Self-Reconfiguration of Modular Robots Using Virtual Forces

Edy Hourany[1], Christian Stephan[1], Abdallah Makhoul[1], Benoit Piranda[1], Bachir Habib[2] and Julien Bourgeois[1]

***Abstract*— Programmable matter is a material that can change its physical properties at will, whether it is its shape, density or conductivity. It can be implemented as an ensemble of micro-robots arranged in space to form a specific shape and having their own computing power. This technology behaves as a distributed system. Each micro-robot is called a module and the whole forms a modular robot. This paper tackles the self-reconfiguration problem by presenting a deterministic planning algorithm that can decide which positions can be filled over multiple iterations using virtual forces. The proposed algorithm implements the Hungarian method to optimize the planning by minimizing the total number of movements of the robots and preventing positions from being blocked. Each module embeds the same algorithm and coordinates with the others using neighbor-to-neighbor communications. Simulation results are conducted to show the effectiveness of the proposed approach.**

I. INTRODUCTION

Modular robots are designed with parts that can be reconfigured to form different shapes and functions. In many cases, such robots are able to reconfigure their own shape autonomously. Modules may be large and unique or small and identical. In the later case, robots begin to resemble to programmable matter that can take any shape. For example, a robot could take a snake shape to crawl through a pipe or take a human shape. In theory, small reconfigurable modules with programmable properties could represent a type of universal robot that can take any shape and perform virtually any function. Such modules may be expensive to develop but are cheap to produce in the long term as they achieve economies of scale [1].

One of the foremost interesting capabilities of modular robots, is the capacity of each module to move towards diverse positions, changing the whole shape and morphology of the grouped modules to achieve different tasks and to adapt to terrain variations. According to [2], this can be achieved either using self-reconfiguration or self-assembly which are hard problems for the following three reasons. First, working with a high number of computational units generates a huge number of possible unique configurations, $(c.w)^n$ where n is the number of modules, c the number of possible connections per module and w the ways of connecting the modules [3]. Second, the modules can move simultaneously, so the branching factor of the tree describing the configurations is $\mathcal{O}(mk)$ with m being the number of possible movements and k the number of modules free to move. Finally, the exploration space of a reconfiguration between two situations is exponential in n which prevents from finding a complete optimal planning.

In this paper, a new self-reconfiguration algorithm for modular robots based on virtual forces is proposed. The main objective is to optimize the shape formation by minimizing the number of movements of the modules. The Virtual Force Algorithm, denoted as VFA for the remainder of the paper, was introduced by [4] to properly place a randomly distributed set of sensors. Once the effective sensors positions are identified, a one time movement with energy consideration incorporated is carried out, i.e., the sensors are redeployed to these positions. This work was inspired by the VFA to identify the attractiveness of empty positions for the free modules.

The remainder of this paper will be as follow: Section II presents the literature review, a brief review of prior research on topics related to self-assembly, self-reconfiguration and virtual forces approach in the robotics world. Section III presents the particularities of the modular robot used and the details about the problem we aim to address. Section IV describes the proposed algorithm. Section V presents the simulation results based on the proposed approach and an elaboration on its performance. Section VI draws the conclusion and outline directions for future works.

II. LITERATURE REVIEW

Modular Self-Reconfigurable Robots (MSR) can be classified in three different types according to [5]: chain, lattice, or mobile reconfiguration. Chain-type MSRs change configuration by attaching and detaching chains of modules to and from themselves, with each chain always attached to the rest of the modules at one or more points. Lattice type robots change shape by moving into positions on a virtual grid, or lattice similar to a chessboard game. The robot only needs to deal with what is occupying the limited number of neighboring positions in the lattice and therefore requires less computation. Lastly, mobile self-reconfiguring robots change shapes by having modules detaching from the ensemble and moving independently until attaching again in another point.

