Centralised Connectivity-Preserving Transformations for Programmable Matter: A Minimal Seed Approach

Matthew Connor*, Othon Michail, Igor Potapov

Department of Computer Science, University of Liverpool, Liverpool, United Kingdom

Abstract

We study a model of programmable matter systems consisting of n devices lying on a 2-dimensional square grid which are able to perform the minimal mechanical operation of rotating around each other. The goal is to transform an initial shape A into a target shape B. We investigate the class of shapes which can be constructed in such a scenario under the additional constraint of maintaining global connectivity at all times. We focus on the scenario of transforming *nice shapes*, a class of shapes consisting of a central line L where for all nodes u in S either $u \in L$ or u is connected to L by a line of nodes perpendicular to L. We prove that by introducing a minimal 3-node seed it is possible for the canonical shape of a line of n nodes to be transformed into a nice shape of $n-1$ nodes. We use this to show that a 4-node seed enables the transformation of nice shapes of size n into any other nice shape of size n in $O(n^2)$ time. We leave as an open problem the expansion of the class of shapes which can be constructed using such a seed to include those derived from nice shapes.

Keywords: Programmable matter, Transformation, Reconfigurable robotics, Shape formation, Centralised algorithms.

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

Email addresses: M.connor3@liverpool.ac.uk (Matthew Connor), Othon.Michail@liverpool.ac.uk (Othon Michail), potapov@liverpool.ac.uk (Igor Potapov)

1. Introduction

Programmable matter refers to matter which can change its physical properties algorithmically. This means that the change is the result following the procedure of an underlying program. The implementation of the program can either be a system level external centralised algorithm or an internal decentralised algorithm executed by the material itself. The model for such systems can be further refined to specify properties that are relevant to real-world applications, for example connectivity, colour [CLS+11] and other physical properties. The result of this is the development of programmable materials such as self-assembling DNA molecules [Dot12, Rot06]. In addition, systems which rely on large collectives of identical robots have been developed, for example the Kilobot system [RCN14] and the Robot Pebbles system [GKR10]. Another interesting implementation is Millimotein [KCL+12], a system where programmable matter folds itself into arbitrary 3D shapes. The CATOMS system [TPB19, TPB20] is a further implementation which constructs 3D shapes by first creating a "scaffolding structure" as a basis for construction. It is expected that applications in further domains such as molecular computers and self-repairing machines may become apparent in the long-term.

As the development of these systems continues, it becomes increasingly necessary to develop theoretical models which are capable of describing and explaining the emergent properties, possibilities and limitations of such systems in an abstract and fundamental manner. To this end, models have been developed for programmable matter. For example, algorithmic self-assembly [Dot12] focuses on programming molecules like DNA to grow in a controllable way, and the Abstract Tile Assembly Model [RW00, Win98], as well as the nubot model [WCG+13] and the SILBOT model [DDD+20], have been developed for this area. The Tile Automata model [CGSW21] is a recent model combining features of both Cellular Automata and the 2-Handed Model of self-assembly. Network Constructors [MS16] uses the Population Protocol model [AAD+06] based on a population of finite-automata interacting randomly as the basis for a new model where the automata are able to create networks by forming connections with each other. The latter model is formally equivalent to a restricted version of chemical reaction networks, which "are widely used to describe information processing occurring in natural cellular regulatory networks" [SCWB08, Dot13]. Finally there is extensive research into the amoebot model [DDG+14, DDG+18, DGR+15, DGR+16], where finite automata on a triangle lattice follows a distributed algorithm to

achieve a desired goal. A recent development of this model [DRS21] introduced concurrency control.

Recent progress in this direction has been made in a previous paper [AMP20], covering questions related to a specific model of programmable matter where nodes exist in the form of a shape on a 2D grid and are capable of performing two specific movements: rotation around each other and sliding a node across two other nodes. They presented 3 problems: transformations with only rotations (Rot-Transformability), transformations with rotations with the restriction that shapes must always remain connected (RotC-Transformability) and transformations with both rotation and sliding movements (RS-Transformability). For Rot-Transformability they prove universal transformation between any pair of colour-consistent shapes which are not blocked, however they leave universal RotC-Transformability as an open problem. Such transformations are highly desirable due to the large numbers of programmable matter systems which rely on the preservation of connectivity, and the minimal, easily implemented nature of rotational-only mechanisms. Nice shapes are a general class, the transformation of which can act a first step towards the achievement of universal transformation, and progress towards more general transformations relying on perimeter traversal. Progress in a very similar direction was made in another paper [AAD$^+$21], which used a similar model but allowed for a greater range of movement, for example "leapfrog" and "monkey" movements. They accomplished universal transformation in $O(n^2)$ movements using a "bridging" procedure which added up to 5 nodes during the procedure as necessary in a manner similar to the seed idea from the previous paper.

2. Contribution

We investigate which families of connected shapes can be transformed into each other via rotation movements without breaking connectivity.

We consider the case of programmable matter on a 2D grid which is only capable of performing rotation movements, defined as the 90° rotation of a node a around one of the two vertices of the edge it shares with a neighbouring edge-adjacent node b, so long as the goal and intermediate cells are empty. All nodes must be *edge connected*, meaning that at every time step there must be a path from any arbitrary node to any other node crossing only spaces occupied by nodes via edges. Our algorithms are *centralised*, using external procedures to transform shapes, and *sequential*, where at most one

node moves per time step, therefore focusing on the questions of the feasibility and complexity of the transformations.

We assume the existence of a *seed*, a group of nodes in a shape S which are placed in empty cells neighbouring a shape A to create a new connected shape which is the unification of S and A. Seeds allow shapes which are blocked or incapable of meaningful movement to perform otherwise impossible transformations. The use of seeds was established in a previous work [MSS19], and more recently shown to enable universal reconfiguration in the context of connectivity preserving transformations [AAD$^+$21], however to our knowledge there has been no attempt to investigate this problem using a seed which is a connected shape fully introduced before the transformation is initiated.

We first study blocked shapes, where our goal is to define the class of shapes which are *blocked*, or incapable of moving any node without a seed. We show that shapes of this class consist of nodes which are surrounded by diagonal lines in the shape of a rhombus, or overlapping rhombuses which may be connected by lines. We then investigate the transformation of nice shapes. A *Nice Shape* (defined in [AMP20]) is a shape S which has a central line L where for all nodes u in S either $u \in L$ or u is connected to L by a line of nodes perpendicular to L. We provide a lower bound of $\Omega(n^2)$ for transforming a line of n nodes into a nice shape in the rotation-only setting. We show that it is possible to transform such a line into a nice shape of $n-1$ nodes using a 3-node seed in $O(n^2)$ time. We then demonstrate that it is possible to transform nice shapes of size n into other nice shapes of size n by using the canonical shape of a line and a 4-node seed in $O(n^2)$ time. We provide an algorithm to implement this transformation and give time bounds for it. We then provide further directions for research.

In Section 3, we formally define the model of connectivity-preserving programmable matter used in this paper. In Section 4 we give our lower bounds. In Section 5 we provide our algorithm for the construction of nice shapes where the colour of nodes added to each side of the line always alternates, then generalise first to all nice shapes and second to the class of shapes made up of nice shapes. In Section 6 we conclude and give directions for potential future research.

3. Model

The programmable matter systems considered in this paper operate on a 2D square grid, with each cell being uniquely referred to by its $y \geq 0$ and $x \geq 0$ coordinates. Such a system consists of a set S of n modules, called nodes throughout. Each node may be viewed as a spherical module fitting inside a cell. At any given time, each node $u \in S$ occupies a cell in the grid $o(u) = (o_y(u), o_x(u)) = (i, j)$ (where i corresponds to a row and j to a column of the grid) and each cell can be occupied by at most one node at a time. At any given time t, the positioning of nodes on the grid defines an undirected neighboring relation $E(t) \subset S \times S$, where $\{u, v\} \in E$ iff $o_y(u) = o_y(v)$ and $|o_x(u) - o_x(v)| = 1$ or $o_x(u) = o_x(v)$ and $|o_y(u) - o_y(v)| = 1$, that is, if u and v are either horizontal or vertical neighbors on the grid, respectively. We say that two nodes are *edge-adjacent* if such a relation exists between them. A more informative and convenient way to define the system at any time t is the mapping $P_t : \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0} \to \{0, 1\}$.where $P_t(i, j) = 1$ iff cell (i, j) is occupied by a node. At any given time t, $P_t^{-1}(1)$ defines a shape. Such a shape is called *connected* if $(S, E(t))$ defines a connected graph.

