# Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure

Pierre Thalamy
Univ. Bourgogne Franche-Comté
FEMTO-ST Institute, CNRS
Montbéliard, France
pierre.thalamy@femto-st.fr

Benoît Piranda
Univ. Bourgogne Franche-Comté
FEMTO-ST Institute, CNRS
Montbéliard, France
benoit.piranda@femto-st.fr

Julien Bourgeois
Univ. Bourgogne Franche-Comté
FEMTO-ST Institute, CNRS
Montbéliard, France
julien.bourgeois@femto-st.fr

## ABSTRACT

In the context of large distributed modular robots, self-reconfiguration is the process of having modules, seen as autonomous agents, acting together and moving to transform the morphology of their physical arrangement to produce a desired shape. However, due to motion constraints, the number of modules that can move concurrently is greatly limited, thus making self-reconfiguration a very slow process.

In this paper, we propose an approach for accelerating self-reconfiguration to build a porous version of the desired shape, using scaffolding. We expand this idea and propose a method for constructing a parametric scaffolding model that increases the parallelism of the reconfiguration, supports its mechanical stability, and simplifies planning and coordination between agents. Each agent has a set of basic rules using only four states which guarantees that module movements and the construction of the scaffold are deterministic. Coupled with an underneath reserve of modules that allows the introduction of rotating quasi-spherical modules at various ground locations of the growing porous structure, our method is able to build the scaffolding structure in $O(N^{\frac{2}{3}})$ time with $N$ the number of modules composing the structure. Furthermore, we provide simulation results showing that our method uses $O(N^{\frac{4}{3}})$ messages with no congestion.

## KEYWORDS

Self-Reconfiguration; Autonomous Robots; Distributed Algorithm

**ACM Reference Format:**
Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. 2019. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

Programmable Matter (PM) [11] is defined as matter that can autonomously alter its physical properties such as its shape or color, as a response to an internal or an external event. While many technologies on the rise claim to be PM, we believe PM based on Modular Self-reconfigurable Robots (MSR) [9] to be the most promising endeavor, owing to the versatility of the systems [4].

*Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.* 

MSR are robots composed of an arbitrary number of individual modules, which can be seen as autonomous agents, that can physically attach to each other and coordinate through communication to achieve a common goal. Though various types of modular robot architecture exist [2], we are interested in lattice-based modular robots, where connected modules are organized in a regular lattice structure, on which modules navigate using their neighbors.

Self-reconfiguration is a notoriously intricate problem [12, 13] which can be stated as finding a series of individual motions (preferably performed in parallel) that can transform an initial arrangement of modular robotic modules (also named *configuration* or *shape*) into a goal one. It is a fundamental algorithm for large modular robots and PM that encompasses a number of non-trivial subproblems, such as defining the goal shape [3, 21, 25, 26], computing a feasible construction plan [7, 27], or coordinating the motion of modules along multiple paths in parallel while avoiding collisions [6, 19]. Solutions have been proposed from various research perspectives such as control theory, computational geometry, multi-agent systems, and biomimetics [1].

Self-reconfiguration can be viewed both as a destruction [7, 10] and a construction process; while we mainly tackle the latter problem in this article, the algorithm proposed in this article could easily be reversed to perform a clean destruction.

Self-reconfiguration is a notably slow process that is very demanding in term of resources. Therefore, in this article, we would like to propose a method for building 3D objects that is both fast and efficient in term of communication and computation. Furthermore, we would like to offer more guarantees on the respect of mechanical constraints throughout the construction.

In order to attain these objectives, we start by redefining the notion of a shape as its *Boundary Representation*, an external surface with a mechanically sound internal organization. This is then transposed into the context of self-reconfiguration as the construction of a scaffolding structure representing the object that satisfies mechanical constraints and provides a structure for supporting a massive number of module movements. We, hence, introduce an algorithm for deterministically constructing this structure based on simple local rules and only 4 agent states.

Such a deterministic approach naturally poses fault-tolerance issues, which will be the topic of future works.

In this article, we obtain the following results. By using scaffolding and coating instead of a filled object, a pyramid requires $\frac{b^3}{3}$ times less modules with $b$ the tile branches length. Our method is able to build the scaffolding structure of a coated pyramid in

$O(N^{\frac{2}{3}})$ time with $N$ the number of modules composing the structure whereas the filled pyramid requires $O(N^{\frac{4}{3}})$ time, that is $N^{\frac{2}{3}}$ times more. Furthermore, we demonstrate that our method uses $O(N^{\frac{4}{3}})$ messages with no congestion as the modules having the highest throughput manage 4 messages/time step and we provide simulation results that confirm this complexity for several scenarios.

## 2 CONTEXT

We mainly perceive objects by their optical and mechanical characteristics. From an optical standpoint, an object can be represented by its external surface, which reflects the light that is provided by an external source. In order to visually materialize an object, it is sufficient to coat the boundary between the interior and the exterior of the object with light-scattering matter (in image synthesis, this is called the Boundary Representation *(BRep)* [8]) model of an object). (See Figure 3.c)

From a mechanical standpoint, however, an object must, thanks to its internal structure, be able to withstand external forces such as gravity or contact forces. An internal structure is therefore practical for the mechanical coherence of an object (Figure 3.b).

In the context of the self-reconfiguration of an arbitrary shape $A$ into a goal shape $B$, every module has to navigate a path from its initial position in configuration $A$ to its final position in configuration $B$. If we consider a filled object then all the possible paths are placed on the external surface of the object, which lowers the number of possible simultaneous movements.

Indeed, moving through the volume of an object multiplies the number of potential paths that can be followed by modules in parallel. This can be guided by a skeletal structure forming a scaffold on the interior of the object, where all these paths can be followed simultaneously by moving modules.

In order to preserve the external aspect of the object, it is then necessary to cover, or coat, the scaffolding structure, which can be done by navigating the many paths provided by the scaffold in parallel.

In this paper, we focus on a solution for the construction of this scaffolding structure.

To do so, we have made the following assumptions:

(1) The object we aim to build resides in a regular Face-Centered Cubic (FCC) lattice which ensures a high density of modules and therefore a compelling visual representation of the shape as it will be more difficult for light to traverse its boundary.
(2) Underneath the object lies a sandbox-like environment, acting as a reserve of modules, and from which modules can be moved at various ground locations of a regular square grid (Figure 3.a).
(3) This sandbox also brings the energy and the initial communication that provides the description of the goal shape.

