A Distributed Self-Assembly Planning Algorithm for Modular Robots

Thadeu Tucci
Univ. Bourgogne Franche-Comté
FEMTO-ST Institute, CNRS
Montbéliard, France
thadeu.tucci@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

A distributed modular robot is composed of many autonomous modules, capable of organizing the overall robot into a specific goal structure. There are two possibilities to change the morphology of such a robot. The first one, self-reconfiguration, moves each module to the right place, whereas the second one, self-assembly docks the modules at the right place. Self-assembly is composed of two steps, (1) identifying the free positions that are available for docking and (2) docking the modules to these positions. This work focuses on the first step. This paper presents a distributed planning algorithm that can decide which positions can be filled and can create any 3D shape, including shapes with internal holes and concavities. Our algorithm consider kinematic constraints and prevents positions from being blocked. Each module embeds the same algorithm and coordinates with the others by means of neighbor-to-neighbor communication.

KEYWORDS

modular robots; programmable matter; self-assembly; sequence planning; distributed algorithm

1 INTRODUCTION

In this work, we target modular robots composed of many homogeneous modules with limited computing resources that can change the way their modules are connected by releasing or docking some of them in order to create a given shape, thus creating intelligent objects. In nature, we have many examples of distributed organized constructions, for example ants, termites and bees. Decomposing the rules that govern these beings represents a big challenge for modular robotic.

One of the most interesting capability of a system of modular robots is the ability of the modules to move towards a different position, that way changing the global shape or morphology of the whole. This is called self-assembly or even self-reconfiguration when modules are always connected during the process.

The expected properties of modular robots [21] are: (1) versatility, modules can self-assemble into many morphologies and can be used to fulfill different tasks, (2) robustness, as a faulty module can be replaced by another, and (3) affordable price, as the mass production of identical modules is likely to reduce the overall cost.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. Andre, S. Koenig (eds.), July 2018, Stockholm, Sweden

https://doi.org/doi

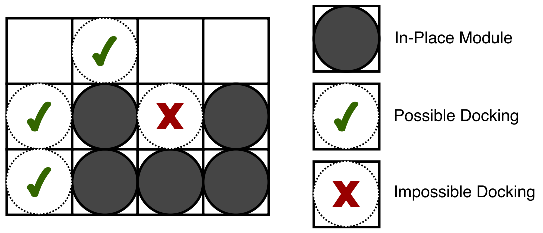

Figure 1: Example of possible and impossible docking positions due to kinematic restrictions.

Self-reconfiguration and self-assembly are hard problems for three reasons. First, the number of possible unique configurations for a modular robot is huge: $(c.w)^n$ where n is the number of modules, c the number of possible connections per module and w the ways of connecting the modules together [9]. In our locomotive example with 61,780 modules, there are $(12 \times 12)^{61,780}$ possible unique configurations for our 12 neighbors modular robot considering isomorphic configurations. Second, as modules can move or dock at the same time, the branching factor of the tree describing the configurations is $O(m^k)$ with m being the number of possible movements and k the number of modules free to move [1]. Third, as a consequence of the previous reason, the exploration space of a reconfiguration between two situations is exponential in n which prevents from finding a complete optimal planning.

There are two ways of designing the self-assembly algorithm: centralized on one module or distributed on every module. As we tackle robots composed of thousands of modules, a centralized algorithm would exceed the available memory on one module whereas a distributed algorithm would scale well. As the algorithm is distributed, each robot can be seen as an autonomous agent.

In our work, we focus on solving the problem of assembling 3D structures that may contain internal holes. While building the whole structure, modules are docking at available places but some positions may become impossible to reach, blocked by other modules.

Previous approaches focus on centralized planning or distributed planning for configurations without internal holes. In this paper, we present a distributed algorithm assisted by a shape description to create close-packed structures and assemble any morphology.

Underwater stochastic assembly has been described on [16]. Robot attraction works by a system of valves that can be controlled. Thanks to the flow of water, free modules can come to the wanted position. Attraction gradient [15] is a method of attraction to avoid that all free modules move concurrently to the same position. It is one of the methods that can be used in conjunction with the algorithm presented in this paper. In both cases, local modules have the knowledge of their neighbor positions that can be filled and accelerate the convergence to the final structure by means of message passing or valves to attract modules to dock on these positions.

The main method consists of an initial seed module that knows the final shape coded into a string and is given a relative position over the target shape. This module waits for the connection of neighbors that are freely around the structure and allows its docking whenever possible based on the rules that will be described. When a neighbor is attached to a module already in place, it receives a position relative to the first one and the final string code of the map.

An efficient encoding of the goal shape description is essential for a good behavior on a modular robots system as the description size can grow linearly with the number of robots present. As well as being memory efficient, it has to be efficient at checking whether positions are inside the goal shape as it is a recurrent task that involve energy consumption and time.

The self-assembly algorithm consists of three steps. The first step is to create the description of the final scene in a spatially efficient way and transfer it to a seed module within its position on the representation. The second step is to structure the plane creation, a solution for a two-dimensional lattice is the first structural goal. Finally, the third step consists in synchronizing the 2D solutions to get a three-dimensional representation.