A self-reconfiguration algorithm is composed of two phases: an assembly planning phase and a movement phase. In [2], the authors propose a distributed algorithm for building a shape by defining on the fly the assembly planning. The advantage of this method is that it does not require any pre-calculation which allows a total adaptability to the situation and a better resistance to errors. The difficulty in this case is to detect and deal with blocking situations so as not to position a robot in a location that will then block the

[1] Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, Montbéliard, France firsname.lastname@univ-fcomte.fr
[2] Department of computer science, Holy Spirit University of Kaslik, Jounieh - Lebanon bachir.habib@uek.edu.lb

others. They propose a distributed algorithm based on long-distance communication between robots, using the point-to-point communication network, in order to manage these situations. The algorithm works even in configurations with multiple holes that are the most complicated to manage.

The authors in [6] proposed an assembly planning algorithm for constructing planar structures out of rectangular modular robots that can dock to each other to form a brick wall pattern. The assembly plan incorporates reachability and and collision constraints and supports distributed assembly so that multiple robots can be docked to a growing structure without centralized coordination.

In their paper, [7], the authors introduced a parallel, asynchronous and fully decentralized distributed algorithm to self-reconfigure robots in 2D vertical lattice from an initial configuration to a goal structure. 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 [8], the authors present a method to organize the iterations of self-assembly agents to generate a consistent assembly of the desired goal structure. The decisions are made by the agent docking in the existing structure and not by the structure. In other term, the decision are made locally by each module in the system.

Another programmable matter module was introduced in [9], where each module has a size of 12 mm per side and is capable of creating a 2D shape by self-disassembly. The authors start with a latched system and the modular robot detaches unnecessary modules.

Thalamy et al. proposed in [10] an approach for accelerating self-reconfiguration by building a porous version of the desired shape, using scaffolding. This method that constructs a parametric scaffolding model, 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 generates that module movements and the construction of the scaffold is deterministic.

In [11], the authors presented a probabilistic self-reconfiguration algorithm for re-configuring rhombic dodecahedron modules residing in a Face Centered Cubic (FCC) lattice, and that can only move using rotations. The authors also proposed the Goal-Ordering method. In this method, the modules use one or two metrics to decide which target location in the goal configuration they should go fill. This method, however, suffers from overcrowding around the open goal positions and is likely to get stuck in local minima and avid converging altogether, especially when the shape is solid and hollow.

Stoy, in his papers [12], [13], 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 [12], or through a gradient descent method [13].

Recently, in [14] the authors propose a self-reconfiguration algorithm based on clustering [15]. The self reconfiguration is done in a parallel manner in each cluster to find the final shape at the end. The proposed algorithm consists in moving the robots that are not in the target shape in the interior of the current configuration in order to fill target free cells.

In this paper, a hybrid algorithm using both centralized and decentralized decisions is proposed. They are both used to reconfigure the robots into a goal structure with the least possible cost. Distributed virtual forces algorithm with a Hungarian optimization approach are used to limit overcrowding around the open goal positions and to avoid getting stuck in local minima.

III. Assumptions and Definitions

The modules used here are 2D node models with 4 connections that can rotate in Clock Wise (CW) and Counter Clock Wise (CCW) directions, also can make a linear or circular translation to move around other modules. The modules are placed on an unlimited grid, oriented along the $\overrightarrow{x}$, $\overrightarrow{y}$. The grid is defined by a large set of cells where each cell can only contain one block. Each cell of the grid has only two different states: empty of full.

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. The following properties are then applied:

- Unique ID;
- Same hardware executing the same software;
- Can get its orientation relative to global referential using embedded sensors;
- Detect the presence of docked neighbors in the adjacent cells;
- Communicates with its connected neighbors only;

Modules can move freely in different directions. They can only rotate in Clock Wise (CW) or Counter Clock Wise (CCW) directions and can perform a translation or a rotation depending on the pivot and near obstacles: so in total, they have 16 possible motions. The movement of the modules needs to satisfy *condition 1.*

Condition 1: A module can move freely without any manual intervention if there is only one pivot in its neighborhood.

Condition 1 defines the case where a node has more than one neighbor that can be used as a pivot and is not able to decide which one to use. Figure 1 shows an example of an invalid case of movement. Indeed, the module 3 has as pivot module 1 and 2, it can not decide by itself which node should be considered as a pivot. Whereas in Figure 2 and Figure 3, module 3, according to Condition 1, can pick the only neighbor it has like a pivot. Figure 2 represents the case of a CW motion with rotation because the left side of the pivot is empty. Figure 3 represents the case of a CCW motion with a translation because the right side of the pivot is filled.