In general, shapes can *transform* to other shapes via a sequence of one or more movements of individual nodes. We consider only one type of movement: rotation. In this movement, a single node moves relative to one or more neighboring nodes. A single rotation movement of a node u is a 90° rotation of u around one of its neighbors. Let (i, j) be the current position of u and let its neighbor be v occupying the cell $(i - 1, j)$ (i.e., lying below u). Then u can rotate 90° clockwise (counterclockwise) around v iff the cells $(i, j + 1)$ and $(i - 1, j + 1)$ ($(i, j - 1)$ and $(i - 1, j - 1)$, respectively) are both empty. By rotating the whole system 90°, 180°, and 270°, all possible rotation movements are defined analogously.

Let A and B be two connected shapes. We say that A transforms to B via a rotation r, denoted $A \xrightarrow{r} B$, if there is a node u in A such that if u applies r, then the shape resulting after the rotation is B. We say that A transforms in one step to B (or that B is reachable in one step from A), denoted $A \to B$, if $A \xrightarrow{r} B$ for some rotation r. We say that A transforms to B (or that B is reachable from A) and write $A \rightsquigarrow B$, if there is a sequence of shapes $A = C_0, C_1, ..., C_t = B$, such that $C_i \to C_{i+1}$ for all $0 \leq i < t$. Rotation is a reversible movement, a fact that we use in our results.

A line is a connected shape where every node lies on the same column or the same row. A nice shape N is defined as a shape which has a central line

L where for all nodes u either $u \in L$ or u is connected to L by a line of nodes perpendicular to L.

Consider a black and red checkered colouring of the 2D grid, like that of a chessboard. Then any shape S consists of $b(S)$ nodes which lie on black cells and $r(S)$ nodes which lie on red cells. Two shapes A and B are *colour consistent* if $b(A) = b(B)$ and $r(A) = r(B)$. Because rotations are the only permissible move, it is impossible for a node to change colour. This is depicted in Figure 1. Therefore, any two shapes for which a solution to Rot-Transformability (and by extension RotC-Transformability) exists must be colour-consistent.

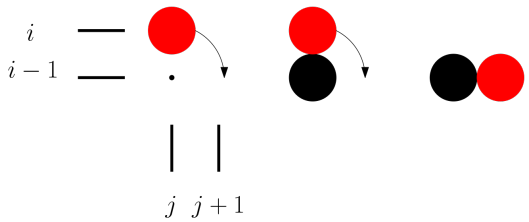


Figure 1: An example of a clockwise rotation movement. A node on the black dot (in row $i-1$) and empty cells at positions $(i, j+1)$ and $(i-1, j+1)$ are required for this movement. Red nodes, used throughout the paper, appear grey in print.

If S is not a nice shape and $S = A \cup B$ where A is a nice shape, we call B the *waste* of the shape S and say that B is $|B|$ waste. A *configuration* of a shape is an arrangement of the nodes of the shape on a 2D grid where each node is uniquely identifiable.

Definition 1. *Let $N_{n-w} = S \cup T$ be the class of shapes with n nodes, where S is a nice shape of size $n - w$ and T is w waste.*

Definition 2. *Let $M_{n-w} = S \cup T$ be the class of shapes with n nodes, where S is a nice shape of size $n - w$ and T is w waste for which the following property holds: For all lines above and below L, where L is the central line of S, the node at the end of each line is the opposite colour to the node at the end of its nearest neighbouring lines.*

4. Infeasible Transformations and the Time Lower Bound

In this section, we cover a series of transformations which are infeasible, meaning that they rely on the ability to move $O(n)$ nodes but exist in a

scenario where moving at most $O(1)$ is possible. We first define the class of shapes which are *blocked*, meaning there is no potential movement available for any node. We then define the class of *k-blocked shapes*, where the set of potential transformations has at most k configurations before any configuration is repeated. We only consider shapes for $k = 0$. Note however, that the end points of a straight line are blocked for $k = 2$, and the whole line for $k = 8$. We show that it is necessary for a seed to have at least 3 nodes if it is to be connected and to enable the movement of more than 5 nodes in a horizontal line. Finally, we provide a lower bound of $\Omega(n^2)$ movements for the problem of transforming a line into a nice shape.

Two nodes are *vertex-adjacent* if their cells share a common vertex. A node w is an *interior* node if for each of the cells x edge-adjacent to w either there is a node occupying x or there are two nodes y and z such that y and z are edge-adjacent to x and vertex-adjacent to w. A node is an *exterior* node if it is not an interior node. These relations are depicted in Figure 2. Note that when rotating nodes around each other, we use a special abbreviation, depicted in Figure 3.

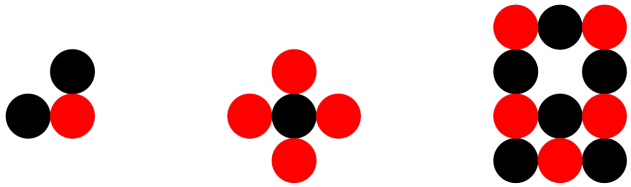


Figure 2: Examples of adjacency and interior/exterior nodes. The black nodes in the first image are edge-adjacent to the red node and vertex-adjacent to each other. The black node in the second image is an interior node surrounded by exterior nodes. The black node in the middle in the third image is also an interior node.

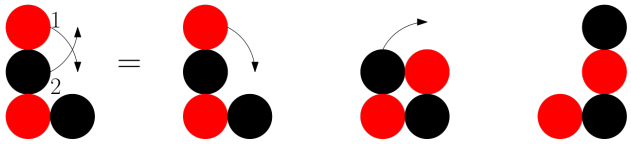


Figure 3: The rotation on the left is an abbreviated version of the rotations on the right, used throughout the paper. The numbers represent the order of rotations.

Lemma 1. *An arbitrary shape A which does not have to preserve connectivity is blocked if and only if there is only 1 node or every exterior node has no edge connections to any other exterior node.*

Proof. A shape with one node is trivially blocked because there is nothing for it to rotate around. Otherwise, a shape consists of interior nodes connected to each other with the possibility of one-node gaps, surrounded by exterior nodes which form diagonal lines due to the edge-adjacency restriction. Interior nodes are blocked by the nodes that surround them, either because the grid space is filled by an edge-adjacent node or the two vertex-adjacent nodes block the rotation movement. Exterior nodes can only rotate around nodes which are edge-connected, which must be interior nodes. The nodes which surround an interior node, whether edge or vertex connected, always block an exterior node from moving, regardless of whether they are interior or exterior nodes themselves. Conversely, if there is an exterior node which is edge-connected to an exterior node, the exterior node can rotate into the empty space which it provides. □

This creates a shape which is similar to one or more overlapping rhombuses, for example Figure 4. Furthermore, with the additional condition of connectivity preservation, it is possible for these shapes to be connected by straight lines resembling a geometric cactus form of a cactus graph with these shapes instead of cycles.

Let S be an arbitrary shape with $B_1 \cup B_2 \cup \ldots \cup B_k = B \subseteq S$ as the set of all shapes which are blocked under the conditions of Lemma 1 which exist within S. Each shape B_i is *maximal*, meaning that $B_i \cup S'$ is a non-blocked shape, for all $S' \subset S$ edge-adjacent to B_i. Let $G(S)$ be a graph formed by first introducing one vertex for every $B_i \subset B$ and then one vertex for every other node in S. For all vertices u and v in $G(S)$, add an edge between them iff their corresponding nodes or blocked shapes in S are edge-adjacent.

