On Geometric Shape Construction via Growth Operations*

Nada Almalki[a,b,*], Othon Michail[a]

[a] *Department of Computer Science, University of Liverpool, Liverpool, UK*
[b] *Department of Information Technology, Taif University, Taif, Saudi Arabia*

Abstract

We study algorithmic *growth processes* under a geometric setting. Each 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, S_{t-1}, generates the next, S_t, always satisfying $|S_t| > |S_{t-1}|$. We define three types of *growth operations* and explore the algorithmic and structural properties of their resulting processes. 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. Moreover, we want to determine whether a given shape S_F can be constructed from a given initial shape S_I using a finite sequence of growth operations of a given type, called a *constructor of* S_F. We give exact and partial characterizations of classes of shapes that can be constructed in polylog n time steps, polynomial-time centralized algorithms for deciding reachability between pairs of input shapes (S_I, S_F) and for generating constructors when S_F can be constructed from S_I, as well as some negative results.

Keywords: Centralized algorithm, Geometric growth operation, Growth process, Programmable matter, Constructor

*A preliminary version of the results in this paper has appeared in [AM22].

*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: N.almalki@liverpool.ac.uk, Nada.m@tu.edu.sa (Nada Almalki), Othon.Michail@liverpool.ac.uk (Othon Michail)

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 [AAD$^+$06, AAER07, MS16], ant colony optimization [CDLN14, Dot12], DNA self-assembly [Dot12, Rot06, RW00, WDM$^+$19], and the algorithmic theory of programmable matter [DGR$^+$16, AAD$^+$21, AMP20, MSS19, FNSS22].

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 growth operations being considered involve generating at most one new node in a specific direction from any existing node. This leads to a shape reconfiguration due to the generation of a set of nodes within it. Nodes (also known as *modules*) in this context represent the individual constituent parts of the system, like the cells of an organism or the individual modules of a reconfigurable robotic system (e.g., the individual DATOMs in the system proposed in [PB21]). The considered 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 [CSG$^+$19], and by the potential of the future development of self-replicating robotic systems.

In a recent study, Mertzios *et al.* [MMS$^+$21] investigated a network-growth process at an abstract graph-theoretic level, free from geometric constraints. Our goal here is to study similar growth processes under a geometric setting and to show how these can be fine-tuned to construct interesting geometric shapes efficiently, i.e., in a number of time steps polylogarithmic in their size. Aiming to focus exclusively on the effect of growth operations, we do not allow any form of shape reconfigurations other than local growth. Preliminary such growth processes, mostly for rectangular shapes, were developed by Woods *et al.* [WCG$^+$13]. Their approach was to first grow such shapes in polylogarithmic time steps and to then 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 only local growth operations.

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.* [MMS$^+$21]. 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 matter systems, because it allows, among others, the system to maintain its strength and coherence and enables sharing of resources between the system's constituent parts. In light of this, all shapes 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 S_I via a sequence of growth operations?* (ii) *Is there a polynomial-time centralized algorithm that can decide if a given final 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?*

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, each node doubles by generating a new node in a given direction, in *RC doubling*, entire columns or rows double, while in *doubling*, individual nodes can double (see Section 2 for detailed definitions of each growth operation).

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 without losing generality, we shall sometimes restrict attention to a single operation per time step (sequential version).

- To avoid having to deal with complicated forms of colliding operations, which is not among the main aims of the present work, 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 (these may also be referred to as $\{up, right, down, left\}$, respectively). For simplicity, we shall exclusively focus on the weaker model in which operations can only occur in the *north* and *east* directions. Our results remain true in the model using all directions. We leave for future research a careful consideration of the model using all directions.

For *full doubling*, we completely characterize the structure of the class of shapes that can be constructed 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, n being the number of nodes in S_F. Note that all time step complexities in this work are functions of the size of the final shape. For *doubling*, we show that some shapes cannot be constructed within 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. Finally, in Section 6, we conclude and give further research directions opened by our work.

1.2. Related work

Recent work has focused on studying the algorithmic principles of reconfiguration, with the potential of developing artificial systems that will

[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*.

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 [Dot12]. Several theoretical models of programmable matter have been developed, including DNA self-assembly and other passively dynamic models [Dot12, Mic18] as well as models enriched with active molecular components [WCG+13]. For another survey of papers related to growth in self-assembly, see [Woo15].

One example of a geometric programmable matter model, which is presented in [DRD+14], 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 [DGR+17] and shape formation [DGR+16, DLFS+20]. Other authors have investigated programmable matter modules that can rotate or slide over neighboring modules through an empty space [DP04, DSY04, MSS19, CMP22], with the goal of capturing the global reconfiguration capabilities of local mechanisms that are feasible to be implemented with existing technology. The authors in [MSS19] proved that the decision problem of transformation between two shapes is in $\mathbf{P}$. In addition, another recent research work [AMP20] 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.* [WCG+13], where a node can rotate a whole line of connected nodes, simulating arm rotation, or the one by Aloupis *et al.* [ACD+08] on Crystalline robots, equipped with powerful lifting capabilities.

A recent study in the field of highly dynamic networks, which is presented in [MMS+21], is partially inspired by the abstract-network approach followed in [MSS20]. 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 growth processes considered in this paper operate on a 2-dimensional square grid. Most of the time we work on a non-negative quadrant but sometimes there are a few exceptions. Each grid position (cross point) is identified by its x and y coordinates, $x \geq 0$ representing the column and $y \geq 0$ the row. Systems of this type consist of n nodes that form a connected shape S, as in Fig. 1. Each node u of shape S is represented by a circle occupying a position on the grid. The process evolves in discrete time steps and in every time step $t \geq 0$, 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.

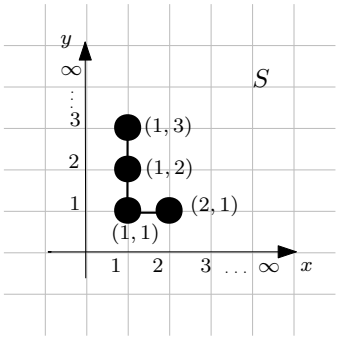


Figure 1: An example of a shape S with four nodes.

A row (respectively column) of a shape S is the set of all nodes of S with the same 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.

We use $w(C_j, u)$ to denote the number of columns of S 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 of S.

Similarly, $s(R_i, u)$ denotes the number of rows of S 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 of S.

For two sets X and Y we denote by $X \times Y$ the Cartesian product of X and Y.

A (p, q)-*rectangle* is a set of the form $A \times B$, where A and B are sets of *consecutive* integers of cardinally p and q, respectively. A set of points is called *rectangle* if it is a (p, q)-rectangle for some A and B of cardinalities p and q, respectively. Given an integer point u and two natural numbers p and q, the (p, q)-*rectangle around* u, denoted $\mathrm{Rec}(u, p, q)$, is the (p, q)-rectangle $\{u_x, u_x + 1, \ldots, u_x + p - 1\} \times \{u_y, u_y + 1, \ldots, u_y + q - 1\}$.

Definition 1 (Translation Operation). *Given a set of integer points Q, the north (south) k-*translation *of Q, for any $k \geq 1$, 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, for any $l \geq 1$, is defined as $\overset{l}{\rightarrow} Q = \{(x + l, y) \mid (x, y) \in Q\}$ ($\overset{l}{\leftarrow} Q$, similarly defined). For example, in Fig. 2, after a full doubling operation the set of points in $C_4 \in (S_{t-1})$ are translated by 3 to the east in S_t.*

Definition 2 (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. This extends inductively to a k-translation, for any $k \geq 1$.*

Throughout, all connections are assumed to be rigid. We also assume that, at the beginning of each time step, every pair of neighboring nodes is connected by a rigid connection. Note that our most general growth operation requires some rigid connections to be removed before the operation is applied.

The basic concept of a growth operation is that a node $u \in S_{t-1}$ generates a new node $u' \in S_t$, in which case we may also say that u *doubles* in time step t. Furthermore, all types of growth operations considered are *linear strength*, in our case meaning that even a single generated node u' is capable of translating (by pushing) any subset of the current shape instance. In

general, an operation is *linear-strength* if for any shape S of size n, the move of a subshape of S of size $O(1)$ (or larger) can cause the move of a subshape of S of size $\Theta(n)$. For example, in [AMP20] a single node can push a line of nodes of length up to $n-1$, in [ACD+08] a single Crystalline atom can push or pull a component of size up to $n-1$, and in [WCG+13] a monomer rotating around a neighboring monomer pulls with it a whole arm of monomers, which can be used to rotate an arm of length $\Theta(n)$ within $O(\log n)$ time steps.

Throughout this paper, l, k will represent the total number of horizontal and vertical growth operations performed, respectively. When an operation is horizontal, the direction d is either *east* or *west* and when vertical, the direction d is either *north* or *south*.