The module we use in our work is a 3D Catom developed in the Claytronics project [5]. This module is a quasi-spherical robot with 12 connectors [10]. Neighborhood of each module is placed in a Face Cubic Centered lattice (FCC) that corresponds to a dense organization of spheres. This theoretical model is currently under development to make a 5 *mm* diameter robot.

In the next section, we review the related works and highlight the difference with our proposal. In section 3, we present the model, the particularities of the modular robot used, and the details about the problem we aim to address. Then, we describe how our idea and algorithm works on section 4. Section 5 is used to discuss the applications of the algorithm and the results of its performance. The conclusion and future works are discussed in section 6.

2 RELATED WORKS

The *Self-assembly planning* proposed in this paper is complementary to previous research on self-assembly and self-reconfiguration systems. Previous methods were proposed although there is a lack of solutions that work in distributed systems and can handle internal holes as can be seen on Table 1, which presents an overview of previous methods.

Wefler and Nagpal [19, 20] proposed an algorithm for distributed construction of structures without holes. These robots carry blocks and deploy them at the right position, avoiding empty spaces in the final object. They use three types of robots displacements to arrive to the proposed positions: random walking, systematic search and gradient-following. In our paper, we extend this idea to allow the construction of structures with holes.

In [16, 17], Tolley et al. worked around the kinematic restrictions and the importance of planning the self-assembly for their underwater robots. To have the order in which these robots should be connected, they start from the virtual assembled shape and remove the possible robots. Once there is no more robot left, the reverse order is used to assemble the modules without blockage. They also proposed an distributed approach to self-assembly layer by layer that limits the number of available positions for docking and the overall time for self-assembly. The robots they use do not have motion but move by the fluid flow they are in, with a control of their valves attracting robots to their allowed positions.

In [13], Seo et al. present an assembly planning algorithm for constructing planar structure out of rectangular modular robots avoiding narrow corridors. Their approach is based on graph properties as topological sort and results in a specific centralized order that is deployed to robots before assembly begins. In [14], they extend their idea to allow models with internal holes.

Jones and Matarić [7] create a transition rule set to self-assembly agents to generate a consistent assembly of a desired goal structure. The decisions are made by the agent docking in the existing structure and not by the structure and, as a result, no attraction rule can be derived.

Stoy et al. show in [15] an algorithm for 3D self-reconfiguration representing the final structure with overlapping bricks automatically generated from a CAD model. They adopted a porous scaffold of modular robots to prevent local minima and ensure that robots do not get stuck. That leads to a porous representation that allows modules to move freely in any direction. In our work we look for a dense representation for the final robot module structure.

Naz et al. published in [8] a parallel, decentralized and asynchronous algorithm for self-reconfiguration in two-dimensional lattice based robot modules. Their algorithm avoids collisions by having a gap of one empty cell between robots that are in transit using communications and can self-reconfigure to almost any compact goal shapes.

In [12], Rubenstein et al. propose a parallel, decentralized and asynchronous algorithm for the Kilobot swarm system to self-reconfigure two-dimensional robots in almost any shape. It has been applied on hardware system with more than a thousand swarm robots. Their system does not need to avoid collisions as it works on a lattice-free system and can construct sparse shapes using what they called collective artificial intelligence.

Gilpin et al. published in [3] about their programmable matter module system where each module has the size of 12mm per side and is capable of creating 2D shapes by self-disassembly. They start with a latched system and the modular robot detach unnecessary modules. In [4] they propose a 3D creation that rely on stochastic forces to self-assemble a close-packed crystalline lattice of modules and then self-disassemble into the specific shape. It is a functional approach that can reduce the complexity but it requires a bigger number of robots to assemble a specific shape as some robots will be discarded by the disassembly.

3 MODEL AND DEFINITION

3.1 Module background

Our modular robot model is abstracted from the model of a Claytronics Atoms, as Catoms[6], where modules are placed in a regular grid, oriented along the $\vec{x}$, $\vec{y}$ and $\vec{z}$ axes. The grid is defined by a large set

Table 1: Robots Planning Algorithm Overview

Author	Architecture	Dimension	Distributed	Compact Representation	Internal Holes
Wefler and Nagpal [19, 20]	Heterogeneous Modular Robots	3D	Yes	Yes	No
Tolley et al. [16, 17]	Underwater Modular Robots	3D	No	Yes	Yes
Stoy et al. [15]	Porous Scaffold Modular Robots	3D	Yes	No	Yes
Seo et al. [13, 14]	Rectangular Modular Robots	2D	No	Yes	Yes
Jones and Matarić [7]	Modular Robots	2D	Yes	Yes	Yes
Naz et al. [8]	Modular Robots	2D	Yes	Yes	No
Rubenstein et al. [12]	Swarm Robots	2D	Yes	No	No
Our method	Modular Robots	3D	Yes	Yes	Yes

of cells where each cell can only contain one block. Each cell of the grid has 2 different states: empty or filled with a block.

