrt{n} nodes into the bottommost row of l_1 (brute-force line formation). Notice that the l_1 nodes change their positions from $(i, j), (i + 1, j + 1), \dots, (i + \sqrt{n} - 1, j + \sqrt{n} - 1)$ into $(i, j), (i + 1, j), \dots, (i + \sqrt{n} - 1, j)$. By the first operation, a horizontal line segment of length \sqrt{n} have been formed, and the base point b_1 keeps place at (i, j) .

- (b) *push*. Pushes the l_1 line segment \sqrt{n} steps towards the leftmost column of the diagonal S_D , i.e., the y column. All \sqrt{n} nodes of l_1 transfer altogether into $(i - \sqrt{n}, j), (i + 1 - \sqrt{n}, j), \dots, (i + \sqrt{n} - 1 - \sqrt{n}, j)$.
- (c) *turn*. Converts the line segment l_1 into diagonal again by moving its \sqrt{n} nodes down to align above the \sqrt{n} nodes of the next following diagonal segment l_2 . This takes place by transferring them into positions $(i - \sqrt{n}, j - \sqrt{n} + 1), (i + 1 - \sqrt{n}, j - \sqrt{n} + 2), \dots, (i + \sqrt{n} - 1 - \sqrt{n}, j)$, (except the bottommost node-base point b_1 which stays still in place at $(i - \sqrt{n}, j)$). By the end of this phase, two parallel diagonal segments l_1 and l_2 have been created, as in Figure 7.

The two parallel segments l_1 and l_2 are consisting of $2\sqrt{n}$ nodes and comprising of $2\sqrt{n}$ vertical lines. Consequently, a new *connected* shape has been formed and is defined below.

► **Definition 14.** A *Ladle* is a connected shape of n nodes consisting of two parts, \mathcal{D} and \mathcal{S} . For a given phase k , where $2 \leq k \leq \lceil \sqrt{n} \rceil$, we have $Ladle_k = \mathcal{D}_k + \mathcal{S}_k$, where \mathcal{D}_k and \mathcal{S}_k are connected via a base point $b_k = (i, j)$, such that:

- \mathcal{D}_k , is a diagonal line containing $n - k\sqrt{n} + 1$ nodes occupying $(x, y), (x + 1, y + 1), \dots, (i, j)$, such that x and y are the leftmost column and the bottommost row of $Ladle_k$, respectively, where $\sqrt{n} < i = j < n - \sqrt{n} + 1$. \mathcal{D}_k is connected to \mathcal{S}_k via its topmost node at (i, j) .
- \mathcal{S}_k , is parallelogram consists of k parallel diagonal segments of size $k\sqrt{n}$ nodes formed \sqrt{n} lines. \mathcal{S}_k is connected to \mathcal{D}_k via its bottommost node at (i, j) , as depicted in Figure 13.



■ **Figure 13** A *Ladle* shape in phase k , where $j' = j + k - 1$.

Throughout this section, we prove that at any phase k of *DLC-Folding*, there are k parallel (*diagonal*) segments containing $k\sqrt{n}$ nodes and forming \sqrt{n} vertical lines. We now prove that the three operations (*turn*, *push* and *turn*) transforms S_D into a *Ladle* by the end of the first phase of *DLC-Folding*.

► **Lemma 15.** Let S_D be a diagonal of order n partitioned into \sqrt{n} segments $l_1, l_2, \dots, l_{\sqrt{n}}$. *DLC-Folding* converts S_D into a *Ladle* by the end of the first phase.

Proof. Consider a diagonal S_D of n nodes as defined previously, which is partitioned into \sqrt{n} segments of length \sqrt{n} each. Now, perform the three operations (*turn*, *push* and *turn*) described above on the topmost segment of S_D (*it is sufficient due to symmetry*), we will obtain a connected shape consists of two parts, a diagonal line whose nodes occupy $(x, y), (x + 1, y + 2), \dots, (x + n - \sqrt{n} - 1, y + n - \sqrt{n} - 1)$ and two parallel diagonal segments

of $2\sqrt{n}$ nodes. Both are connected via the base point $(x + n - \sqrt{n} - 1, y + n - \sqrt{n} - 1)$, which is the topmost node of the diagonal part and the bottommost of the two parallel diagonal segments. As a result, one can easily find out that the new shape constructed by the end of the first phase meets all conditions mentioned in Definition 14, therefore, it is a *Ladle*. ◀

The following lemma shows that the three operations of *DLC-Folding* hold in any phase k , where $2 \leq k < \sqrt{n}$, such that in phase $k + 1$, the size of \mathcal{S}_k increases by \sqrt{n} , and conversely the \mathcal{D}_k length decreases by \sqrt{n} .

► **Lemma 16.** *Consider a Ladle of n nodes in phase k , where $1 < k \leq \sqrt{n}$. Then, in phase $k + 1$, DLC-Folding increases the size of \mathcal{S}_k by \sqrt{n} and decreases the length of \mathcal{D}_k by \sqrt{n} .*

Proof. The size of the *Ladle* $= |n|$ must be the same each phase and all time over transformations. In phase k , a *Ladle_k* consists of two parts, $\mathcal{D}_k = |n - k\sqrt{n} + 1|$ and $\mathcal{S}_k = |k\sqrt{n}|$, where both are connected via a common node (i, j) (see Definition 14). Now, perform the three operations (*turn*, *push* and *turn*) on the \mathcal{S}_k part that contains k segments of length \sqrt{n} aligned diagonally on top of each other. Let assume that the k segments form \sqrt{n} vertical lines since the same argument applies symmetrically to a different orientation. First, we move all vertical \sqrt{n} lines downwards to the bottommost row i of \mathcal{S}_k , which shall form k horizontal lines by completely filling in the k bottom rows of \mathcal{S}_k . Therefore, those horizontal lines create a rectangle, as depicted in Figure 14 (a). Then, the second operation pushes the k vertical lines of the rectangle \sqrt{n} steps horizontally towards the x leftmost column of the *Ladle_k*, as in Figure 14 (b). Lastly, the strategy completes folding by translating the vertical \sqrt{n} lines downwards, until each of them stays above a node of the next following segment (*notice that every vertical line is moved except the rightmost one*), see Figure 14 (c). By the end of phase $k + 1$, a new *Ladle* has been created, which is consisting of $\mathcal{D}_{k+1} = |n - (k - 1)\sqrt{n} + 1|$ and $\mathcal{S}_{k+1} = |(k + 1)\sqrt{n}|$ connected via the common b_{k+1} base point at $(i - \sqrt{n}, j - \sqrt{n})$. Hence, we conclude that in phase $k + 1$, the size of \mathcal{S}_{k+1} increased by \sqrt{n} nodes, while the length of \mathcal{D}_{k+1} decreased by \sqrt{n} , and this holds trivially and inductively for any phase k , where $1 < k \leq \sqrt{n}$. ◀

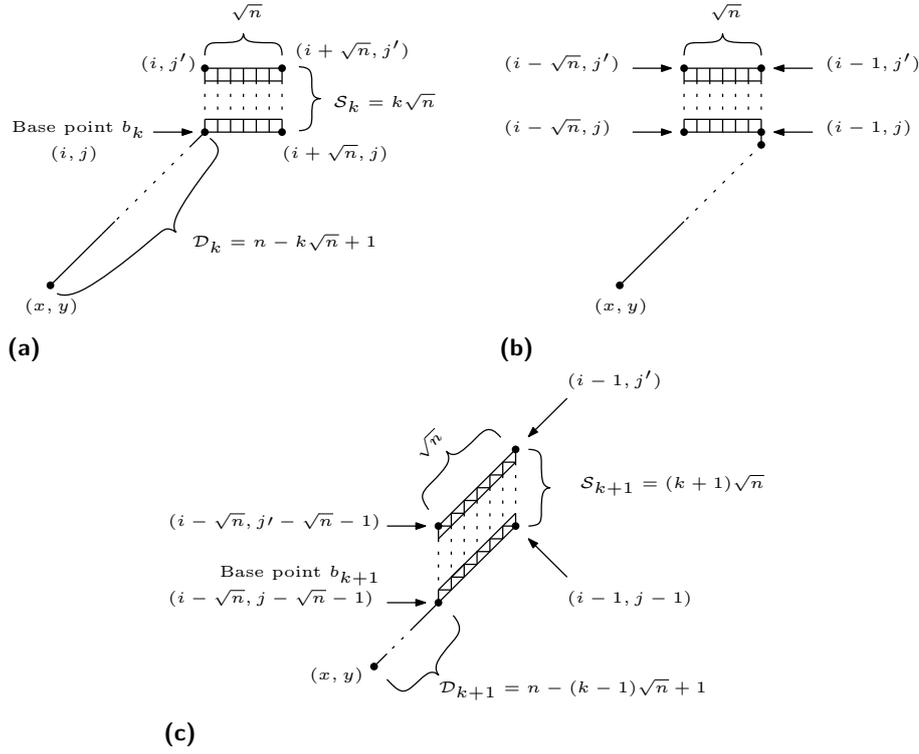
Now, we prove that *DLC-Folding* transforms S_D into a *nice* shape in \sqrt{n} phases.

► **Lemma 17.** *Given a diagonal S_D of order n partitioned into \sqrt{n} segments, DLC-Folding converts S_D into a nice shape in \sqrt{n} phases.*

Proof. By following Lemma 15, S_D converts into a *Ladle₂* which consists of two parts $\mathcal{D}_2 = |n - 2\sqrt{n} + 1|$ and $\mathcal{S}_2 = |2\sqrt{n}|$. Then, by Lemma 16, through the final phase $k = \sqrt{n}$, all segments are being folded diagonally over each other, therefore the diagonal part of the *Ladle* will be exhausted $\mathcal{D}_{\sqrt{n}} = \phi$, whilst the parallelogram part acquires all n nodes, $\mathcal{S}_{\sqrt{n}} = |n|$. The resulting shape of \sqrt{n} vertical (horizontal) lines at the end of the final phase is no longer *Ladle* and complies perfectly with all standards and properties of *nice shapes*, see Definition 11; as a result, it is a *nice* shape. ◀

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

► **Lemma 18.** *Given a diagonal S_D of order n partitioned into \sqrt{n} segments, DLC-Folding folds the topmost (bottommost) segment in $O(n)$ steps.*



■ **Figure 14** Folding a $Ladle_k$ over phase k , where $j' = j + k - 1$, see Lemma 16 for further explanation.