## 3 RELATED WORKS

One particularly tricky aspect of self-reconfiguration stated earlier is the coordination of motions between concurrently moving modules so that they avoid blocking the motion of each other or attempting to simultaneously move into the same space. Some authors such as Naz et al. [19] proposed to leave a gap between moving modules through communication-based coordination in order to limit the risk of collisions. While this is powerful and practical for the rotating motion of 2D space modules, it cannot be applied efficiently to 3D space as is.

Furthermore, most existing solutions to the self-reconfiguration problem in large modular robots consider simple module geometries (such as different flavors of cubes [5, 15, 17, 28, 31]), and actuators capable of performing both translation and rotation motions. Yet, the self-reconfiguration of modules with more complex geometries [29, 30] has proven itself much harder, especially when only the latter form of motion is possible [18], as modules are more likely to prevent the motion of another due to blocking constraints, especially when they have rigid non-deformable bodies.

The most relevant reconfiguration work to our module geometry is the one of Yim et al. [29], where the authors proposed a probabilistic self-reconfiguration algorithm for reconfiguring rhombic dodecahedron modules residing in a FCC lattice, and that can only move using rotation. They proposed a method named *Goal-Ordering*, in which modules use one or two metrics to decide which target location in the goal configuration they should go fill. Their method however, suffers from overcrowding around the open goal positions, and is likely to get stuck in local minima and avoid converging altogether, especially in solid and hollow shapes.

In order to ease the motion constraints of self-reconfiguration and increase motion parallelism, Kotay and Rus [16] proposed to engineer the interior of the target shape by discretizing it into repeating hollow multi-module sub-structures named *tiles*, that would leave large tunnels for modules to navigate the structure in parallel, at the cost of a large increase in the granularity of the shape. This approach is referred to as *Scaffolding*. In this work, we are also interested in scaffolding as a way to increase the parallelism potential of the reconfiguration, as well as to ease coordination between moving modules and avoid blocking issues due to overcrowding.

More recent works have also considered a similar approach to lowering the complexity of reconfiguration for cubic modules capable of both rotation and translation. Støy approximated the target shape with a porous representation made by removing individual modules from its volume in a manner that would guarantee an absence of local minima, and hollow or solid sub-configurations. He, then, proposed to use local rules and cellular automata to perform the reconfiguration in [24], or through a gradient descent method in [25].

Furthermore, an aspect of self-reconfiguration that should be further considered is the mechanical constraints imposed on the system. Two main types of mechanical failures are identified in [14]: (1) *loss of stability* due to a shift in the center of mass of the system; (2) *structural failure*, caused by the breaking of a bond between modules after an excessive stress. While Hołobut and Lengiewicz also present a distributed procedure for predicting if the next reconfiguration step will cause a structural failure in their paper, to the best of our knowledge, no self-reconfiguration algorithm truly considers mechanical constraints in their design—when these same constraints might prevent them from working in practice. With our novel approach to scaffolding, we aim to make progress towards achieving mechanically sound reconfigurations by structural design, hence without having to resort to such software methods that involve a costly performance overhead.

## 4 MODULAR ROBOTIC MODEL

In this work, we consider a modular robot made from an arbitrary number of quasi-spherical modules named *3D Catoms*, that move by rotating on the surface of neighbors, and connect to up to 12 neighbor modules, one on each of their connectors (see Figure 1). We strongly advise readers to browse through [23] for a better sense of the geometry and motion mechanisms of *3D Catoms*. Our project is a follow-up of the *Claytronics* project [11] and *Catom* stands for *Claytronics atoms*. *3D Catoms* reside in a 3-dimensional grid described as a *Face-Centered-Cubic Lattice (FCC)*, with coordinates in $\mathbb{Z}^3$. *3D Catoms* are symmetrical and therefore their orientation does not matter. Positions on the grid are referred throughout the paper as lattice *cells*, or simply as *positions*.

A *3D Catom* $M$ is able to move from one cell to a free neighbor position by rotating on the surface of another *3D Catom* $P$ acting as a *pivot*. This corresponds to a change of latching connectors on the surface of $P$; a rotation can therefore be described as a couple $\langle C_i, C_j \rangle_P$, where $C_i$ and $C_j$ refer to connectors with identifier $i, j \in [0, 11]$. These connectors are also used by *3D Catoms* to communicate with their immediate neighbors, the only mode of communication available to them. Furthermore, a rotation $\langle C_i, C_j \rangle_P$ can be performed through one of two paths on the surface of $P$, either by rotation on an hexagonal face, or through an octagonal face ($R_h$ and $R_o$ on Figure 1, respectively). We provide a YouTube video illustrating this mode of motion (see footnote[1]).

A *3D Catom* is said to be *mobile* if it can reach at least one of its neighbor cells according to its motion constraints. The set of all motion constraints imposed on a module $M$ that seeks to move to a neighbor position $N$ is identical to the one introduced by Yim et al. in *Proteo* [29].

Due to the geometry of the modules and the resulting blocking constraints, it is not possible to bridge the gap between two lines of modules growing towards each other, therefore the construction of a shape that has all its elements connected on each layer needs to be grown from a single initial point, and with a carefully designed set of construction rules, likely resulting in a diagonal growth of the volume of the object.

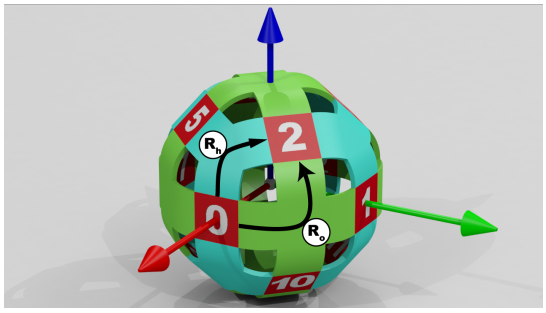


**Figure 1: Two possible paths that can be used to perform motion $\langle C_0, C_2 \rangle_P$ on the surface of a module $P$.**

[1]Youtube video illustrating *3D Catom* motions https://youtu.be/IZh-5p1dbKk

## 5 SCAFFOLDING

Our scaffold model is defined as an arrangement of canonical components sharing a common structure named *tiles*.