We first assume that modules are organized in a 2D square lattice and can only communicate with their directed neighbors. Any communication to distant modules needs a certain number of hops. Modules have sensors to detect without communication the presence of docked neighbors.

The position P_c of the cell c in the grid is given by coordinates in $\mathbb{Z}^2$. Each block can be directly connected to up to four neighbors in a 2D lattice. We will generalize the method to 3D models at Section 4.2 considering connected overlapping layers.

The following properties are applied to each module:

- Same hardware executing the same program;
- Unique id;
- Is able to get its orientation relative to global referential using embedded sensors;
- Detect the presence of docked neighbors in the adjacent cells;
- Communicate with its connected neighbors.

Modules can move freely in different environments, as swarm or underwater robots, or connected and assisted by neighbors. In both cases, the movement of docking in a final position must satisfy the rule of docking (see Rule 1).

RULE 1. *A module can dock on a cell only if there is no two adjacent cells in symmetrically opposed planes to that cell that are occupied.*

Rule 1 defines a kinematic restriction applied to 2D and 3D lattice. Indeed, we assume there must be enough space to allow modules to arrive at the cell. Figure 3 shows an example of case that can be restricted by Rule 1 if modules A and B are part of different contiguous row.

Let $\mathcal{G}$ be the goal shape, we assume that each module B_i stores locally a copy of $\mathcal{G}$. A module B_i has a position $P_i(x_i, y_i)$, with $(x_i, y_i) \in \mathbb{Z}^2$.

These modules can be extended to use a FCC 3D lattice which is made of a regular square grid on each horizontal layer $\overrightarrow{XY}$ and interleaved modules on $\overrightarrow{XZ}$ and $\overrightarrow{YZ}$ axes. Using this lattice allow modules to have up to 12 neighbors and to create a dense organization of quasi-spherical modules [10] (Figure **??**).

3.2 Problem definition

The main idea is to start from a single initial seed module S_0 which stores $\mathcal{G}$ and has a position in the target object. This module attracts many others to assemble an intermediate set of modules in the connectors of S_0 that must have a neighbor according to $\mathcal{G}$ and must ensure that all positions in $\mathcal{G}$ can still be reached. S_0 sends $\mathcal{G}$ to its new neighbors with their relative coordinates. One after another, new modules in the structure attracts new neighbors to build a dense target object.

Modules should have enough memory to store a description of the goal shape $\mathcal{G}$. As a consequence we use an optimized shape description. Stoy and Nagpal[15] propose a vector method that uses overlapping bricks to overcome the memory and scale problem. The model defines an approximation of the shape that can be used to scale the size of the representation although some quality loss. Another

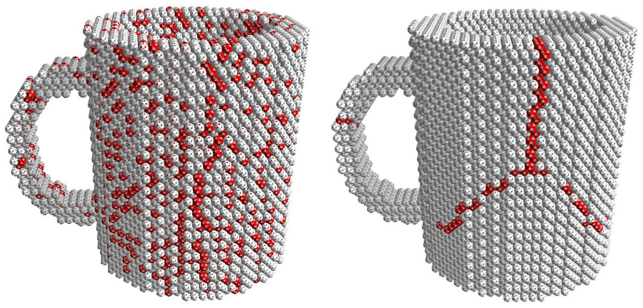
Figure 2: Importance of sequence planning shown on a Mug model made by 12,000 modules. Red modules represent docking problems using two simple planning algorithm. On the left, choosing a stochastic order produces 3,691 modules that does not verify the docking rule. On the right, filling regularly each module neighborhood in sequence results in 231 positions that could not be filled.

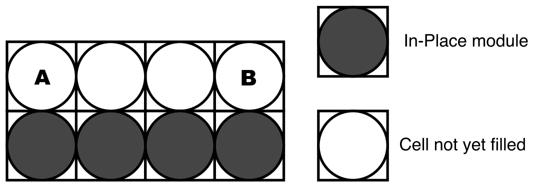

Figure 3: Modular robots and kinematic constraints in a row: only one contiguous line of blocks can be allowed. Modules A and B cannot be part of different contiguous line to avoid positions to get blocked.

vector method [18] for programmable matter has been proposed which requires low memory and with no quality loss. It uses a CSG model as description of the scene and the associated tree can be easily compressed to be stored in each module.

The robots can attract others only when this action will not cause future positions to get blocked and lead to holes on the system. For example, in Figure 3 we show that if cells A and B becomes occupied before the center cell, creating two contiguous rows, this will lead to a future position getting blocked. At each row just one contiguous line of robots should be possible to avoid blocked positions. In another general example, Figure 2 shows the construction of a mug without an algorithm that avoid these blocked cells.

The condition of having only one contiguous line of robots by row is the first of two conditions to avoid unwanted holes in the final object. The second condition is attracting a node in an existent row only when one corner of its cells is on a final state, as represented in Figure 5.