Proof. In the first phase, we perform *turn*, *push* and *turn* on the first topmost (bottommost) segment of S_D of length \sqrt{n} . The first operation (*turn*) is a brute-force line formation that is trivially computed by:

$$1 + 2 + \dots + (\sqrt{n} - 1) = \frac{\sqrt{n}(\sqrt{n} - 1)}{2} = \frac{n - \sqrt{n}}{2}, \quad (5)$$

Then, the second operation *pushes* a line of \sqrt{n} length in \sqrt{n} line steps. Again, the last operation costs as much as the first *turn*, namely $\frac{n - \sqrt{n}}{2}$. Altogether, the total cost required to fold the first segment thoroughly is at most:

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

◀

► **Lemma 19.** *By the end of phase k , for all $1 < k \leq \sqrt{n}$, DLC-Folding folds $Ladle_k$ in $O(n)$ steps.*

Proof. In phase k , $Ladle_k$ holds two parts, $\mathcal{D}_k = |n - k\sqrt{n} + 1|$ and $\mathcal{S}_k = |k\sqrt{n}|$. In the first operation of *DLC-Folding*, by exploiting the linear mechanisms, all \sqrt{n} lines of \mathcal{S}_k translate in a distance equals to (5), namely $\frac{n - \sqrt{n}}{2}$. Now, the \sqrt{n} lines have moved and formed another k lines in a different orientation. Therefore, in the second operation, we push those k lines \sqrt{n} steps in a total of:

$$k\sqrt{n} = O(n), \quad (6)$$

And the third move is completing the process by (folding) the \sqrt{n} lines diagonally above the next segment, with the same cost of (5):

$$\frac{n - \sqrt{n}}{2} = O(n), \quad (7)$$

With this, by summing (5), (6) and (7), the total steps performed by the end of phase k is given by:

$$\begin{aligned} t_k &= \frac{n - \sqrt{n}}{2} + k\sqrt{n} + \frac{n - \sqrt{n}}{2} = n - \sqrt{n} + k\sqrt{n} \\ &= O(n). \end{aligned} \quad (8)$$

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

Altogether, Proposition 12 and Lemmas 18 and 19, the running time of *DLC-Folding* is,

► **Theorem 20.** *Given an initial connected diagonal of n nodes, DLC-Folding solves the DIAGONALTOLINECONNECTED problem in $O(n\sqrt{n})$ steps.*

Proof. By Lemma 18, *DLC-Folding* creates a *Ladle* in a total of:

$$T_1 = \frac{n - \sqrt{n}}{2}. \quad (9)$$

Now, Lemma 19 provides the running time of phase k , for all $1 < k \leq \sqrt{n}$, therefore the total run for all phases is computed by (*except the first phase*):

$$\begin{aligned} T_2 &= \sum_{i=1}^{\sqrt{n}-1} n - \sqrt{n} + i\sqrt{n} = n\sqrt{n} - 2n - \sqrt{n} + \sum_{i=1}^{\sqrt{n}-1} i\sqrt{n} \\ &= n\sqrt{n} - 2n - \sqrt{n} + \sqrt{n} \sum_{i=1}^{\sqrt{n}-1} i = n\sqrt{n} - 2n - \sqrt{n} + n \left(\frac{\sqrt{n}-1}{2} \right) \\ &= n\sqrt{n} - 2n - \sqrt{n} + \left(\frac{n\sqrt{n} - n}{2} \right) = \frac{n\sqrt{n} - 5n - 2\sqrt{n}}{2} \\ &= O(n\sqrt{n}). \end{aligned} \quad (10)$$

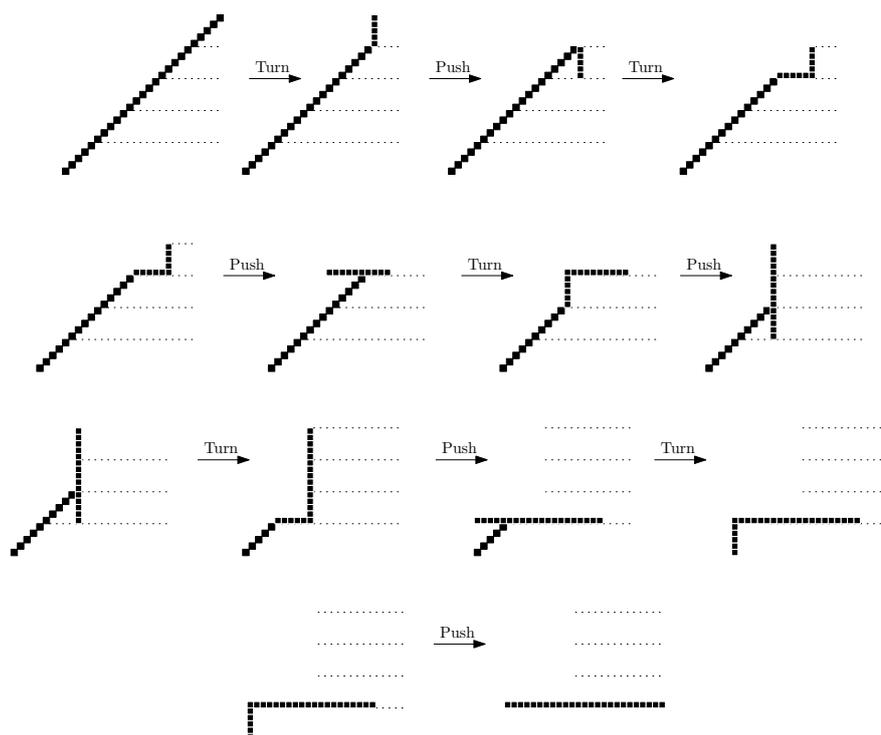
Then, the total cost for all phases of *DLC-Folding* is given by summing (9) and (10) :

$$\begin{aligned} T_3 &= T_1 + T_2 \\ &= \frac{n - \sqrt{n}}{2} + \frac{n\sqrt{n} - 5n - 2\sqrt{n}}{2} = \frac{2n - 2\sqrt{n} + 2n\sqrt{n} - 10n - 4\sqrt{n}}{4} \\ &= \frac{2n\sqrt{n} - 8n - 6\sqrt{n}}{4} = \frac{n\sqrt{n} - 4n - 3\sqrt{n}}{2} \\ &= O(n\sqrt{n}). \end{aligned} \quad (11)$$

Finally, the resulting shape of *DLC-Folding* is a *nice* shape, which transforms into a line S_L in $O(n)$ steps (see Proposition 12), then the total cost T required to transform S_D into S_L , is bounded above by:

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

◀



■ **Figure 15** Shows all transformations of *DLC-Extending* on a diagonal of 25 nodes.

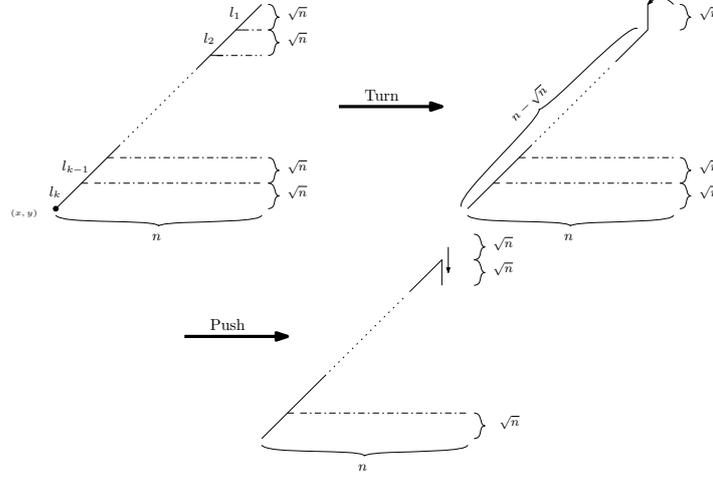
3.3 Preserving Connectivity through Extending

The current transformation, called *DLC-Extending*, is another strategy to transform the diagonal S_D into a line S_L in $O(n\sqrt{n})$ steps, with preserving connectivity throughout transformations. As mentioned earlier, S_D is partitioned into \sqrt{n} segments of length \sqrt{n} each. The implementation of this strategy lasts for \sqrt{n} phases equal to the number of segments. In each phase, we perform two types of movements, *turn* and *push*. Generally speaking, *DLC-Extending* starts building the spanning line by first performing a line formation on the *topmost* (*bottommost*) diagonal segment of S_D ; after that, convert the rest of the diagonal segments into lines and include them to the main spanning line, sequentially one after the other. Notice that *DLC-Extending* extends the main spanning line gradually every phase by a length of \sqrt{n} . Figure 15 shows the performance of *DLC-Extending* on a diagonal of 25 nodes.