Let $\mathcal{G}$ be the goal shape, we assume that each module s_i stores locally a copy if $\mathcal{G}$. A module s_i has a position

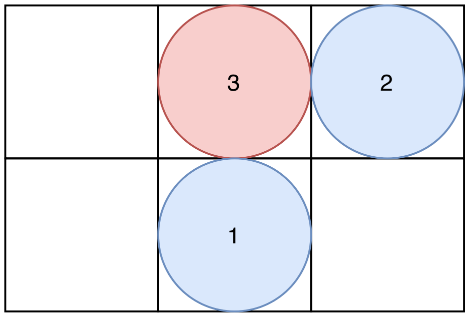

Fig. 1. Invalid Node Motion according to Condition 1. The red module (module 3) is a module with 2 connections. Thus a pivot for rotation is not possible.

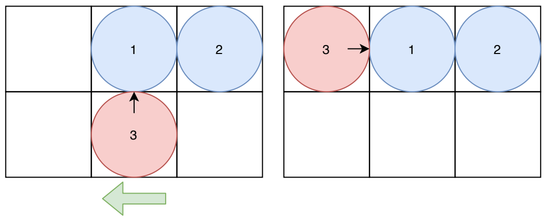

Fig. 2. Clock Wise (CW) direction and rotation. The red module (module 3) is a module with 1 connection. Thus a rotation pivot can be identified and a rotation is then possible.

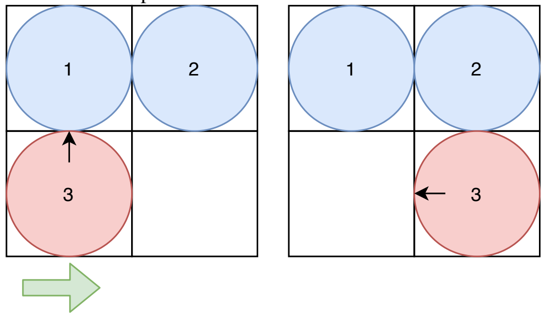

Fig. 3. Counter Clock Wise (CCW) direction and translation. The red module (module 3) is a module with 1 connection. Thus, a pivot can be identified and a translation is then possible.

$p_i(x_i, y_i)$, with $(x_i, y_i) \in \mathbb{Z}^2$. Furthermore, in this paper, it is assumed that:

- The size of the system, as well as the initial shape of the system, is unknown.
- The leader is elected and can be any module in the system being at any position.
- Communication is only possible between adjacent neighboring modules.
- Each module is aware of the connections it has at any given moment, thus the loss of any message in the middle of the transmission is improbable and therefore ignored.
- Each module is aware of the goal shape.
- Each module can scan the grid and can check if a certain position (x, y) is empty or filled.
- Each module has access to a scheduler queue where they can push events to be executed either instantly or in a future time.

IV. Proposed Algorithm

The problem treated in this paper is the following: having a goal shape $\mathcal{G}$, the free modules present in the grid will define the potential target positions inside $\mathcal{G}$ that they may fill by applying the proposed algorithm. The leader will pick the free modules and the goal positions to be filled in a way that minimizes the cost of the whole structure. However, after moving some modules, the leader would be able to pick other potential nodes and fill new empty positions. Therefore, the process will repeat itself until no further solutions are found by the leader.

In this paper, a deterministic self-reconfiguration algorithm, is proposed, based on virtual forces and composed of two major phases: (a) Distributed calculation of the potential movements of the free nodes, and (b) Centralized optimization of the overall movements of all these nodes.

A. *Distributed Virtual Force Algorithm (VFA)*

The first phase of the algorithm is split into three main steps:

1) VFA phase: each free node detects the free positions in the goal structure and for each of them, it defines a weight as a result of the attractive and repulsive forces towards this free position.
2) Exploration phase: the leader explores the whole system of modules while at the same time builds a tree structure rooted at the leader module itself.
3) System potential movements reporting phase: This phase is launched in parallel to the previous phase, in a manner where the algorithm collects all the free nodes' potential movements and their weights to report the total potential movements from the whole system to the leader module.