### 5.1 Anatomy of a Scaffold Tile

A tile is composed of a core made of a *root* module surrounded by 4 *support* modules (placed to help others to vertically traverse the structure), and a number of branches connecting it to other tiles. The length of these branches $b$ is a parameter of our model, it is determined by the mechanical capabilities of the structure, in relation to the connector strength of the *3D Catom* hardware. However, $b$ has to be greater than 4 as the space would be too tight for module motion otherwise.
We can express the number of modules that compose a tile using the relation $N_{\text{tile modules}} = 6(b-1) + 5$. In the remaining figures, we will use $b = 6$, which yields $N_{\text{tile modules}} = 35$. This number can be compared to the fully filled bounding box of the tile that would use $b^3$ modules (216 with $b = 6$).

Each tile is an arrangement of modules that follows the axes of the FCC grid. As shown in Figure 2.b, the center node $R$ (also referred to as *Tile Root*, drawn in white) is connected to:

- a branch made of modules $\{X_i | i \in [1..b-1]\}$ (in orange) following the $\overrightarrow{x}$ axis;
- a branch $\{Y_i | i \in [1..b-1]\}$ (drawn in green in Figure 2.c) following the $\overrightarrow{y}$ axis;
- four branches $\{Z_i | i \in [1..b-1]\}$, $\{RightZ_i | i \in [1..b-1]\}$, $\{RevZ_i | i \in [1..b-1]\}$, $\{LeftZ_i | i \in [1..b-1]\}$ (in blue in Figure 2.e) following axes $\overrightarrow{z}$, $(1,-1,1)$, $(-1,-1,1)$ and $(-1,1,1)$, respectively.

Finally, the tile is also composed of four support modules named $SZ$, $SRightZ$, $SRevZ$ and $SLeftZ$ at respective positions $(1,1,0)$, $(1,-1,0)$, $(-1,-1,0)$ and $(-1,1,0)$ relative to $R$ (in yellow on Figure 2.b).

We also define eight special empty positions below the tile, placed near the tip of incoming vertical branches named *Entry Point Locations (EPL)*, and shown as transparent cells on Figure 2.a. EPL are the positions that allow modules to enter a growing tile from the lower levels. When a module enters one of these positions (as a *FreeAgent*), it stops and either requests a goal destination into this tile from the *Coordinator* of this tile (i.e., *Tile Root*), or in some cases waits for some condition to clear before moving to its destination. There are two types of entry points, appearing in purple and blue on Figure 2.a: the blue EPL are temporary and will become unreachable as branches $X$ and $Y$ start growing, they are used for introducing future Support modules into the tile; purple EPL are the main entry points that are used for the rest of the life cycle of the tile, but can only be used once support modules are in place.

Our scaffolding is then made of regularly placed instances of partial or complete tiles that can be connected to up to 6 neighbor tiles through modules $X_{b-1}$, $Y_{b-1}$, $Z_{b-1}$, $RightZ_{b-1}$, $RevZ_{b-1}$ and $LeftZ_{b-1}$ as shown in Figure 2.f. Partial tiles are tiles whose number of branches is lower than 6, as found on the borders of the object.

### 5.2 Hierarchical Organization

We define an *h-pyramid* as a pyramid with a height of $h$ tiles (that is to say $b(h-1)+1$ modules from the base to the tip). For instance,

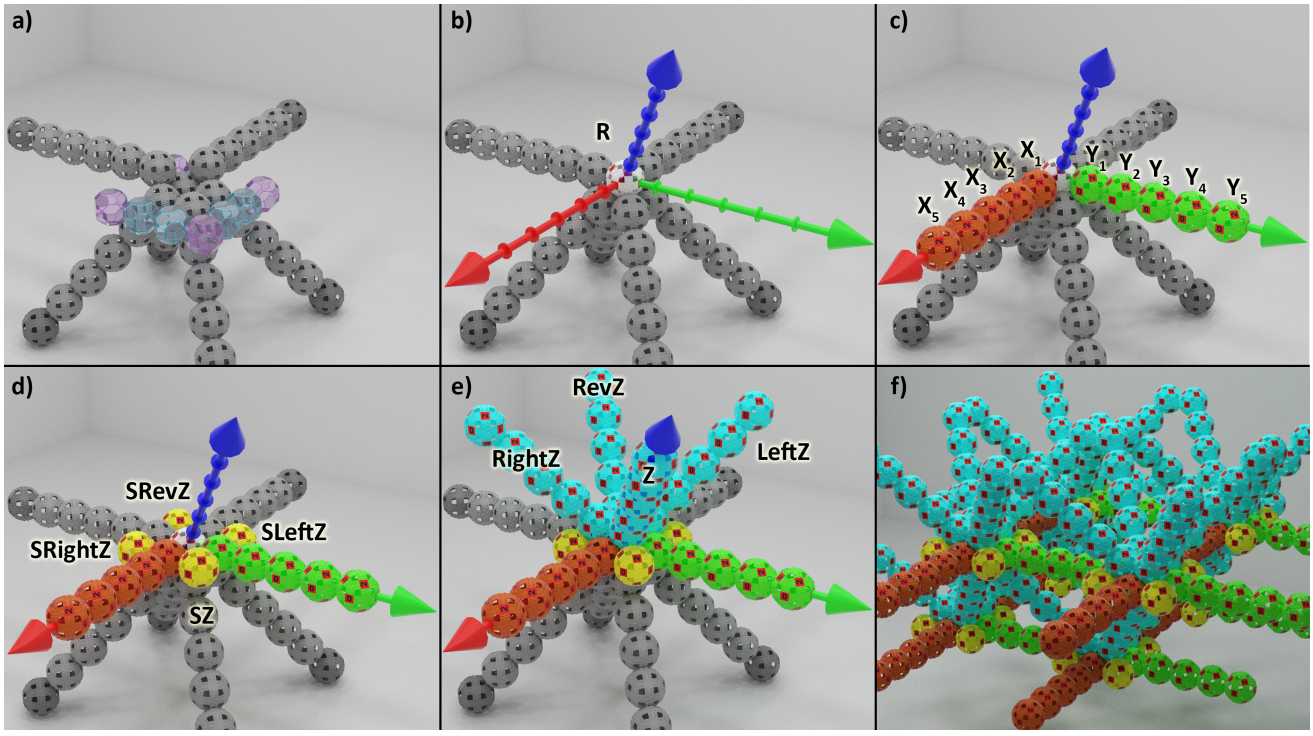


**Figure 2: 3D structure of a tile with $b = 6$. a) Existing structure made from previous tile. Transparent cells represent entry points into the tile; b) White module represents the root $R$ of the new tile, origin of the local coordinates system; c,d,e) 3D position of branch and support modules of the tile; f) Assembly of multiple tiles to construct a scaffold.**

in the example of the 4-pyramid shown on Figure 2, the scaffold results from the assembly of 30 scaffold tiles.