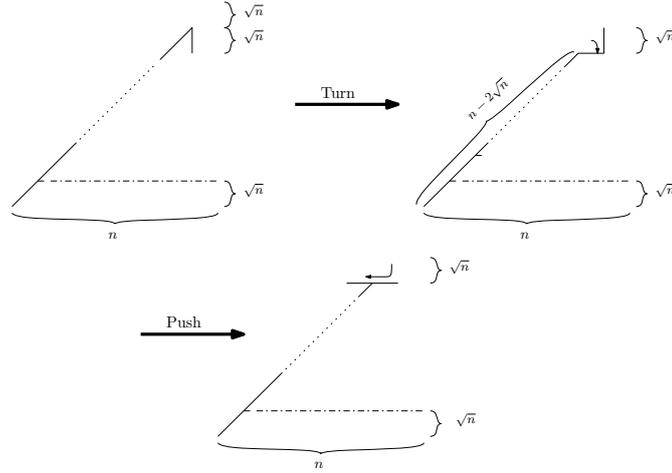
3.3.1 Formal Description

Consider an initial diagonal S_D of n nodes defined and partitioned as in Section 3.2.1. In phase k , for all $1 \leq k \leq \sqrt{n}$, we perform two line operations, *turn* and *push*, on a diagonal segment $l_k \in S_D$ of length \sqrt{n} nodes. Assume l_k occupies $(i, j), (i+1, j+1), \dots, (i+\sqrt{n}-1, j+\sqrt{n}-1)$, where $i = x + h_k$ and $j = y + h_k$, for $h_k = n - k\sqrt{n}$. Here, the bottommost node $b_k = (i, j)$ is the base point of l_k . Due to symmetry, we only show transformations starting vertically from the topmost segment of S_D . In the first phase, *DLC-Extending* forms a line at the topmost segment l_1 , by performing a brute-force line formation to transfer all nodes into the leftmost column of l_1 . Consequently, all nodes move from $(i, j), (i+1, j+1), \dots, (i+\sqrt{n}-1, j+\sqrt{n}-1)$ into $(i, j), (i, j+1), \dots, (i, j+\sqrt{n}-1)$. Secondly, it *pushes* this *line* segment \sqrt{n} steps towards

the bottommost row of S_D to occupy $(i, j - \sqrt{n}), (i, j + 1 - \sqrt{n}), \dots, (i, j + \sqrt{n} - 1 - \sqrt{n})$. By the end of the first phase, *DLC-Extending* shall construct a spanning line of length \sqrt{n} . For example, Figure 16 demonstrates the two operations (*turn* and *push*) of the first phase.



■ **Figure 16** The process of the two operations (*turn* and *push*) in the first phase.

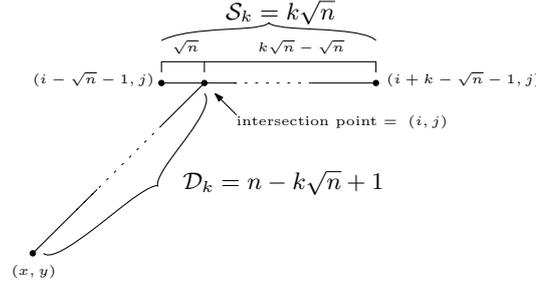


■ **Figure 17** The process of the two operations (*turn* and *push*) in the second phase.

In the second phase, *DLC-Extending* turns l_2 into a *line* by sending all l_2 nodes towards its bottommost row (*in a sequential order*). Observe that l_1 and l_2 are now connected perpendicularly via $(i, j - \sqrt{n})$ (Figure 17), which then provides the ability to *push* l_1 into l_2 a distance of \sqrt{n} vertically towards the leftmost column of S_D . By the end of this phase, the length of the spanning line is extended by \sqrt{n} . As a result, we have obtained a specific *connected* shape, called \mathcal{T} -*shape* and defined as follows;

► **Definition 21.** A \mathcal{T} -*shape* is a *connected shape* of n consisting of two parts, \mathcal{D} and \mathcal{S} . Both are *connected* via a common intersection point (i, j) . Figure 18 shows an example of \mathcal{T} -*shape* in phase k for all $1 < k < \sqrt{n}$, such that \mathcal{T} -*shape* $_k = \mathcal{D}_k + \mathcal{S}_k$, where

- (a) - \mathcal{D}_k , is a diagonal line containing $n - k\sqrt{n} + 1$ nodes and occupying $(x, y), (x + 1, y + 1), \dots, (i, j)$, such that x and y are the leftmost column and the bottommost row of $\mathcal{T_shape}_k$, respectively, where $\sqrt{n} < i = j < n - \sqrt{n} + 1$.
- (b) - \mathcal{S}_k , is a horizontal or vertical line of length $k\sqrt{n}$ nodes occupying $(i - \sqrt{n} - 1, j), (i + 1 - \sqrt{n} - 1, j), \dots, (i, j)$ or $(i, j - \sqrt{n} - 1), (i, j + 1 - \sqrt{n} - 1), \dots, (i, j)$, respectively, where $i' = i + k\sqrt{n} - 1$ and $j' = j + k\sqrt{n} - 1$.



■ **Figure 18** An example of a $\mathcal{T_shape}$ in phase k .

We turn now to prove correctness of *DLC-Extending*. First, we show that the first and second phase converts a diagonal S_D into a $\mathcal{T_shape}$.

► **Lemma 22.** *Let S_D be a diagonal of order n partitioned into $\lceil \sqrt{n} \rceil$ segments $l_1, l_2, \dots, l_{\lceil \sqrt{n} \rceil}$. By the end of the second phase, *DLC-Extending* converts S_D into a $\mathcal{T_shape}$.*

Proof. By performing the above two main operations (*turn* and *push*) on the two topmost segments, l_2 and l_2 , respectively, we shall acquire a connected shape of two parts: 1) A diagonal at $(x, y), (x + 1, y + 1), \dots, (i, j)$, where $i = x + n - 2\sqrt{n} - 2$ and $j = y + n - 2\sqrt{n} - 2$, and 2) A horizontal or vertical line of length $2\sqrt{n}$ occupying $(i - 2\sqrt{n}, j), \dots, (i + 2\sqrt{n}, j)$ or $(i, j - 2\sqrt{n}), \dots, (i, j + 2\sqrt{n})$, respectively. Observe that the two parts will have the same common intersection point at (i, j) in all cases. Therefore, the resulting shape of the second phase meet all properties of Definition 21, and hence, we conclude that this shape is a $\mathcal{T_shape}$. ◀

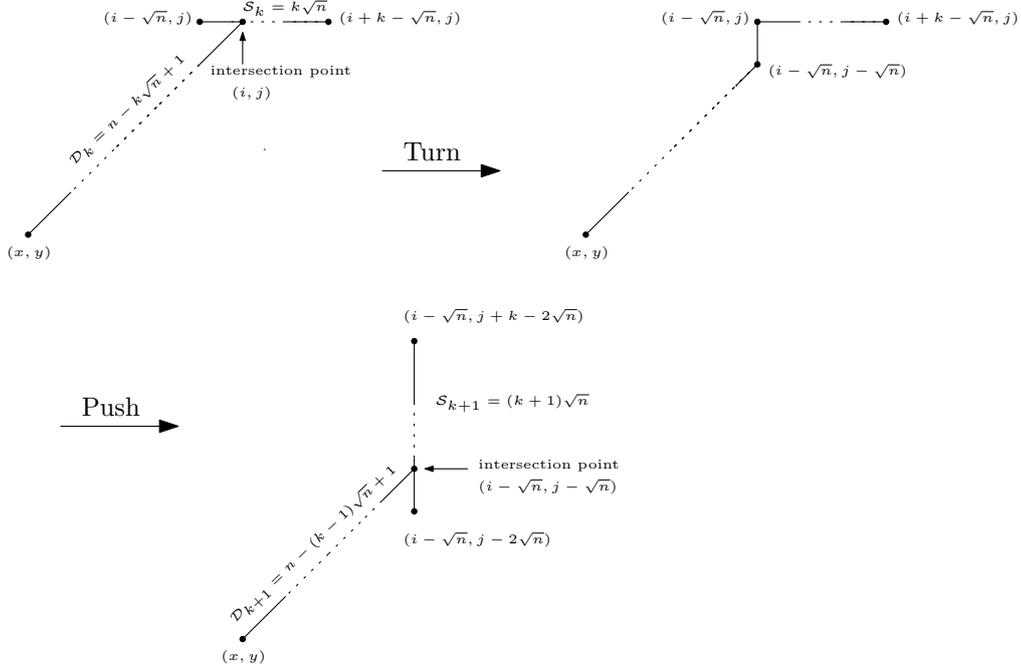
The following lemma shows that the two operations of *DLC-Extending* hold in any phase k , for all $2 \leq k < \sqrt{n}$.

► **Lemma 23.** *Let a $\mathcal{T_shape}_k$ of n nodes be in phase k , for all $2 \leq k < \sqrt{n}$. Therefore, in phase $k + 1$, *DLC-Extending* increases and decreases the length of \mathcal{S} and \mathcal{D} by \sqrt{n} , respectively.*

Proof. By Definition 21, $\mathcal{T_shape}_k$ in phase k holds two segments, a diagonal $\mathcal{D}_k = \lfloor n - k\sqrt{n} \rfloor$ occupies $(x, y), \dots, (i, j)$ and vertical (horizontal) line $\mathcal{S}_k = \lfloor k\sqrt{n} \rfloor$ at $(i - \sqrt{n}, j), \dots, (i + k - \sqrt{n}, j)$, where both intersect at a common point (i, j) , as shown in Figure 19 top-left. Assume that \mathcal{S}_k is horizontal (due to symmetry, it is sufficient to focus on only one orientation). Therefore, we *turn* the topmost diagonal segment of \mathcal{D}_k by performing a *line formation* to collect all \sqrt{n} nodes at the leftmost column in that segment. Therefore, they move from cells $(i - \sqrt{n} + 1, j - \sqrt{n} + 1), \dots, (i - 1, j - 1)$ into $(i - \sqrt{n} + 1, j - \sqrt{n} + 1), \dots, (i - \sqrt{n} + 1, j - 1)$. Notice that the shape still maintains connectivity.

Next, we *push* \mathcal{S}_k to include the new constructed *line* segment, which subsequently extends \mathcal{S}_k vertically with an increase of \sqrt{n} in length, in order to occupy $(i - \sqrt{n}, j -$

$2\sqrt{n}), \dots, (i - \sqrt{n}, j + k - 2\sqrt{n})$. In phase $k + 1$, we obtain a new *connected* shape consists of \mathcal{D}_{k+1} of length $n - (k - 1)\sqrt{n}$ and \mathcal{S}_{k+1} of $(k + 1)\sqrt{n}$, and they both intersect at a common point $(i - \sqrt{n} - 1, j)$, as shown in Figure 19. Therefore, we derive that *DLC-Extending* in phase $k + 1$ decreases the length of \mathcal{D}_k by \sqrt{n} and increases \mathcal{S}_k by \sqrt{n} . ◀



■ **Figure 19** All transformations of *DLC-Extending* on \mathcal{T} -shape from phase k to $k + 1$, see Lemma 23 for more details.