4 DISTRIBUTED SELF-ASSEMBLY ALGORITHM

Algorithm 1: Algorithm input and function detailed for border following over module on position P.

```
Input:
  $\mathcal{G}$ // global goal shape
  $P$ // position of $B_i$
  $dir \in \{N, E, S, W\}$ // initial direction of following
  $searchDir \in \{CW, CCW\}$ // direction of rotation

Function borderFollowing($P, dir, searchDir$):
    $j \leftarrow (dir + 3) \mod 4$; // predecessor direction
    for $i \in [0...3]$ do
        $Q \leftarrow P + NextDir(j, searchDir)$;
        if $Q \in \mathcal{G}$ then
            borderFollowing($Q, j, searchDir$);
            return;
        end
        $j \leftarrow (j + 1) \mod 4$;
    end
end
```

The use of an efficient method for scene encoding is a critical aspect that can reduce the memory, the time of transfer and the energy used in a system of self-reconfiguration modular robots. In this work we use a compact method based on CSG [11] know as CSG4PM *Constructive Solid Geometry for Programmable Matter* [18] that is a vector description that ensures scalability, fidelity and reduced memory when compared to other methods. This method consists in defining a tree of objects that can be combined using boolean operators in order to model the final solid object. Depth-first search algorithm on the tree can be used to solve the in/out problem as knowing if the cell C is inside the model or not.

Algorithm 2: Algorithm functions for modules attraction.

```
Function northAttraction($B_i$):
    if isNorthSeed($B_i$) // Rule 2
     then
        sendAttractSignalTo($\langle x_i, y_i + 1\rangle$);
end

Function southAttraction($B_i$):
    if isSouthSeed($B_i$) // Rule 3
     then
        sendAttractSignalTo($\langle x_i, y_i - 1\rangle$);
end

Function westAttraction($B_i$):
    if $West(B_i) \in \mathcal{G}$ then
        if $SouthWest(B_i) \in \mathcal{G} \wedge$
          $hasModuleOnSouthConnector(B_i)$ then
            getAuthorizationToAttract($South(B_i)$, $WEST$);
        else if isOnInternalHole($B_i$) then
            borderFollowingToGetAuthorizationToAttract();
        else
            sendAttractSignal($\langle x_i, y_i - 1\rangle$);
        end
end

Function eastAttraction($B_i$):
    if $East(B_i) \in \mathcal{G}$ then
        if $NorthEast(B_i) \in \mathcal{G} \wedge$
          $hasModuleOnNorthConnector(B_i)$ then
            getAuthorizationToAttract($North(B_i)$, $EAST$);
        else if isOnInternalHole($B_i$) then
            borderFollowingToGetAuthorizationToAttract();
        else
            sendAttractSignal($\langle x_i, y_i + 1\rangle$);
        end
end

Function getAuthorizationToAttract($B_i, direction$):
    if $direction = WESTt$ then
        if isConnected($West(B_i)$) then
            sendAuthorizationToAttract($North(B_i)$, $WEST$) ;
    else if $direction = EASTt$ then
        if isConnected($East(B_i)$) then
            sendAuthorizationToAttract($South(B_i)$, $EAST$) ;
end

Function sendAuthorizationToAttract($B_i, direction$):
    if $direction = WESTt$ then
        sendAttractSignal($\langle x_i - 1, y_i\rangle$);
    else if $direction = EASTt$ then
        sendAttractSignal($\langle x_i + 1, y_i\rangle$);
    end
end
```

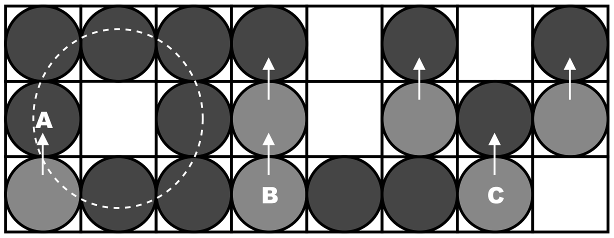

Figure 4: Light-gray modules represent seeds which are responsible for attracting a new module on its north row. Each contiguous row can have only one seed to avoid blocked positions. The rightmost module is alway elected, except for Module A, since it is part of an internal hole.

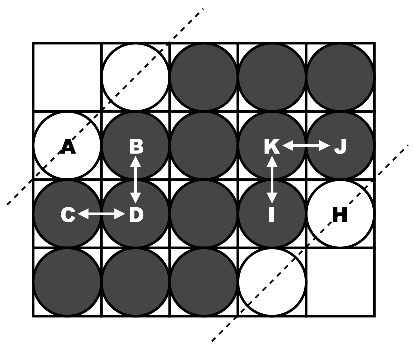

Figure 5: Cells that should be in place before attracting neighbors. Before attracting a module in cell A, modules B and C should already be in place. B and C are not direct neighbors, so to communicate they need a common neighbor module to know if both are already in place. The construction method fills space by adding modules along the diagonal line in both directions simultaneously (NW and SE).