1) Phase 1: Virtual Force Algorithm: In modular robots, the proposed approach considers each free node as a "source of force" for all the empty positions in the target structure. This force can be either positive (attractive) or negative (repulsive). On one hand, the attractive force is defined as being inversely proportional to the number of moves required by this free node to reach a certain empty position in a specific direction (CW or CCW); indeed, the more moves the node needs to reach the empty position the less the force exerted is attractive. On the other hand, the repulsive forces are defined as the existence of an invalid path in a certain direction according to Condition 1.

In a neighborhood of M modules, let $j \in (CW : 0, CCW : 1)$ and let the total force exerted by a node s_i on an empty position p_k in a certain direction j be denoted by $\overrightarrow{F_{ijk}}$ which is a vector whose orientation is determined by the vector sum of all the forces exerted by s_i in the direction j on the empty position p_k. If $\overrightarrow{F_{ijkA}}$ and $\overrightarrow{F_{ijkR}}$ are considered, respectively, the attractive force and repulsive force exerted by s_i in the direction j on the empty position p_k, the total $\overrightarrow{F_{ijk}}$ can now be expressed as:

$$\overrightarrow{F_{ijkA}} + \overrightarrow{F_{ijkR}} = \overrightarrow{F_{ijk}} \tag{1}$$

$\overrightarrow{F_{ijkA}}$ as defined previously is inversely proportional to the number of moves required by si to reach the empty position p_k while ignoring the Condition 1. Let be n the number of these moves and M the number of modules:

$$\overrightarrow{F_{ijkA}} = +\frac{M}{n} \tag{2}$$

$\overrightarrow{F_{ijkR}}$ is the existence of invalid path in a certain direction according to the Condition 1. It can be expressed as follows:

$$\overrightarrow{F_{ijkR}} = \begin{cases} 0, & \text{if from } j \text{ to } p_k \text{ is a valid path} \\ -M, & \text{if from } j \text{ to } p_k \text{ is an invalid path} \end{cases} \tag{3}$$

A positive value for $\overrightarrow{F_{ijk}}$ expresses the attractiveness of the motion of s_i in the direction j towards p_k, whereas a negative value for $\overrightarrow{F_{ijk}}$ expresses that such a motion would result in a blocking position for s_i and therefore should not be taken into consideration as a possible motion for s_i.

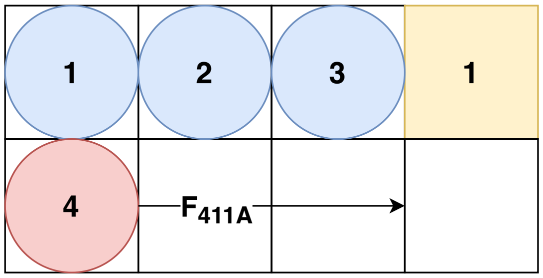

Fig. 4. Attractive force with 3 required movements

In Figures 4, 5, and 6, the red module is a free module and the blue ones are in the goal shape. The orange squares are empty positions in the goal shape that should be filled. Figures 4 and 5 present attractive forces with the variation of the number of motions needed to reach the target. Indeed, in Figure 4 the node $s4$ in the direction 1 needs 3 moves to reach the empty position p1 with a total number of 3 modules in the figure, therefore the length of the vector $\overrightarrow{F_{411A}}$ should equal to $4/3$. Whereas in Figure 5 the node $s2$ in the direction