Here, we show that the last phase \sqrt{n} of *DLC-Extending* converts a \mathcal{T} -shape into a spanning line S_L .

▶ **Lemma 24.** *Given a diagonal S_D of n nodes and partitioned into \sqrt{n} segments, *DLC-Extending* transforms a \mathcal{T} -shape into a spanning line S_L by the end of the last phase.*

Proof. It follows from Lemmas 22 and 23. At any phase k , for all $2 \leq k < \sqrt{n}$, \mathcal{D} decreases and \mathcal{S} increases by \sqrt{n} . Thus, we shall obtain the following in phase $\sqrt{n} - 1$;

$$\begin{aligned}
 \mathcal{T}\text{-shape}_{\sqrt{n}-1} &= \mathcal{D}_{\sqrt{n}-1} + \mathcal{S}_{\sqrt{n}-1} \\
 &= [n - (\sqrt{n} - 1)\sqrt{n}] + [(\sqrt{n} - 1)\sqrt{n}] \\
 &= [n - (n - \sqrt{n})] + [(n - \sqrt{n})] \\
 &= [\sqrt{n}] + [(n - \sqrt{n})].
 \end{aligned}$$

As a result, we acquire $\mathcal{D}_{\sqrt{n}-1}$ of length \sqrt{n} and $\mathcal{S}_{\sqrt{n}-1}$ of $n - \sqrt{n}$ nodes. At this moment, in the last phase \sqrt{n} , we can trivially *turn* and *push* the diagonal $\mathcal{D}_{\sqrt{n}-1} = |\sqrt{n}|$ and include it to $\mathcal{S}_{\sqrt{n}-1}$ by the line formation and pushing mechanism, in order to form the final spanning line S_L . ◀

To sum up, the total running time for all \sqrt{n} phases of *DLC-Extending* is analysed below.

► **Theorem 25.** *Given an initial connected diagonal of n nodes, DLC-Extending solves the DIAGONALTO LINECONNECTED problem in $O(n\sqrt{n})$ steps.*

Proof. By Lemmas 22, 23 and 24, *DLC-Extending* carries out transformations in \sqrt{n} phases. In phase k , for all $1 \leq k \leq \sqrt{n}$, the two main operations, *turn* and *push*, is performed. Here, the first operation *turns* a corresponding segment l_k by applying a brute-force line formation on its \sqrt{n} nodes (*except the base point*) as follows;

$$\begin{aligned} t_1 &= 1 + 2 + \dots + (\sqrt{n} - 1) = \sum_{i=1}^{\sqrt{n}-1} i \\ &= \frac{\sqrt{n}(\sqrt{n} - 1)}{2} = \frac{n - \sqrt{n}}{2} \\ &= O(n), \end{aligned} \tag{12}$$

Multiply 12 by \sqrt{n} to obtain the total *turns* for all \sqrt{n} phases:

$$\begin{aligned} T_1 &= t_1 \cdot \sqrt{n} = \frac{n - \sqrt{n}}{2} \cdot \sqrt{n} = \frac{n\sqrt{n} - n}{2} \\ &= O(n\sqrt{n}). \end{aligned} \tag{13}$$

The second operation of phase k *pushes* the spanning line by $(k - 1)\sqrt{n}$ steps horizontally or vertically in at most (see Lemma 9):

$$t_2 = 2 \cdot (k - 1)\sqrt{n} = k\sqrt{n} - \sqrt{n}, \tag{14}$$

Now, observe that the last phase only *turns* and *pushes* the last diagonal segment of length \sqrt{n} (*dose not push the whole spanning line, see Lemma 24*). Therefore, the total of *push* operations for all \sqrt{n} phases is the summation of 14 plus pushing the last segment of length \sqrt{n} :

$$\begin{aligned} T_2 &= \sum_{i=1}^{\sqrt{n}-1} t_2 + 2\sqrt{n} \\ &= \sum_{i=1}^{\sqrt{n}-1} i\sqrt{n} + \sqrt{n} = \sqrt{n} \sum_{i=1}^{\sqrt{n}-1} i + \sqrt{n} = \sqrt{n} \cdot \frac{n - \sqrt{n}}{2} + \sqrt{n} = \frac{n\sqrt{n} - n}{2} + \sqrt{n} \\ &= O(n). \end{aligned} \tag{15}$$

Finally, putting 13 and 15 together, *DLC-Extending* completes its course in a total cost T computed by,

$$\begin{aligned} T &= T_1 + T_2 \\ &= \frac{n\sqrt{n} - n}{2} + n + \sqrt{n} = \frac{n\sqrt{n} - n + 2n + 2\sqrt{n}}{2} = \frac{n\sqrt{n} - n + 2\sqrt{n}}{2} \\ &= O(n\sqrt{n}). \end{aligned}$$

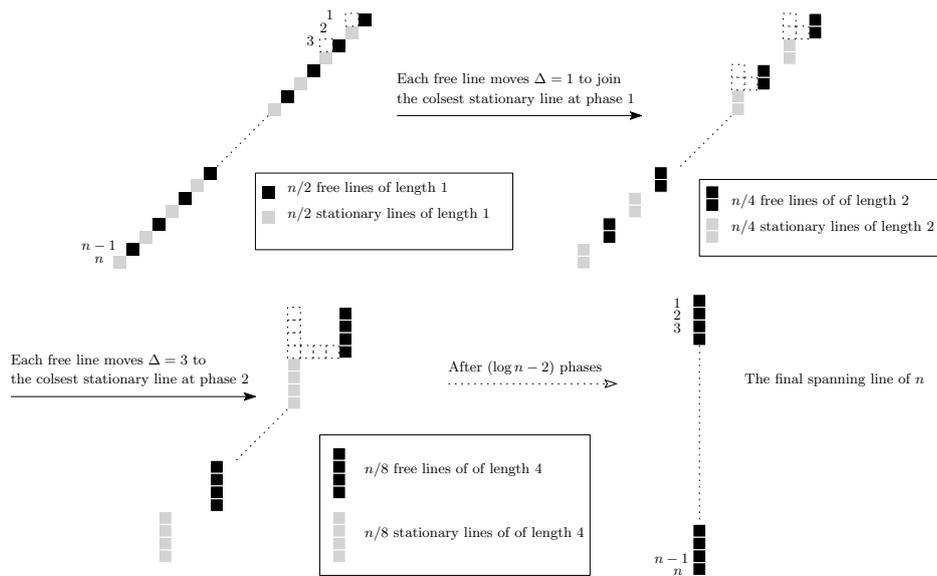
◀

3.4 An $O(n \log n)$ -time Transformation

We now investigate another approach (called *DL-Doubling*) for DIAGONALTO LINE (i.e., without necessarily preserving connectivity). The main idea is as follows. The initial

configuration can be viewed as n lines of length 1. We start (in *phases*) to successively double the length of lines (while halving their number) by matching them in pairs through shortest paths, until a single spanning line remains. Let the lines existing in each phase be labelled $1, 2, 3, \dots$ from top-right to bottom-left. In each phase, we shall distinguish two types of lines, *free* and *stationary*, which correspond to the odd ($1, 3, 5, \dots$) and even ($2, 4, 6, \dots$) lines from top-right to bottom-left, respectively. In any phase, only the *free* lines move, while the *stationary* stay still. In particular, in phase i , every free line j moves via a shortest path to merge with the next (top-right to bottom-left) stationary line $j + 1$. This operation merges two lines of length k into a new line of length $2k$ residing at the column of the stationary line. In general, at the beginning of every phase i , $1 \leq i \leq \log n$, there are $n/2^{i-1}$ lines of length 2^{i-1} each. These are interchangeably free and stationary, starting from a free top-right one, and at distance 2^{i-1} from each other. The minimum number of steps by which any free line of length k_i , $1 \leq k_i \leq n/2$, can be merged with the stationary next to it is roughly at most $4k_i = 4 \cdot 2^i$ (by two applications of turning of Lemma 9). By the end of phase i (as well as the beginning of phase $i + 1$), there will be $n/2^i$ lines of length 2^i each, at distances 2^i from each other. The total cost for phase i is obtained then by multiplying $n/2^i$ free lines, each is paying at most $4 \cdot 2^i$ to merge with the next stationary, thus, a linear cost in each one of $\log n$ phases in total.

See Figure 20 for an illustration of *DL-Doubling*.



■ **Figure 20** The process of the $O(n \log n)$ -time *DL-Doubling*. Nodes reside inside the black and grey cells.