4.1 2D Self-Assembly Algorithm

The algorithm follows these steps: First, the goal object $\mathcal{G}$ is encoded using the CSG4PM method and transferred to the first seed module. From the initial seed module it can try to attract modules on its sides connected to cells that are in $\mathcal{G}$. For each of these sides it requires specific rules that can be solved using the goal shape description, the sensor of neighbor presence and neighbor-to-neighbor communication.

A typical case of forbidden position is created if a row on the goal shape description has more than one contiguous row of modules on the system. Only one module, that is called seed, is responsible for attracting a module to a next or previous line.

Modules can decide if they are seed without any communication but using the shape description of $\mathcal{G}$. To decide if a module is a seed, it applies the Rules 2 and 3, for north and south seeds respectively. The first rule elects the rightmost module that has its north position inside the shape description as seed. A row can have more than one seed, for example seed modules B and C in Figure 4. A module can decide if it is a seed too if its north-east position is not inside the shape description and its north position is. The same can be done for their opposite direction, a module can be a seed for its previous row. In this case they use their south and south-west positions as reference.

The following rules define a module B_i in its position as a north or south seed:

Rule 2.

$$\begin{aligned} isNorthSeed(B_i) \implies & \neg isNorthLineOnMerge(B_i) \wedge \\ & \langle x_i, y_i + 1 \rangle \in \mathcal{G} \wedge \\ & (\langle x_i + 1, y_i + 1 \rangle \notin \mathcal{G} \vee \\ & \langle x_i + 1, y_i \rangle \notin \mathcal{G}) \end{aligned}$$

Rule 3.

$$\begin{aligned} isSouthSeed(B_i) \implies & \neg isSouthLineOnMerge(B_i) \wedge \\ & \langle x_i, y_i - 1 \rangle \in \mathcal{G} \wedge \\ & (\langle x_i - 1, y_i - 1 \rangle \notin \mathcal{G} \vee \\ & \langle x_i - 1, y_i \rangle \notin \mathcal{G}) \end{aligned}$$

Figure 4 have in light-gray seeds for their corresponding north position. Module marked as B is a seed due to the fact that its north position is in the shape description and their north-east position is not. Module C is seed as it is the rightmost module of the line and its north position is inside the shape. An exception is the module A which is in a case of merging line, a consequence of the internal hole, and will be discussed later.

A module can attract another to its west side when the cell located on its south-west has finished. Modules in a regular grid do not have direct communication with modules in their diagonal, as the case of the south-west location. A module on its south can be used as intermediate to this communication when south module is present (Figure 5). Otherwise, if there is no module on its south, it can be a case of internal hole and the module sends a message over the inner border, following the Algorithm 1 to communicate and have the certitude the module is already in place before attracting on its west side.

The same method is applied when attracting a module on its east side but with the symmetrically opposed verification. It is based on the presence of the module on its north-east location, when it is in $\mathcal{G}$. If the north-east cell is not inside the shape description it can attract a module immediately. The Algorithm 2 shows the rules to attract neighbors on all its four sides.

Modules in the internal border may not yet be in place when executing a border following algorithm. In this case, the current module creates a queue of messages which is transmitted after the docking, sent along with module position and goal scene description. Algorithm 1 shows how border following works on a distributed environment.

The following data is used and transmitted when a module arrives at its final position:

- Target scene description
- Relative position over the target scene
- Queue of messages for merge synchronization

Module A on Figure 4 is not an elected seed as, in this case of internal hole, it would have two seeds for the same north row and may lead to a situation as foreseen in Figure 3. The algorithm verifies that is a point of row merge using the Rule 4. Thereafter, it should check if it is a case of internal or external hole. It can be done by running the same algorithm (see Algorithm 1) and counting the number of turns it made to complete the hole. No communication is necessary as it only needs the scene description.

RULE 4.

$$\begin{aligned} isNorthLineOnMerge(B_i) \implies & (\langle x_i + j, y_i\rangle \notin \mathcal{G} \wedge \\ & \langle x_i + j, y_i + 1\rangle \in \mathcal{G} \wedge \\ & \langle x_i + n + 1, y_i\rangle \in \mathcal{G} \wedge \\ & \langle x_i + n + 1, y_i + 1\rangle \in \mathcal{G}) \\ & \forall j \in \{1, \dots, n\} \end{aligned}$$

RULE 5.

$$\begin{aligned} isSouthLineOnMerge(B_i) \implies & (\langle x_i - j, y_i\rangle \notin \mathcal{G} \wedge \\ & \langle x_i - j, y_i + 1\rangle \in \mathcal{G} \wedge \\ & \langle x_i - n - 1, y_i\rangle \in \mathcal{G} \wedge \\ & \langle x_i - n - 1, y_i - 1\rangle \in \mathcal{G}) \\ & \forall j \in \{1, \dots, n\} \end{aligned}$$

Figure 6 shows the construction of an object with internal hole. The red module detects it is in a merge situation and awaits to be sure the other side is in place before continuing. Modules change their color to blue to show the route of messages from East to West, and to green when they are transmitting a response. The message follows the border of the hole and requires only as extra information the previous direction of the message.