Definition 3 (Growth Operation). *A* growth operation *o is an operation that when applied on a shape instance S_{t-1}, for all time steps $t \geq 1$, yields a new shape instance $S_t = o(S_{t-1})$, 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*. For the sake of clarity, we now provide a high-level overview of these operations, with their more technical definitions appearing in the respective sections, that is, in Sections 3, 4, and 5.

A *full doubling* operation is a growth operation in which every node $u \in S_{t-1}$ generates a new node $u' \in S_t$, that is, $|S_t| = 2|S_{t-1}|$. Then, *row and column doubling*, abbreviated *RC doubling*, is a growth operation where, in each time step t, a subset of columns (rows) of the shape is selected and these are fully doubled. Finally, the most general version of these operations is the *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. 2.

Definition 4 (Reachability Relation). *For 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.*

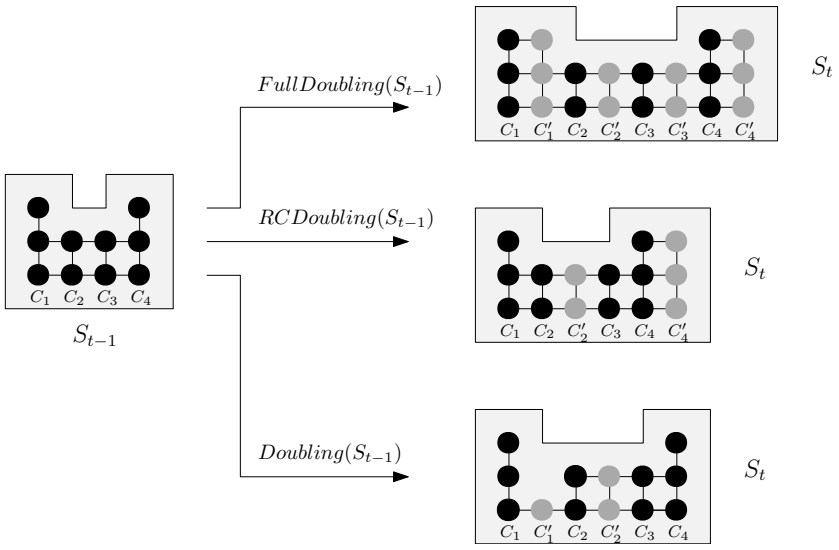


Figure 2: Illustration of the new shape S_t obtained by applying different types of growth operations on the same shape S_{t-1}, where the direction of growth is east. In this paper, nodes in S_{t-1} are usually colored black and the newly generated nodes gray.

Definition 5 (Constructor). *A* constructor $\sigma = (o_1, o_2, \ldots, o_{t_{last}})$ *is a finite sequence of doubling operations of a given type*[2]*, applied from a given initial shape S_I to yield a final shape S_F. We then say that σ is a constructor of S_F from S_I.*

Remark 1. *Note that, in Definition 5, the directions of different o_i's, $1 \leq i \leq t_{last}$, need not be the same.*

2.1. Problem Definitions

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

ClassCharacterization. Identify the family of shapes S_F that can be obtained from a given initial connected shape S_I via a sequence of growth

[2]A reasonable formal representation of an operation of a given type in a time step, consists of a direction together with a complete description of the points that must double in that time step. In (general) doubling, as will become evident when it is formally defined in Section 5, a description of the connections that must be removed before the operation is applied should additionally be provided.

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 constructor σ of S_F from S_I. In the special case of this problem in which S_I is always a singleton, the input is just S_F.

3. Full Doubling

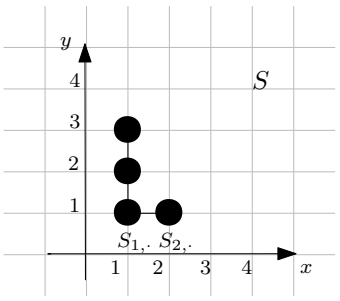
In this section, after providing a formal definition of the full doubling operation, we investigate the ClassCharacterization problem under this operation. We do this first for the special case of $|S_I| = 1$ (in Section 3.1), and then for the more general case of $|S_I| \geq 1$ (in Section 3.2).

Definition 6 (Full Doubling). *After applying a* full doubling operation *on a shape S a* new shape S' *is obtained, depending only on the direction d of the operation:*

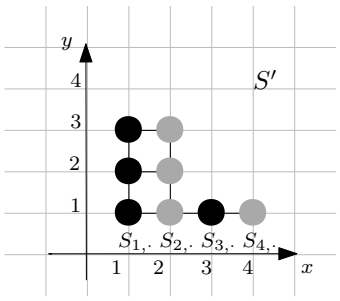
1. *If the direction d of the* full doubling operation *is* east*, then for every column $S_{j,\cdot}$ of S a new column is generated to the* east *of $S_{j,\cdot}$. The effect of applying this to all columns is that every column $S_{j,\cdot}$ of S is translated to the east by $j-1$, such that $S'_{2j-1,\cdot}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

Intuitively, if a full doubling operation is applied 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.

Fig. 3 depicts an example where S consists of two columns $S_{1,\cdot}$ and $S_{2,\cdot}$. Applying a full doubling operation in the east direction results in S', where $S_{1,\cdot}$ (which becomes $S'_{1,\cdot}$) generates $S'_{2,\cdot}$ and $S_{2,\cdot}$ is translated by 1 to the east and becomes $S'_{3,\cdot}$, which in turns generates $S'_{4,\cdot}$.



(a) Initial shape S.



(b) S' after a full doubling operation.

Figure 3: Illustration example of the new shape S' after applying a full doubling operation on the columns of S, in the east direction.

3.1. Singleton Initial Shape: $|S_I| = 1$

This section characterizes shapes that can be obtained by a *full doubling* operation that starts with a singleton initial shape S_I, that is, satisfying $|S_I| = 1$. The characterization is straightforward and its main purpose is to illustrate the dynamics of the *full doubling* operation.

Proposition 1. *Let l and k be natural numbers. A full doubling constructor with l horizontal and k vertical full doubling operations that starts from a single node constructs a rectangle of size $2^l \times 2^k$.*

Proof. We will prove the statement by induction on the number $t = l + k$ of operations. For $t = 0$ the statement is trivial.

Let $t > 0$ and assume that the proposition holds for all constructors with $t - 1$ full doubling operations.

Let $\sigma = (o_1, o_2, \ldots, o_t)$ be any full doubling constructor with exactly t operations and let l and k be the number of horizontal and vertical full doubling operations in this constructor, respectively, that is, $t = l + k$. Assume first that the last operation in σ is a horizontal full doubling and consider the constructor $\sigma' = (o_1, o_2, \ldots, o_{t-1})$ consisting of the first $t - 1$ operations of σ, where in this case $t - 1 = (l - 1) + k$. By the induction hypothesis, σ' constructs a rectangle of size $2^{l-1} \times 2^k$. The horizontal full doubling operation that is applied in time step t, doubles all 2^{l-1} columns of the rectangle, thus creating a rectangle of size $2^l \times 2^k$, as desired.

The case when the last operation of σ is a vertical doubling is proved similarly. Let us assume that the last full doubling operation of σ is a vertical

doubling and consider that the constructor $\sigma' = (o_1, o_2, \ldots, o_{t-1})$ consisting of the first $t-1$ operations of σ, where in this case $t-1 = l+(k-1)$. By the induction hypothesis, σ' constructs a rectangle of size $2^l \times 2^{k-1}$. The vertical full doubling operation that is applied in time step t, doubles all 2^{k-1} columns of the rectangle, thus creating a rectangle of size $2^l \times 2^k$, as desired (see Fig. 4).

□

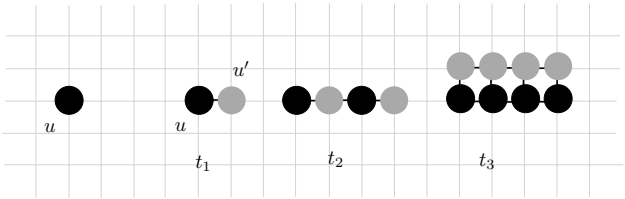


Figure 4: Construction overview of a rectangle where $n = 2^2 \times 2$.

