On Geometric Shape Construction via Growth Operations

Nada Almalki and Othon Michail[0000−0002−6234−3960]

Department of Computer Science, University of Liverpool, UK
{N.Almalki, Othon.Michail}@liverpool.ac.uk

Abstract. In this work, we investigate novel algorithmic *growth processes*. Our system runs on a 2-dimensional grid and operates in discrete time steps. The growth process begins with an initial shape of nodes $S_I = S_0$ and, in every time step $t \geq 1$, by applying (in parallel) one or more *growth operations* of a specific type to the current shape-instance S_{t-1}, generates the next instance S_t, always satisfying $|S_t| > |S_{t-1}|$. Our goal is to characterize the classes of shapes that can be constructed in $O(\log n)$ or polylog n time steps, n being the size of the final shape S_F, and determine whether a shape S_F can be constructed from an initial shape S_I using a finite sequence of *growth operations* of a given type, called a *constructor of* S_F.
In particular, we propose three *growth operations*, *full doubling*, *row and column doubling*, which we call *RC doubling*, and *doubling*, and explore the algorithmic and structural properties of their resulting processes under a geometric setting. For *full doubling*, in which, in every time step, every node generates a new node in a given direction, we completely characterize the structure of the class of shapes that can be constructed from a given initial shape. For *RC doubling*, in which complete columns or rows double, our main contribution is a linear-time centralized algorithm that for any pair of shapes S_I, S_F decides if S_F can be constructed from S_I and, if the answer is yes, returns an $O(\log n)$-time step constructor of S_F from S_I. For the most general *doubling* operation, where a subset of individual nodes can double, we show that some shapes cannot be constructed in sublinear time steps and give two universal constructors of any S_F from a singleton S_I, which are efficient (i.e., up to polylogarithmic time steps) for large classes of shapes. Both constructors can be computed by polynomial-time centralized algorithms for any shape S_F.

Keywords: Centralized algorithm · Geometric growth operations · Programmable matter · Constructor.

1 Introduction

The realization that many natural processes are essentially algorithmic, has fueled a growing recent interest in formalizing their algorithmic principles and in developing new algorithmic approaches and technologies inspired by them. Examples of algorithmic frameworks inspired by biological and chemical systems are population protocols [5, 6, 24], ant colony optimization [9, 14], DNA self-assembly [14, 25, 26, 28], and the algorithmic theory of programmable matter [1, 3, 22].

Motivated by these advancements and by principles of biological development which are apparently algorithmic, we introduce a set of geometric *growth processes* and study their algorithmic and structural properties. These processes start from an initial shape of nodes S_I, possibly a singleton, and by applying a sequence of *growth operations* eventually develop into well-defined global geometric structures. The considered *growth operations* involve at most one new node being generated by any existing node in a given direction and the resulting reconfiguration of the shape as a consequence of a set of nodes being generated within it. This node-generation primitive is also inspired by the self-replicating capabilities of biological systems, such as cellular division, their higher-level processes such as embryogenesis [7], and by the potential of the future development of self-replicating robotic systems.

In a recent study, Mertzios *et al.* [20] investigated a network-growth process at an abstract graph-theoretic level, free from geometric constraints. Our goal here is to study similar *growth processes* under a geometric setting and show how these can be fine-tuned to construct interesting geometric shapes efficiently, i.e., in polylogarithmic time steps. Aiming to focus exclusively on the effect of *growth operations*, we do not allow any other form of shape reconfigurations apart from local growth. Preliminary such *growth processes*, mostly for rectangular shapes, were developed by Woods *et al.* [27]. Their approach was to first grow such shapes in polylogarithmic time steps and then to transform them into arbitrary geometric shapes and patterns through additional reconfiguration operations, the latter essentially capturing properties of molecular self-assembly systems. Like them, we study the problem of constructing a desired final shape S_F starting from an initial shape S_I via a sequence of shape-modification operations. However, in this work the considered operations are restricted to local *growth operations*. To the best of our knowledge, the structural characterization and the underlying algorithmic complexity of constructing geometric shapes by *growth operations*, have not been previously considered as problems of independent interest.

1.1 Our Approach and Contribution

In this work, our main objective is to study *growth operations* in a centralized geometric setting. Applying a sequence of such operations in a centralized way, yields a centralized geometric growth process. Our model can be viewed as an applied, geometric version of the abstract network-growth model of Mertzios *et al.* [20]. The considered model is discrete and operates on a 2D square grid. Connectivity preservation is an essential aspect of both biological and of the so-inspired robotic and programmable matters, because it allows the system to maintain its strength and coherence and enables sharing of resources between devices in the system [8, 22]. In light of this, all the shapes discussed in this work are assumed to be connected and the considered *growth operations* cannot break the shape's connectivity.

For all types of considered operations, the study revolves around the following main questions: (i) *What is the class of shapes that can be constructed efficiently from a given initial shape via a sequence of growth operations?* (ii) *Is there a polynomial-time centralized algorithm that can decide if a given target shape S_F can be constructed from a given initial shape S_I and, whenever the answer is positive, return an efficient constructor of S_F from S_I?*

The *growth operations* considered in this paper are characterized by the following additional properties:

- In general, more than one *growth operation* can be applied at the same time step (parallel version). To simplify the exposition of some of our results and without losing generality, we shall sometimes restrict attention to a single operation per time step (sequential version).
- To avoid having to deal with colliding operations, we restrict attention to single-direction *growth operations*. That is, for each time step t, a direction $d \in \{\text{north, east, south, west}\}$ is fixed and any operation at t must be in direction d. For clarity of presentation of the results in this work, we shall focus mostly on the *east* and *north* directions of the considered operations. Due to the nature of these operations, generalizing to all four directions is immediate.