4.2 3D Self-Assembly

In this subsection we propose a solution for a three-dimensional lattice self-assembly.

For our 3D self-assembly we assume there can be only one initial point of start for a contiguous next plane. The module that is responsible for attracting another to the next plane is called 3D seed. Each module can locally determine if it is a 3D seed using $\mathcal{G}$ description, without neighbor-to-neighbor communication.

Modules can use the same algorithm to follow the inner border and always select the module with the lowest value on $\vec{x}$ axis and then with the lowest value on $\vec{y}$ axis. For each module on the actual plane they check if the respective next plane module is on a border and is the lowest. In the case where the next plane is bigger than the actual plane, the lowest module of the exterior border of the actual plane is a seed if there is a module on top of it.

A seed module can attract for the next plane only when the current plane has finished, therefore a consensus on when plane have finished is necessary. Using the given self-assembly algorithm generates an implicit spanning tree that is used to send data to the first module of the plane. Planes can know how many modules they have using the description scene. This number of modules is used to confirm the plane is completed.

This method is efficient as each plane starts its construction immediately after it arrives at a final state, but it does not handle objects with loops as two planes growing separately and merging in a certain point. A solution to objects with loops is synchronizing the planes, constructing each plane one after another. To synchronize plane construction, when a plane is subdivided in many local connected areas,

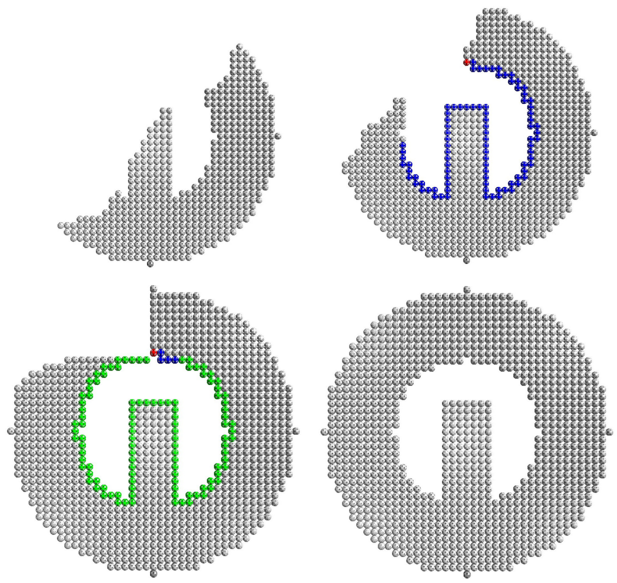

Figure 6: Example of constructing a model with internal hole in a two-dimensional square lattice. The task is parallelized as many positions can be filled at the same time. The modular robot drawn in red detects a section of line merge made by the internal hole. It sends a message following the inner border (blue modules) and waits for an unlock message. At the moment the message arrives in the south-west position of the red module, the module sends a response in the reverse path (green). This verification is necessary before a module docks on the red module west side in order to have no position blocked.

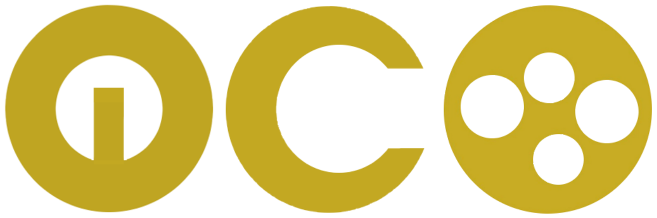

Figure 7: Three classes of target structure: Power Button, Letter C and Bumpy respective representations.

we define a tree of seeds linking 3D seeds of connected planes. Then when a local connected area has finished, a message ($EndOfPlane$) is sent to the root of the tree of 3D seeds. This root 3D seed sends the authorization to create the next plane after having received all $EndOfPlane$ messages.

5 EXPERIMENTAL EVALUATION

We have implemented and evaluated the algorithm using VisibleSim [2], a C++ simulator for modular robots. VisibleSim enables the users to run event-based simulations. An event is a task executed by the module simulating one of its actuator, for example attracting

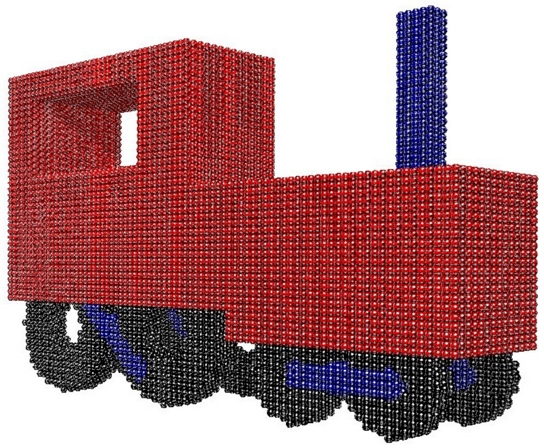

Figure 8: Locomotive example made by 61,780 Catoms constructed using the distributed 3D self-assembly algorithm. Each cell in the target shape could be filled with modules.

a new robot and sending a message. Messages and events can be pushed to the system with communication and actuator simulated delays. In order to test our algorithm, we use random delays with a large range which change the orders of docking, making each evaluation executed on the system unique.