3.4.1 Formal Description

Consider an initial diagonal line S_D of n nodes occupying cells $(x_1, y_1), \dots, (x_n, y_n)$, therefore, we can define the odd and even nodes into two different types of line of length 1, *free* and *stationary*. In any phase of *DL-Doubling*, the permission of move is only given to *free* lines, i.e., *stationary* lines cannot move in that phase. The $\lceil \frac{n}{2} \rceil$ nodes in $(x_1, y_1), (x_3, y_3), \dots, (x_{n-1}, y_{n-1})$ are *free*, on the contrary, $(x_2, y_2), (x_4, y_4), \dots, (x_n, y_n)$ are occupied by $\lceil \frac{n}{2} \rceil$ *stationary* lines. Since *DL-Doubling* keeps doubling the length of lines from

1 up to n , it consequently lasts for $\log n$ phases. In the first phase $i = 1$, $1 \leq i \leq \log n$, each of the $\lceil \frac{n}{2} \rceil$ *free* lines of length 1 moves a distance of $\Delta = 1$ to their left to occupy $(x_2, y_1), (x_4, y_3), \dots, (x_n, y_{n-1})$, in order to merge with the next following $\lceil \frac{n}{2} \rceil$ *stationary* lines. As a result, we obtain $\lceil \frac{n}{2} \rceil$ (*vertical*) lines of length 2, where their bottommost nodes occupying $(x_2, y_1), (x_4, y_3), \dots, (x_n, y_{n-1})$, as depicted in the top of Figure 20. *Due to symmetry, it is sufficient to show transformations occurring on one orientation, i.e., horizontal doubling holds.*

In the second phase, we divide the *vertical* $\lceil \frac{n}{2} \rceil$ lines of length 2 each into $\lceil \frac{n}{4} \rceil$ *free* and $\lceil \frac{n}{4} \rceil$ *stationary* lines interchangeably, starting from a free top-right one at (x_{n-2}, y_{n-2}) , and at distance $\Delta = 2^{2-1} = 2$ from each other. Now, the $\lceil \frac{n}{4} \rceil$ *free* lines must move at least $\Delta = 3$ steps to line up with the next following *stationary* line to the left. Consequently, all $\lceil \frac{n}{4} \rceil$ *free* joint *stationary* lines, which forms $\lceil \frac{n}{4} \rceil$ (*vertical*) lines of length 4 each. Eventually, when *DL-Doubling* repeats this process $\log n$ phases, it will form the goal spanning line S_L of order n by the end of phase $\log n$.

Here, we should also take into account all turns and delays of which a *free* line asks for moving and reallocating above the next following *stationary* line to the left. In particular, each *free* line must first convert from *vertical* into *horizontal*, move one further step to fill the empty cell above the topmost node of the *stationary* and finally converts again into *vertical*. By Lemma 9, at any phase i , for all $1 \leq i \leq \log n$, a *free* line of length k_i , where $1 \leq k_i \leq n/2$, performs at most $4k_i = 4 \cdot 2^i$ steps to merge with closest *stationary* line. Therefore, the following lemma shows that holds in all $\log n$ phases.

► **Lemma 26.** *By the end of phase i , for all $1 \leq i \leq \log n$, DL-Doubling has created $n/2^i$ lines, each of length 2^i , by performing $O(n)$ steps in that phase.*

Proof. We use induction to prove this argument. Following the above reasoning of *DL-Doubling* in Section 3.4.1, by the end of first phase (base case), each of $\frac{n}{2}$ *free* lines of length 1 moves only $\Delta = 1$ step to form $\frac{n}{2}$ lines of length 2, which costs at most $1 \cdot \frac{n}{2} = O(n)$ steps in total. Hence, the base case is true for phase 1. Now, assume that holds in phase i , for all $1 \leq i \leq \log n$, where each free line of length 2^{i-1} pays at most $4 \cdot 2^{i-1} = 2^{i+1} - 3$ steps to match with the closest *stationary* lines (two applications of Lemma 9), therefore, a total cost t_i required to form $\frac{n}{2^i}$ lines of length 2^i in phase i is given as follows;

$$\begin{aligned} t_i &= \frac{n}{2^i} \cdot (2^{i+1} - 3) \\ &= \frac{2^{i+1} \cdot n}{2^i} - \frac{3n}{2^i} = 2n - \frac{3n}{2^i} = n \left(2 - \frac{3}{2^i} \right) \\ &= O(n), \end{aligned} \tag{16}$$

Now, we prove that this must hold in phase $i + 1$,

$$\begin{aligned} t_{i+1} &= \frac{n}{2^{i+1}} \cdot (2^{i+2} - 3) \\ &= \frac{2^{i+2} \cdot n}{2^{i+1}} - \frac{3n}{2^{i+1}} = 2n - \frac{3n}{2^{i+1}} = n \left(2 - \frac{3}{2^{i+1}} \right) \\ &= O(n). \end{aligned} \tag{17}$$

As a result, our assumption is also true for phase i , therefore, from 16 and 17, we conclude that *DL-Doubling* pays at most $O(n)$ steps in each phase. ◀

Utilising Lemma 26, we can now formulate the following:

► **Theorem 27.** *DL-Doubling transforms any diagonal S_D of order n into a line S_L in $O(n \log n)$ steps.*

Proof. *DL-Doubling* constructs the goal spanning line S_L of length n in $\log n$ phases, while S_L increases exponentially in each phase. Given that, the total running time T of this transformation is computed by summing all steps of 16 over all $\log n$ phases, as follows:

$$\begin{aligned}
 T &= \sum_{i=1}^{\log n} t_i \\
 &= \sum_{i=1}^{\log n} n(2 - \frac{3}{2^i}) \\
 &= 2n \log n - 3n \sum_{i=1}^{\log n} \frac{1}{2^i},
 \end{aligned} \tag{18}$$

Let us compute the right summation of 18,

$$\sum_{i=1}^{\log n} \frac{1}{2^i} = \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^{\log n}} \right), \tag{19}$$

By multiplying 19 by 2 and subtracting it again by itself,

$$\begin{aligned}
 &\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^{\log(n-1)}} \right) - \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^{\log(n-1)}} + \frac{1}{2^{\log(n)}} \right) \\
 &= 1 - \frac{1}{2^{\log(n)}},
 \end{aligned} \tag{20}$$

Then, by plugging 20 into 18, this yields,

$$\begin{aligned}
 2n \log n - 3n \left(1 - \frac{1}{2^{\log n}} \right) &= 2n \log n - 3n + \frac{3n}{2^{\log n}} = n \left(2 \log n + \frac{3}{2^{\log n}} - 3 \right) \\
 &= O(n \log n).
 \end{aligned}$$

Thus, it has been proved that *DL-Doubling* transforms S_D of n nodes into S_L in a total of $O(n \log n)$ steps. ◀

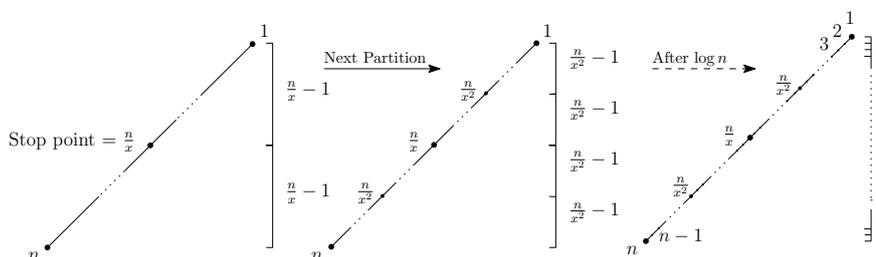
3.5 An $O(n \log n)$ -time Transformation Based on Recursion

An interesting observation for DIAGONALTOLINE (i.e., without necessarily preserving connectivity), is that the problem is essentially self-reducible. This means that any transformation for the problem can be applied to smaller parts of the diagonal, resulting in small lines, and then trying to merge those lines into a single spanning line. An immediate question is then whether such recursive transformations can improve upon the $O(n \log n)$ best upper bound established so far. The extreme application of this idea is to employ a full uniform recursion (call it *DL-Recursion*), where S_D is first partitioned into two diagonals of length $n/2$ each, and each of them is being transformed into a line of length $n/2$, by recursively applying to them the same halving procedure. Finally, the top-right half has to pay a total of at most $4(n/2) = 2n$ to merge with the bottom-left half and form a single spanning line (and the same is being recursively performed by smaller lines).

More formally, consider a diagonal S_D of n nodes where the bottom-left and top right nodes occupy (x_1, y_1) and (x_n, y_n) , respectively. Then, the goal is to collect all nodes at

the leftmost column, say x_n . The collection can be arranged in a recursive way by creating stop points (partitions) on S_D in which each stop point always creates equal partitions of the same length. This can be parametrized by $\frac{n}{x}$ for each partition, where $2 \leq x \leq n$. For example, if $x = 2$, we have a stop point that halves S_D into 2 partitions of length $\lceil \frac{n}{2} \rceil$. As a consequence, the first node on the top will stop at the middle of S_D and wait for all nodes to its right to gather at that point (*column*) and then continue directly to the gathering (*column*) x_n .

Now, let us repeat the same process on each of the x partitions recursively, by considering the partition as a diagonal of length roughly $\frac{n}{x} - 1$, which is divided into x sub-partitions each of length $\frac{n}{x^2}$ roughly. Every recursion shrinks the partitions by a factor of $\frac{1}{x}$. For example, in the $x = 2$ case, we halve the length of the partitions every time we subdivide, therefore, we will end the recursion when it arrives at partitions of length 1, which will happen after $\log n$ repetitions. That occurs similarly for the general x case by simply end after $\log_x n$ repetitions. For example, Figure 21 demonstrates this procedure.



■ **Figure 21** Shows all steps of subdividing the diagonal S_D recursively by a factor of $\frac{1}{x}$, where $x = 2$.

Next, we draw an abstract underlying tree of the partitioning process to trace all necessary computations required to travel from the diagonal S_D into a bottommost left column x_n . Figure 22 presents the tree of n nodes and weighted edges indicate the minimum distances (shortest paths) Δ between them, with a degree and depth of $\log_2 n$, which is also the number of phases that are needed to cease the segmentation recursively. Here, node n is the root of the tree occupies the target column at x_n , and it has $\log n$ child nodes (*stop points*) $n - 1, n - 2, n - 4, \dots, \frac{n}{2}$ of distances $\Delta = 1, 2, 4, \dots, \frac{n}{2}$, respectively. Now, the distance Δ between a parent u and child node v defines two basic properties: 1) the number of sub-child nodes (siblings) of v , namely each child node v gets $\log \Delta(u, v)$ sub-child nodes, and 2) the maximum cost by which a child node v requires to merge with its parent u , and it is computed by $(2^{\Delta+1} - 3)$ (a reader may consult Lemma 26 and Theorem 27). For example, the $\frac{n}{2}$ child node needs $(2^{\frac{n}{2}+1} - 3)$ steps to join its parent node n in a distance of $\Delta = n - \frac{n}{2} = \frac{n}{2}$, and at the same time, this tells us that this child node is also holding $\log \frac{n}{2}$ sub-child nodes. On the contrary, $n - 1$ node got no sub-child since $\log 1 = 0$, so it is a leaf and requires $(2^{1+1} - 3) = 1$ steps to reach its parent node n . The same idea follows for other sub-trees, such as $\frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \dots, 2$.