Let us consider a $h$-pyramid shape composed of $N_{tiles}$ scaffold tiles assembled together in the FCC lattice. This shape is made of 5 modules at the tip of the pyramid (constituting the top tile at height $h$), 4 tiles under these modules at height $h-1$, and again 4 overlapping tiles under each of these tiles and so on until reaching the base level at height 1).

This $h$-pyramid is composed of tiles whose origins $R_{i,j,k}$ are placed at:

$$R_{i,j,k}(b \times i, b \times j, b \times k) \text{ with } \begin{cases} 0 \leq i \leq h-k \\ 0 \leq j \leq h-k \\ 0 \leq k \leq h \end{cases}$$

In our example shown on Figure 3, for a 4-pyramid with $b = 6$, the number of modules comprising the scaffold is 630 and the coating uses 760 additional modules.

$$N_{tiles} = \sum_{i=1}^{h} i^2 = \frac{h^3}{3} + \frac{h^2}{2} + \frac{h}{6} \quad (1)$$

Then, we can express the number of modules used to construct the scaffold of the *h-pyramid* (including support modules drawn in yellow in Figure 3.b). First, consider $N_i$ the number of modules from tiles at level $i$ of the pyramid, as they appear on Figure 3.b. We sum $i$ segments along the hich axis composed of one white module plus $(i-1)$ groups of 1 white and $b-1$ red modules; $(i-1)$ segments along the $\overrightarrow{y}$ axis made of $(b-1$ green modules); $4(i-1)^2$ ascending branches of $(b-1)$ blue modules; and $4i^2$ support modules in yellow.

$$N_i = i\left((i-1)b + 1 + (i-1)(b-1)\right) + 4(b-1)(i-1)^2 + 4i^2 \quad (2)$$

$$N_{modules} = \sum_{i=1}^{h} N_i = (2b - \frac{1}{3})h^3 + (\frac{9}{2} - 2b)h^2 + \frac{5}{6}h \quad (3)$$

And the number of modules comprising the coating of the pyramid ($h \geq 2$) as:

$$N_{coating} = 4 \sum_{i=1}^{(h-1)b+1} i = 2\left(b^2h^2 + (3b - 2b^2)h - 3b + 2\right) \quad (4)$$

Note that from $h = 6$ and on, more modules are required for building the structure than for building the coating of the pyramid.

Since that due to the geometrical constraint of *3D Catoms*, we need to enforce a strict construction order of the tiles of the pyramid (bottom to top, left to right, front to back), we can construct a graph connecting these tiles, and a spanning tree within that graph expressing the precedence of the tiles in term of their construction order.

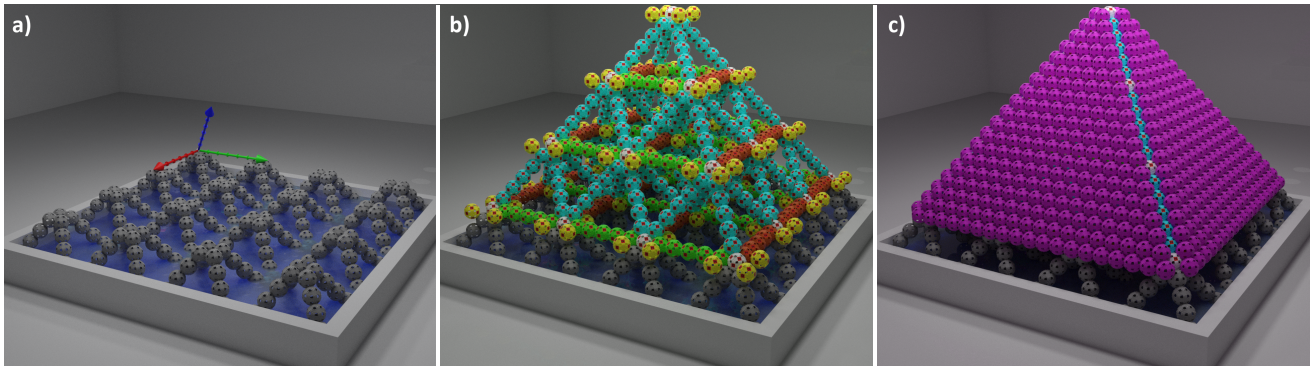


**Figure 3: Construction of a 3D model using scaffolding ($b = 6$). a) Support structure; b) Scaffold of the 4-pyramid; c) Coated 4-pyramid, after removal of support modules.**

The scaffold formation problem fits into our idea of a larger and more comprehensive self-reconfiguration scheme, introduced in the next section.

## 5.3 Self-Reconfiguration Scheme

In the context of the self-reconfiguration into an arbitrary goal shape $G$ from an empty sandbox, our approach consists in a sequence of several phases:

(1) **Scaffold construction:** If the desired shape is not convex by itself or when connected to the sandbox, we build a scaffold encompassing a tile approximation of the convex hull of the union of the object and the base of the sandbox (e.g, in Figure 4.a, the native white scaffolding of the sphere is complemented by a red scaffolding, filling the gap between the object and the base), and fill holes in the shapes. Under a carefully designed construction plan, this supports the mechanical stability of intermediate configurations.
(2) **Removing excess modules:** Non-essential modules are removed from the shape, and stored onto the scaffold, ready to be used as coating later on.
(3) **Coating:** Construction of the surface of the shape, using excess modules and additional modules called in from the sandbox (see Figure 4.b).