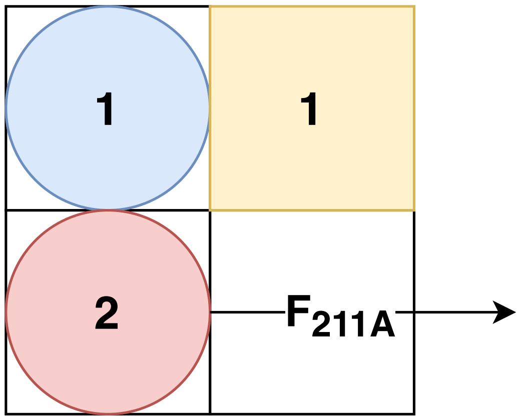

Fig. 5. Attractive force with 1 required movement

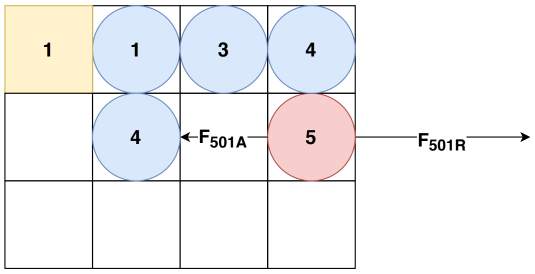

Fig. 6. VF with a repulsive force

1 needs 1 move to reach the empty position $p1$ with a total number of 2 modules in the figure, therefore the length of the vector $\overrightarrow{F_{211A}}$ should equal to $2/1$.

Figure 6 shows a case of attractive and repulsive forces. The node $s5$ in the direction 0 needs 4 moves to reach the empty position $p1$ if we ignore the Condition 1 with a total number of 5 modules in the figure, therefore the length of the vector $\overrightarrow{F_{501A}}$ should equal to $5/4$. Whereas the path to $p1$ following the direction 0 would cause a blocking position of $s5$ between $s2$ and $s4$, so the length of $\overrightarrow{F_{501R}}$ would be equal to 5 in the opposite direction of the motion. In this case, the sum of the forces $\overrightarrow{F_{501}}$ would be negative and ignored in the remaining phases of the algorithms. Theoretically, if Condition 1 is not valid the value of the repulsive force will be greater than the attractive one and the sum of forces would be negative, therefore in such a case, it is useless to calculate the attractive force. Let's define $\mathcal{L}$ as the $2D$ grid, $\mathcal{G}$ the goal shape we aim to obtain, $\mathcal{D}$ the set of possible directions, $\mathcal{P}$ the set of available modules in $\mathcal{L}$ and $\mathcal{T}$ the set of forces calculated by s.

2) Phase 2: Exploration: Inspired from [16], this phase's goal is to explore the whole system of modules while building a logical tree structure rooted at the leader module along the way. The algorithm starts with the initiation from the leader module and by using 3 types of messages:

1) Type 1 messages represent the discovery of new modules
2) Type 2 messages reply by confirming that a new module has been discovered and it is one of the children of the sender module
3) Type 3 messages reply by neglecting the fact that a module is to be discovered and notify the sender module that the destination module is already discovered

Also, to notice that type 2 messages are the only way to expand the tree structure of the system.

The leader module starts by specifying that it is now discovered and that it has no parent, and would finally transmit type 1 messages to each of its neighbors. Whenever a module receives a type 1 message it initializes its parent to be the sender module and replies by a type 2 message which notifies the parent that this node is now one of its children, and finally proceeds by sending type 1 messages to its neighbors (excluding the already known to be parent neighbor). If it is not the first time that it receives a type 1 message (the module is already discovered by another module) it sends back a reply message of type 3, which notifies the sender module that this module is not its child.

3) Phase 3: System Potential Movements Reporting: In this phase, that is run in parallel with the previous phase, the main idea is that whenever a module is discovered and all its subsequent children finish discovering their respective sub-trees (or if this module does not have any children), this module should report its sub-tree potential modules movements to its parent. The sub-tree potential module's movements of a given module are the potential modules movements of the tree structure that is rooted in this module. Thus, this process of sub-tree potential module movements reporting will emerge from the leaves to the leader module. Type 4 messages are now introduced. These messages are only passed from child to parent and they carry out a set value named that represents the sub-tree potential modules movements of the child node. After a module has been discovered, and it has sent out type 1 messages to all of its neighbors (potential children) and it received back all the reply messages (either type 2 or type 3), it would if all neighbors replied with type 3 messages or if it has no neighbors other than its parent (both cases represent the case of a leaf) send a type 4 message to its parent, with the set composed at the first phase. If one or more neighbors replied with a type 2 message (i.e., the node has at least one child) wait for all children to send their sub-tree potential modules movements, and only when they do, proceed by sending a type 4 message to its parent, which now carries its sub-tree potential modules movements that is the aggregation of all sub-tree potential modules movements received from its children with its own. This process keeps on repeating until eventually all sub-tree potential module movements would be transmitted from child to parent and the leader module would be transmitted from child to parent and the leader module would receive the final total system potential module movements. The Algorithm's part concerning the second phase is presented in Algorithm 1.