Having said that, the steps of any transformation strategy that solves the above recursion problem can generally be reordered without affecting the cost, and each stop point takes place in the reordered version. That is because the abstract tree representation remains conveniently invariant, (*inherits the same cost*), for any arrangement by which a transformation can exploit to collect nodes into the target point (*column* or *row*). However, this recursion proceeds in $\log n$ cost phases, where in each phase i , for all $1 \leq i \leq \log n$, we upper bound a cost that the transformation A pays at most to move all nodes in each phase (even though A may move

incurring T_1 . In particular, we analyse the recurrence relation:

$$\begin{aligned} T_n &= 2 \cdot T_{n/2} + 2n = 2(2 \cdot T_{n/4} + n) + 2n = 4 \cdot T_{n/4} + 4n = 4(2 \cdot T_{n/8} + n/2) + 4n \\ &= 8T_{n/8} + 6n = \dots = 2^i \cdot T_{n/2^i} + 2i \cdot n = \dots = 2^{\log n} \cdot T_{n/2^{\log n}} + 2(\log n)n \end{aligned}$$

Since $T_1 = 1$, we get,

$$\begin{aligned} T_n &= n \cdot T_1 + 2n(\log n) = n + 2n(\log n), \\ &= O(n \log n). \end{aligned}$$

Finally, we give the following theorem,

► **Theorem 28.** DL-Recursion transforms any diagonal S_D of order n into a line S_L of the same order in $O(n \log n)$ steps.

4 Universal Transformations

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

In this section, we develop a universal transformation, called *U-Box-Partitioning*, which exploits line movements in order to transform *any* initial connected shape S_I into *any* target shape S_F of the same order n , in $O(n\sqrt{n})$ steps. Due to reversibility (Lemma 10), it is sufficient to show that any initial connected shape S_I can be transformed into a spanning line (implying then that any pair of shapes can be transformed to each other via the line and by reversing one of the two transformations). We maintain our focus on transformations that are allowed to break connectivity during their course.

Observe that any initial connected shape S_I of order n can be enclosed in an appropriately positioned $n \times n$ square (called a *box*). Our universal transformation is divided into three phases:

Phase A: Partition the $n \times n$ box into $\sqrt{n} \times \sqrt{n}$ sub-boxes (n in total in order to cover the whole $n \times n$ box). For each sub-box move all nodes in it down towards the bottommost row of that sub-box as follows. Start filling in the bottommost row from left to right, then if there is no more space continue to the next row from left to right and so on until all nodes in the sub-box have been exhausted (resulting in zero or more complete rows and at most one incomplete row). Moving down is done via shortest paths (where in the worst case a node has to move distance $2\sqrt{n}$); see Figure 23.

Phase B: Choose one of the four length- n boundaries of the $n \times n$ box, say w.l.o.g. the left boundary. This is where the spanning line will be formed. Then, transfer every line via a shortest path to that boundary (incurring a maximum distance of $n - \sqrt{n}$ per line).

Phase C: Turn all lines (possibly consisting of more than one line on top of each other), by a procedure similar to that of Figure 6 (e), to end up with a spanning line of n nodes on the left boundary.

However, there are two variants of $\sqrt{n} \times \sqrt{n}$ sub-boxes:

1. *Occupied sub-box:* Denoted by s and contains k nodes of S_I , where $1 \leq k \leq n$.
2. *Unoccupied sub-box:* An empty sub-box (has no nodes).

Now, we show some important properties of *occupied sub-boxes*. Given an *occupied sub-box* s of k nodes, where $1 \leq k \leq n$, therefore, the maximum number of lines which can be formed inside s is at most $\lceil \frac{k}{\sqrt{n}} \rceil$. As mentioned above, those k lines can be aligned horizontally at bottommost rows or vertically at leftmost columns of the occupied sub-box, such as Figure 23. The following lemma gives an upper bound on the number of sub-boxes that any connected S_I can occupy.

► **Lemma 29.** *Any connected shape S_I of order n occupies at most $O(\sqrt{n})$ sub-boxes.*

Proof. It follows directly from Corollary 38, which states that for a given connected shape S_I of n nodes enclosed by a square box of size $n \times n$ and any uniform partitioning of that box into sub-boxes of dimension d , then, it holds that S_I can occupy at most $O(\frac{n}{d})$ sub-boxes. Here, *U-Box-Partitioning* is dividing the $n \times n$ square box into $\sqrt{n} \times \sqrt{n}$ sub-boxes of dimension $d = \sqrt{n}$, therefore, S_I occupies at most $\frac{n}{\sqrt{n}} = O(\sqrt{n})$ sub-boxes. ◀

Below, we prove correctness and analyse the running time of phase A.

► **Lemma 30.** *Starting from any connected shape S_I of order n , Phase A completes in $O(n\sqrt{n})$ steps each.*

Proof. By Lemma 29, let S_I be a connected shape of n nodes occupies \sqrt{n} sub-boxes of size $\sqrt{n} \times \sqrt{n}$ each, and $s \in S_I$ be any *occupied sub-box* of k nodes, where $1 \leq k \leq n$. Then, s performs a trivial line formation to collect all k nodes at its bottommost (or leftmost) boundary. Consider the worst case of a node occupies the top-right corner and wants to gather at the bottom-left of s , therefore, it is incurring a distance (*shortest path*) of at most $\Delta = 2\sqrt{n}$ to arrive there. Now, the k nodes fill in the \sqrt{n} bottommost row from left to right, and then start filling in the next row from left to right and so on until all nodes in s is exhausted. For example, Figure 23 shows the line formation at the bottommost rows of a 6×6 sub-box of $k = 11$ nodes. Consequently, s forms at least one *complete* line of length \sqrt{n} or one *incomplete* of less than \sqrt{n} . Recall that S_I is connected. So there are at most \sqrt{n} *occupied sub-boxes*, which means there are at most $\frac{n}{\sqrt{n}} = O(\sqrt{n})$ lines (*complete or incomplete*) inside all those *occupied sub-boxes*. This is trivial to prove, assume that the \sqrt{n} *occupied sub-box* formed more than \sqrt{n} lines, resulting in $|S_I| > n$, which is a contradiction.



■ **Figure 23** An example of a brute-force line formation to collect all k nodes at bottommost rows of a sub-box of size 6×6 containing $k = 11$ nodes.

With that, the total cost t_1 required to form a line l of w nodes, for all $1 \leq l, w \leq \sqrt{n}$, inside s is:

$$\begin{aligned}
 t &= w \cdot 2\sqrt{n} = \sqrt{n} \cdot 2\sqrt{n} = 2n \\
 &= O(n),
 \end{aligned}
 \tag{21}$$

Multiply 21 by \sqrt{n} , to obtain total steps T_1 to form all \sqrt{n} lines inside all *occupied*

sub-boxes;

$$\begin{aligned}
 T_1 &= \sqrt{n} \cdot t \\
 &= \sqrt{n} \cdot 2n = 2n\sqrt{n} \\
 &= O(n\sqrt{n}).
 \end{aligned} \tag{22}$$

Finally, we conclude that Phase A finishes its course in $O(n\sqrt{n})$ steps. ◀

In phase B, set any (length- n) boundary of the $n \times n$ square box as the *gathering boundary* of all lines formed inside the *occupied sub-boxes* in phase A. Then, the following lemma computes the total steps required to transfer all those lines to that *gathering boundary*.

► **Lemma 31.** *Starting from any connected shape S_I of order n , Phase B completes in $O(n\sqrt{n})$ steps.*

Proof. It follows from Lemmas 29 and 30. Let S_I be a connected shape of order n enclosed by a $n \times n$ box and then partitioned into $\sqrt{n} \times \sqrt{n}$ *occupied sub-boxes* of k nodes each, where $1 \leq k \leq n$. By phase A, there are l lines, for all $1 \leq l \leq \sqrt{n}$, formed inside all \sqrt{n} *occupied sub-boxes*. Without loss of generality, define the left border of the $n \times n$ box as the gathering boundary for all those lines. Now, the distance between any *line* inside an *occupied sub-box* and the defined boundary is no longer than $n - \sqrt{n}$. Thus, the number of steps (distance δ) by which a *line* requires to reach that boundary is therefore $\delta \leq n - \sqrt{n}$. As there are at most \sqrt{n} lines in the *occupied sub-boxes*, then the total steps T_2 for all l lines to transfer and arrive at the left border is at most,

$$\begin{aligned}
 T_2 &= l \cdot \delta \\
 &= \sqrt{n} \cdot (n - \sqrt{n}) = n\sqrt{n} - n \\
 &= O(n\sqrt{n}).
 \end{aligned} \tag{23}$$

By the end of phase B, all \sqrt{n} lines have transferred and arrived at the length- n gathering boundary, where each contributes a node to that boundary. Therefore, in phase C, all those lines ought to push and include themselves to the length- n border, in order to form the goal spanning line of length n nodes. Formally, we give the following Lemma.

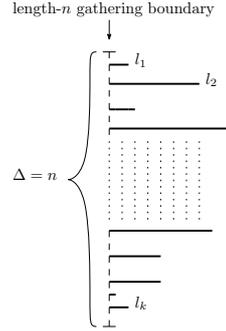
► **Lemma 32.** *Consider any length- n boundary and n nodes forming k lines, where $1 \leq k \leq n$, that are perpendicular to that boundary. Then, by line movements, the k lines require at most $O(n)$ steps to form a line of length n on that boundary. This implies that Phase C is completed in $O(n)$ steps.*

Proof. See the example in Figure 24, where the length- n gathering border denoted by the dashed line, and the lines $\{l_1, l_2, \dots, l_k\}$ of n nodes are depicted by bold black lines, where $1 \leq k \leq \sqrt{n}$. Without doubt, the n nodes shall completely fill up the border of length $\Delta = n$. Now, pick the first line l_1 of k_1 nodes and start to push k_1 into the topmost point of the boundary, until either 1) k_1 are exhausted, or 2) reaching the topmost point of the boundary and still have nodes waiting to be pushed, therefore, it can easily begin to push them down towards the bottommost point of the boundary. By performing the same strategy for all other lines, they shall eventually fill in the length- n border completely by n nodes.