We study three *growth operations*, *full doubling*, *RC doubling*, and *doubling*, where full doubling is the most restricted and doubling the most general one. In *full doubling*, in every time step, every node generates a new node in a given direction, in *RC doubling*, complete columns or rows double, and in *doubling* up to individual nodes can double.

For *full doubling*, we completely characterize the structure of the class of shapes that are reachable from any given initial shape. For *RC doubling*, our main contribution is a linear-time centralized algorithm that for any pair of shapes S_I, S_F decides if S_F can be constructed from S_I and, if the answer is yes, returns an $O(\log n)$-time step constructor of S_F from S_I. For *doubling*, we show that some shapes cannot be constructed in sublinear time steps and give two universal constructors of any S_F from a singleton S_I, which are efficient (i.e., up to polylogarithmic time steps). for large classes of shapes. Both constructors can be computed by polynomial-time centralized algorithms for any shape S_F.[1]

In Section 1.2, we discuss the related literature. Section 2 presents all definitions that are used throughout the paper. Sections 3, 4, and 5 present our results for *full doubling*, *RC doubling*, and *doubling*, respectively.

[1] Note that there are two distinct notions of time used in this paper. One represents the time steps of a growth process, while the other represents the running time of a centralized algorithm deciding reachability between shapes and returning constructors for them. We shall always distinguish between the two by calling the former *time steps* and the latter *time*.

1.2 Related work

Recent work has focused on studying the algorithmic principles of reconfiguration, with the potential of developing artificial systems that will be able to modify their physical properties, such as reconfigurable robotic ensembles and self-assembly systems. For example, the area of algorithmic self-assembly of DNA aims to understand how to train molecules to modify themselves while also controlling their own growth [14]. Several theoretical models of programmable matter have been developed, including DNA self-assembly and other passively dynamic models [14, 21] as well as models enriched with active molecular components [27].

One example of a geometric programmable matter model, which is presented in [12], is known as the *Amoebot* and is inspired by amoeba behavior. In particular, programmable matter is modeled as a swarm of distributed autonomous self-organizing entities that operate on a triangular grid. Research on the Amoebot model has made progress on understanding its computational power and on developing algorithms for basic reconfiguration tasks such as coating [11] and shape formation [10, 13]. Other authors have investigated cycle-shaped programmable matter modules that can rotate or slide a device over neighboring devices through an empty space [15, 16, 22, 8], with the goal of capturing the global reconfiguration capabilities of local mechanisms that are feasible to be implemented with existing technology. The authors in [22] proved that the decision problem of transformation between two shapes is in **P**. In addition, another recent research work [3] investigated a linear-strength mechanism through which a node can push a line of one or more nodes by one position in a single time step. Other linear-strength mechanisms are the one by Woods *et al.* [27], where a node can rotate a whole line of connected nodes, simulating arm rotation, or the one by Aloupis *et al.* [4] on crystalline robots, equipped with powerful lifting capabilities.

A recent study in the field of highly dynamic networks, which is presented in [20], is partially inspired by the abstract-network approach followed in [23]. The authors completely disregard geometry and develop a network-level abstraction of programmable matter systems. Their model starts with a single node and grows the target network G to its full size by applying local operations of node replication. Local edges are only activated upon a node's generation and can be deleted at any time but contribute negatively to the edge-complexity of the construction. The authors develop centralized algorithms that generate basic graphs such as paths, stars, trees, and planar graphs and prove strong hardness results. We similarly focus on centralized structural and algorithmic characterizations as a first step that will promote our understanding of such novel models and will facilitate the future development of more applied constructions, like fully distributed ones.

2 Model and Preliminaries

The programmable matter systems considered in this paper operate on a 2-dimensional square grid. Each grid position (cross point) is identified by its x and y coordinates, $x \geq 0$ representing the row and $y \geq 0$ the column. Systems of this type consist of n nodes that form a connected shape S. Each node u of shape S is represented by a circle occupying a position on the grid. Time consists of discrete time steps and in every time step $t \geq 1$, zero or more *growth operations* can occur depending on the type of operation considered. At any given time step t, each node $u \in S$ is determined by its coordinates (u_x, u_y) and no two nodes can occupy the same position at the same time step. Two distinct nodes $u = (u_x, u_y)$ and $v = (v_x, v_y)$ are *neighbors* if $u_x \in \{v_x - 1, v_x + 1\}$ and $u_y = v_y$ or $u_y \in \{v_y - 1, v_y + 1\}$ and $u_x = v_x$, that is, if they are at orthogonal distance one from each other. In that case, we are assuming that, unless explicitly removed, a connection (or edge) uv exists between u and v.

Definition 1 (Row and Column of a Shape). *A row (respectively column) of a shape S is the set of all nodes of S with the same fixed y-coordinate (respectively x-coordinate).*

By $S_{\cdot,i}$ we denote the row of S consisting of all nodes whose y-coordinate is i, i.e., $S_{\cdot,i} = \{(x, i) \mid (x, i) \in S\}$. Similarly, column j *of S, denoted as $S_{j,\cdot}$, is the set of all nodes of S whose x-coordinate is j, i.e., $S_{j,\cdot} = \{(j, y) \mid (j, y) \in S\}$. When the shape S is clear from context, we will refer to $S_{\cdot,i}$ as R_i and to $S_{j,\cdot}$ as C_j.*

Definition 2 (Translation Operation). *Given a set of integer points Q, the north (south) k-*translation *of Q is defined as $\uparrow_k Q = \{(x, y + k) \mid (x, y) \in Q\}$ ($\downarrow_k Q$, similarly defined). The east (west) l-*translation *of Q is defined as $\xrightarrow{l} Q = \{(x + l, y) \mid (x, y) \in Q\}$ ($\xleftarrow{l} Q$, similarly defined).*

Definition 3 (Rigid Connection). *A connection uv between two nodes u and v of a shape S is* rigid *if and only if a 1-translation of one node in any direction d implies a 1-translation of the other in the same direction, unless uv is first removed.*

Throughout, all connections are assumed to be rigid.