Remark 2. *From Proposition 1, it follows that the final shape of a full doubling constructor depends on the number of horizontal and vertical operations and not on the order of these operations.*

3.2. An Arbitrary Connected Initial Shape: $|S_I| \geq 1$

In this section, we characterize the class of 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 7 (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:*

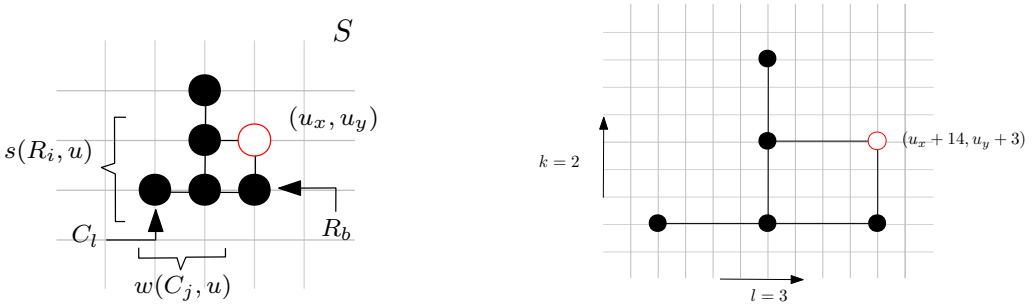
1. *First, the coordinates of $|S|$ points of $F_{l,k}(S)$ are determined as a function of the coordinates of the points of S. For each $u \in S$ the coordinates of $u' \in F_{l,k}(S)$ are given by $(u_x + (2^l - 1)w(C_j, u), u_y + (2^k - 1)s(R_i, u))$, as in Fig. 5 (a and b).*
2. *Generate the Cartesian 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*, when applied on an input shape S, is a shape $F_{l,k}(S) = \bigcup_{u \in S} Rec(u', 2^l, 2^k)$, where u' is a function of u in the union (see Fig. 6 for an illustration).*

Remark 3. *An equivalent, more compact definition of* reconfiguration function *is that given two integers $l, k > 0$, a* reconfiguration function $F_{l,k}$ *maps an input shape S to the output shape $F_{l,k}(S)$ as follows:*
For every node $u \in S$, let u' be the point $(u_x + (2^l - 1)w(C_j, u), u_y + (2^k - 1)s(R_i, u))$. Then,

$$F_{l,k}(S) := \bigcup_{u \in S} \mathrm{Rec}(u', 2^l, 2^k).$$

However, our next results rely on the former definition of reconfiguration function*.*



(a) Input shape S.

(b) After step 1 of *reconfiguration function.*

Figure 5: Step 1 of a *reconfiguration function* for $l = 3$ and $k = 2$. Node $u = (u_x, u_y) \in S$ (red node in (a)) will give a new node $u' = (u'_x + 14, u'_y + 3) \in F_{3,2}(S)$ (red node in (b)) and similarly for the other nodes. For an interpretation of colors the reader is referred to the online version of the article.

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

Proof. By definition, $F_{l,k}(S)$ gives a new point $u' = (u_x + (2^l - 1)w(C_j, u), u_y + (2^k - 1)s(R_i, u))$ for each $u \in S$ and a rectangular set of points $R(u') = \{u'_x, u'_x + 1, \ldots, u'_x + (2^l - 1)\} \times \{u'_y, u'_y + 1, \ldots, u'_y + (2^k - 1)\}$ originating at

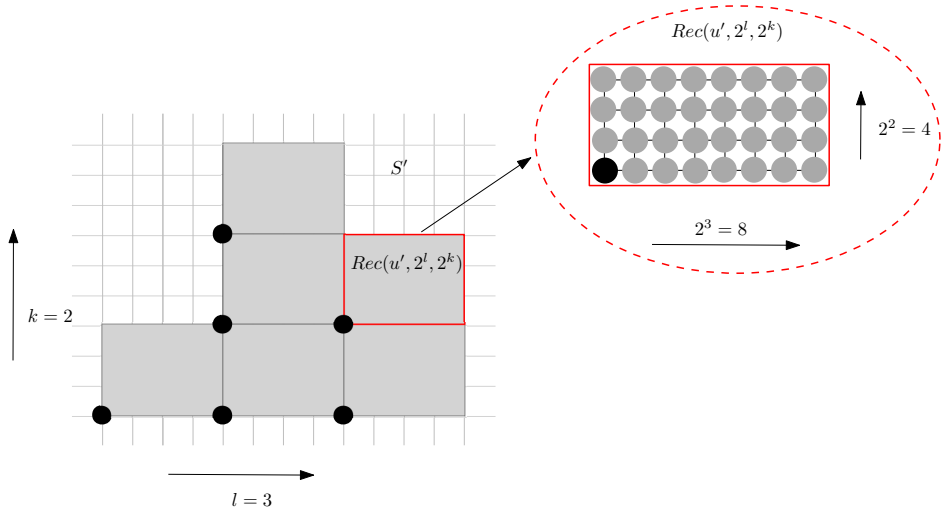


Figure 6: An example of the output shape S' after applying the *reconfiguration function* $F_{l,k}$ on shape S, where S is represented by the black bold nodes and were compressed before applying $F_{l,k}$.

u'. This gives a new shape S'. Applying $F_{l',k'}$ to each u' gives a new point u'', such that

$$\begin{aligned}
u'' &= (u'_x + (2^{l'} - 1)w(C_j, u'), u'_y + (2^{k'} - 1)s(R_i, u')) \\
&= (u_x + (2^l - 1)w(C_j, u) + (2^{l'} - 1)w(C_j, u'), u_y + (2^k - 1)s(R_i, u) + \\
&\quad (2^{k'} - 1)s(R_i, u')) \\
&= (u_x + (2^l - 1)w(C_j, u) + (2^{l'} - 1)2^l w(C_j, u), u_y + (2^k - 1)s(R_i, u) + \\
&\quad (2^{k'} - 1)2^k s(R_i, u)) \\
&= (u_x + (2^{l'+l} - 1)w(C_j, u), u_y + (2^{k'+k} - 1)s(R_i, u)).
\end{aligned}$$

The set of all these points u'' is, thus, equivalent to the set of points returned by step (1) of $F_{l'+l,k'+k}(S)$.

For the remaining points, take again any $u \in S$. The rectangle of u'' returned by $F_{l'+l,k'+k}(S)$ consists of all points enclosed within the following four corners: (u''_x, u''_y),

$$\begin{aligned}
&(u''_x + (2^{l'+l} - 1), u''_y + (2^{k'+k} - 1)) = \\
&(u_x + (2^{l'+l} - 1)w(C_j, u) + (2^{l'+l} - 1), u_y + (2^{k'+k} - 1)s(R_i, u) + (2^{k'+k} - 1)) = \\
&(u_x + (2^{l'+l} - 1)(w(C_j, u) + 1), u_y + (2^{k'+k} - 1)(s(R_i, u) + 1)), \qquad (1)
\end{aligned}$$

$(u''_x, u_y + (2^{k'+k} - 1)(s(R_i, u) + 1))$, and $(u_x + (2^{l'+l} - 1)(w(C_j, u) + 1), u''_y)$.

We have already shown that the point returned by $F_{l',k'}(F_{l,k}(S))$ for u is the same to the one returned by $F_{l'+l,k'+k}(S)$, that is, (u''_x, u''_y). It is sufficient to show that the opposite diagonal point returned by $F_{l',k'}(F_{l,k}(S))$ for u is the same as the point of Eq. 1.

Applying $F_{l,k}(S)$ gives for $u \in S$ a new point $u' = (u_x+(2^l-1)w(C_j, u), u_y+(2^k-1)s(R_i, u))$ and the top-right corner of the rectangle of u' is the point $v' = (u'_x+2^l-1, u'_y+2^k-1) = (u_x+(2^l-1)(w(C_j, u)+1), u_y+(2^k-1)(s(R_i, u)+1))$. Next, applying $F_{l',k'}$ to S' gives for v' a new point v'', whose x coordinate is

$$\begin{aligned} v''_x &= v'_x + (2^{l'} - 1)w(C_j, v') \\ &= v'_x + (2^{l'} - 1)(2^l w(C_j, u) + (2^l - 1)) \end{aligned} \tag{2}$$

and its y coordinate is $v''_y = v'_y + (2^{k'} - 1)(2^k s(R_i, u) + (2^k - 1))$. The top-right corner w of the rectangle of v'' is obtained by adding $2^{l'} - 1$ to both these coordinates, thus, yielding

$$\begin{aligned} w_x &= v'_x + 2^l(2^{l'} - 1)(w(C_j, u) + 1) \\ &= u'_x + (2^l - 1) + 2^l(2^{l'} - 1)(w(C_j, u) + 1) \\ &= u_x + (2^l - 1)(w(C_j, u) + 1) + 2^l(2^{l'} - 1)(w(C_j, u) + 1) \\ &= u_x + (2^{l'+l} - 1)(w(C_j, u) + 1) \end{aligned} \tag{3}$$

and, similarly, $w_y = u_y + (2^{k'+k} - 1)(s(R_i, u) + 1)$. Thus, w is the same point as the one returned by Eq. 1.

The lemma now follows by observing that the rectangle formed by $F_{l',k'}(F_{l,k}($ $S))$ within the area defined by the points (u''_x, u''_y), (u''_x, w_y), (w_x, w_y), and (w_x, u''_y) is missing no points. □

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

Proof. We will prove by induction that applying any sequence of l east and k north full doubling operations on shape S_I results in a shape S_F which is equivalent to the output of the reconfiguration function $F_{l,k}(S_I)$.