4) Centralized Hungarian Optimization: At this point, the leader has gathered all the system's potential movements of free modules towards empty positions in the goal structure. We define the cost of the algorithm as being the total number of moves by all the different free modules. The problem that needs to be solved now is to fill the highest number of empty positions while optimizing the cost of the algorithm: it can be formulated as an assignment problem which can be solved using the Hungarian Method [17]. The Hungarian method is one of the most popular polynomial-time algorithms for solving classical assignment problem. As stated in the original algorithm published in 1955 : "Assuming that the numerical scores are available for the performance of each of n persons on each of n jobs, the assignment problem is the quest for an assignment of persons to jobs so that the sum of the n scores so obtained is as large as possible" [18]. In this paper, the people are considered the modules, the jobs are the empty positions to be filled and the performance is the result of the Virtual Forces phase. Theoretically, the Kuhn-Munkres algorithm (Hungar-

Algorithm 1: System Potential Movements Reporting

```
$subtree\_movements\_sent \leftarrow false$
$subtree\_movements \leftarrow \emptyset$

if received message then
    switch message.type do
        case 1 do
            if $is\_discovered = true$ then
                ...
            else
                ...
                if neighbours.size = 0 AND
                 !subtree_movements_sent then
                    CHECK()
                       // Check for possible
                     moves
                else
                    foreach neighbour in neighbours
                     do
                        send type 1 message to
                         neighbour
                    end
                end
            end
        end
        case 2 do
            ...
        end
        case 3 do
            ...
            if neighbours.size = 0 AND
             !subtree_movements_sent then
                CHECK()
                      // Check for possible
                 moves
            end
        end
        case 4 do
            child ← find message.origin in children
            $child.subtree\_movements \leftarrow$
             $message.subtree\_movements$
            $subtree\_movements \leftarrow$
             $subtree\_movements \cup$
             $<\{message.subtree\_movements\}>$
            if neighbours.size = 0 AND
             !subtree_movements_sent then
                CHECK()
                      // Check for possible
                 moves
            end
        end
    end
end
```

ian Method) is guaranteed to reach the global optimal. When the leader has all the potential movements, it will launch the Kuhn-Munkres algorithm twice: once in each direction (CW and CCW). To prevent unexpected collisions between moving blocks, the approach needs to pick one direction at a time. The algorithm is searching for the direction with the overall lowest possible cost on the system. Having that, the leader will be able to identify which modules should be moved to which positions. When the leader moves certain blocks, it may open the way to other blocked modules to be able to move, so the whole algorithm may be repeated on several iterations until no further potential system movements are provided by the environment exploration and reporting phases. The overall algorithm of the proposed approach is presented in Algorithm 2.

Algorithm 2: Overall proposed approach

```
$\mathcal{S} \leftarrow \emptyset$
$best\_cost \leftarrow +\infty$
$best\_solution \leftarrow \emptyset$

$\mathcal{T} \leftarrow$ distributed algorithm of the first section
while $\mathcal{T}$ is not empty do
    foreach direction in $\{CW, CCW\}$ do
        $\mathcal{S} \leftarrow$ KUHN-MUNKRES(direction)
        if $best\_cost > \mathcal{S}.cost$ then
            $best\_cost \leftarrow \mathcal{S}$.cost
            $best\_solution \leftarrow \mathcal{S}$.solution
        end
        foreach $s$ in $\mathcal{S}$ do
            Scheduler.push(Movement Event, $s$)
        end
    end
    $\mathcal{T} \leftarrow$ distributed algorithm of the first section
end
```

V. Simulation Results

The proposed algorithm was implemented and evaluated using VisibleSim [2] a modular robots simulator. To study the performance of the proposed algorithm, four metrics were taken into account in our simulations: the number of iterations needed by the Kuhn-Munkres algorithm; the number of moves made by all the modules; the number of messages exchanged among the modules during the self-reconfiguration process and the number of events processed by the simulator.