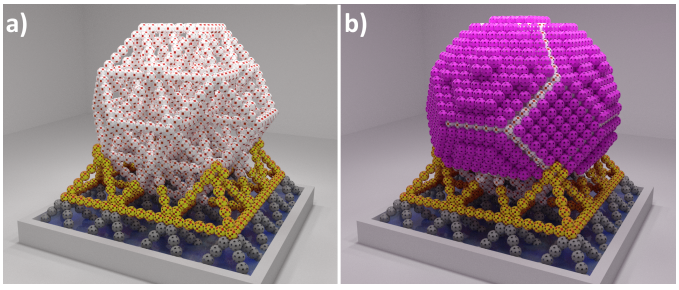


**Figure 4: Complete self-reconfiguration scheme example for the construction of a more complex shape, a sphere. a) Extended scaffold; b) Coating of the white part of the scaffold.**

## 5.4 Construction Agent Roles

Each tile is composed of $\{6i + 5 | i \in [0..b-1]\}$ modules (see Figure 2) that must be inserted in a specific order so as to avoid deadlocks. When docked as a *tile component*, some of these modules endorse active roles, while others are simply passive structural components.

We consider that during its life, a module can be in four different states. For each of these state it runs a corresponding agent code:

(1) *Idle* modules are sandbox modules which are waiting to be called in to partake in the reconfiguration by modules from the growing structure.
(2) *Free Agent* modules are *Idle* modules that have been called in and entered the reconfiguration scene, waiting to be assigned a goal position or in motion to their assigned position.
(3) *Coordinators* are modules docked in the root position of a tile, and which are responsible for scheduling the construction of their tile. More specifically, the role of the *Coordinator* is to ensure that modules arrive at specific branches in an order compatible with the construction order of the components of the tile, and inform incoming modules on where they are needed.
(4) *Relay* modules are docked robots whose only role is to forward messages between *FreeAgents* transiting through its tile and the local *Coordinator*. This role can be endorsed by supports or vertical branch tip modules.

## 5.5 Tile Construction Process

The tile construction process is performed by modules arriving at the various EPL of the vertical incident branches of parent tiles (i.e., *FreeAgent* modules), and coordinated by the root module of the tile (i.e., *Coordinator* module).

The construction is started by the arrival of the coordinator into the tile root position, which can only happen once the branches incident to its tile are complete. While waiting for this condition to clear, the future coordinator awaits on one of the entry points of the *RevZ* branch. If there are no incoming branches whose completion to wait for, as on some corner cases, it can directly proceed to its position; otherwise, it keeps waiting for the tip module of the last incoming branch to notify it that the tile is ready to start growing.

**Algorithm 1:** Distributed control algorithm for the *FreeAgent* module role.

```
Function reachedNewTileEntryPoint():
    coordinatorPos = getNearestTileRoot(getPosition());
    relayModule = findSupportOrBranchTipNeighbor();
    sendMessage(relayModule, REQUEST_GOAL_POSITION);

Function planNextRotation():
    nbh = getNeighborhood();
    nextPosition = matchLocalRules(nbh, goalPosition, step);
    rotateTo(nextPosition);

Event Handler ROTATION_END:
    if getPosition() == goalPosition then
        if isScaffoldComponent(getPosition()) then
            agentRole = agentRoleForComponent(getPosition());
        else
            reachedNewTileEntryPoint();
            return;
    else
        step++;
        planNextRotation();

Message Handler PROVIDE_GOAL_POSITION(rcvdPosition):
    step = 0;
    goalPosition = rcvdPosition;
    planNextRotation();
```

When the coordinator gets into position, it immediately sends down all the vertical branches below it a message expressing the requirements of the construction of its tile: an 8-bit word indicating which branches it has to build, and whether or not it will need to provide a tile root through its *RevZ* branch. This message is routed all the way down to the four ground coordinators located under the current tile, where ground coordinators can then summon *Idle* modules from the sandbox according to the requirements and send them up towards the growing root. It is assumed that the four ground coordinators are able to share a common notion of time, that allows them to temporally coordinate their feeding of modules to the system.

When a previously *Idle* module is called in for construction, it endorses the *FreeAgent* (FA) role, behaving as specified by Algorithm 1, getting routed from tile to tile by local coordinators and locally navigating each tile from an entry point to a destination using a set of local rules common to all modules. Every time a FA module enters a new tile, it updates its coordinate system to use positions relative to the local coordinator, set as its origin. If the module is just transiting through the tile, the position returned by the coordinator will be an *EPL*, otherwise it will be one of the $6b + 4$ non-root tile components. The local rules can be seen as a dictionary whose key is the tuple $\langle N, P_{Goal}, Step \rangle$, and $P_{disp}$ is the value. $N$ is a 12-bit representation of the local neighborhood of the module; $P_{Goal}$ is the goal position of the module in the tile (relative to the position of the coordinator); $Step$ is there to avoid rule-matching collision between rules and denotes how many motions the module has already performed in the tile; $P_{disp}$ is the displacement corresponding to the rotation that the module has to perform.

Once a FA module has reached its position as a component of the tile, it updates its state based on its new position, and when relevant, notifies waiting modules that they can resume their motion.

### 5.6 Messaging

There are four kinds of messages being exchanged in a distributed manner during the reconfiguration process detailed in the last section:

(1) **REQUEST_GOAL_POSITION:** Sent through *Relay* modules to the local *Coordinator* by a *FreeAgent* module arriving at a tile entry point location, to request a destination within this tile.
(2) **PROVIDE_GOAL_POSITION:** Response to a goal request by a *Coordinator* to a *FreeAgent*. Follows the same path as the request. Contains a grid position to be used as goal by the receiver.
(3) **TILE_INSERTION_READY:** Sent by the last arrived horizontal branch tip module to the *FreeAgent* and future tile root waiting at one of *RevZ* EPL of the neighbor tile.
(4) **INITIATE_FEEDING:** Sent by a freshly arrived *Coordinator* down all of its incident vertical branches to express its resource requirements to 4 lower-level *Coordinators* connected to the sandbox—i.e., how many modules it needs for building its tile.

## 6 ANALYSIS

### 6.1 Algorithmic Complexity

In this section we study the time and message complexities of the reconfiguration method in the case of the pyramid. Results can then be generalized to more complex shapes in subsequent works.

Let's consider throughout this section the construction tree of the pyramid, whose vertices are the tiles of the pyramid, and whose edges denote the precedence in the construction order of these tiles. The root of the tree is the tile at position $(0, 0, 0)$. An edge between a father and a child vertex means that the start of the construction of the child is triggered by the completion of the father. We determine for each of the edges, the time elapsed between the construction of the father and the one of the child. For instance, within a tile, the $X$ branch is the first branch to be built, and its completion will allow the child tile at position $(b, 0, 0)$ relative to tile $X$ to get its construction process under way.