Observe that, in our case, the k lines connect to length- n boundary by k nodes. Therefore, there is still $n - k$ nodes need to be pushed into the boundary. According to Lemma 9, each of

the $n - k$ nodes requires 2 steps to be included, therefore all $n - k$ nodes shall perform a total of $2(n - k)$ steps to fully filled up the boundary of length n . Following that, for any k lines of n nodes that are connected perpendicularly to a border of length- n , *U-Box-Partitioning* pushes the n nodes of k lines into the length- n border by line mechanisms in a total T_3 of at most,

$$\begin{aligned} T_3 &= 2(n - k) = 2(n - \sqrt{n}) \\ &= O(n). \end{aligned} \tag{24}$$



■ **Figure 24** The dashed line indicates the length- n gathering boundary of the $n \times n$ box, whilst the bold black lines represent the k lines of n nodes.

Now, we prove that any connected shape S_I transforms into a line S_L in at most $O(n\sqrt{n})$ steps.

► **Lemma 33.** *U-Box-Partitioning transforms any connected shape S_I into a straight line S_L of the same order n , in $O(n\sqrt{n})$ steps.*

Proof. By the above Lemmas 29, 30 and 32, we sum 22, 23 and 24 to compute the overall moves T as follows;

$$\begin{aligned} T &= T_1 + T_2 + T_3 \\ &= O(n\sqrt{n}) + O(n\sqrt{n}) + O(n) \\ &= O(n\sqrt{n}). \end{aligned} \tag{25}$$

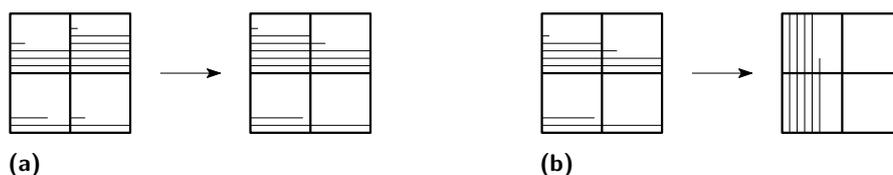
This provides an upper bound $O(n\sqrt{n})$ of *U-Box-Partitioning* to transform any arbitrary connected shape S_I into a single spanning line S_L of the same number of nodes.

Putting Lemma 33 and reversibility (Lemma 10) together gives:

► **Theorem 34.** *For any pair of connected shapes S_I and S_F of the same order n , transformation U-Box-Partitioning can be used to transform S_I into S_F (and S_F into S_I) in $O(n\sqrt{n})$ steps.*

4.2 An $O(n \log n)$ -time Universal Transformation

We now present an alternative universal transformation, called *U-Box-Doubling*, that transforms any pair of connected shapes, of the same order, to each other in $O(n \log n)$ steps.



■ **Figure 25** (a) Pushing left in each $2^i \times 2^i$ sub-box. (b) Cleaning the orientation by aligning (filling) the leftmost columns.

Given a connected shape S_I of order n , do the following. Enclose S_I into an arbitrary $n \times n$ box as in *U-Box-Partitioning* (Section 4.1). For simplicity, we assume that n is a power of 2, but this assumption can be dropped. Proceed in $\log n$ phases as follows: In every phase i , where $1 \leq i \leq \log n$, partition the $n \times n$ box into $2^i \times 2^i$ sub-boxes, disjoint and completely covering the $n \times n$ box. Assume that from any phase $i - 1$, any $2^{i-1} \times 2^{i-1}$ sub-box is either empty or has its k , where $0 \leq k \leq 2^{i-1}$, bottommost rows completely filled in with nodes, possibly followed by a single incomplete row on top of them containing l , where $1 \leq l < 2^{i-1}$, consecutive nodes that are left aligned on that row. This case holds trivially for phase 1 and inductively for every phase. That is, in odd phases, we assume that nodes fill in the leftmost columns of boxes in a symmetric way. Every $2^i \times 2^i$ sub-box (of phase i) consists of four $2^{i-1} \times 2^{i-1}$ sub-boxes from phase $i - 1$, each of which is either empty or occupied as described above.

Consider the case where i is odd, thus, the nodes in the $2^{i-1} \times 2^{i-1}$ sub-boxes are bottom aligned. For every $2^i \times 2^i$ sub-box, move each line from the previous phase that resides in the sub-box to the left as many steps as required until that row contains a single line of consecutive nodes, starting from the left boundary of the sub-box, as shown in Figure 25 (a). With a linear procedure similar to that of Figure 12 (and of *nice shapes*), start filling in the columns of the $2^i \times 2^i$ sub-box from the leftmost column and continuing to the right. If an incomplete column remains, push the nodes in it to the bottom of that column; see Figure 25 (b) for an example.

The case of even i is symmetric, the only difference being that the arrangement guarantee from $i - 1$ is left alignment on the columns of the $2^{i-1} \times 2^{i-1}$ sub-boxes and the result will be bottom alignment on the rows of the $2^i \times 2^i$ sub-boxes of the current phase. This completes the description of the transformation. We first prove correctness:

► **Lemma 35.** *Starting from any connected shape S_I of order n , U-Box-Doubling forms by the end of phase $\log n$ a line of length n .*

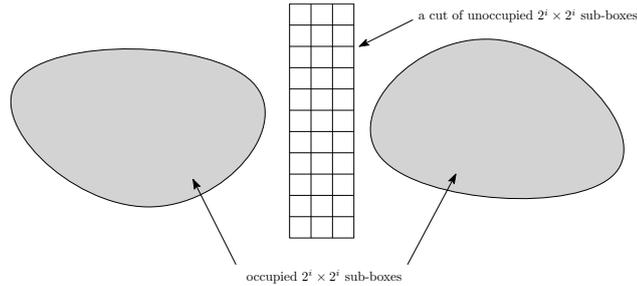
Proof. In phase $\log n$, the procedure partitions into a single box, which is the whole original $n \times n$ box. Independently of whether gathering will be on the leftmost column or on the bottommost row of the box, as all n nodes are contained in it, the outcome will be a single line of length n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

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

► **Lemma 36.** *In every phase i , the “super-shape” formed by the occupied $2^i \times 2^i$ sub-boxes is connected.*

Proof. By induction on the phase number i . For the base of the induction, observe that for $i = 0$ it holds trivially because the initial S_I is a connected shape. Assuming that it holds for phase $i - 1$, we shall now prove that it must also hold for phase i . By the inductive

assumption, the occupied $2^{i-1} \times 2^{i-1}$ sub-boxes form a connected super-shape. Observe that, by the way the original $n \times n$ box is being repetitively partitioned, any box contains complete sub-boxes from previous phases, that is, no sub-box is ever split into more than one box of future phases. Additionally, observe that a sub-box is occupied iff any of its own sub-boxes (of any size) had ever been occupied, because nodes cannot be transferred between $2^i \times 2^i$ sub-boxes before phase $i + 1$. Assume now, for the sake of contradiction, that the super-shape formed by $2^i \times 2^i$ sub-boxes is disconnected. This means that there exists a “cut” of unoccupied $2^i \times 2^i$ sub-boxes as in Figure 26. Replacing everything by $2^{i-1} \times 2^{i-1}$



■ **Figure 26** An example of a “cut” of unoccupied $2^i \times 2^i$ sub-boxes.

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

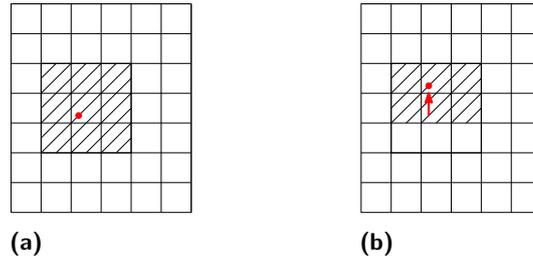
Next, we give an upper bound on the number of occupied sub-boxes in a phase i .

► **Lemma 37.** *Given that U-Box-Doubling starts from a connected shape S_I of order n , the number of occupied sub-boxes in any phase i is $O(\frac{n}{2^i})$.*

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

In order to simplify the argument, whenever S_I occupies another unoccupied sub-box, we will award it a constant number of additional occupations for free and only calculate the additional distance (in nodes) that the shape has to cover in order to reach another unoccupied sub-box. In particular, pick any node of S_I and consider as freely occupied that sub-box and the 8 sub-boxes surrounding it, as depicted in Figure 27 (a). Giving sub-boxes for free can only help the shape, therefore, any lower bound established including the free sub-boxes will also hold for shapes that do not have them (thus, for the original problem). Given that free sub-boxes are surrounding the current node, in order for S_I to occupy another sub-box, at least one surrounding $2^i \times 2^i$ sub-box must be exited. This requires covering a distance of at least 2^i , through a connected path of nodes.

Once this happens, S_I has just crossed the boundary between an occupied sub-box and an unoccupied sub-boxes. Then, by giving it for free at most 5 more unoccupied sub-boxes, S_I has to pay another 2^i nodes to occupy another unoccupied sub-box; see Figure 27 (b). We then continue applying this 5-for-free strategy until all n nodes have been used.



■ **Figure 27** (a) A node of shape S_I in red and the occupied sub-boxes that we give for free to the shape. (b) The shape just exited the sub-box with arrow entering an unoccupied sub-box. By giving the 5 horizontally dashed sub-boxes for free, a distance of at least 2^i has to be travelled in order to reach another unoccupied sub-box.