Theorem 1. *An arbitrary shape S is blocked under the condition of connectivity preservation if the graph $G(S)$ is a tree, and every leaf in G is a blocked shape.*

Proof. By Lemma 1, each of the blocked shapes is incapable of movement. Because G is a tree, any vertex v in G which does not correspond to a blocked shape cannot be part of a cycle. Because every leaf is a blocked shape, all such v must be interior vertices with at least two edges. When a node rotates, it can only maintain edge connectivity with the node it rotates around. Therefore, the rotation of any of the corresponding nodes would violate connectivity, equivalent to bisecting G into two disconnected components. □

We define a *connected* seed to be a seed which is a connected shape by itself. We next show that a connected seed of size $s < 3$ on a line of length n occupying the grid spaces $(0, 0)$ to $(n - 1, 0)$ can only move a constant number of nodes (5). Note that if the seed is disconnected a 2-node seed is able to enable non-trivial movement by taking positions such that they can work with both ends of the line at the same time. The position of the seed can also be symmetrical so long as the destination of the pairs is also mirrored.

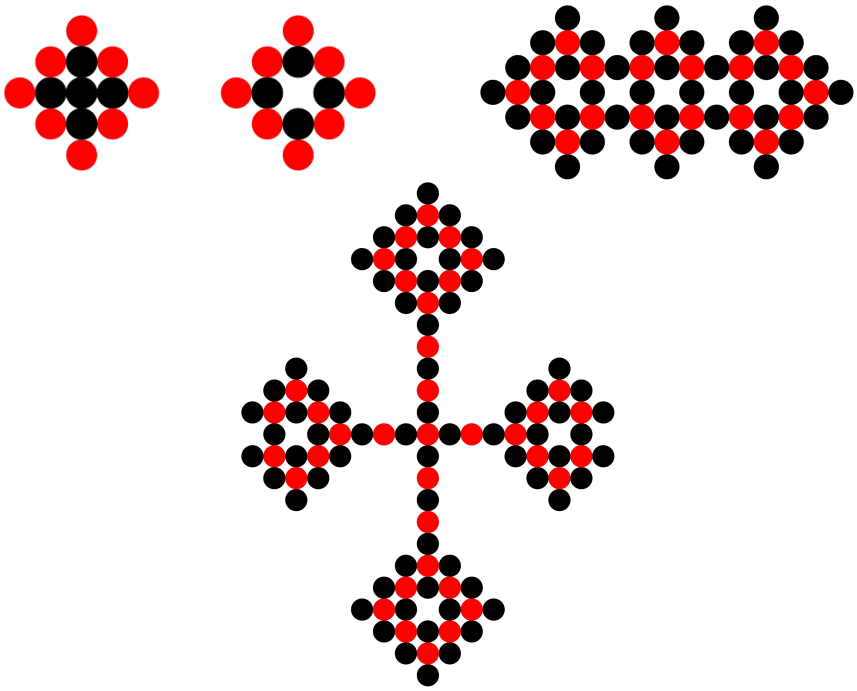


Figure 4: Examples of blocked shapes.

Lemma 2. *Any line of nodes S of length n can move at most five nodes from the line with any k-seed of size $k < 3$ nodes.*

Proof. A line without seeds, with the connectivity preserving condition and with only rotation movements cannot do anything other than rotate the two nodes at each end point. With a one node seed, the only possible action is for the node to be positioned in the cell $(2, 1)$ (or any equivalent symmetrical position) and rotate the end node at $(0, 0)$ to $(1, 1)$ to form a pair. This is equivalent to having a two node seed on a line of length $n - 1$. With a

two node seed, it can only interact with an end node and with each node in the positions $(0,1),(1,1)$ or $(1,1),(2,1)$ (or any symmetrical position). In the former case, the end node can only rotate around the node in $(0,1)$ because it depends on it to maintain connectivity. In the latter case, the end node can rotate to $(0,1)$. This allows the node in $(1,0)$ and the node next to it (i.e. in $(2,0)$) to rotate. However, they cannot move much without breaking connectivity thanks to a reliance on the nodes in $(3,0)$ and $(3,1)$ for connectivity which restricts movement.

Therefore, if we start with a one node seed, form a two node pair, rotate the node in $(0,1)$ to $(2,1)$, move the two nodes in $(1,0)$ and $(2,0)$ and the node at the other end of the line, we have exhausted all possibilities to maximise the number of moving nodes without using a seed of size $k \geq 3$. □

4.1. Time and Seed Lower Bounds for Line Transformations

We now give a lower bound on the running time of any strategy which transforms a line into a nice shape.

Lemma 3. *There exists a nice shape such that any strategy which transforms a line of n nodes into the nice shape requires $\Omega(n^2)$ time steps in the worst case.*

Proof. Our goal is to transform the line of length n into a nice shape with two lines of length $n/2$, one horizontal line and one vertical line above and perpendicular to the node in the center of the horizontal line.

Let c be the node in the line which the vertical line will be constructed above.

To avoid breaking connectivity, it is necessary for M to transfer nodes from the ends of the line to the space above c. Each of these nodes must perform $\lceil n/2 \rceil$ movements assisted by M. While the distance to the c grows shorter with each node transferred, the line above c grows longer. Therefore, given that $\lceil n/2 \rceil$ nodes must move towards and onto the vertical line, the total number of movements m is given by $\lceil n/2 \rceil \cdot \lceil n/2 \rceil = \Omega(n^2)$. □

Since Theorem 1 shows that blocked shapes (including blocked nice shapes, see the top-left shape of Figure 4) exist, it is necessary for any algorithm which solves RotC-Transformability for nice shapes to be aided, and we aid this transformation with a seed. By Lemma 2, this seed must be of size ≥ 3 to enable non-trivial transformations of a straight line. Finally, Lemma 3 shows that $O(n^2)$ is the best time bound we can hope for.

5. Transformation for Nice Shapes

In this section, we investigate the possibilities related to the transformation of shapes which are connectivity preserving. We focus on the problem of converting a nice shape of $O(n)$ nodes into any other nice shape of $O(n)$ nodes using an $O(1)$ seed. We do this by showing we can transform the canonical shape of a line with $O(n)$ nodes into any nice shape. Due to reversibility, it follows that any nice shape can be transformed into such a line, and then into another nice shape. More specifically, we first provide a solution for the variant of this problem (which we call M) where all the lines perpendicular to a central line L in the nice shape are such that the node at the end of each line is the opposite colour to the node at the end of its nearest neighbouring lines. We then prove that slight modifications to the method of construction allow for the class of all nice shapes to be constructed. Our methods construct a shape which is a union of a nice shape with constant waste $O(1)$.

We start with a shape S which is a line of length n occupying the cells $(0, 0)$ to $(n-1, 0)$. We are allowed to attach at an arbitrary position a k-node seed forming an arbitrary connected shape of size k to the line. We use a 3-node seed as this is the minimum size which allows us to move more than 5 nodes without breaking connectivity. This initial configuration is depicted in Figure 5. It is possible for our results to apply to a disconnected 2-node seed with a slightly modified procedure but with higher waste. We place the seed in a specific position as the connected 3-node seed is incapable of movement. We sketch the line to nice shape proof in the following subsection.

Figure 5: The line with the seed attached.