The basic concept of *growth operation* is that a node $u \in S_t$ generates a new node $u' \in S_{t+1}$. In particular, we are exploring three specific *growth operations* a *full doubling*, *row and column doubling* and *doubling*. In most cases, every node $u \in S_t$ is colored black while its generations are colored gray at the next time step $t + 1$ after any type of *growth operations*. Furthermore, any type of *growth operation o* is equipped with a *linear-strength* mechanism, which is the ability of a generated node u' to translate its connected component on a growing direction in a single time step.

Throughout this paper, l, k will represent the total number of horizontal and vertical *growth operations* performed (respectively). By horizontal, the direction d is either *east* or *west*, while the vertical d is either *north* or *south*.

Definition 4 (Growth Operation). *A* growth operation *o is an operation that when applied on a shape instance S_t, for all time steps $t \geq 1$, yields a new shape instance $S_{t+1} = o(S_t)$, such that $|S_t| > |S_{t-1}|$.*

In this work, we consider three specific types of *growth operations* moving from the most special to the most general: *full doubling* , *RC doubling* and *doubling* operation. For the sake of clarity, we will provide a high-level overview of these three operations, with the more technical versions appearing in their respective Sections 3, 4 and 5. First, a *full doubling* operation is a growth operation in which every node $u \in S_t$ generates a new node $u' \in S_{t+1}$, that is, $|S_{t+1}| = 2|S_t|$. Then, *row and column doubling* denoted by *RC doubling*, is a growth operation where in each time step t, a subset of columns (rows) is selected and these are fully doubled. Finally, the most general version of these operations is a *doubling* operation, in which, in each time step t, any subset of the nodes can double in a given direction. The differences between these three operations are highlighted in Fig. 1.

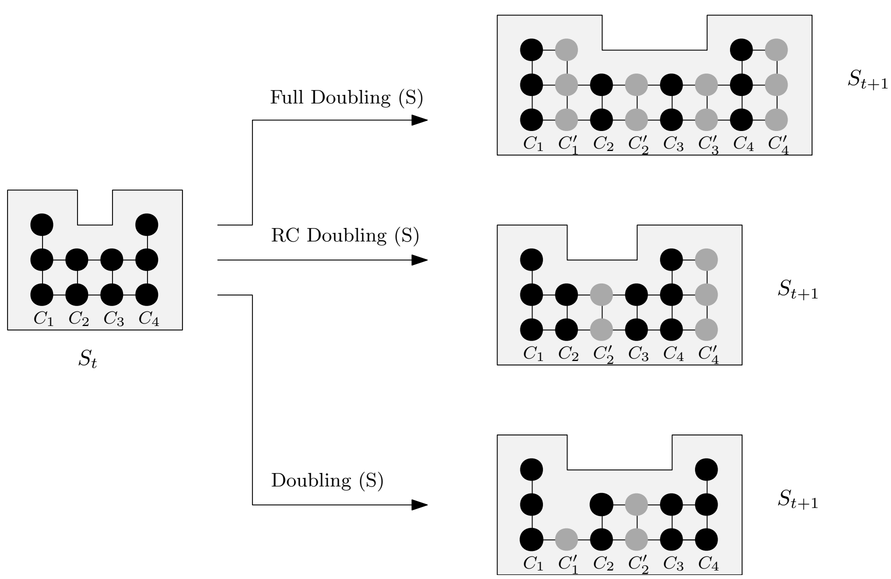

Fig. 1: Illustration of the results S_{t+1} when applying different growth operations on the same shape S_t where the direction of growing is east.

Definition 5 (Reachability Relation). *Given a growth operation of a given type, we define a reachability relation $\rightsquigarrow$ on pairs of shapes S, S' as follows. $S \rightsquigarrow S'$ iff there is a finite sequence $\sigma = o_1, o_2, \ldots, o_{t_{last}}$ of operations of a given type for which $S = S_0, o_1, S_1, o_2, S_2 \ldots, S_{(t_{last}-1)}, o_{t_{last}}, S_{t_{last}} = S'$, where $S_i = o(S_{i-1})$ for all $1 \leq i \leq t_{last}$. Whenever we want to emphasize a particular such sequence σ, we write $S \overset{\sigma}{\rightsquigarrow} S'$ and say that σ* constructs *shape S' from S.*

Definition 6 (Constructor). *A* constructor $\sigma = (o_1, o_2, \ldots, o_{t_{last}})$ *is a finite sequence of doubling operations of a given type,* $1 \leq i \leq t_{last}$.

Remark 1. Note that in Definition 6 the directions of different o_i's do not need to be the same.

2.1 Problem Definitions

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

Class characterization. Identify the family of shapes S_F that can be obtained from a given initial connected shape S_I via a sequence of *growth operations* of a given type.

ShapeConstruction. Given a pair of shapes (S_I, S_F) decide if $S_I \rightsquigarrow S_F$. If yes, compute a sequence σ that constructs S_F from S_I. In the special case of this problem in which S_I is by assumption a singleton, we shall assume that the input is just S_F.

3 Full Doubling

In this section, after providing a formal definition of the full doubling operation, we investigate the class characterization problem under this operation. Note that when the initial shape consists of a single node, i.e., $|S_I| = 1$, the characterization is straightforward. Section 3.1 discusses the more general case where $|S_I| \geq 1$.