To show that our approach works on different styles of shapes, we have conducted configurations of varying complexity. A video of these simulations is provided [1] to better illustrate the algorithm and the reconfiguration results. We considered two configuration scenarios while varying the total number of modules. The first scenario is to transform a straight chain line modular robot to a "UFC" [2] like shape

[1]Youtube video: $https://youtu.be/fwKvd_F3zDk$

[2]Acronym of our University: University of Franche Comté

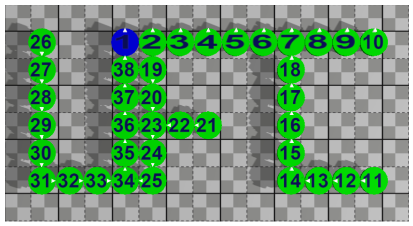
Fig. 7. The goal shape of UFC

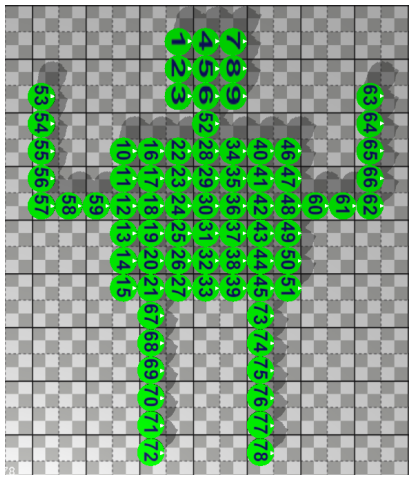
Fig. 8. The goal shape of a humanoid

and the second to transform it to a humanoid shape as shown in Figures 7 and 8 respectively.

The obtained results are presented in Figures 9 and 10. These figures show the number of iterations, events, messages and moves for the two configurations "UFC" like shape and humanoid shape while varying the total number of modules. In the two scenarios, we notice that the number of messages and moves to change the shape of a modular robot using our algorithm is linear according to the total number of modules in the system which is a very encouraging result comparing to the results presented in [2].

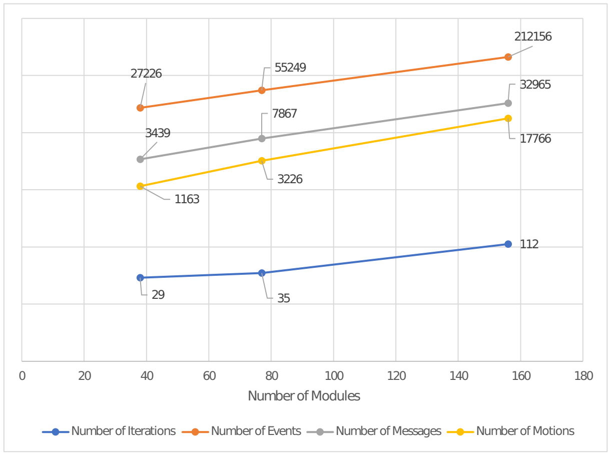

Fig. 9. Simulation Results of the "UFC"-like shape.

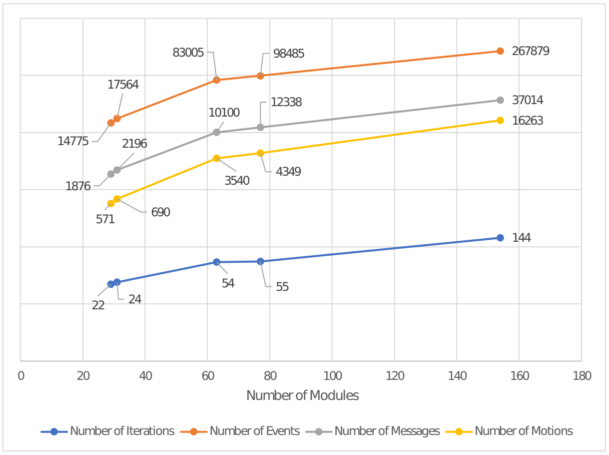

Fig. 10. Simulation Results of the humanoid shape.