5.1. Line to Nice Shape

We begin by considering the transformation of a line into a shape from M_{n-w}. The restriction of this class guarantees that no node with the "wrong" colour is ever in the position to block construction. The process of construction is therefore simpler. A method of dealing with these nodes, introduced later, will allow the restriction to be dropped, yielding the construction of N_{n-w}. Our first result is the following theorem:

Theorem 2. *A line of length n can be transformed to any given nice shape in the class M_{n-1} using a 3-node seed in $O(n^2)$ time.*

To solve this problem, we follow a strategy of having nodes rotate onto the horizontal line with the help of the 3-node seed and then constructing lines perpendicular to the horizontal line using the nodes. Additionally, we move 4 nodes below the line and on the opposite side to the seed. These nodes can then replicate the behaviour of the seed on the other side of the line, allowing for construction to occur below as well as above the line. Because their behaviour is the same, we refer to the seed and the group of nodes on the other side of the line as *builders*. As a result, the horizontal line becomes the central line L of the nice shape, and the vertical lines become the lines of nodes perpendicular to L. Finally, the seed and a single node which aid construction cannot be incorporated into the final shape and are discarded as waste.

To prove that this is possible, we define three algorithmic procedures. The first procedure, *RaiseNodes*, allows a builder to move two nodes at a time from the horizontal line. These nodes combine with the builder to form a 5-node cluster. This cluster can be broken if necessary into a 3-node line and a 2-node line, allowing the 2-node line to move by having each node rotate around the other. The second procedure, *MirrorSeed*, is the procedure for creating the second builder below the horizontal line. It accomplishes this by moving two of the 2-node lines to the end of the horizontal and then rotating nodes in such a way that the four nodes are "pushed" through the horizontal and to the other side. The final procedure, *DepositNodes*, collects nodes from the horizontal line and deposits them in any reachable location. We will show that the set of reachable locations enables the construction of any nice shape.

As a result, we end up with nice shapes where the central column L corresponds to what is left of the original horizontal line. However, the resulting nice shapes $\mathcal{M} \subset \mathcal{N}$, where $\mathcal{N}$ is the set of all nice shapes, have only even lines. This is due to the construction procedures, which place two nodes at a time. We therefore provide additional movements that allow us to expand the set of nice shapes which can be constructed to include all nice shapes. We perform this for a special case and then generalise to drop this assumption and get any nice shape.

5.2. RaiseNodes

We use a 3-node seed in the cells $(2,1)$, $(3,1)$, $(4,1)$ for our operations as, by Lemma 2, a two node seed is incapable of helping nodes to move.

We call the first operation *RaiseNodes*. For this operation we use the 3-node seed to move nodes from the horizontal line such that they are on top of the horizontal line. In the process, the 3-node seed moves along the horizontal line such that each node moves from its original position $(x,1)$ to $(x+2,1)$.

We can raise two nodes at a time as a pair. The result can also be interpreted as a shape consisting of 5 nodes, which we refer to as a 5-node seed. Moving the pair of nodes once they are on the line is a trivial process. Each node rotates around the other node, alternating their relative positions within the two node shape.

The following lemma shows that these operations are possible.

Lemma 4. *Using a 3-node seed in the cells* $(2,1)$ *to* $(4,1)$*, it is possible to move 2 nodes from the line such that the 3-node seed is converted into a 5-node seed.*

Proof. The seed can only be placed in the cells specified as a three node line is incapable of translating to another position.

First, the leftmost node of the horizontal line at $(0,0)$ must rotate above the line. Then the third node in the seed at $(3,1)$ can rotate right, creating a space for the node just raised from the line, which then takes its place. By moving the two nodes in $(4,1)$ and $(2,1)$ one space right in the same manner, the four nodes above the line have moved two spaces to the right, creating room for another node to be raised from the line. □

Figure 6 depicts the process.

In addition, we can move the 5-node line to the right by following a series of specific rotations (see Figure 7). Furthermore, the technique of moving the 4-node line to the right from Lemma 4 can be used to move any line of even length. We can therefore move any line of odd length to the right by first splitting it into a 5-node line and a line of even length and then moving them separately. As a result, the process of raising nodes from the line can be repeated indefinitely so long as the nodes on the line have the space to be moved out of the way of the operation.

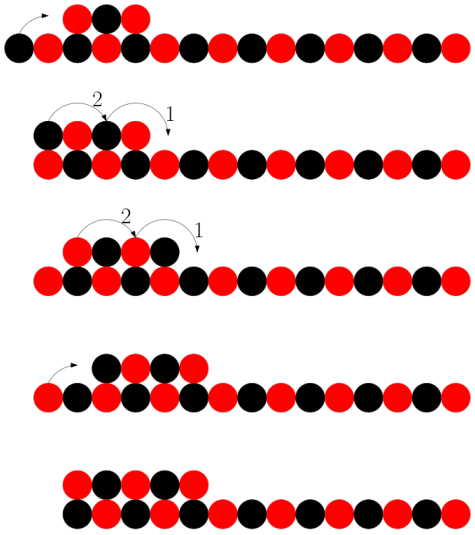


Figure 6: Raising nodes from the line. All numbers refer to the sequence of operations. Multiple rotations around the same node are represented by a single long arrow.

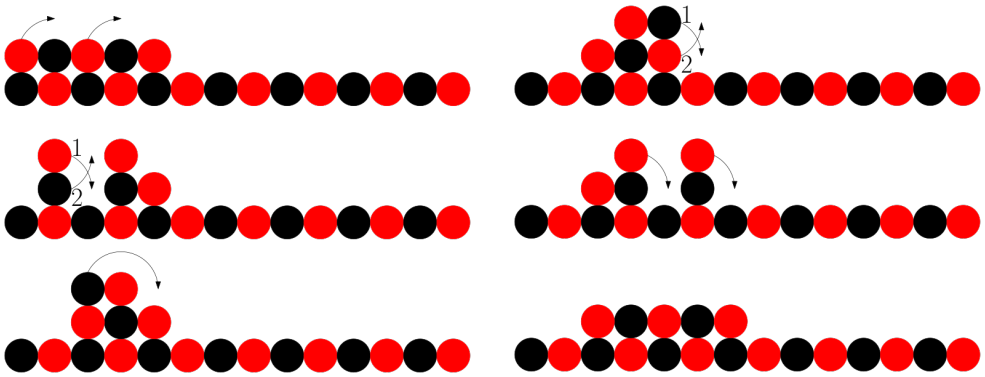


Figure 7: Moving a line of 5 nodes. Figures should be read vertically.

5.3. MirrorSeed

We now use RaiseNodes for our next operation, *MirrorSeed*, to place four nodes at the opposite side of the line (i.e. $(n-4,1)$ to $(n-1,1)$) and then push them through and below the line, creating a four node mirror of our original seed in the cells $(n-4,-1)$ to $(n-1,-1)$. Having a mirror of the original seed allows us to perform construction operations on the bottom of the horizontal line. We do this in 3 steps: raise four nodes using RaiseNodes twice to create a 7-node line, position four of the nodes at the end of the line and rotate the nodes and those at the end of the line such that the four nodes move through (not around) the line and to the other side.

Lemma 5. *Using a 3-node seed in the cells* $(1,1)$ *to* $(3,1)$*, above a line* L *of length* n *it is possible to create a 4-node line in the cells immediately below the nodes* $(n-4,0)$ *to* $(n-1,0)$*.*

Proof. We first move the 4 leftmost nodes in S, S_0 to S_3 to the top of the line. We do this by raising S_0 and S_1, and then repeat the procedure a second time with the next two nodes S_2 and S_3. We now have 4 nodes a square above the end of the line. By rotating them around each other in pairs we can place them in the cells $(n-4,1)$ to $(n-1,1)$. We can then "push" the nodes to the other side of S by following the procedure depicted in Figure 8. The result is four nodes in the cells $(n-4,-1)$ to $(n-1,-1)$ □

5.4. DepositNode

Next, we present *DepositNode*, a sub-procedure using the 3-node seed to create a 5-node shape and move a node from the horizontal line to any empty cell which the shape can reach, provided the 5-node shape has the correct colouring, defined as having 3 nodes of the colour which will fill the cell.

We raise two nodes from the line, use this shape to deposit a node and move the other 4 nodes as a square back to the left. By leaving the cells above and below the two leftmost nodes in the line empty we can rotate the leftmost node, merging with the square to create a new 5-node shape. We can therefore repeat the process of moving for each node one at a time. In addition, this sub-procedure can be applied to the builder on the other side of the line.