For the base case, let first $l = 1$ and $k = 0$, that is, a horizontal full doubling operation applied to S_I. By the definition of this operation (see Definition 6), the obtained shape is $S'_I = \bigcup_{1 \le j \le C}(\overset{j-1}{\rightarrow} S_{I\,j,\cdot} \cup \overset{j}{\rightarrow} S_{I\,j,\cdot})$. So,

given any $u \in S_I$, say in the jth column of S_I, u is translated right $j-1$ times, thus yielding a new point $u' = (u_x + j - 1, u_y) = (u_x + w(C_j, u), u_y)$, as required by (1) of Definition 7. Moreover, u is translated right j times to give another new point $u'' = (u_x + j, u_y) = (u'_x + 1, u_y)$, as required by (2) of Definition 7 for sets $\{u'_x, u'_x + 1\} \times \{u_y\}$. Thus, $S'_I = F_{1,0}(S_I)$ holds. A similar argument for $F_{0,1}(S_I)$ completes the proof of the base case.

For the induction hypothesis, let us assume that after l east and k north full doubling operations, starting from S_I, the obtained shape is $S = F_{l,k}(S_I)$. For the inductive step, we first consider the $(l+1)$th east operation is applied to S to give a new shape S'. By replacing in the base case S_I by S, we get that $S' = F_{1,0}(S) = F_{1,0}(F_{l,k}(S_I)) = F_{l+1,k}(S_I)$, where the second equality follows from the inductive hypothesis and the last equality from additivity of the reconfiguration function (Lemma 1).

For the second argument of $F_{l,k+1}(S_I)$, let us assume the $(k+1)$th north operation is applied to S to give a new shape S'. By replacing in the base case S_I by S, we get that $S' = F_{0,1}(S) = F_{0,1}(F_{l,k}(S_I)) = F_{l,k+1}(S_I)$, where the second equality follows from the inductive hypothesis and the last equality from additivity of the reconfiguration function (Lemma 1), and this ends the proof. □

4. RC Doubling

After a formal definition of the *RC doubling* operation, in this section, we study both the ClassCharacterization 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, for the yes instances, to additionally return an $O(\log n)$-time step constructor of S_F from S_I.

Definition 8 (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.*

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 9 (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,*

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

Fig. 7 and 8 illustrate Definition 9.

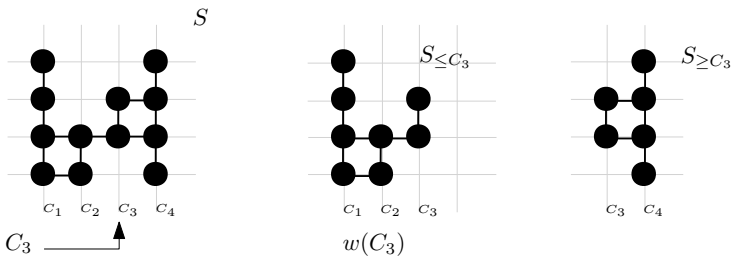


Figure 7: An example of a single RC doubling operation (Definition 9), where C_3 is the selected column to be doubled in the *east* direction.

Proposition 2 (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 RC doubling-operations.*

Proof. The "if" part follows trivially, because single RC doublingis a special case of *RC* (parallel) row and column doubling.

We now prove the "only if" part. That is, we will show that for any sequence σ of *RC* doubling operations generating S_F from S_I, there is a

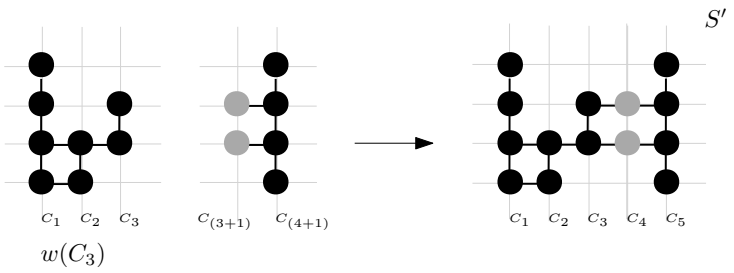


Figure 8: The final shape S' after a *single RC doubling operation*.

sequence of single *RC* operations σ' generating S_F from S_I. For every *RC* row/column operation o in σ, we add to σ' a break down of o into a sequence of individual operations of the columns (rows, resp.) in the subset. We do this for a west to east (bottom to top, resp.) ordering of the columns (rows, resp.), even though any permutation would do equally well. We will prove the equivalence for east column operations as all other operations simply follow by rotating the whole system by 90°, 180°, and 270°. Let $D = \{d_1, d_2, \ldots, d_{|D|}\}$ be the set of column indices in an RC column operation applied to a shape S and denote by o_D the operation and by $o_D(S)$ the shape obtained by applying o_D to S. Similarly, denote by $o_d(S)$ the shape obtained by applying a single doubling of column C_d, for $d \in D$, to shape S. Then it is sufficient to show that

$$o_D(S) = o_{d_{|D|}}(o_{d_{|D|-1}}(o(\ldots o_{d_2}(o_{d_1}(S))\ldots))).$$

It is sufficient to show that the right hand side of the above equation translates every column C_j of S, $w(C_j)$ times (the copies of columns C_d, $d \in D$, will then be trivially translated by $w(C_j)+1$ as also done by the left hand side of the equation).

By definition, $o_d(S) = S_{\leq C_d} \cup (\overset{1}{\rightarrow} S_{\geq C_d})$ translates all columns to the east of C_d by 1. Therefore, any given column C_j of S, will be translated by 1 for each of the column operations of $o_{d_{|D|}}(o_{d_{|D|-1}}(o(\ldots o_{d_2}(o_{d_1}(S))\ldots)))$ that happen to its west. The number of those is equal to the number of indices in D which are less than j, which is, by definition, equal to $w(C_j)$. □

Definition 10 (Consecutive Column/Row Multiplicities). *Given a shape S and a column C_j (row R_i) of S which is either the leftmost column, i.e., $j = 1$ (bottom-most row, i.e., $i = 1$), or $C_{j-1} \neq C_j$ ($R_{i-1} \neq R_i$), 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 11 (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 (see Fig. 9).*

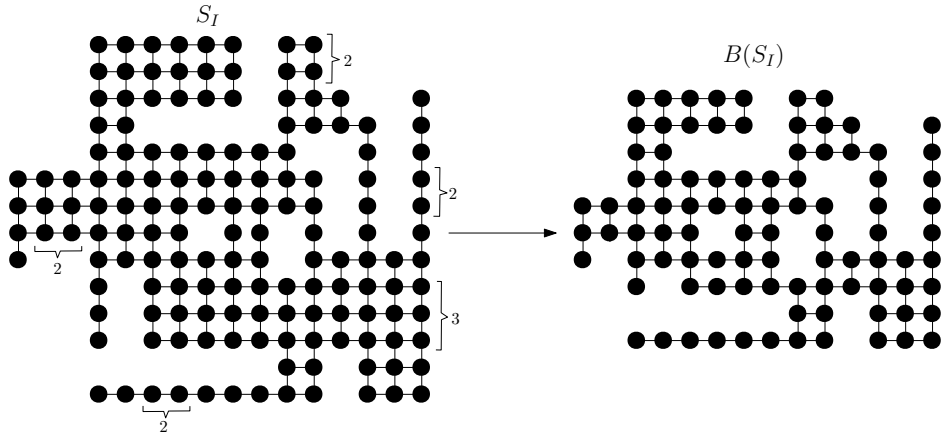


Figure 9: An example of extracting the baseline shape $B(S_I)$ from a shape S_I.

Observation 1. *Every shape S that has a given shape B as its baseline can be obtained by successively doubling the original columns and rows of B, thus, creating consecutive multiplicities of these columns and rows. Focusing on any given column C (row R) of B, we denote by $M_S(C)$ ($M_S(R)$) the consecutive multiplicity of that particular column (row) of B in S (and not its total multiplicity, in case two or more identical copies of a column or row exist in non-consecutive coordinates of B).*

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 2, 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, (see Fig. 10). □

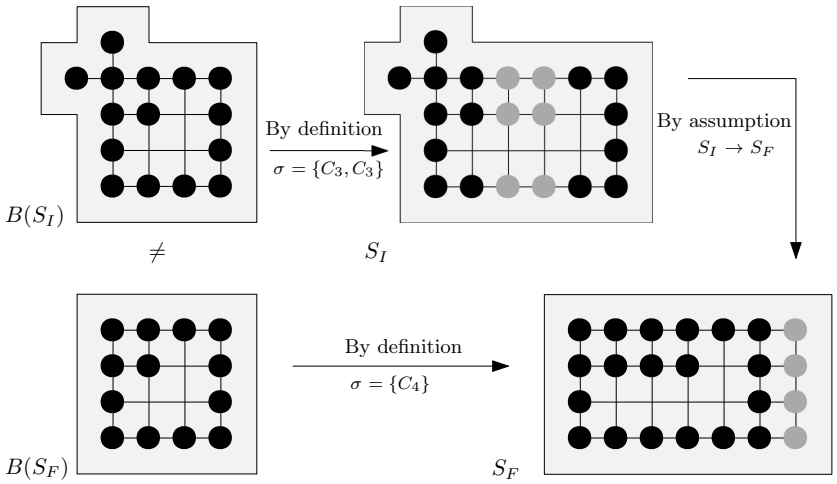


Figure 10: Theorem 2 illustration example.