We assume that the duration of the construction of a tile only depends on the number of modules forming it, it is therefore built in constant time. By studying displacement rules, we can deduce that the sum of the waiting time and the motion time necessary for a $R$ module to reach its position depends on its height $i$ in the construction tree, which can be expressed as:

$$T_{tile}(i) = [24 + 6b + 2b(i-1)] \times ts = [24 + 4b + 2b \times i] \times ts \quad (5)$$

where $ts$ is the duration of an unitary *3D Catom* motion, constituting a time step.

Theorem 1. *In the case of the h-pyramid, the height of construction tree is* $3(h-1)$.

Proof. Given that the construction of the tile at $(0,0,i+b)$ requires the construction of 4 lower tiles: $(0,0,i)$, $(b,0,i)$, $(0,b,i)$ and $(b,b,i)$, the height of the higher vertex in the tree is equal to the height of the lower vertex + 3, therefore $height(0,0,h) = 3(h-1)$.

Considering that the depth of two vertices placed in the same plane is $2(h-1)$ in the construction tree, which is indeed lower than $3(h-1)$, we can deduce that the previous path $(0,0,i)$ to $(0,0,i+b)$ is a critical path. □

Theorem 2. *The reconfiguration time of the reconfiguration of the h-pyramid is* $O(N^{\frac{2}{3}})$.

Proof. Using Equation 5, we can express the time required to construct the $h^{th}$ level of the $h-pyramid$ in number of motion times as:

$$T = \sum_{i=1}^{h} 24 + 4b + 2b \times i = 24h + b(5h + h^2) \tag{6}$$

We conclude that the reconfiguration time is $O(h^2)$ time steps. Using Equation 3, and considering that the parameter $b$ is a positive constant, we can assume that there exist two positive real numbers $\{p,q\} \in \mathbb{R}^2$ verifying: $p \times h^3 < N < q \times h^3$.

Then, we deduce bounds for $h$:

$$\left(\frac{N}{q}\right)^{\frac{1}{3}} < h < \left(\frac{N}{p}\right)^{\frac{1}{3}}$$

Combining with previous Equation 6, we deduce bounds for the motion time $T$:

$$6\left(\frac{N}{q}\right)^{\frac{2}{3}} + 54\left(\frac{N}{p}\right)^{\frac{1}{3}} < T < 6\left(\frac{N}{p}\right)^{\frac{2}{3}} + 54\left(\frac{N}{p}\right)^{\frac{1}{3}}$$

We conclude that the reconfiguration time is $O(N^{\frac{2}{3}})$. □

Theorem 3. *The complexity of the number of messages* $N_{messages}$ *sent to schedule the construction of a N modules pyramid is* $O(N_{modules}^{\frac{4}{3}})$.

Proof. Each module sends 4 kinds of messages during a reconfiguration: $N_{messages} = N_{RGP} + N_{PGP} + N_{TIR} + N_{IF}$. Where *RGP*, *PGP*, *TIR* and *IF* messages denote the messages detailed in Section 5.6. At worst case $N_{TIR} = c \times \sum_{i=1}^{h} i^2 = N_{tiles}$, $c$ is a small constant. The number $m$ of *IF* messages sent by a module depends on the level $i$ of its docking tile: $m = 4b \times (h-1)$. Then,

$$N_{IF} = \sum_{i=1}^{h} i^2\,(4b \times (h-i)) = \frac{4b}{12}(h-1)h^2(h+1)$$

Similarly, messages *RGP* and *PGP* are sent $k=3$ or $k=4$ times every time a FA module enters a tile, except if it will become root, therefore using Equation 2,

$$N_{RGP} = N_{PGP} = k \times \sum_{i=1}^{h} \left((N_i - i^2) \times i\right)$$

As $N_{IF}$, $N_{RGP}$, and $N_{PGP}$ are $O(h^4)$, we can deduce as in the previous proof that $N_{messages}$ is $O(N_{modules}^{\frac{4}{3}})$. □

## 6.2 Comparison with a Filled Pyramid

In this section, we compare our scaffolding approach that builds porous objects with the construction of filled objects.

For the *h-pyramid*, scaffolding uses $N_{modules}$ (cf. Equation 3) modules whereas a filled shape uses:

$$N_{modules}^{compact} = \sum_{i=1}^{b(h-1)+1} i^2 = \frac{2b^3(h-1)^3 + 9b^2(h-1)^2 + 13b(h-1) + 6}{6}$$

We observe that the filled pyramid requires $\frac{b^2}{6}$ times more modules than the one using scaffolding.

In order to evaluate the number of motions for building a filled configuration, we notice that for each layer $j$ we have a number of motions of:

$$N_j^{layer} = j^2 + \sum_{i=1}^{j-1} 2i \times j$$

As we have $n = b(h-1)+1$ layers, and each *3D Catom* has to climb $j+1$ layers, then we have a total number of movements for building the filled *h-pyramid* of:

$$N_{motions}^{filled} = \sum_{j=1}^{b(h-1)+1} \left(h - j + j^2 + \sum_{i=1}^{j-1} 2i \times j\right) = O(h^4)$$

Using an algorithm similar to Tucci et al. [27] to cover each plane of the pyramid, we can build two distinct parts of the structure in parallel, hence:

$$T^{filled} = O(N^{\frac{4}{3}})$$

However, this does not take into account the construction of the surface of the pyramid. Nevertheless, as the algorithm does not use the vertical border branches of level $i$ to construct the tiles of level $i+1$, we can use these branches $[1..i]$ to transport modules from the sandbox to fill the borders during the construction of upper tiles. Then, only the last tile at the top of the pyramid will remain to be covered, which can be done at the end of the construction of the scaffold.

To summarize, the reconfiguration time of the construction of the *h-pyramid* with scaffolding is $O(N^{\frac{2}{3}})$ and the visual aspect is similar to the one of the fully filled pyramid whose construction takes $O(N^{\frac{4}{3}})$ time.

# 7 EXPERIMENTS

The experiments were made on the VisibleSim [22] simulator. They consist in building pyramids of various base sizes ($4\times4$, $5\times5$ ... $9\times9$) from an equal-size sandbox.