We provide pseudo-code to describe this process. We input an instance of the nice shape which is under construction and the co-ordinates of a destination. The algorithm then moves the nodes such that a node is placed in

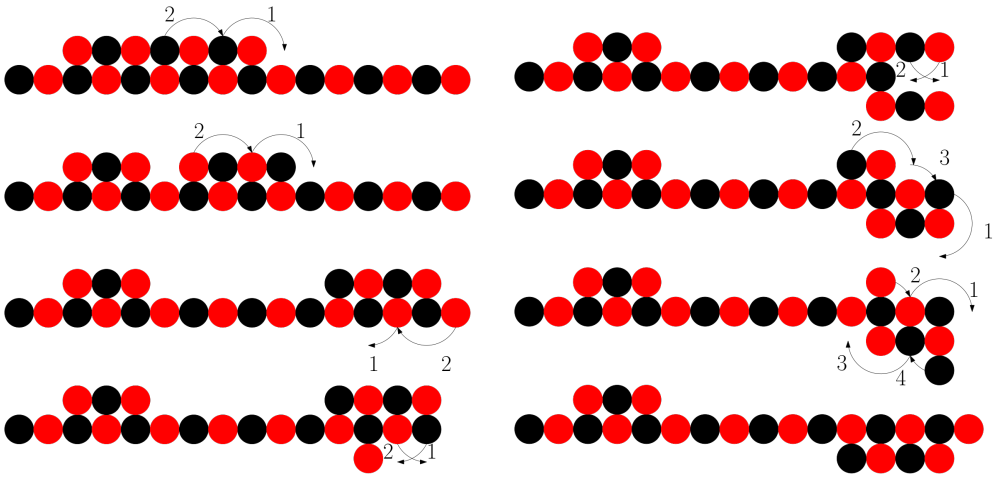


Figure 8: Pushing the nodes through the line.

the destination by the relevant builder, creating the output. In this way, the construction of the nice shape takes place over a series of phases, where each phase $1 \leq x \leq j$ corresponds to the movement of the xth node to the xth destination.

If the seed nodes are making their first transfer then they need to raise two nodes to become a builder with 5 nodes. After that, they only need to raise one node at a time. getLineHead() gets the node which is currently leftmost in the line. move() causes the builder to transfer one of its node to the destination. It must be the same colour but does not need to be the exact node. getParity() gets the colour of the node which must be placed first. rotateBlacks(seed) rotates each of the black coloured nodes in the seed rightwards. badParity occurs when the leftmost node of the line is not the colour which must be placed next. This situation represents a worst case scenario for node placement.

Our strategy is to demonstrate that the moves each builder can make are sufficient to be able to construct a nice shape. For ease of understanding, we provide visual representations of the movement we intend to accomplish. In this example, we show that it is possible to deposit the node at the end of the horizontal line.

Lemma 6. *A 3-node seed on any line S of length n, where n is an even number, can transfer a node the other end of the line.*

Algorithm 1 DepositNode

```
1: destination ← (x, y) //co-ordinates node will be deposited on
2: //If this is the first transfer for the seed nodes
3: if (badParity) == true then
4:     pair ← RaiseNodes
5:     //If the first node must be red
6:     if getParity() == "red" then
7:         //the node at the front when the seed is formed
8:         node ← seed[3]
9:     else
10:        //rotate the black nodes to place one at the front
11:        rotateBlacks(seed)
12:        //the node at the front after the rotation
13:        node ← seed[2]
14: else
15:    //the current leftmost node of the line
16:    node ← getLineHead()
17: move(node, destination)
```

Proof. We position the 3-node seed in the same grid spaces as in Lemma 5. We can then follow the process in Figure 9 to achieve our result. Note that we must raise three nodes from the line to deposit a node successfully, as we must be able to choose which colour is deposited first. We do this by using the RaiseNodes process such that three nodes are raised from the line. The other node is then the first node selected by getLineHead(). The process of moving right two spaces is repeatable, these repetitions are omitted. This entire process can be performed symmetrically by the nodes at the bottom of the line by only raising two nodes from the line. □

In this way, we can place a node of the colour we prefer onto both sides of the line. It is then possible to (see Figure 10) transfer the builder to a vertical line. By positioning the builder carefully we can ensure that the movement is equivalent to crossing a line of even length. Therefore the process of adding another node can be performed on vertical lines, such as the ones we will build for our nice shapes.

To build any vertical line, we must first show that it is possible for DepositNode to construct lines of length 4 above the horizontal line. After that, because it is possible for the builders to shift onto a 4-node line, the situation becomes that of depositing a node at the end of a line.

Lemma 7. *Using a 3-node seed in the cells* $(1, 1)$ *to* $(3, 1)$*, above a line* L *of length* n *it is possible to create another line of length* 4 *above any* $u_i \in L$*.*

Proof. For this situation we have two scenarios: one where the colouring is correct and another where it is incorrect. We first consider the correct colouring and then show how to deal with the incorrect colouring.

For the first node, we simply deposit the node using DepositNode above u_i. The next node is deposited above u_{i-1} and rotated to be above the first. The next two nodes are more difficult, so we have provided Figure 11 to illustrate the process.

In the case where the colouring is incorrect, we deposit the incorrect node anywhere to the right directly above L and collect a second node from L. We can then merge the 5-node shape with the node we deposited temporarily to create 6-node shape. Then 5 nodes of the correct colouring can split from the shape we created and deposit the node.

The 4-node square can then return to the node that was left behind and use it as the next node for depositing. In this way, the 5-node shape is capable of "selecting" its colouring. □