We studied the number of messages by the scaling factor and the number of simultaneous attracting positions for four classes of target structure (called "power button", "letter c", "bumpy" and "2D locomotive") presented in Figure 7 and Figure 8. We made a more realistic experimentation using a locomotive 3D model shown in Figure 8, described by 61,780 Catoms. In order to have a proper comparison with the other target structures we used a lateral view of the locomotive for a 2D representation. Efficiency of our algorithm is affected by geometrical and topological characteristics of the goal shape:

- The "power button" model proposes a simple case of internal hole (homeomorphic to a torus) with concave parts;
- The "letter c" model is a concave shape homeomorphic to a sphere;
- The "bumpy" model contains several internal holes at once;
- The "2D locomotive" groups many complex areas with holes, convex and concave parts.

For each representation three versions were used, each one scaling the size of the structure by two. As for the representation of the power button, we start with a structure using 815 robots, scale it to $3,318$ and $13,233$ modules. Through our experiments, we show the effectiveness of our method in terms of number of messages and number of available docking positions at the same time.

5.1 Messages Evaluation

The number of messages scales linearly with the number of modules as shown in Figure 9. The messages on the system are composed of an initial communication, where a recently docked module sends a message to a neighbor module, asking for initial data as a relative position. This initial communication increases the number of messages by $2n$. Communication is also used to define when a position can be occupied and requires more communication depending on the target structure. Therefore, by experimental results, the proposed algorithm requires an average of $5n$ messages. For example, the bumpy structure has more internal holes and requires a greater number of messages, but no significant difference was found between the four classes of the target structure.

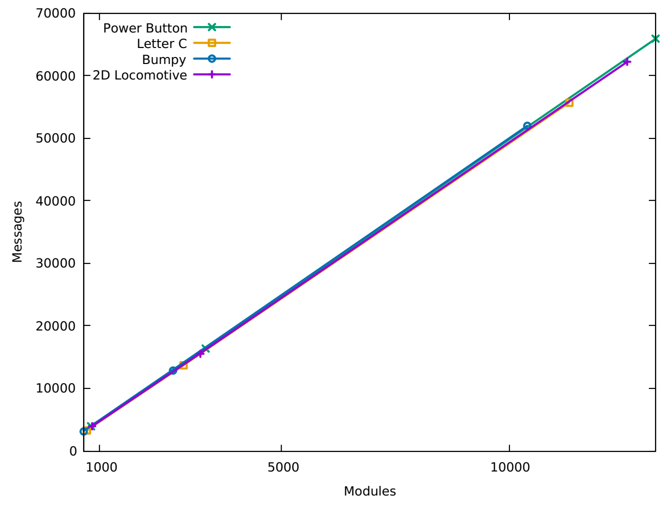

Figure 9: Number of messages for four topologically different representations on three different scales. The number of messages remains linear and similar across all representations.

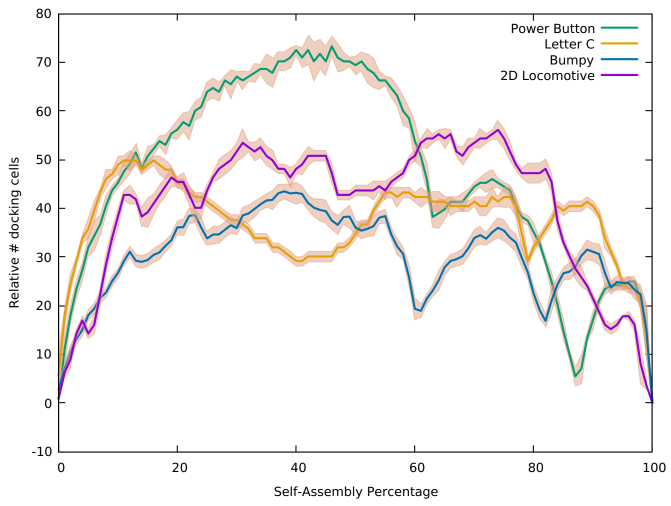

Figure 10: Relative number of available docking positions over the percentage of construction. It shows parallelism of the method by having as many docking positions as possible at the same time.

5.2 Available Docking Positions Evaluation

The number of available positions attracting modules can speed up the convergence to the final object as many modules can arrive at the same time in different positions. The movement of free modules to their correct position has a large impact in the overall time of the self-assembly. The algorithm attracts new modules in the diagonal of the target structure, and placing the initial seed at a position relative to the middle of the structure can make the convergence twice as fast as there are two diagonals, one facing the west side and the other facing east. The number of available docking positions is related with the goal morphology and the position of the initial seed.

Be D the number of available positions and N the total number of modules, $\frac{D}{\sqrt{N}}$ is a relation of the number of available docking positions that does not change when scaling the target. Figure 10 shows the relative number of available docking positions by the construction percentage. As power button can have up to 4 diagonals being filled at the same time it has, at a certain step of the construction, more available docking positions.