Definition 7 (Full Doubling). *After applying a* full doubling operation *on* S *a* new shape S' *is obtained, depending only on the direction* d *of the operation:*

1. *If the direction* d *of a* full doubling operation *is an* east*, then for every column* $S_{j,\cdot}$ *of* S *a new column is generated to the* east *of* $S_{j,\cdot}$*. The effect of applying this to all columns is that every column* $S_{j,\cdot}$ *of* S *is translated to the east by* $j-1$*, such that* $S'_{2j-1,\cdot} \stackrel{j-1}{=\!\!\rightarrow} S_{j,\cdot}$*, and generates the new column* $S'_{2j,\cdot} \stackrel{j}{=\!\!\rightarrow} S_{j,\cdot}$*. Therefore, the new shape* S' *of this doubling operation is* $S' = \bigcup_j (S'_{2j-1,\cdot} \cup \ S'_{2j,\cdot})$.
2. *If the direction* d *of a* full doubling operation *is a* north*, then for every row* $S_{\cdot,i}$ *of* S *a new row is generated to the north of* $S_{\cdot,i}$*. The effect of applying this to all rows is that every row* $S_{\cdot,i}$ *of* S *is translated to the north by* $i-1$*, such that* $S'_{\cdot,2i-1} =\uparrow_{i-1} S_{\cdot,i}$*, and generates the new row* $S'_{\cdot,2i} =\uparrow_i S_{\cdot,i}$*. Therefore, the new shape* S' *of this doubling operation is* $S' = \bigcup_i (S'_{\cdot,2i-1} \cup \ S'_{\cdot,2i})$.

In other words, if a full doubling operation is performed on S in the east direction, then a set of columns equal to the original is generated. Every original column is translated by the number of original columns to its west and its own copy is generated to the east of its final position. Similarly for rows.

3.1 An Arbitrary Connected Initial Shape $|S_I| \geq 1$

In this section we characterize shapes that can be obtained by a sequence of full doubling operations from an arbitrary connected initial shape S_I, where $|S_I| \geq 1$.

Definition 8 **($w(C_j, u)$).** *Let $w(C_j, u)$ denote the number of columns to the left (west) of a node u in a column j, that is, $w(C_j, u) = u_x - C_l$, where C_l is the leftmost column.*

Definition 9 **($s(R_i, u)$).** *Let $s(R_i, u)$ denote the number of rows below (south) of a node u in a row i, that is, $s(R_i, u) = u_y - R_b$, where R_b is the bottom-most row.*

Definition 10 (Reconfiguration Function). *Given two integers $l, k > 0$, we define a reconfiguration function $F_{l,k}$ that maps a shape to another shape as follows:*

1. *First, the coordinates of $|S|$ points of $F_{l,k}(S)$ are determined as a function of the coordinates of the points of S. For each $u \in S$ the coordinates of $u' \in F_{l,k}(S)$ are given by $(u_x + (2^l - 1)w(C_j, u), u_y + (2^k - 1)s(R_i, u))$.*
2. *Generate the Cartesian product around u' such that, $Rec(u', 2^l, 2^k) = \{u'_x + 1, \ldots, u'_x + (2^l - 1)\} \times \{u'_y + 1, \ldots, u'_y + (2^k - 1)\}$ originating at u'. Adding all points of these rectangles to $F_{l,k}(S)$ completes the definition of $F_{l,k}(S)$.*

The output of the reconfiguration function *after these two phases is a shape S, such that $F_{l,k}(S) = \bigcup_{u \in S} Rec(u', 2^l, 2^k)$, as presented in Fig. 2 (note that u' is a function of u in the union).*

Lemma 1 (Additivity of Reconfiguration Function). *For all shapes S and all $l, k, l', k' \geq 0$ it holds that $F_{l',k'}(F_{l,k}(S)) = F_{l'+l,k'+k}(S)$.*

Theorem 1. *Given any initial shape S_I and any sequence of l east and k north full doubling operations, the obtained shape is $S_F = F_{l,k}(S_I)$.*

4 RC Doubling

After a formal definition of the *RC doubling* operation, in this section, we study both the class characterization and the ShapeConstruction problems. In particular, we develop a linear-time centralized algorithm to decide the feasibility of constructing S_F from S_I and to return a constructor of S_F from S_I if one exists, both within $O(\log n)$-time steps.

Definition 11 (RC Doubling). *A row and column doubling is a growth operation where a direction $d \in \{east, west\}$ ($d \in \{north, south\}$) is fixed and all nodes of a subset of the columns (rows, respectively) of shape S generates a new node in d direction.*

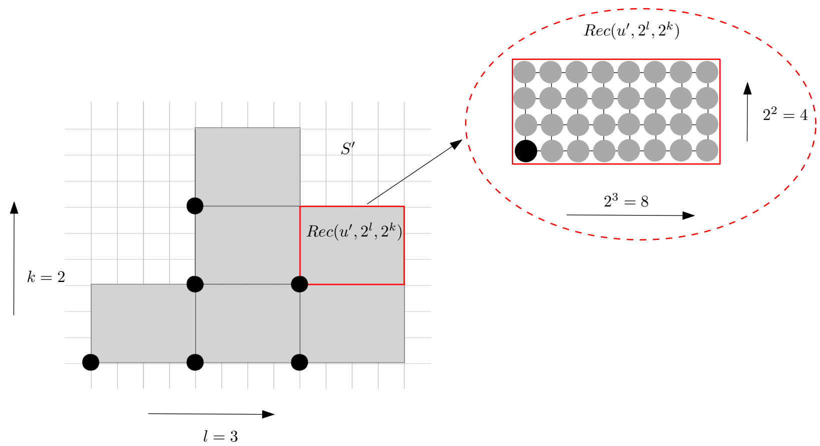

Fig. 2: An example of the output shape S' after applying the *reconfiguration function* $F_{l,k}(S)$.