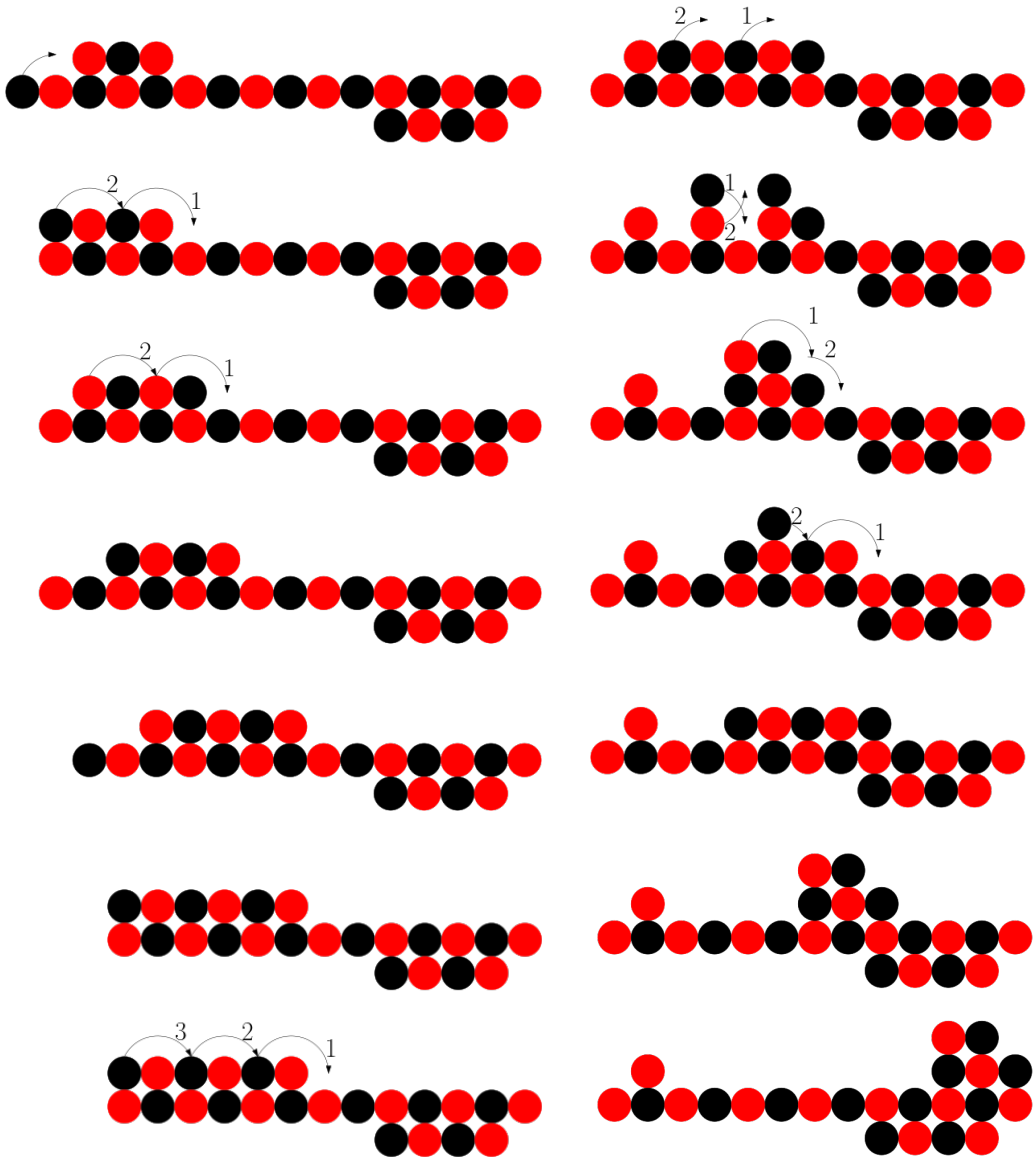


Figure 9: Moving a node across the line. To reach the second configuration in the left column, we raise three nodes by using the RaiseNodes process twice and stopping the second process early. In this figure, we place the black node. If we want to place the red node somewhere on the line, we can omit the first rotation in the second subfigure and create a 5-node seed with a red node at the front.

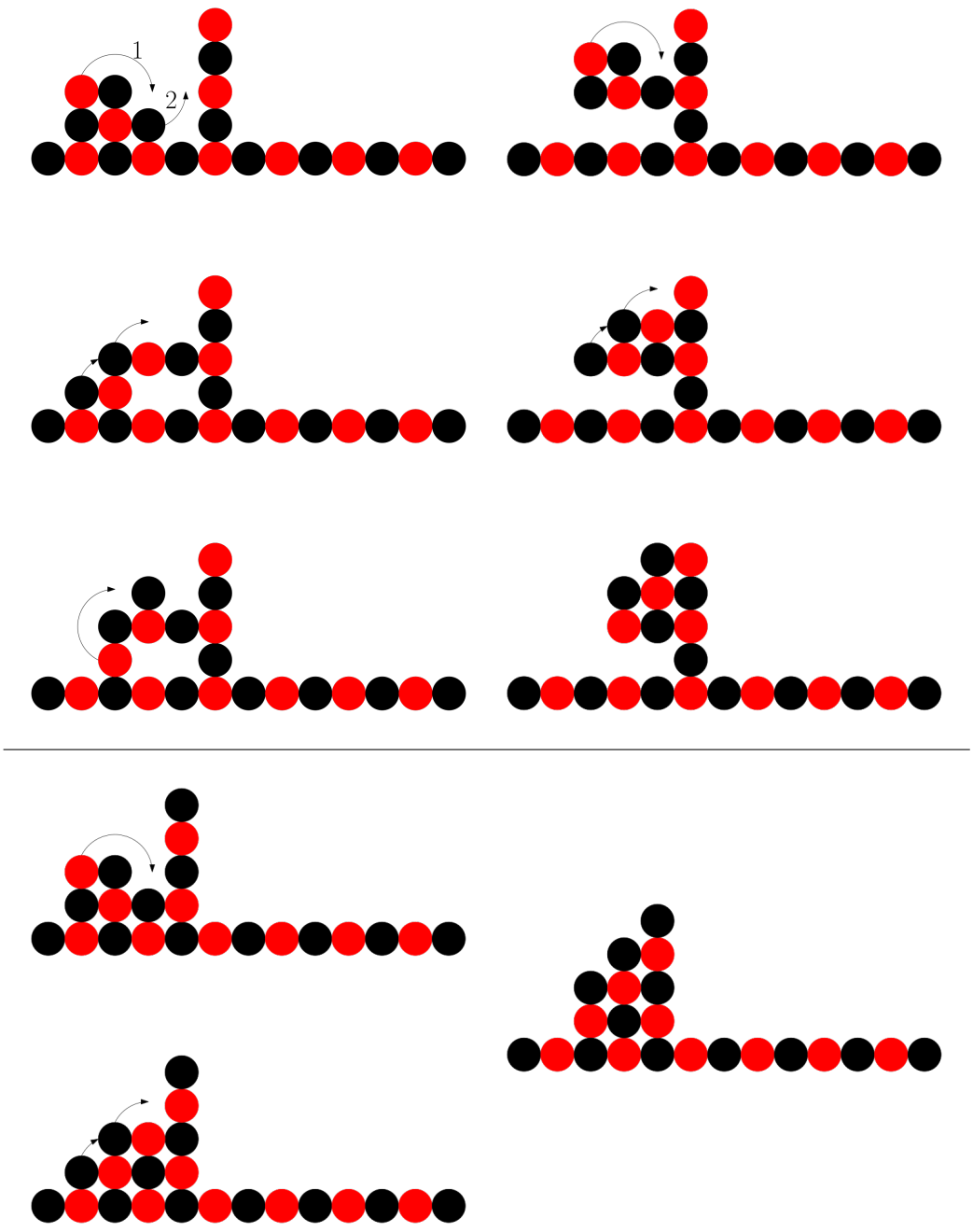


Figure 10: An example of moving the builder onto a vertical line. When the sequence of colours in the line is more convenient, a much simpler process can be used.

The next step we must take is to show that we can select the parity of the construction. This is a necessary process to guarantee that the initial processes of construction do not result in a scenario where two nodes of the same colour must be placed at the same time.

Lemma 8. *Using a 3-node seed in the cells* $(1, 1)$ *to* $(3, 1)$*, above a line* L *of length* n *it is possible to construct a two node line above* L *regardless of the colour of the node in* $(0, 0)$*.*

Proof. We begin by raising two nodes from L. We then use four of these nodes to form a 4-node square. We do this such that the node which is not part of the square is the opposite colour to the node at the end of L (i.e. in $(2, 0)$) We then have two scenarios depending on whether the first node we intend to place is the colour of the node on the line or the node in $(2, 0)$. If it is the same as the node on the line, we place the node on the line in the correct position and then place the node from the line above it. If the first node is different, we take the node from the line and form a 6-node block with the other five on the line. We can then place both nodes in the correct order. □