VI. Conclusions

This paper presented an algorithm for a system of modular robots that is capable of self-reconfiguration based on a distributed virtual forces approach and a centralized optimization method. The proposed algorithm tries to move the maximum number of free modules to empty positions inside a goal shape over multiple iterations if required. The results show that the algorithm has a linear consumption of messages over the distributed phase and will increase the resources consumption with the number of iterations required which is related to how much the initial shape is different from the goal shape. In a future work, a possible improvement of the Kuhn-Munkres algorithm is to use the technique based on sparsity structure of the cost matrix as proposed in [19]. This may improve the complexity of our approach. Another future direction is to study a self-reconfiguration solution for 3D lattice modular robots.

Acknowledgment

The authors would like to acknowledge the National Council for Scientific Research of Lebanon (CNRS-L) for granting a doctoral fellowship to Edy Hourany.
This work has been supported by the EIPHI Graduate School (contract ANR-17-EURE-0002) and the ISITE-BFC project (ANR-15-IDEX-03).

References

[1] Tommaso Toffoli and Norman Margolus. Programmable matter: concepts and realization. *Physica. D, Nonlinear phenomena*, 47(1-2):263–272, 1991.

[2] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. A distributed self-assembly planning algorithm for modular robots. In *17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*, pages 550–558, 2018.

[3] Michael Park, Sachin Chitta, Alex Teichman, and Mark Yim. Automatic configuration recognition methods in modular robots. *The Int. Jour. of Robotics Research*, 27(3-4):403–421, March 2008.

[4] Y. Zou and Krishnendu Chakrabarty. Sensor deployment and target localization based on virtual forces. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 2, pages 1293–1303 vol.2, 2003.

[5] M. Yim, Ying Zhang, and D. Duff. Modular robots. *IEEE Spectrum*, 39(2):30–34, 2002.

[6] Jungwon Seo, Mark Yim, and Vijay Kumar. 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)*, pages 1016–1021. IEEE, 2013.

[7] André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263. IEEE, 2016.

[8] Chris Jones and Maja J Mataric. From local to global behavior in intelligent self-assembly. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 721–726. IEEE, 2003.

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

[10] Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. Distributed Self-reconfiguration using a Deterministic Autonomous Scaffolding Structure. Research Report 2843, UBFC, 2019.

[11] Mark Yim, Ying Zhang, John Lamping, and Eric Mao. Distributed Control for 3D Metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.

[12] K Stoy. Using cellular automata and gradients to control self-reconfiguration. *Robotics and Autonomous Systems*, 54(2):135–141, 2006.

[13] K. Stoy and R. Nagpal. Self-reconfiguration using directed growth. In *In Proc. 7th Int. Symp. on Distributed Autonomous Robotic Systems*, pages 1–10, 2004.

[14] Mohamad Moussa, Benoit Piranda, Abdallah Makhoul, and Julien Bourgeois. Cluster-based distributed self-reconfiguration algorithm for modular robots. In *35th International Conference on Advanced Information Networking and Applications (AINA 2021)*, may 2021.

[15] Jad Bassil, Mohamad Moussa, Abdallah Makhoul, Benoît Piranda, and Julien Bourgeois. Linear distributed clustering algorithm for modular robots based programmable matter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*, pages 3320–3325. IEEE, 2020.

[16] Joseph Assaker, Abdallah Makhoul, Julien Bourgeois, and Jacques Demerjian. A unique identifier assignment method for distributed modular robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*, pages 3304–3311. IEEE, 2020.

[17] Hong Cui, Jingjing Zhang, Chunfeng Cui, and Qinyu Chen. Solving large-scale assignment problems by kuhn-munkres algorithm. In *2nd Int. Conf. Advances Mech. Eng. Ind. Inform.(AMEII 2016)*, 2016.

[18] Harold W Kuhn. Kuhn hw. the hungarian method for the assignment problem. naval research logistic quaterly 1955; 2: 83-97. *Naval Research Logistic Quaterly*, 2:83–97, 1955.

[19] Hong Cui, Jingjing Zhang, Chunfeng Cui, and Qinyu Chen. Solving large-scale assignment problems by kuhn-munkres algorithm. 2016.

Acronyms

CCW	Counter Clock Wise. 2, 3, 6
CW	Clock Wise. 2, 3, 6
FCC	Face Centered Cubic. 2
VF	Virtual Forces. 4, 5
VFA	Virtual Force Algorithm. 1, 3