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

$$N_k \geq l \cdot 2^i \tag{26}$$

But, that leads to

$$k = 8 + l \cdot 5 \Rightarrow l = \frac{k - 8}{5}. \tag{27}$$

Thus, from (26) and (27):

$$N_k \geq \frac{k - 8}{5} \cdot 2^i. \tag{28}$$

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

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

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

As a corollary of this, we obtain:

► **Corollary 38.** *Given a uniform partitioning of $n \times n$ square box containing a connected shape S_I of order n into $d \times d$ sub-boxes, it holds that S_I can occupy at most $O(\frac{n}{d})$ sub-boxes.*

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

► **Lemma 39.** *Starting from any connected shape of n nodes, U-Box-Doubling performs $O(n \log n)$ steps during its course.*

Proof. We prove this by showing that in every phase i , $1 \leq i \leq \log n$, the transformation performs at most a linear number of steps. We partition the occupied $2^i \times 2^i$ sub-boxes into two disjoint sets, B_1 and B_0 , where sub-boxes in B_1 have at least 1 *complete line* (from the previous phase), i.e., a line of length 2^{i-1} , and sub-boxes in B_0 have 1 to 4 *incomplete lines*, i.e., lines of length between 1 and $2^{i-1} - 1$. For B_1 , we have that $|B_1| \leq n/2^{i-1}$. Moreover, for every complete line, we pay at most 2^{i-1} to transfer it left or down, depending on the parity of i . As there are at most $n/2^{i-1}$ such complete lines in phase i , the total cost for this is at most $2^i \cdot (n/2^{i-1}) = n$.

Each sub-box in B_1 may also have at most 4 incomplete lines from the previous phase, as in Figure 25 left, where at most two of them may have to pay a maximum of 2^{i-1} to be transferred left or down, depending on the parity of i (as the other two are already aligned). As there are at most $n/2^{i-1}$ sub-boxes in B_1 , the total cost for this is at most $2 \cdot 2^{i-1} \cdot (n/2^{i-1}) = 2n$.

Therefore, the total cost for pushing all lines towards the required border in B_1 sub-boxes is at most:

$$n + 2n = 3n. \tag{29}$$

For B_0 , we have (by Lemma 37) that the total number of occupied sub-boxes in phase i is at most $5(n/2^i) + 8$, therefore, $|B_0| \leq 5(n/2^i) + 8$ (taking into account also the worst case where every occupied sub-box may be of type B_0). There is again a maximum of 2 incomplete lines per such sub-box that need to be transferred a distance of at most 2^{i-1} , therefore, the total cost for this to happen in every B_0 sub-box is at most:

$$2 \cdot 2^{i-1} \left(5 \cdot \frac{n}{2^i} + 8 \right) = 5n + 8 \cdot 2^i \leq 13n. \tag{30}$$

By paying the above costs, all occupied sub-boxes have their lines aligned horizontally to their left or vertically to their bottom border, and the final task of the transformation for this phase is to apply a linear procedure in order to fill in the left (bottom) border of the $n \times n$ box. This procedure costs at most $2k$ for every k nodes aligned as above (Lemma 9), therefore, in total at most:

$$2n. \tag{31}$$

This completes the operation of *U-Box-Doubling* for phase i . Putting (29), (30) and (31) together, we obtain that the total cost T_i , in steps, for phase i is,

$$\begin{aligned} T_i &\leq 3n + 13n + 2n \\ &= 18n. \end{aligned}$$

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

$$\begin{aligned} T &\leq 18n \cdot \log n \\ &= O(n \log n). \end{aligned}$$

◀

Finally, together Lemma 35, Lemma 39 and reversibility (Lemma 10) imply that:

► **Theorem 40.** *For any pair of connected shapes S_I and S_F of the same order n , transformation U-Box-Doubling can be used to transform S_I into S_F (and S_F into S_I) in $O(n \log n)$ steps.*

5 Conclusions

In this work, we studied a new linear-strength model extending upon the model of [21, 30]. The nodes can now move in parallel by translating a line of any length by one position in a single time-step. This model, having the model of [21, 30] as a special case, adopts all its transformability results (including universal transformations). Then, our focus naturally turned to investigating if pushing lines can help achieve a substantial gain in performance (compared to the $\Theta(n^2)$ of those models). Even though it can be immediately observed that there are instances in which this is the case (e.g., initial shapes in which there are many long lines, thus, much initial parallelism to be exploited), it was not obvious that this holds also for the worst case. By identifying the diagonal as a potentially worst-case shape (essentially, because in it any parallelism to be exploited does not come for free), we managed to first develop an $O(n\sqrt{n})$ -time transformation for transforming the diagonal into a line, then to improve upon this by two transformations that achieve the same bound while preserving connectivity, and finally to provide an $O(n \log n)$ -time transformation (that breaks connectivity). Going one step further, we developed two universal transformations that can transform any pair of connected shapes to each other in time $O(n\sqrt{n})$ and $O(n \log n)$, respectively.

There is a number of interesting problems that are opened by this work. The obvious first target (and apparently intriguing) is to answer whether there is an $o(n \log n)$ -time transformation (e.g., linear) or whether there is an $\Omega(n \log n)$ -time lower bound matching our best transformations. We suspect the latter, but do not have enough evidence to support or prove it. The tree representation of the problem that we discuss in Section 3.5 (see, e.g., Figure 22), might help in this direction. Moreover, we didn't consider parallel time in this paper. If more than one line can move in parallel in a time-step, then are there variants of our transformations (or alternative ones) that further reduce the running time? In other words, are there parallelisable transformations in this model? In particular, it would be interesting to investigate whether the present model permits an $O(\log n)$ parallel time (universal) transformation, i.e., matching the best transformation in the model of Aloupis *et al.* [2]. It would also be worth studying in more depth the case in which connectivity has to be preserved during the transformations. In the relevant literature, a number of alternative types of grids have been considered, like triangular (e.g. in [14]) and hexagonal (e.g., in [38]), and it would be interesting to investigate how our results translate there. Finally, an immediate next goal is to attempt to develop distributed versions of the transformations provided here.

References

- 1 Greg Aloupis, Nadia Benbernou, Mirela Damian, Erik D Demaine, Robin Flatland, John Iacono, and Stefanie Wuhler. Efficient reconfiguration of lattice-based modular robots. *Computational geometry*, 46(8):917–928, 2013.
- 2 Greg Aloupis, Sébastien Collette, Erik D Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhler. Reconfiguration of cube-style modular robots using $O(\log n)$ parallel moves. In *International Symposium on Algorithms and Computation*, pages 342–353. Springer, 2008.
- 3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, March 2006.
- 4 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

- 5 Aaron T Becker, Erik D Demaine, Sándor P Fekete, Jarrett Lonsford, and Rose Morris-Wright. Particle computation: complexity, algorithms, and logic. *Natural Computing*, pages 1–21, 2017.
- 6 Julien Bourgeois and Seth Copen Goldstein. Distributed intelligent mems: progresses and perspectives. *IEEE Systems Journal*, 9(3):1057–1068, 2015.
- 7 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.
- 8 Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012. doi:10.1137/100796534.
- 9 Alejandro Cornejo, Fabian Kuhn, Ruy Ley-Wild, and Nancy Lynch. Keeping mobile robot swarms connected. In *Proceedings of the 23rd international conference on Distributed computing, DISC’09*, pages 496–511, Berlin, Heidelberg, 2009. Springer-Verlag.
- 10 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.
- 11 Joshua J Daymude, Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.
- 12 Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin-Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2006.
- 13 Erik D Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.
- 14 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot—a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 220–222, 2014.
- 15 Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, page 21. ACM, 2015.
- 16 Zahra Derakhshandeh, Robert Gmyr, Andréa W. 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. doi:10.1007/s00446-019-00350-6.
- 18 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Line recovery by programmable particles. In *Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN ’18*, pages 4:1–4:10, New York, NY, USA, 2018. ACM. doi:10.1145/3154273.3154309.
- 19 David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55:78–88, 2012.
- 20 Shawn M Douglas, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William M Shih. Self-assembly of dna into nanoscale three-dimensional shapes. *Nature*, 459(7245):414, 2009.
- 21 Adrian Dumitrescu and János Pach. Pushing squares around. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 116–123. ACM, 2004.

- 22 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.
- 23 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.
- 24 Sándor Fekete, Andréa W Richa, Kay Römer, and Christian Scheideler. Algorithmic foundations of programmable matter (Dagstuhl Seminar 16271). In *Dagstuhl Reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. Also in *ACM SIGACT News*, 48.2:87–94, 2017.
- 25 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3(2):1–185, 2012.
- 26 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.
- 27 Robert A Hearn and Erik D Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.
- 28 Ara N Knaian, Kenneth C Cheung, Maxim B Lobovsky, Asa J Oines, Peter Schmidt-Neilsen, and Neil A Gershenfeld. The milli-motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1447–1453. IEEE, 2012.
- 29 Evangelos Kranakis, Danny Krizanc, and Euripides Markou. The mobile agent rendezvous problem in the ring. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–122, 2010.
- 30 Othon Michail, George Skretas, and Paul G. Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 2019. doi:<https://doi.org/10.1016/j.jcss.2018.12.001>.
- 31 Othon Michail and Paul G. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016. doi:<http://dx.doi.org/10.1007/s00446-015-0257-4>.
- 32 Othon Michail and Paul G Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2), 2018.
- 33 An Nguyen, Leonidas J 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 W. K. Rothmund 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, 2000. doi:10.1145/335305.335358.
- 35 Paul WK Rothmund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.
- 36 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.
- 37 Masahiro Shibata, Toshiya Mega, Fukuhito Ooshita, Hirotugu Kakugawa, and Toshimitsu Masuzawa. Uniform deployment of mobile agents in asynchronous rings. *Journal of Parallel and Distributed Computing*, 119:92–106, 2018.
- 38 Jennifer E Walter, Jennifer L Welch, and Nancy M Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.
- 39 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.
- 40 Damien Woods, Ho-Lin 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.

- 41 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010.
- 42 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.
- 43 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.