We define an RC doubling operation *for columns in the east direction and the other cases can be similarly defined. The operation is applied to a shape S and will yield a new shape S'. Let J be the set of indices (ordered from west to east) of all columns of S and D its subset of indices of the columns to be doubled by the operation. For any $j \in J$, let $w(D, j) = |\{j' \in D \mid j' < j\}|$, i.e. $w(D, j)$ denotes the number of doubled columns to the west of column j. Then the new shape S' is defined as:*

$$S' = (\bigcup_{j \in J} \overset{w(C_j)}{\rightarrow} C_j) \cup (\bigcup_{j \in D} \overset{w(C_j)+1}{\rightarrow} C_j)$$

That is, every doubled column C_j, for $j \in D$, generates a copy of itself to the east. The result is that every column C_j, for $j \in J$, is translated east by $w(C_j)$ and additionally the final position of the copy of C_j, for $j \in D$, is an east $(w(C_j)+1)$ translation of C_j.

Definition 12 (Single RC Doubling Operation). *Let $d \in J$ be the index of the single doubled column. Define $S_{\leq C_d}$ ($S_{\geq C_d}$) to be the set of columns to the west (east, resp.) of column C_d, inclusive. That is, $S_{\leq C_d} = \bigcup_{j \in J, j \leq d} C_j$ ($S_{\geq C_d} = \bigcup_{j \in J, j \geq d} C_j$, resp.). Then,*

$$S' = S_{\leq C_d} \cup (\overset{1}{\rightarrow} S_{\geq C_d}).$$

Proposition 1 (Serializability of Parallel Doubling). *A shape S_F can be generated from a shape S_I through a sequence of RC (parallel) doubling operations iff it can be generated through a sequence of single row/column doubling operations.*

Definition 13 (Consecutive Column/Row Multiplicities). *Given a shape S and a column C_j (row R_i) of S which is either the leftmost column (bottommost row) (i.e., $j = 1$) or $C_{j-1} \neq C_j$ ((i.e., $i = 1$) or $R_{i-1} \neq R_i$) (where equality is defined up to horizontal (vertical) only translations of columns (rows)), the multiplicity $M_S(C_j)$ ($M_S(R_i)$) of column (row) C_j (R_i), is defined as the maximal number of consecutive identical copies of C_j (R_i) in S to the right (top) of C_j (R_i) (inclusive).*

Definition 14 (Baseline Shape). *The* baseline shape $B(S)$ *of a shape S, is the shape obtained as follows. For every column C_j of S with $M_S(C_j) > 1$, remove all consecutive copies of C_j to its right (non-inclusive) and compress the shape to the left to restore connectivity. Then for every row R_i of S with $M_S(R_i) > 1$, remove all consecutive copies of R_i to its top (non-inclusive) and compress the shape down to restore connectivity. Observe that all columns and rows of $B(S)$ have multiplicity 1. Moreover, any shape whose columns and rows all have multiplicity 1 is called a baseline shape.*

4.1 $S_I \rightsquigarrow S_F$ Constructor

Theorem 2. *A shape S_I can generate a shape S_F through a sequence of RC doubling operations iff $B(S_I) = B(S_F) = B$ and for every column C and row R of B it holds that $M_{S_F}(C) \geq M_{S_I}(C)$ and $M_{S_F}(R) \geq M_{S_I}(R)$.*

Proof. To prove that the condition is sufficient, we can w.l.o.g. restrict attention to single RC doubling operations (as these are special cases of RC doubling operations). Then, for every column C of B for which $M_{S_F}(C) > M_{S_I}(C)$ holds, we double the west-most copy of column C in S_I, $M_{S_F}(C) - M_{S_I}(C)$ times to the east. Similarly, for rows. It is not hard to see that any sequence of these operations applied to S_I, yields S_F.

For the necessity of the condition, we need to show that if S_I can generate S_F through a sequence of RC doubling operations, then $B(S_I) = B(S_F) = B$ and the multiplicities are as described in the statement. We first observe that, by Proposition 1, S_I can also generate S_F through a sequence of single RC doubling operations. So, it is sufficient to show that violation of any of the conditions would not allow for a valid sequence of single RC doubling operations.

Let us first assume that $B(S_I) = B(S_F) = B$ holds, but $M_{S_F}(C) \geq M_{S_I}(C)$ does not, that is, $M_{S_F}(C) < M_{S_I}(C)$ for some column C of B. Then, there is no way of obtaining S_F from S_I as this would require deleting $M_{S_I}(C) - M_{S_F}(C)$ copies of C. Similarly, if $M_{S_F}(R) \geq M_{S_I}(R)$ is violated.

Finally, assume that $B(S_I) \neq B(S_F)$ and that $S_I \rightsquigarrow S_F$ still holds. By definition of baseline shapes, $B(S_I) \rightsquigarrow S_I$ and $B(S_F) \rightsquigarrow S_F$ hold, thus, we have $B(S_I) \rightsquigarrow S_I \rightsquigarrow S_F$ and $B(S_F) \rightsquigarrow S_F$. That is, there is a sequence of single column/row operations starting from $B(S_I)$ and another starting from $B(S_F)$ that eventually make the two shapes equal (starting originally from two unequal baseline shapes). So, there must be a pair σ and σ' of such sequences minimizing the maximum length $\max_{\sigma,\sigma'}(|\sigma|, |\sigma'|)$ until the two shapes first become equal.