6 CONCLUSION

We presented a distributed algorithm for a system of modular robots that is capable of generating an attraction list of its available positions. Our algorithm prevents positions from becoming impossible to reach and is able to create close-packed structures with internal holes that are applicable to a variety of modular robotic systems. One of the tasks that requires lots of time in a self-assembly system is to find the exact location for the modules to dock. We show with our results that the algorithm can have many simultaneous docking positions and accelerate the creation of a model with a linear number of messages.

In future works, we hope to extend the actual plane-by-plane algorithm to be able to have more available parallel docking positions on a 3D lattice. We would also like to combine this planning with algorithms to control module movements. Finally, it would be interesting to have the algorithm implemented and tested on an existing hardware.

ACKNOWLEDGMENTS

The authors would like to thank the Brazilian Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), 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 Mobilitech project.

REFERENCES

[1] Jérôme Barraquand and Jean-Claude Latombe. 1991. Robot Motion Planning: A Distributed Representation Approach. *The International Journal of Robotics Research* 10, 6 (Dec. 1991), 628–649. https://doi.org/10.1177/027836499101000604

[2] Dominique Dhoutaut, Benoît Piranda, and Julien Bourgeois. 2013. Efficient simulation of distributed sensing and control environments. In *IEEE Internet of Things (iThings/CPSCom)*. IEEE, 452–459.

[3] Kyle Gilpin, Ara Knaian, and Daniela Rus. 2010. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2485–2492.

[4] Kyle Gilpin and Daniela Rus. 2010. Modular Robot Systems. *IEEE Robotics Automation Magazine* 17, 3 (Sept. 2010), 38–55. https://doi.org/10.1109/MRA.2010.937859

[5] Seth Goldstein and Todd Mowry. 2004. Claytronics: A Scalable Basis For Future Robots. *RoboSphere 2004* (Nov. 2004).

[6] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. 2005. Programmable matter. *Computer* 38, 6 (2005), 99–101.

[7] Chris Jones and Maja J. Mataric. 2003. From local to global behavior in intelligent self-assembly. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, Vol. 1. 721–726 vol.1. https://doi.org/10.1109/ROBOT.2003.1241679

[8] 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.

[9] Michael Park, Sachin Chitta, Alex Teichman, and Mark Yim. 2008. Automatic Configuration Recognition Methods in Modular Robots. *The International Journal of Robotics Research* 27, 3-4 (March 2008), 403–421. https://doi.org/10.1177/0278364907089350

[10] Benoit Piranda and Julien Bourgeois. 2016. Geometrical study of a quasi-spherical module for building programmable matter. In *DARS 2016, 13th International Symposium on Distributed Autonomous Robotic Systems.*

[11] Aristides AG Requicha. 1980. *Representations of rigid solid objects*. Springer.

[12] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. 2014. Programmable self-assembly in a thousand-robot swarm. *Science* 345, 6198 (2014), 795–799.

[13] Jungwon Seo, Mark Yim, and Vijay Kumar. 2013. Assembly planning for planar structures of a brick wall pattern with rectangular modular robots. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. 1016–1021. https://doi.org/10.1109/CoASE.2013.6653996

[14] Jungwon Seo, Mark Yim, and Vijay Kumar. 2016. Assembly sequence planning for constructing planar structures with rectangular modules. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 5477–5482. https://doi.org/10.1109/ICRA.2016.7487761

[15] Kasper Stoy and Radhika Nagpal. 2007. Self-Reconfiguration Using Directed Growth. In *Distributed Autonomous Robotic Systems 6*, Rachid Alami, Raja Chatila, and Hajime Asama (Eds.). Springer Japan, 3–12. DOI: 10.1007/978-4-431-35873-2_1.

[16] Michael T. Tolley and Hod Lipson. 2010. Fluidic manipulation for scalable stochastic 3D assembly of modular robots. In *2010 IEEE International Conference on Robotics and Automation*. 2473–2478. https://doi.org/10.1109/ROBOT.2010.5509664

[17] Michael T Tolley and Hod Lipson. 2011. On-line assembly planning for stochastically reconfigurable systems. *The International Journal of Robotics Research* 30, 13 (Nov. 2011), 1566–1584. https://doi.org/10.1177/0278364911398160

[18] 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*. ACM, 256–261.

[19] Justin Werfel, Yaneer Bar-Yam, Daniela Rus, and Radhika Nagpal. 2006. Distributed construction by mobile robots with enhanced building blocks. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2787–2794. https://doi.org/10.1109/ROBOT.2006.1642123

[20] Justin Werfel and Radhika Nagpal. 2008. Three-Dimensional Construction with Mobile Robots and Modular Blocks. *The International Journal of Robotics Research* 27, 3-4 (March 2008), 463–479. https://doi.org/10.1177/0278364907084984

[21] Mark Yim, Ying Zhang, and David Duff. 2002. Modular robots. *IEEE Spectrum* 39, 2 (2002), 30–34.