Proposition 3. *For any shape S, there is a unique baseline shape $B(S)$.*

Proof. Let us assume a shape S where there are multiple baseline shapes, consider two of those shapes, $B_1(S)$ and $B_2(S)$ where $B_1(S) \neq B_2(S)$, by Definition 11, all columns and rows of these baseline shapes have multiplicity 1. Thus, these baseline shapes $B_1(S)$ and $B_2(S)$ are equal since they have the same multiplicity of the same column and the same row. Therefore, this is a contradiction, and there is only one baseline shape $B(S)$. □

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 4 (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. □

Observation 2. *To construct any shape S of size n, it requires at least $\lceil \log n \rceil$ 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.

After that, Algorithm 1, 2, and 3 shows the pseudo code that briefly formulates this procedure. 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.

The next Algorithm 1 does not depend on the order of removing consecutive columns/rows multiplicities or whether compress columns first or rows.

Algorithm 2 determines the feasibility of growing S_I to S_F, and then we use that decision to compute the constructor σ in Algorithm 3.

Algorithm 1: Baseline Shape B(S)

Input : S
Output: $B(S)$

```
/* Baseline function to check the multiplicity of every
   column and row of a shape S and return the B(S) after
   remove duplication.                                    */
Function Baseline-Shape(S):
    for every column (C_j of S) do
        if the multiplicity M_S(C_j) of column C_j is more than one
         then
            (C_j ← C_{j+1})
    for every row (R_i of S) do
        if the multiplicity M_S(R_i) of row R_i is more than one then
            (R_i ← R_{i+1})
    return B(S)
```

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

Theorem 3. *Algorithm 3 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.*

Proof. In order to prove that the running time of the constructor $S_I \overset{\sigma}{\rightsquigarrow} S_F$ is linear, we will analyse the pseudo code of Algorithm 1, 2 and 3. First, in Algorithm 1, we determine the baseline shape $B(S)$ by comparing every column C_j of shape S from left to right, that is, C_j to C_{j+1}, C_{j+2} until we find the first one C_{j+x} which is not equal to C_j. Then, we start from C_{j+x} and do the same for C_{j+x+1}, C_{j+x+2} until again we find the first one which is not equal to C_{j+x} and so on. Thus, every column is involved to at most one comparison to a column to its right, that is, the number of comparisons is $C_{|J|-1}$, the same argument holds for rows.

In order to determine the multiplicities of columns, we compare two columns C_j and C_{j+x} point-to-point until either the points are exhausted or the first pair of unequal points is found, which means that every point $(u_x, u_y) \in C_j$ is involved in at most one comparison to another point. There-

Algorithm 2: Decision

Input : (S_I, S_F)
Output: True if S_F can be obtained from S_I via a sequence of RC doubling operations; False otherwise

```
1 Compute the baseline of S_I, Baseline-Shape(S_I)
2 Compute the baseline of S_F, Baseline-Shape(S_F)
3 if B(S_I) is equal to B(S_F) then
4     return True
5 else
6     return False
```

fore, the total number of comparisons equals $|S|$, and the total number of time steps is linear.

For Algorithm 2, we compare $B(S_I)$ and $B(S_F)$, by translating them into the same origin and assigning the bottom-most point of their leftmost column to $(0, 0)$ and accordingly updating the coordinates of all other points. This procedure yields a decision which is based on Algorithm 1, which is also linear. Following that Algorithm 3, there is a variable $m(C_j)$ for each column C_j, that keeps track of the number C_j copies, and if the comparison of C_j and C_{j+x} is true, we increase this variable. The total number of increments for $m(C_j)$ is at most equal to the number of columns C_j, thus, C_J. Then, at each point of these duplicated columns $m(C_j)$, we subtract from it at most twice (once for columns and once for rows), for a total of two operations per point. Therefore, the overall running time for returning a sequence σ for the growth process is equal at most the number of nodes of S. □

5. Doubling

This section studies *doubling* operations in their most general form, where up to individual nodes can be involved in a growth operation. We define two sub-types of general doubling operations, the rigidity-preserving, and rigidity-breaking doubling operation. We start by defining a special type of a rigidity-breaking doubling operation in which, in every time step, a single node doubles. This special type is particularly convenient for the ClassCharacterization problem, as it can provide a (slower but simple) way to simulate other types of doubling operations. It also serves as an easier starting point toward the definition of the more general operations.

Algorithm 3: Constructor $S_I \overset{\sigma}{\rightsquigarrow} S_F$

Input : Decision, from Algorithm 2
Output: Constructor σ

while *Decision* **do**
  /* Find the constructor σ for columns. */
  for *every column (C_j of S_F)* **do**
    for *every column (C_j of S_I)* **do**
      if *the index of every column C_j of S_F and S_I are equal* **then**
        if *multiplicity of $M_{S_F}(C_j)$ is greater than $M_{S_I}(C_j)$* **then**
          compute the difference in $m(C_j)$
          append ($m(C_j)$ to σ)
compute the maximum of σ in max_{value}
/* Count the steps k for doubling columns. */
if $k = \log max_{value}$ mod2 *does not equal* 0 **then**
  add one extra step to k to double the remaining columns that are not power of 2
/* $m(C_j)$ in the returned σ is an abbreviation of the operations required to generate $m(C_j)$ copies by successive RC doubling operations of all copies of C_j starting from a single original copy. */
return $\sigma = \{m(C_1), m(C_2), \ldots, m(C_j)\}$

Moreover, 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 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.

Given a connected shape S and two neighboring nodes $u, v \in S$, assume that, in the current time step, u will double in the direction of v and call that direction d. Let $S(u)$ and $S(v)$ be the maximal connected subshapes of S containing u but not v and v but not u, respectively. In general, any such operation partitions S into two connected subshapes $S'(u) \subseteq S(u)$ and $S'(v) \subseteq S(v)$ containing u and v, respectively, where $S'(u)$ is the subshape

of all nodes that will stay put (*stable subshape*), while $S'(v)$ is the subshape of all nodes that will translate by 1 in direction d (*moving subshape*). This translation by 1 is due to v being pushed by the node generated by u and the rigidity of connections in $S'(v)$.

Definition 12 (Collision). *A doubling operation is said to cause a* collision*, if any of the following holds:*

- *After applying the operation, a node of $S'(v)$ lies over a node of $S'(u)$.*
- *There are two nodes $w, z \in S$ where z is a neighbor of w in direction d via a rigid connection, such that $w \in S'(u)$ and $z \in S'(v)$.*

An operation is called collision-free*, if it causes no collision.*

We now define the partitioning into $S'(u)$ and $S'(v)$ of the doubling operation through a coloring process. Note that different ways of partitioning and of deciding which rigid connections to remove, may yield different types of doubling operations, i.e., different resulting dynamics. In what follows, let d be w.l.o.g. "right" and C_j, C_{j+1} be the columns of u, v, respectively. At each time step t, before doubling u towards v, we compute a coloring of shape S in phases, as follows:

- Phase 1: Color black a maximal connected subshape of $S(u)$ that contains no node from columns C_m, for all $m \geq j + 1$.
- Phase 2: From the remaining nodes, color red a maximal connected subshape of $S(v)$.
- Phase 3: All the remaining uncolored nodes are colored black.

Set $S'(u)$ and $S'(v)$ to be the black and red, respectively, subshapes of S.

Definition 13 (Bicolor Edges). *Any edge of shape S having one endpoint in the stable subshape $S'(u)$ and the other endpoint in the moving subshape $S'(v)$, is called a* bicolor edge *(see Fig. 11).*

After computing the coloring of shape S, remove all bicolor edges and double u towards v. Before the end of the time step, introduce a rigid connection between each pair of neighboring nodes who are lacking one.