Call S_t and $S'_{t'}$ the dynamically updated shapes by σ_t and $\sigma'_{t'}$, respectively. In what follows we omit the time step subscripts. Let us assume w.l.o.g. that it is the last step t_{min} of σ that first satisfies $S = S'$ and that this step is a doubling of a column C. Thus, after step t_{min}, both S and S' contain an equal number of at least two consecutive copies of C. But the only way a shape can first obtain two consecutive copies of a column is by doubling one of its columns, thus, there must be a previous single column doubling operation in σ' that doubled column C (note that, at that point, C could have been a subset of the final version of the column). Deleting that operation from σ' and the last operation at t_{min} from σ, yields a new pair of sequences that satisfy $S = S'$ at some $t \leq t_{min} - 1$, thus, contradicting minimality of the (σ, σ') pair. We must, therefore, conclude that $S_I \rightsquigarrow S_F$ cannot hold in this case. □

Lemma 2. *For any S_I, S_F satisfying the conditions of Theorem 2, there is a constructor from S_I to S_F using at most $2 \log n$ time steps, where n is the total number of nodes in S_F.*

Proof. Since there is a constructor from S_I to S_F, then, by Theorem 2, $B(S_I) = B(S_F) = B$ and for every column C and row R of B it holds that $M_{S_F}(C) \geq M_{S_I}(C)$ and $M_{S_F}(R) \geq M_{S_I}(R)$. By Definition 5 (in Section 2) $S_I \rightsquigarrow S_F$, S_F can be obtained by applying on every column C and row R of S_I as many *RC doubling operations* as required to make its multiplicity equal to S_F. W.l.o.g. we only show this process applied to columns.

Let C be a column of B. Starting from $M_{S_I}(C)$ copies of C in S_I we want to construct the $M_{S_F}(C)$ copies of C in S_F. Note that neither $M_{S_F}(C)$ nor $M_{S_F}(C) - M_{S_I}(C)$ are necessarily powers of 2. Then, let 2^k be the greatest power of 2, such that $M_{S_I}(C)2^k < M_{S_F}(C)$, i.e., $M_{S_I}(C)2^k < M_{S_F}(C) < M_{S_I}(C)2^{k+1}$. Then, from the second inequality, it holds that $M_{S_F}(C) - M_{S_I}(C)2^k < M_{S_I}(C)2^{k+1} - M_{S_I}(C)2^k$ and this leads to $M_{S_F}(C) - M_{S_I}(C)2^k < M_{S_I}(C)2^k$, which means that if we construct $M_{S_I}(C)2^k$ columns then columns remaining to be constructed to reach $M_{S_F}(C)$ will be less than the constructed ones.

So, we construct $M_{S_I}(C)2^k$ columns (including the original column) by always doubling, within $k \leq \log(M_{S_F}(C))$ steps. Once we have those, we double in one additional time step $M_{S_F}(C) - M_{S_I}(C)2^k$ of those to get a total of $M_{S_F}(C)$ columns within $k + 1 \leq \log(M_{S_F}(C))$ steps. If we set $M_{S_F}(C)$ to be the maximum multiplicity of S_F, then for every column $C' \neq C$, its multiplicity $M_{S_F}(C') \leq M_{S_F}(C)$ can be constructed in parallel to the multiplicity of C, thus, within these $\log(M_{S_F}(C))$ steps. And similarly for rows. As $M_{S_F}(C) \leq n$ and $M_{S_F}(R) \leq n$, where n is the number of nodes of S_F, it holds that all column and row multiplicities can be constructed within at most $2 \log n$ time steps. □

We now present an informal description of a linear-time algorithm for ShapeConstruction. The algorithm decides whether a shape S_F can be constructed from a shape S_I and, if the answer is positive, it returns an $O(\log n)$-time step constructor.

Given a pair of shapes (S_I, S_F), do the following:

Step 1 Determine the *baseline shapes* $B(S_I)$ and $B(S_F)$ of S_I and S_F, respectively. Then compare $B(S_I)$ with $B(S_F)$ and, if they are equal, proceed to Step 2, otherwise return *No* and terminate.

Step 2 Since we have $B = B(S_I) = B(S_F)$, if for all columns C (rows R) of B it holds that $M_{S_I}(C) \leq M_{S_F}(C)$ and $M_{S_I}(R) \leq M_{S_F}(R)$ then proceed to Step 3, else return *No* and terminate.

Step 3 Output the constructor defined by Lemma 2.

Finally, together Proposition 1, Theorem 2, and Lemma 2 imply that:

Theorem 3. *The above algorithm is a linear-time algorithm for* ShapeConstruction *under RC doubling operations. In particular, given any pair of shapes* (S_I, S_F)*, when* $(S_I \rightsquigarrow S_F)$ *the algorithm returns a constructor* σ *of* S_F *from* S_I *of* $O(\log n)$*-time steps.*

5 Doubling

This section studies *doubling* operations in their most general form, where a subset of individual nodes can be involved in a growth operation. We start with a formal definition of two sub-types of general doubling operations and then investigate both the class characterization and ShapeConstruction problems. By focusing on the special case of a singleton S_I, we give a universal linear-time step (i.e., slow) constructor and, on the negative side, prove that some shapes cannot be constructed in sublinear time steps. Our main results are then two universal constructors that are efficient (i.e., polylogarithmic time steps) for large classes of shapes. Both constructors can be computed by polynomial-time centralized algorithms for any input S_F.

5.1 Rigidity in Doubling Operations

Given a shape S and two neighboring nodes $u, v \in S$, let $S(u)$ and $S(v)$ be the maximal connected sub-shapes of S containing u but not v and v but not u, respectively. When u is doubling in the direction of v, call that direction d, rigidity of connections (see Definition 3) implies that any $w \in S(u) \setminus S(v)$ must remain in its position while any $z \in S(v) \setminus S(u)$ must translate by 1 in direction d. For any node in $S(u) \cap S(v)$ these two actions would contradict each other. Such nodes belong to a $u, v, \dots, u$ cycle, and any such cycle must break or grow in at least one of its connections, in addition to the connection uv which will by assumption grow. In this paper, we focus on the case where all these cycles break (or grow) at the (C_j, C_{j+1}) cut. Depending on how we choose to treat such cycles, we shall define two sub-types of general *doubling operations*: *rigidity-preserving doubling* and *rigidity-breaking doubling*. Intuitively, in the former for all affected edges e in the (C_j, C_{j+1}) cut a node is generated over e, while in the latter any subset of those edges can simply break.