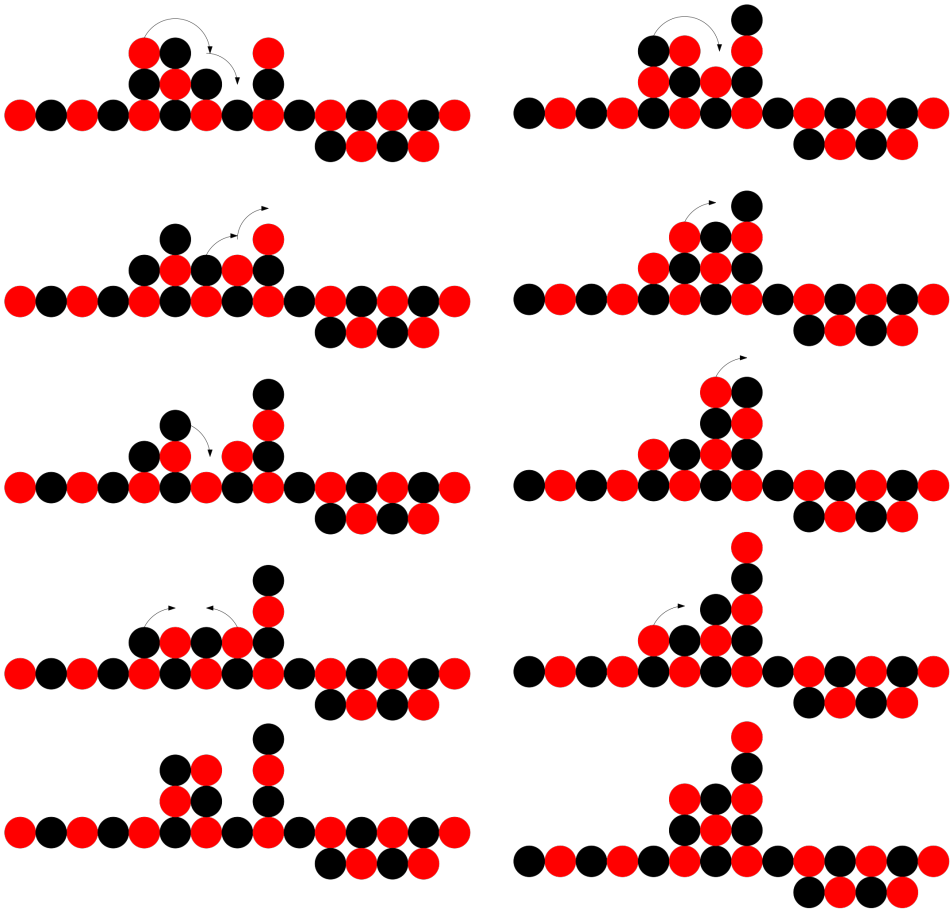
5.5. Construction of a subset of nice shapes

We now have all of the lemmas that are necessary to prove that it is possible to construct a specific subset of the class of nice shapes. We first present an upper bound on the time for constructing nice shapes using our algorithm. We then prove that using our sub-procedures we can construct a nice shape using a line and a 3-node seed, and finally we show that process is reversible using a 4-node seed.

Lemma 9. *The transformation of a horizontal line of* n *nodes into any nice shape requires* $O(n^2)$ *time steps.*

Proof. The RaiseNodes and MirrorSeed algorithmic procedures perform a sequence of specific movement and as such are $O(1)$ time. The DepositNode procedure moves 5 nodes, deposits a node and returns with 4 nodes. Therefore we must perform $5n + 4n$ moves to transfer one node, and in the worst case, we must build a vertical line of length n above the last node $(n - 1, 0)$ in the horizontal line. This means each node must move past $n - 1$ nodes to reach their destination.

Therefore the process is bound by the speed of DepositNode, which is $(5n + 4n) \cdot (n - 1) = O(n^2)$ time steps. □



(a) Adding the third node

(b) Adding the fourth node

Figure 11: Adding the last two nodes.

Theorem 2. *A line of length n can be transformed to any given nice shape in the class M_{n-1} using a 3-node seed in $O(n^2)$ time.*

Proof. The seed is positioned above the second, third and fourth nodes in the horizontal line, at $(1, 1)$, $(2, 1)$, and $(3, 1)$. We first use RaiseNodes twice to raise 4 nodes from the line and then use MirrorSeed to create a 4 node builder below the horizontal line. Then, we use DepositNode to construct the 5 node builder.

The 5 node builder can deposit a node in the construction area and move back to the end of the line by having each node rotate around each other. It is then able to take another node from the horizontal line by positioning itself two node spaces away from the end of the line and rotating the last node such that it is connected to the seed.

We are therefore able to follow a procedure for constructing vertical lines one node at a time. The construction proceeds for each side of L in phases $0 \leq i \leq |L|$, where phase i corresponds to the construction of the column above the node L_i.

The entire process is mirrored for the bottom of the shape using DepositNode for the builder on the bottom. The builder on the bottom waits until the builder on the top is finished and then starts lifting nodes from the same side of the line. By moving the other builder slightly it is possible to avoid the situation where it disconnects from the line.

Finally, one of the builders places the nodes of the other builder, and is then discarded, leading to a waste of 1 node. By Lemma 9, the whole process is completed in $O(n^2)$ time. □

Theorem 3. *A nice shape in the class M_n can be transformed to any given nice shape from M_n using a 4-node seed in $O(n^2)$ time.*

Proof. The transformation can be made reversible by assuming that the 4 nodes which are discarded at the end of the transformation constitute a 4 node seed for transforming the nice shape into a line. We can then construct a line of length n by following the process in reverse, and from there construct a nice shape of size n. □

5.6. Construction of any nice shape

We now show how to extend this to the class of all nice shapes. We follow a broadly similar procedure to the one in Theorem 2. The key difference is that we first create the *foundation*, a layer of nodes above and below the

horizontal line. We place a node at the start of every vertical line which starts with the same node colour that the previous vertical line built would end with. We then proceed as normal. First we prove that the foundation is sufficient for constructing any colour-consistent nice shape. Then we prove that the 5 node builder is capable of crossing the foundation to deposit nodes.

Lemma 10. *For any nice shape constructed from a line, for all lines perpendicular to L with an odd number of nodes there is at most one line which cannot be paired with another line which ends in the other colour.*

Proof. We have the initial line which is either odd or even. We can move nodes out in pairs to build lines. It is possible to build lines which are odd by splitting a pair and distributing its nodes between two odd lines. Such lines can therefore be paired.

However, there are two ways that an extra odd line can be created. First, when the horizontal line is odd, we can support one odd vertical line by extracting the extra node. Second, when the horizontal line is even, we can also split a pair with the horizontal, making it odd. If both are attempted at the same time the resulting lines will end in different colours and therefore can be paired. As a result, at most one line which is odd cannot be paired. □

Lemma 11. *Any* 5 *node builder which is constructing lines can cross the foundation to do so.*

Proof. When moving a builder carrying a node across the foundation, there are 3 scenarios the builder can encounter.

In the first scenario, there is a node in (x, y) which is the same colour as the node being carried in $(x-2, y)$. In this case, the builder must deposit the node in $(x-4, y)$ and collect the node it has encountered. Then, when the builder is returning without carrying a node, it must shift the node it deposited from $(x-4, y)$ to (x, y).

In the second scenario, the node at (x, y) is a different colour and the cell $(x+1, y)$ is empty. For this scenario, the node which is being carried rotates into $(x+1, y)$. Then the builder's nodes rotate around each other to be above $(x-1, y-1)$ and $(x-2, y-1)$. Then the top two nodes in $(x-1, y+1)$ and $(x-2, y+1)$ rotate around each other such that the node in $(x+1, y)$ is the node being carried by the 5 node builder.

In the third scenario, there is a series of nodes beginning with the node (x, y), with alternating colours blocking the builder. In this case, we first

identify the node n which is the node furthest right in the series with the same colour as the node the builder is carrying. Then the top two nodes of the builder in $(x-2, y+1)$ and $(x-3, y+1)$ rotate until they are positioned such that they form a 5 node builder with n.

Each of these processes are depicted in Figure 12.