The following definition sums this up:

Definition 14 (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. If the neighboring position of u in direction d is unoccupied a new node is generated in that position. If it is occupied by a node v, a new node is generated in that position, all bicolor edges (as computed above) are removed, $S'(v)$ translates by 1 in direction d, and $S'(u)$ stays put. The operation is complete by introducing a rigid connection between each pair of neighboring nodes lacking one.*

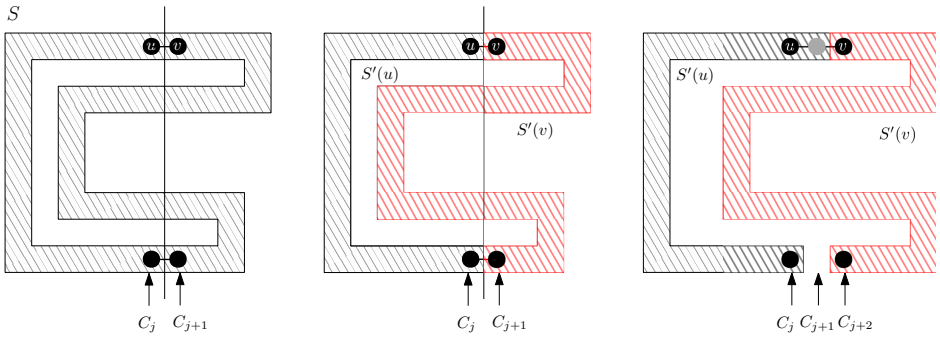


Figure 11: An example of breaking a bicolor edge in order to perform a doubling operation on node u.

Proposition 4. *All bicolor edges lie in the (C_j, C_{j+1}) cut and have the same color orientation, which is black in their C_j endpoint and red in their C_{j+1} endpoint.*

Proof. Let $e = wz$ be a bicolor edge such that z is the right neighbor of w. Observe first that e cannot lie completely to the left side of the cut. This is because if the red colored during phase 2 were uncolored with (i) a black neighbor in phase 1, then it would have been colored black in phase 1, and (ii) an uncolored neighbor in phase 1, that neighbor would have also been colored red in phase 2, both cases contradicting the assumption of e being bicolor.

Moreover, e cannot lie completely to the right side of the cut. When an endpoint of e is colored red during phase 2, the other endpoint must also be uncolored, because no black is added to the right of the cut in phase 1.

That endpoint would have also been colored red in phase 2, contradicting the assumption of e being bicolor.

We next prove that there can be no bicolor $e = wz$ with opposite color orientation. As proved, e must lie in the cut. For the sake of contradiction, assume that w is red and z is black. Note that z lies to the right of the cut, so it must have been uncolored at the end of phase 1. Then, coloring w red would imply coloring z red, thus contradicting the bicolor assumption for e.

It follows that all bicolor edges lie in the (C_j, C_{j+1}) cut and have the same color orientation. □

Proposition 5. *A single-node doubling operation on any connected shape S is collision-free.*

Proof. Assume two neighboring nodes $u, v \in S$, and a doubling operation on node u towards v. After computing the coloring process on S at time step t, we have $u \in S'(u)$ and $v \in S'(v)$, which means that u belongs to the *stable subshape* and v in the *moving subshape*. After that, any edge e that connects $S'(u)$ and $S'(v)$ is a bicolored edge and is removed at the same time step t before u doubles.

By Definition 12, there are two ways in which a doubling operation can cause a collision; by resulting in a node lying over another or a rigid connection to break.

For the first, a node of $z \in S'(v)$ can only lie over a node of $w \in S'(u)$ after $S'(v)$'s translation, if w was a neighbor of z in the direction of the translation. But this would imply that zw is a bicolor edge of opposite color orientation, thus violating Proposition 4.

For the second, if $w \in S'(u)$ and $z \in S'(v)$ are neighbors before $S'(v)$'s translation, then wz is a bicolor edge. Thus, the corresponding rigid connection is removed by the operation before the translation moves z away from w. As a consequence, no rigid connection breaks. □

In this paper, we focus on the case where every node of bicolor edges e at (C_j, C_{j+1}) either breaks its connection with its neighbors (or grows by doubling). Depending on how we choose to treat this, 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}) a node is generated over e, while in the latter any subset of those edges can simply break.

Definition 15 (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. 12).*

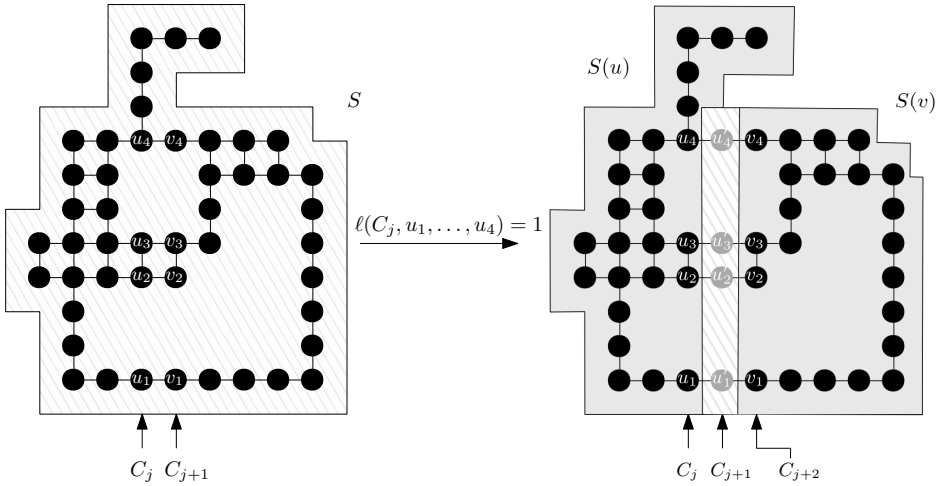


Figure 12: An illustration of Definition 15, in which all nodes of (C_j) must double and the sub-shape $S(v)$ must be shifted to the east by one.

Definition 16 (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. 13 and 11).*

Observation 3. *Observe that general doubling operations allow for the translation of components which can extend to both sides of the doubling node(s) (for example, the red component in Fig. 11). It is easy to see that such a translation of a $S'(v)$ can never be blocked by a component $S'(u)$ which stays put (e.g., the black component in Fig. 11). Because if a node in $S'(v)$ had a neighbor in the direction of translation belonging to the component $S'(u)$, then this would violate the maximality of $S'(u)$.*

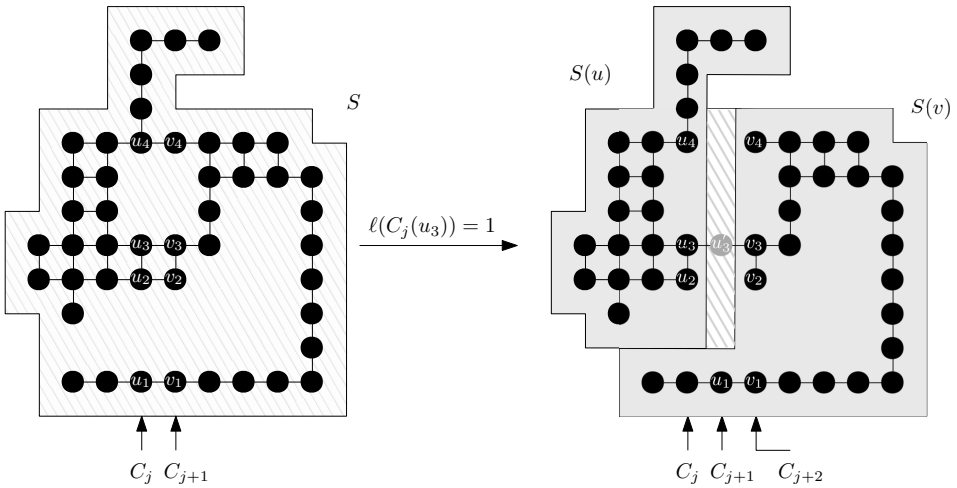


Figure 13: An illustration of Definition 16, where there is one node $u_3 \in C_j$ doubles to the east and shifts the connected component in the same direction, while other edges in C_j are removed.

Proposition 6. *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.*

Proof. Consider a pair of shapes S_I, S_F, to obtain the final shape S_F starting with a singleton initial shape S_I, compute a spanning tree as shown in Fig. 14. Then, select any node as its root, this selected root will correspond to the initial single node of S_I. Since a node can generate at most one new node per time step. Thus, each phase should consist of at most 4 time steps t. Then, in every phase generate the next level of a *breadth first search* of the spanning tree T. Due to the fact that the next *breadth first search* level always concerns positions that are empty in the shape and are adjacent to a node that has already been generated. Therefore, it is clear that it is possible to fill all these positions without the need to push any existing node.