We start with a special case of the rigidity-breaking doubling operation in which, in every time step, a single node doubles. This special case is particularly convenient for the class characterization problem, as it can provide a (slower but simple) way to simulate both types of doubling operations. It also serves as an easier starting point towards the definition of the more general operations.

Definition 15 (Single-Node Doubling). *A* single-node doubling operation *is a growth operation in which at any given time step t, a direction $d \in \{north, east, south, west\}$ is fixed and a single node u of shape S doubles in direction d.*

Consider w.l.o.g. an east doubling operation applied on $u = (u_x, u_y) \in C_j$ of S. If u has no east neighbor in S, then, u generates a new node $u' = (u_x+1, u_y) \in C_{j+1}$ and the obtained shape is $S \cup \{u'\}$. Otherwise, u has a neighbor $v \in C_{j+1}$ of S which will need to translate by 1 in the east *direction together with some sub-shape of $S(v)$. We identify the maximal connected sub-shape $S'(u) \subseteq S(u)$ that contains no node from columns C_m, for all $m \geq j+1$, and the maximal connected sub-shape $S'(v) \subseteq S(v) \setminus S'(u)$. That is, $S'(u)$ contains all nodes on u's side that must stay put, while, from the remaining nodes, $S'(v)$ contains all nodes that must translate by 1. Any bicolor edge (one whose one endpoint is in $S'(v)$ and the other endpoint in $S'(u)$; we call these the* bicolor edges associated with *uv) must be an edge of the (C_j, C_{j+1}) cut. We remove all bicolor edges in order to perform the operation.*

Definition 16 (Rigidity-Preserving Doubling Operation). *A rigidity-preserving doubling operation is a generalization of a single-node doubling operation. In every time step a direction d is fixed and, for any node u that doubles towards a neighbor v in direction d and for all bicolor edges e associated with uv, a node is generated over e (see Fig. 3).*

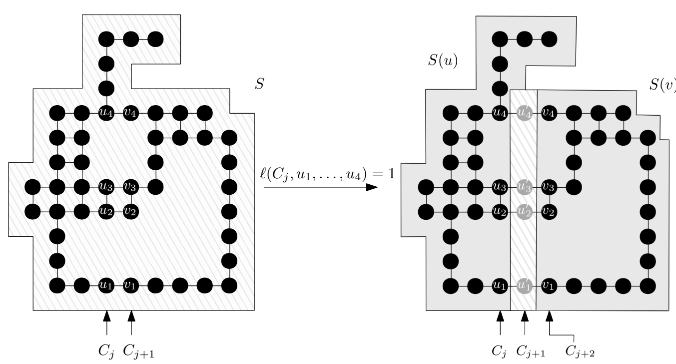

Fig. 3: An illustration of Definition 16, in which all nodes of (C_j) must double and the sub-shape $S(v)$ must be shifted to the east by one.

Definition 17 (Rigidity-Breaking Doubling Operation). *A rigidity-breaking doubling operation is a generalization of a single-node doubling operation. In every time step a direction d is fixed and, for any node u that doubles towards* *a neighbor v in direction d and for all bicolor edges e associated with uv, either a node is generated over e or e is removed (see Fig. 4).*

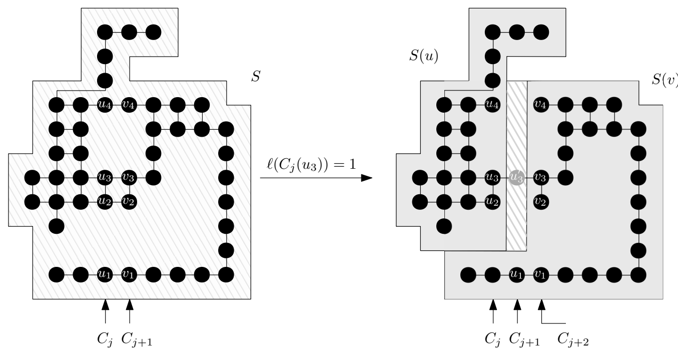

Fig. 4: An illustration of Definition 17, where there is one node $u_3 \in C_j$ doubles to the east and shifts the connected component in the same direction, while other edges in C_j are removed.

5.2 Universal Constructors of S_F

Proposition 2. *For any shapes S_I and S_F, where $S_I \subseteq S_F$, there is a linear-time step constructor of S_F from S_I.*

We call any $L \geq 1$ consecutive nodes connected horizontally or vertically an *L-line.*

Proposition 3. *If a 3-line is ever generated, it must be preserved in the final shape S_F, that is, rigidity-preserving doubling operation will never break the 3-line.*

A staircase is a shape S, in which each step consists of at least 3 consecutive nodes, whereas an exact-staircase consists of two nodes.

Proposition 4. *A staircase of size n requires $\Omega(n)$ time steps to be generated by rigidity-preserving doubling operations.*

Proposition 5. *A rigidity-preserving or rigidity-breaking doubling operation cannot build an exact staircase shape S within a sublinear time.*

Next, by putting together the universal linear-time steps constructor of Proposition 2 for doubling and the logarithmic-time steps constructor of Theorem 2 for RC doubling, we get the following general and faster constructor for doubling.

Theorem 4. *Given any connected target shape S_F, there is an $[O(|B(S_F)|) + O(\log |S_F|)]$-time step constructor of S_F from $S_I = \{u_0\}$ through doubling operations. Moreover, there is a polynomial-time algorithm computing such a constructor on every input S_F.*

The constructor of Theorem 4 is fast as a function of $n = |S_F|$, when $|S_F| - |B(S_F)|$ is large. For example, for all S_F for which $|B(S_F)| = O(\log |S_F|)$ holds, it gives a logarithmic-time steps constructor of S_F. It is also a fast constructor for all shapes S_F that have a relatively small (geometrically) *similar* shape S_I under *uniform scaling*. Note that shape similarity can be decided in linear time [19, 2]. In such cases, S_I can again be constructed in linear time steps from a singleton, followed by a fast construction of S_F from S_I via full doubling in all directions in a round-robin way.