The video available on YouTube[2] shows recordings of these simulations. One notable aspect of these recordings is that the growth of the scaffold progresses in a roughly diagonal manner. This is due to the enforced construction order that makes sure that there are never two branches growing facing each other, as our module geometry would not be able to bridge the gap between them. Furthermore, it is clear from this video that our method is able to leverage the potential for parallelism of such a modular robotic system, with a large number of modules moving concurrently to

[2] YouTube video of simulation at https://youtu.be/1pvNBQlcVGE

reach their goal positions, approximately following a bell curve with surges of parallelism, as can be seen on Figure 5.

In this set of experiments, we will study the following properties of our method: motion parallelism, number of messages exchanged between any two modules, temporal distribution of these messages. Other properties of the method can be theoretically derived, which was done in the previous section.

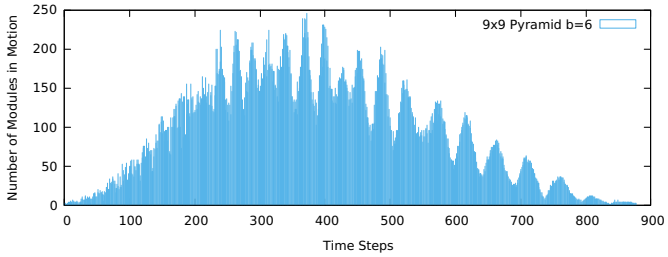


**Figure 5: Motion parallelism over time during the reconfiguration into a 9-pyramid (7905 modules).**

In the following figures, time is represented as time steps relative to the rotation time of a module. One time step corresponds to the time it takes for a single module to move from one position of the grid to a neighbor one. Also, we assume that communication time is negligible relative to the motion time of a module.

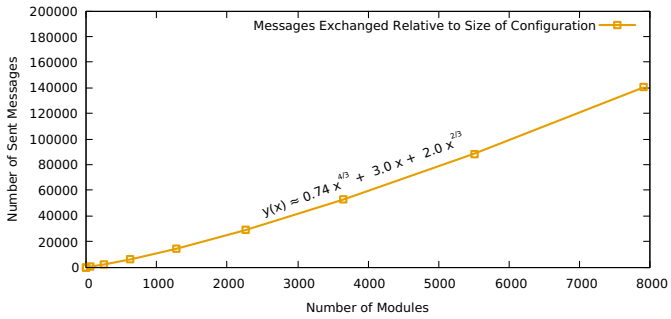


**Figure 6: Number of messages exchanged during scaffold construction for various sizes of goal configurations.**

Figure 6 displays the total number of messages exchanged between any two neighbor modules during reconfiguration, as a function of the number of modules required to build various sizes of pyramids. We verify that the curve given by simulations for 1x1 to 9x9 base tiles is compliant with the expression of the complexity of Theorem 3.

Furthermore, Figure 7 provides information on the temporal distribution of these messages, by showing the maximum throughput observed by each module during a reconfiguration. It shows that a maximum of four messages are sent within a single time step (this corresponds to INITIATE_FEEDING messages by newly arrived *Coordinators*, being sent down all incident vertical branches at once). On average, the maximum message rate per module is 1.325, which together show that a congestion of the network, whose avoidance is a critical aspect of such large distributed systems [20], cannot occur.

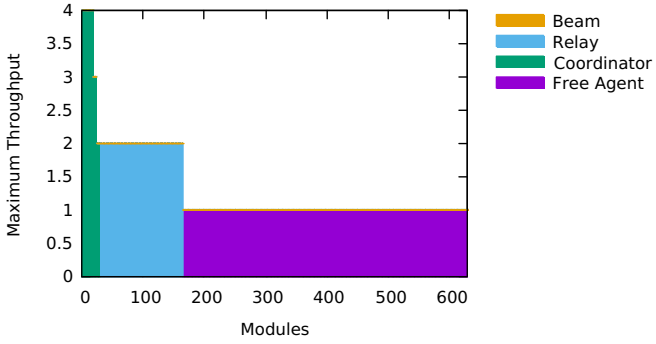


**Figure 7: Maximum number of messages sent per module in a single time step for every module during a reconfiguration into a 4-pyramid**

## 8 CONCLUSION AND FUTURE WORKS

In this paper, we introduced of novel approach to the distributed scaffold-based self-reconfiguration of large modular robotic ensembles, using a parameterizable scaffold model, local rules, and simple coordination. We defined the purpose and geometry of our scaffold, proposed a distributed and deterministic method for constructing it from a sandbox of modules, and explained how its construction fits into a larger self-reconfiguration scheme that involves the coating of the structure. We provided an analysis of this method with the example of the construction of a pyramid, as well as a set of experiments, which showed that our approach can perform self-reconfiguration into a porous version of an object in $O(N^{\frac{2}{3}})$ time, leveraging the potential for parallel motion of our modules, and using $O(N^{\frac{4}{3}})$ messages, with no possibility of network congestion. We envision as future works to replace the resource requests sent by arriving tile roots, by a continuous feeding of *3D Catoms* up every branch of the scaffold, which could be interrupted by tile roots when they stop requiring resources. We believe this would allow us to reach a $O(N^{\frac{1}{3}})$ reconfiguration time. Furthermore, we are interested in generalizing this method to any shape, which requires additional coordination at the tile level to allow modules to traverse the tile horizontally without colliding. The scaffold coating algorithm for rendering the surface of objects also needs to be designed. Finally, an interesting topic is the complete removal of tile root modules, which would direct us towards fully-decentralized and probabilistic construction methods, and possibly allow a faster growth order to emerge.

## ACKNOWLEDGMENTS

This work was partially supported by the ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, ISITE-BFC project (ANR-15-IDEX-03), Labex ACTION program (ANR-11-LABX-01-01), and the Mobilitech project.