For more general $S_I \subseteq S_F$, we already have the nodes in S_I and the nodes in $S_F \setminus S_I$ must be generated in order to construct S_F. We compute a spanning forest T of $S_F \setminus S_I$, each (maximal) connected component of which is a tree T_i spanning that component. For every such tree T_i we pick a neighbor u_i of T_i in S_I, which is guaranteed to exist by maximality of the components of T. Then from every u_i we set u_i as the root of $T_i \cup \{u_i\}$ and run the process we already have for singleton S_I, in order to construct T_i. □

We call any $L \geq 1$ consecutive nodes connected horizontally or vertically

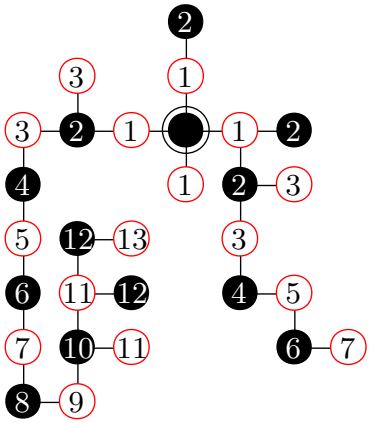


Figure 14: Constructing any shape S_F in linear time steps using BFS.

an *L-line*.

Proposition 7. *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.*

Proof. Let us consider a horizontal w.l.o.g. 3-line, since there are two possible directions of rigidity-preserving doubling operation that can be applied to the 3-line, either *north* or *east*, we will first assume a *north* rigidity-preserving doubling operation that breaks the 3-line into two distinct rows. As illustrated in Fig. 15 (a), there must be a row below the 3-line to perform this operation and break it. Since all nodes of the 3-line belong to the same component and such an assumption of applying this operation and break the 3-line cannot hold because it contradicts the definition of rigidity-preserving doubling operation (see Definition 15 where all nodes below the 3-line component must push the complete 3-line to the north and hence never break).

Now, let us assume the other case of applying an *east* rigidity-preserving doubling operation to break the 3-line, where a break means that there will be an unoccupied position between two consecutive nodes of 3-line, which means a column C_j splits the line into two parts.(i.e., it must be going through a node of the 3-line). Applying this operation by one of the nodes of 3-line, this will only expand the 3-line to the *east* and never break it. So, there must be a row below or above the 3-line. Let us consider it is below w.l.o.g, this implies that the node of C_j who generates to the east and pushed right and the

node of the 3-line breaking to the right must belong to the same component defined in the area to the right of the column, as in Fig. 15 (b). Therefore, this contradicts the fact that rigidity-preserving doubling operation when pushing a component must generate nodes in all attachment points, and this completes the proof. □

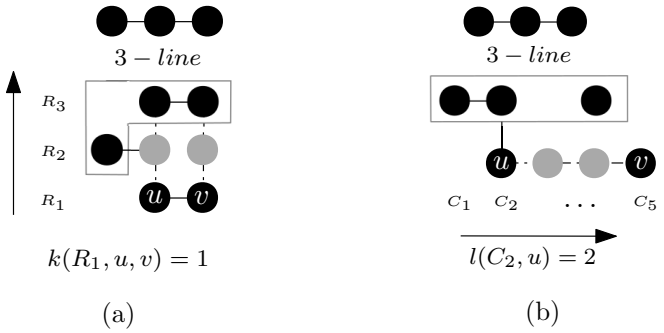


Figure 15: Maintaining the 3-line shape in the final construction.

Growing some shapes from a singleton shows challenges due to its unique characteristics. For example, the "staircase" shape illustrates a worst-case of the growth operations. The structure of the staircase, consisting of steps with a number of consecutive nodes, creates complexities not present in simple shapes. The symmetrical pattern within staircase shapes requires more flexibility in the growth operations. Furthermore, the growth of the staircase shapes is restricted to specific directions, making it more challenging to achieve desired configurations compared to shapes with more flexible growth orientations. Therefore, the complexities arising from the growing of staircase shapes provide us with deeper insights into our growth operations and indicate their limitations.

Definition 17 (Staircase). *A staircase is a shape S, in which each step consists of at least 3 consecutive nodes.*

Proposition 8. *A staircase of size n requires $\Omega(n)$ time steps to be generated by rigidity-preserving doubling operations.*

Proof. Starting from a shape $|S| = 1$ and performing a rigidity-preserving doubling operation at the first time step t_1, continuing the same operation at the next time step, we observe that at t_3 we have two possible situations,

either building a square of 4 nodes or generating nodes at the two endpoints. The initial attempt of constructing a square will fail because adding any node will result in a 3-line (see Proposition 7).

Consequently, we proceed with the second where we reach a point of generating nodes at the two endpoints of the constructed staircase, as shown in Fig. 16. As a result, growth can only be at most 2 nodes it each time step t, which is linear in total. □

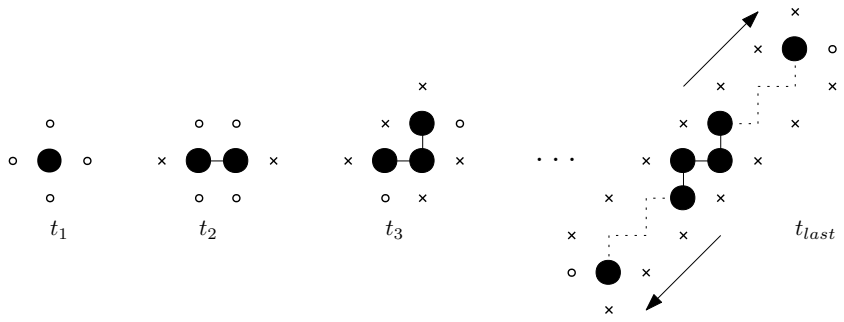


Figure 16: Building a staircase in linear time steps.

Definition 18 (Exact-staircase). *An exact-staircase is a special case of staircase shape, in which each step consists of two nodes.*

Proposition 9. *A rigidity-preserving or rigidity-breaking doubling operation cannot build an exact staircase shape S within a sublinear number of time steps.*

Proof. Assume that the last time step t_{last} of a rigidity-preserving doubling operation will lead to an exact-staircase S of n nodes. We claim that this only occurred if only the two endpoints of the staircase is involved in this operation at t_{last}. To prove the above claim, assume now, for the sake of contradiction, that at least one internal node u was involved in t_{last} of a *north* rigidity-preserving doubling operation, and the final shape constructed was an exact staircase. Then, node v must have been the node generated by u at step t_{last} and v did not exist in t_{last-1}. But this implies that the part of the staircase above u could not have been connected to u in t_{last-1}, as shown in Fig. 17 (a), because if it were connected above u, then the generation of v would have pushed it one row to the north as illustrated in Fig. 17 (b). As

this does not hold, we must assume that the top part of the staircase cannot have been connected to u, and this contradicts the fact that the assumed type of construction cannot break connectivity.

As a consequence, this contradicts the assumption that the internal node u may have involved in t_{last}. Therefore, only the two endpoints of the staircase can generate in t_{last}, as a result, in every step the shape can only grow by at most 2, giving a $t \geq \lfloor (n/2) \rfloor$. □

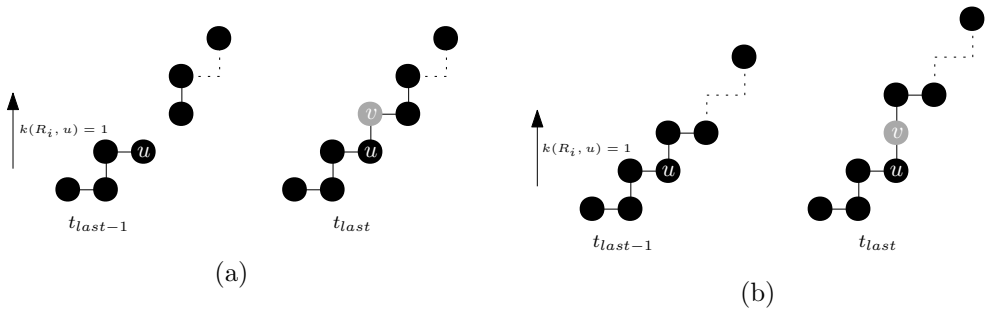


Figure 17: An explanation of Proposition 9.

Observation 4. *Any construction that rigidity-preserving doubling operation performs can also be constructed by rigidity-breaking doubling but not the opposite.*

Next, by putting together the universal linear-time steps constructor of Proposition 6 for doubling and the logarithmic-time steps constructor of Theorem 2 (corresponding to Algorithm 3) for RC doubling, we get the following general and faster constructor for doubling.