Any foundation must consist of any of these three scenarios arranged in a sequence. Therefore, by following the correct process in the scenario the builder crosses the foundation and places a node of the correct colour. Then while returning any nodes deposited can be shifted, creating a new foundation which is equivalent to the original. □

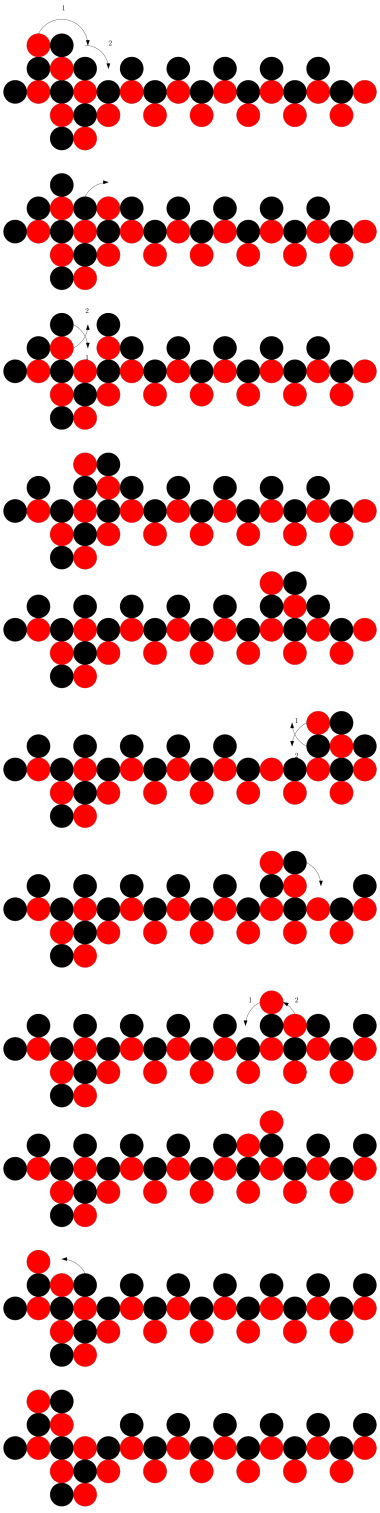
We are now in the position to prove our main result, that it is possible to construct any nice shape from any other nice shape using a seed of size 4. Let N_n be the subclass of nice shapes which is colour consistent to a line of length n.

Theorem 4. *A line of length n can be transformed to any given nice shape N_{n-1} using a 3-node seed in $O(n^2)$ time.*

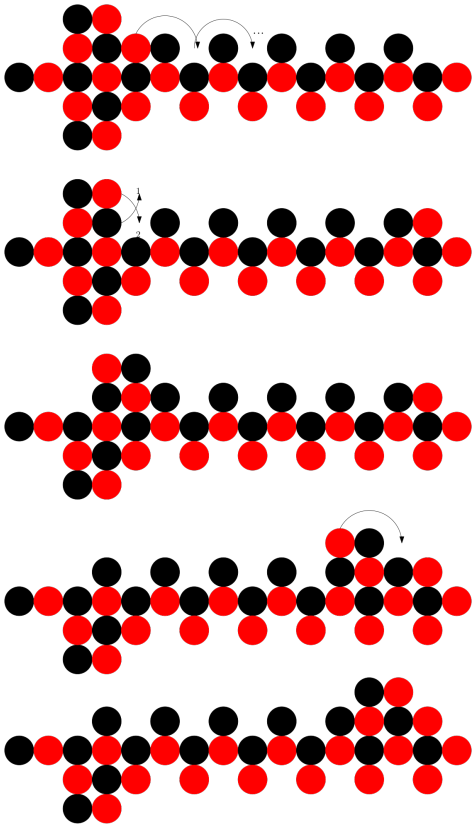
Proof. The initial steps of the procedure are as in Theorem 2. When we have created both builders, we then create the foundation by placing each node in the foundation from right to left. We alternate between the builders as necessary. By Lemma 10, we know that the scenario where the colours we need to place do not match what is available will never occur. By Lemma 11, we know that the existence of the foundation does not impede construction. We are then able to follow a procedure for constructing vertical lines as before. Finally, the last builder is discarded as before. □

Theorem 5. *A nice shape of n nodes can be transformed to any given nice shape N_n using a 4-node seed in $O(n^2)$ time.*

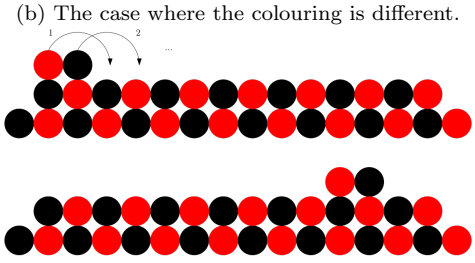
Proof. By Theorem 4, we can construct a nice shape from a line using a 3-node seed with 1 node as waste. By reversibility, we can start with a 4-node seed and construct a line of length n. It is then possible to construct another nice shape using the line. □



(a) The case where the colouring is the same.



(b) The case where the colouring is different.



(c) The case where the colours alternate.

Figure 12: Moving the builder across the foundation.

6. Conclusions

Some open problems follow from the findings of our work. The most obvious is expanding the class of shapes which can be constructed using minimal seeds to those which can be derived from nice shapes. This could possibly be expanded by transferring nodes along the perimeter of a nice shape with the help of bridging nodes or by compressing them. In the long run this could lead to characterisations of the classes of connectivity-preserving shapes which can be constructed using only rotation for a given seed. Another important question is the impact that switching to a decentralised model of transformations will have on the results, especially because most programmable matter systems which model real-world applications implement programs in this way. This in turn could lead to real-world applications for the efficient transformation of programmable matter systems.

References

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

[AAD+21] H. A. Akitaya, E. M. Arkin, M. Damian, E. D. Demaine, V. Dujmović, R. Flatland, M. Korman, B. Palop, I. Parada, A. v. Renssen, and V. Sacristán. Universal Reconfiguration of Facet-Connected Modular Robots by Pivots: The O(1) Musketeers. *Algorithmica*, 83[5]:1316–1351, May 2021.

[AMP20] A. Almethen, O. Michail, and I. Potapov. Pushing lines helps: Efficient universal centralised transformations for programmable matter. *Theoretical Computer Science*, 830-831:43–59, August 2020.

[CGSW21] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Verification and computation in restricted Tile Automata. *Natural Computing*, November 2021.

[CLS+11] X. Chen, L. Li, X. Sun, Y. Liu, B. Luo, C. Wang, Y. Bao, H. Xu, and H. Peng. Magnetochromatic Polydiacetylene by Incorporation of Fe3O4 Nanoparticles. *Angewandte Chemie International Edition*, 50[24]:5486–5489, 2011.

[DDD+20] G. D'Angelo, M. D'Emidio, S. Das, A. Navarra, and G. Prencipe. Asynchronous Silent Programmable Matter Achieves Leader Election and Compaction. *IEEE Access*, 8:207619–207634, 2020.

[DDG+14] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Amoebot - a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, SPAA '14, pages 220–222, Prague, Czech Republic, June 2014. Association for Computing Machinery.

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

[DGR+15] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An Algorithmic Framework for Shape Formation Problems in Self-Organizing Particle Systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, NANOCOM' 15, pages 1–2, Boston, MA, USA, September 2015. Association for Computing Machinery.

[DGR+16] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal Shape Formation for Programmable Matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, pages 289–299, Pacific Grove, California, USA, July 2016. Association for Computing Machinery.

[Dot12] D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55[12]:78–88, December 2012.

[Dot13] D. Doty. Timing in chemical reaction networks. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 772–784. Society for Industrial and Applied Mathematics, December 2013.

[DRS21] J. J. Daymude, A. W. Richa, and C. Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. Technical Report arXiv:2105.02420, arXiv, May 2021. arXiv:2105.02420 [cs] type: article.

[GKR10] K. Gilpin, A. Knaian, and D. Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *2010 IEEE International Conference on Robotics and Automation*, pages 2485–2492, May 2010. ISSN: 1050-4729.

[KCL+12] A. N. Knaian, K. C. Cheung, M. B. Lobovsky, A. J. Oines, P. Schmidt-Neilsen, and N. A. Gershenfeld. The Milli-Motein: A self-folding chain of programmable matter with a one centimeter module pitch. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1447–1453, October 2012. ISSN: 2153-0866.

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

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

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

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

[RW00] P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 459–468, Portland, Oregon, USA, May 2000. Association for Computing Machinery.

[SCWB08] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7[4]:615–633, December 2008.

[TPB19] P. Thalamy, B. Piranda, and J. Bourgeois. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 140–148, Montreal QC, Canada, May 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[TPB20] P. Thalamy, B. Piranda, and J. Bourgeois. 3D Coating Self-Assembly for Modular Robotic Scaffolds. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11688–11695, October 2020. ISSN: 2153-0866.

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

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