# REFERENCES


[1] Hossein Ahmadzadeh and Ellips Masehian. 2015. Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization. *Artificial Intelligence* 223 (June 2015), 27–64. https://doi.org/10.1016/j.artint.2015.02.004

[2] Hossein Ahmadzadeh, Ellips Masehian, and Masoud Asadpour. 2016. Modular Robotic Systems: Characteristics and Applications. *Journal of Intelligent & Robotic Systems* 81, 3-4 (March 2016), 317–357. https://doi.org/10.1007/s10846-015-0237-8

[3] Dongyang Bie, Yanhe Zhu, Xiaolu Wang, Yu Zhang, and Jie Zhao. 2016. L-systems driven self-reconfiguration of modular robots. *International Journal of Advanced Robotic Systems* 13, 5 (Sept. 2016), 172988141666934. https://doi.org/10.1177/1729881416669349

[4] Julien Bourgeois, Benoit Piranda, Andre Naz, Nicolas Boillot, Hakim Mabed, Dominique Dhoutaut, Thadeu Tucci, and Hicham Lakhlef. 2016. Programmable matter as a cyber-physical conjugation. In *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 002942–002947. https://doi.org/10.1109/SMC.2016.7844687

[5] Zack Butler and Daniela Rus. 2003. Distributed Planning and Control for Modular Robots with Unit-Compressible Modules. *The International Journal of Robotics Research* (2003), 699–715. https://doi.org/10.1177/0278364903022900 2

[6] Robert Fitch and Zack Butler. 2008. Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots. *The International Journal of Robotics Research* 27, 3-4 (2008), 331–343. https://doi.org/10.1177/0278364907085097

[7] R. Fitch, Z. Butler, and D. Rus. 2003. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. 2460–2467. https://doi.org/10.1109/IROS.2003.1249239

[8] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. 1990. *Computer Graphics: Principles and Practice (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[9] T. Fukuda and Y. Kawauchi. 1990. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. IEEE Comput. Soc. Press, 662–667. https://doi.org/10.1109/ROBOT.1990.126059

[10] Kyle Gilpin, Ara Knaian, and Daniela Rus. 2010. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. IEEE, 2485–2492. https://doi.org/10.1109/ROBOT.2010.5509817

[11] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. 2005. Programmable matter. *Computer* 38, 6 (2005), 99–101. http://ieeexplore.ieee.org/abstract/document/1439465/

[12] A. A. Gorbenko and V. Yu. Popov. 2012. Programming for modular reconfigurable robots. *Programming and Computer Software* 38, 1 (Jan. 2012), 13–23. https://doi.org/10.1134/S0361768812010033

[13] F. Hou and W. M. Shen. 2010. On the complexity of optimal reconfiguration planning for modular reconfigurable robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. https://doi.org/10.1109/ROBOT.2010.5509642

[14] Paweł Hołobut and Jakub Lengiewicz. 2017. Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. 2103–2109. https://doi.org/10.1109/ICRA.2017.7989242

[15] Hiroshi Kawano. 2018. Distributed Tunneling Reconfiguration of Sliding Cubic Modular Robots in Severe Space Requirements. In *DARS 2018, 14th International Symposium on Distributed Autonomous Robotic Systems*. 14.

[16] K. D. Kotay and D. L. Rus. 2000. Algorithms for self-reconfiguring molecule motion planning. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, Vol. 3. 2184–2193. https://doi.org/10.1109/IROS.2000.895294

[17] Jakub Lengiewicz and Paweł Hołobut. 2018. Efficient collective shape shifting and locomotion of massively-modular robotic structures. *Autonomous Robots* (Feb. 2018). https://doi.org/10.1007/s10514-018-9709-6

[18] Othon Michail, George Skretas, and Paul G. Spirakis. 2017. On the Transformation Capability of Feasible Mechanisms for Programmable Matter. *arXiv:1703.04381 [cs]* (March 2017). http://arxiv.org/abs/1703.04381 arXiv: 1703.04381.

[19] André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. 2016. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*. IEEE, 254–263. http://ieeexplore.ieee.org/abstract/document/7778628/

[20] André Naz, Benoît Piranda, Thadeu Tucci, Seth Copen Goldstein, and Julien Bourgeois. 2018. Network Characterization of Lattice-Based Modular Robots with Neighbor-to-Neighbor Communications. In *Distributed Autonomous Robotic Systems*, Vol. 6. Springer International Publishing, Cham, 415–429. http://link.springer.com/10.1007/978-3-319-73008-0_29

[21] Florian Pescher, Benoıt Piranda, Stephane Delalande, and Julien Bourgeois. 2018. Molding a Shape-Memory Polymer with Programmable Matter. In *DARS 2018, 14th International Symposium on Distributed Autonomous Robotic Systems*. 13.

[22] Benoit Piranda. 2016. VisibleSim: Your simulator for Programmable Matter. In *Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271)*. Dagstuhl.

[23] Benoit Piranda and Julien Bourgeois. 2018. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots* (Feb. 2018). https://doi.org/10.1007/s10514-018-9710-0

[24] Kasper Støy. 2006. Using cellular automata and gradients to control self-reconfiguration. *Robotics and Autonomous Systems* 54, 2 (2006), 135 – 141. https://doi.org/10.1016/j.robot.2005.09.017

[25] Kasper Støy and Radhika Nagpal. 2007. Self-Reconfiguration Using Directed Growth. In *Distributed Autonomous Robotic Systems 6*. 3–12. https://doi.org/10.1007/978-4-431-35873-2_1

[26] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. 2017. Efficient Scene Encoding for Programmable Matter Self-reconfiguration Algorithms. In *Proceedings of the Symposium on Applied Computing*. 256–261. https://doi.org/10.1145/3019612.3019706

[27] Thadeu Tucci, Benoit Piranda, and Julien Bourgeois. 2018. A Distributed Self-Assembly Planning Algorithm for Modular Robots. In *International Conference on Autonomous Agents and Multiagent Systems) (AAMAS )*. Association for Computing Machinery (ACM), Stockholm, Sweden.

[28] S. Vassilvitskii, M. Yim, and J. Suh. 2002. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, Vol. 1. 117–122 vol.1. https://doi.org/10.1109/ROBOT.2002.1013348

[29] Mark Yim, Ying Zhang, John Lamping, and Eric Mao. 2001. Distributed Control for 3D Metamorphosis. *Autonomous Robots* 10, 1 (Jan. 2001), 41–56. https://doi.org/10.1023/A:1026544419097

[30] Echi Yoshida, Satoshi Murata, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. 1998. A distributed method for reconfiguration of a three-dimensional homogeneous structure. *Advanced Robotics* 13, 4 (1998). https://doi.org/10.1163/156855399X00234

[31] Yanhe Zhu, Dongyang Bie, Xiaolu Wang, Yu Zhang, Hongzhe Jin, and Jie Zhao. 2017. A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata. *J. Parallel and Distrib. Comput.* 102 (2017), 80 – 90. https://doi.org/10.1016/j.jpdc.2016.11.016