Theorem 4. *Given any connected final 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.*

Proof. By Proposition 6, the baseline $B(S_F)$ of S_F can be constructed from a singleton S_I within $O(|B(S_F)|)$ time steps through doubling operations. Then, by Theorem 2, S_F can be constructed from its baseline $B(S_F)$ within $O(\log |S_F| - |B(S_F)|))$ time steps, yielding $O(\log |S_F|)$ time steps in the worst-case, through RC doubling operations. As RC doubling is a special case of

doubling, the latter constructor is also one using doubling operations, thus, the theorem follows. □

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* (which is, in our case, equivalent to getting by full doubling in all directions from S_I to S_F). Shape similarity is to be defined for orthogonal (i.e., rectilinear) shapes in a way analogous to its definition for general polygons: two shapes S_I and S_F being *similar* if S_I can be made equal (up to translations) to S_F through uniform scaling. Note that shape similarity can be decided in linear time [Man76, AT78]. 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 polygon 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 [IA86, Kei00]. These algorithms, given an orthogonal polygon S (i.e., whose edges are horizontal or vertical), 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 final 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.*

Proof. Given S_F we use the $O(n^{3/2} \log n)$-time algorithm of [IA86] to partition S_F into rectangles. As all shapes considered in this paper are orthogonal polygons, S_F is a valid input to the algorithm, thus, the algorithm returns a partition of S_F consisting of the minimum number h of rectangles, $S_1, S_2, \ldots, S_h$.

We define a graph $G' = (V', E')$ associated with those rectangles. In particular, $V' = \{S_1, S_2, \ldots, S_h\}$ and $E' = \{S_i S_j \mid S_i, S_j \in V'$ and a node

$u \in S_i$ is an orthogonal neighbor to a node $v \in S_j\}$, that is, the graph G' has a vertex for each rectangle and has an edge between two vertices iff the corresponding rectangles are vertically or horizontally adjacent. Note that G' is a connected graph. If not, then G' would consist of at least two connected components G'_i. Each G'_i corresponds to a subset of the rectangles, so that each rectangle belongs to a single G'_i. Moreover, the rectangles corresponding to the vertices of G'_i form a partition of a shape W_i which is a subshape of S_F. These imply that $S_F = \bigcup_i W_i$, where the W_is are pairwise disconnected shapes, thus contradicting connectivity of S_F. So, it must hold that G' is connected.

Therefore, we can compute a spanning tree T of G', which we shall then use to define the constructor of S_F. Note that T is a spanning tree of the corresponding rectangles, given their adjacency relation. We set as the root of T a maximum-area rectangle S'_0 of the partition. The sought constructor is based on a BFS traversal of T starting from S'_0. The constructor first constructs S'_0 through a constructor of Theorem 3, in a number of time steps logarithmic in $|S'_0|$. Then, for the set $N(S'_0) = \{S' \mid S'_0S' \in E'\}$ of rectangles adjacent to S'_0, we compute the set of points $c(S', S'_0) = \{(x, y) \mid S' \in N(S'_0)$ and (x, y) are the coordinates of a corner-node of S' which is adjacent to a node of $S'_0\}$. For every $S' \in N(S'_0)$ we set one of the points in $c(S', S'_0)$ as the initial point from which S' will be constructed. Then we construct S' directly in its final position by a constructor of Theorem 3. Note that the selection of these points and the non-overlapping nature of the partition ensure that these growth processes cannot overlap.

Whenever a rectangle S' in the next level of the spanning tree can be constructed from more than one initial points (either adjacent to the same or to distinct rectangles of the previous level), then one of those points is chosen arbitrarily. All rectangles of a given level are constructed in parallel, thus the time steps paid per level of T are the time steps associated with the largest rectangle of that level. The process continues in this way until the furthest level is constructed.

In the worst case, the h rectangles are constructed sequentially, each, rather crudely, in $O(\log |S_F|)$ time steps, for a total of $O(h \log |S_F|)$ time steps. □

Observe that for those shapes S_F for which h is constant or $O(\log |S_F|)$, the constructor returned by Theorem 5 is of logarithmic or polylogarithmic time steps, respectively.

6. Conclusion

In this work, we studied the graph-growth mechanisms proposed by Mertzios *et al.* [MMS+21] under a 2D geometric setting. In particular, we investigated some potential forms of growth operations, from the most special to the most general, namely *full doubling*, *RC doubling*, and *doubling*. We developed a linear-time algorithm that decides the feasibility of growing any pair of shapes S_I, S_F through *RC doubling*, and, for the yes instances, additionally returns an $O(\log n)$-time step constructor of S_F from S_I. Further, in the (general) doubling operation, for any connected final shape S_F, we provide an $[O(|B(S_F)|) + O(\log |S_F|)]$-time step constructor and an $O(h \log |S_F|)$-time step constructor starting from a singleton S_I. On the other hand, we proved that some shapes cannot be constructed in sublinear time steps.

A number of interesting problems are opened by this work. The obvious first target is to obtain an optimal constructor and to develop an algorithm that optimizes the running time. This problem appears to be computationally hard, hence it might be worth trying to establish the hardness of the problem. A variety of grid types, such as triangular (e.g., in [DRD+14]), have been explored in the relevant literature, and it would be interesting to study how our growth operations apply there. Another possible direction is to develop a distributed model of the growth operations described here. On some more specific technical questions, we do not yet know what is the class of shapes that are constructible in (poly)logarithmic time steps starting from a singleton through a (general) *doubling* operation. Moreover, we did not study ShapeConstruction for any pair of shapes S_I, S_F through a (general) *doubling* operation.

Acknowledgements

The authors would like to thank Viktor Zamaraev for useful conversations during the various stages of developing this work.

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, 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, et al. Universal reconfiguration of facet-connected modular robots by pivots: the O(1) musketeers. *Algorithmica*, 83[5]:1316–1351, 2021.

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20[4]:279–304, 2007.

[ACD+08] G. Aloupis, S. Collette, E. D. Demaine, S. Langerman, V. Sacristán, and S. Wuhrer. Reconfiguration of cube-style modular robots using O(logn) parallel moves. In *International symposium on algorithms and computation*, pages 342–353. Springer, 2008.

[AM22] N. Almalki and O. Michail. On geometric shape construction via growth operations. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks*, pages 1–17. Springer, 2022.

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

[AT78] S. G. Akl and G. T. Toussaint. An Improved Algorithm to Check for Polygon Similarity. *Inf. Process. Lett.*, 7[3]:127–128, 1978.

[CDLN14] A. Cornejo, A. Dornhaus, N. Lynch, and R. Nagpal. Task allocation in ant colonies. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2014.

[CMP22] M. Connor, O. Michail, and I. Potapov. Centralised connectivity-preserving transformations for programmable matter: a minimal seed approach. *Theoretical Computer Science*, 2022.

[CSG+19] M. M. Chan, Z. D. Smith, S. Grosswendt, H. Kretzmer, T. M. Norman, B. Adamson, M. Jost, J. J. Quinn, D. Yang, M. G. Jones, et al. Molecular recording of mammalian embryogenesis. *Nature*, 570[7759]:77–82, 2019.

[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*, pages 289–299, 2016.

[DGR+17] Z. Derakhshandeh, R. Gmyr, W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.

[DLFS+20] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33[1]:69–101, 2020.

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

[DP04] A. Dumitrescu and J. Pach. Pushing squares around. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 116–123, 2004.

[DRD+14] Z. Derakhshandeh, A. Richa, S. Dolev, C. Scheideler, R. Gmyr, and T. Strothmann. Brief announcement: Amoebot-a new model for programmable matter. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2014*, pages 220–222. Association for Computing Machinery, 2014.

[DSY04] A. Dumitrescu, I. Suzuki, and M. Yamashita. Formations for fast locomotion of metamorphic robotic systems. *The International Journal of Robotics Research*, 23[6]:583–593, 2004.

[FNSS22] S. P. Fekete, E. Niehs, C. Scheffer, and A. Schmidt. Connected reconfiguration of lattice-based cellular structures by finite-memory robots. *Algorithmica*, 84[10]:2954–2986, 2022.

[IA86] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM Journal on Computing*, 15[2]:478–494, 1986.

[Kei00] J. M. Keil. Polygon decomposition. *Handbook of computational geometry*, 2:491–518, 2000.

[Man76] G. K. Manacher. An application of pattern matching to a problem in geometrical complexity. *Inf. Process. Lett.*, 5[1]:6–7, 1976.

[Mic18] O. Michail. Terminating distributed construction of shapes and patterns in a fair solution of automata. *Distributed Computing*, 31[5]:343–365, 2018.

[MMS+21] G. B. Mertzios, O. Michail, G. Skretas, P. G. Spirakis, and M. Theofilatos. The complexity of growing a graph. *arXiv preprint arXiv:2107.14126*, 2021.

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

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

[MSS20] O. Michail, G. Skretas, and P. G. Spirakis. Distributed computation and reconfiguration in actively dynamic networks. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 448–457, 2020.

[PB21] B. Piranda and J. Bourgeois. Datom: A deformable modular robot for building self-reconfigurable programmable matter. In *International Symposium Distributed Autonomous Robotic Systems*, pages 70–81. Springer, 2021.

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

[RW00] P. W. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468, 2000.

[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*, pages 353–354, 2013.

[WDM+19] D. Woods, D. Doty, C. Myhrvold, J. Hui, F. Zhou, P. Yin, and E. Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567[7748]:366–372, 2019.

[Woo15] D. Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373[2046]:20140214, 2015.