Finally, we give an alternative constructor, based on a partitioning of an orthogonal shape into the minimum number of rectangles. Note that there are efficient algorithms for the problem, e.g., an $O(n^{3/2} \log n)$-time algorithm [17, 18]. These algorithms, given an orthogonal polygon S, partition S into the minimum number h of rectangles $S_1, S_2, \ldots, S_h$, "partition" meaning a set of pairwise non-overlapping rectangles which are sub-polygons of S and whose union is S.

Theorem 5. *Given any connected target shape S_F, there is an $O(h \log |S_F|)$-time step constructor of S_F from $S_I = \{u_0\}$ through doubling operations, where h is the minimum number of rectangles in which S_F can be partitioned. Moreover, there is a polynomial-time algorithm computing such a constructor on every input S_F.*

References

1. Akitaya, H.A., Arkin, E.M., Damian, M., Demaine, E.D., Dujmović, V., Flatland, R., Korman, M., Palop, B., Parada, I., Renssen, A.v., et al.: Universal reconfiguration of facet-connected modular robots by pivots: the O(1) musketeers. Algorithmica **83**(5), 1316–1351 (2021)
2. Akl, S.G., Toussaint, G.T.: An Improved Algorithm to Check for Polygon Similarity. Inf. Process. Lett. **7**(3), 127–128 (1978)
3. Almethen, A., Michail, O., Potapov, I.: Pushing lines helps: Efficient universal centralised transformations for programmable matter. Theoretical Computer Science **830**, 43–59 (2020)
4. Aloupis, G., Collette, S., Demaine, E.D., Langerman, S., Sacristán, V., Wuhrer, S.: Reconfiguration of cube-style modular robots using O(logn) parallel moves. In: International symposium on algorithms and computation. pp. 342–353. Springer (2008)
5. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed computing **18**(4), 235–253 (2006)
6. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing **20**(4), 279–304 (2007)

7. Chan, M.M., Smith, Z.D., Grosswendt, S., Kretzmer, H., Norman, T.M., Adamson, B., Jost, M., Quinn, J.J., Yang, D., Jones, M.G., et al.: Molecular recording of mammalian embryogenesis. Nature **570**(7759), 77–82 (2019)
8. Connor, M., Michail, O., Potapov, I.: Centralised connectivity-preserving transformations for programmable matter: a minimal seed approach. Theoretical Computer Science (2022). https://doi.org/https://doi.org/10.1016/j.tcs.2022.09.016
9. Cornejo, A., Dornhaus, A., Lynch, N., Nagpal, R.: Task allocation in ant colonies. In: International Symposium on Distributed Computing. pp. 46–60. Springer (2014)
10. Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T.: Universal shape formation for programmable matter. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures. pp. 289–299 (2016)
11. Derakhshandeh, Z., Gmyr, R., Richa, W., Scheideler, C., Strothmann, T.: Universal coating for programmable matter. Theoretical Computer Science **671**, 56–68 (2017)
12. Derakhshandeh, Z., Richa, A., Dolev, S., Scheideler, C., Gmyr, R., Strothmann, T.: Brief announcement: Amoebot-a new model for programmable matter. In: 26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2014. pp. 220–222. Association for Computing Machinery (2014)
13. Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y.: Shape formation by programmable particles. Distributed Computing **33**(1), 69–101 (2020)
14. Doty, D.: Theory of algorithmic self-assembly. Communications of the ACM **55**(12), 78–88 (2012)
15. Dumitrescu, A., Pach, J.: Pushing squares around. In: Proceedings of the twentieth annual symposium on Computational geometry. pp. 116–123 (2004)
16. Dumitrescu, A., Suzuki, I., Yamashita, M.: Formations for fast locomotion of metamorphic robotic systems. The International Journal of Robotics Research **23**(6), 583–593 (2004)
17. Imai, H., Asano, T.: Efficient algorithms for geometric graph search problems. SIAM Journal on Computing **15**(2), 478–494 (1986)
18. Keil, J.M.: Polygon decomposition. Handbook of computational geometry **2**, 491–518 (2000)
19. Manacher, G.K.: An application of pattern matching to a problem in geometrical complexity. Inf. Process. Lett. **5**(1), 6–7 (1976)
20. Mertzios, G.B., Michail, O., Skretas, G., Spirakis, P.G., Theofilatos, M.: The complexity of growing a graph. arXiv preprint arXiv:2107.14126 (2021)
21. Michail, O.: Terminating distributed construction of shapes and patterns in a fair solution of automata. Distributed Computing **31**(5), 343–365 (2018)
22. Michail, O., Skretas, G., Spirakis, P.G.: On the transformation capability of feasible mechanisms for programmable matter. Journal of Computer and System Sciences **102**, 18–39 (2019)
23. Michail, O., Skretas, G., Spirakis, P.G.: Distributed computation and reconfiguration in actively dynamic networks. In: Proceedings of the 39th Symposium on Principles of Distributed Computing. pp. 448–457 (2020)
24. Michail, O., Spirakis, P.G.: Simple and efficient local codes for distributed stable network construction. Distributed Computing **29**(3), 207–237 (2016)
25. Rothemund, P.W.: Folding DNA to create nanoscale shapes and patterns. Nature **440**(7082), 297–302 (2006)
26. Rothemund, P.W., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 459–468 (2000)

27. Woods, D., Chen, H.L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: Proceedings of the 4th conference on Innovations in Theoretical Computer Science. pp. 353–354 (2013)
28. Woods, D., Doty, D., Myhrvold, C., Hui, J., Zhou, F., Yin, P., Winfree, E.: Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. Nature **567**(7748), 366–372 (2019)