Distributed Configuration Recognition for 2D Lattice-Based Modular Robots

Jad Bassil, Benoît Piranda, Abdallah Makhoul, and Julien Bourgeois

Abstract Modular robots consist of multiple individual robotic modules that are connected to form various configurations. They are also equipped with locomotion capabilities so they can move to change their interconnections and self-reconfigure into different configurations. In this paper, we propose a distributed algorithm for configuration recognition designed for large-scale 2D lattice-based modular robots. The algorithm consists of searching the set of borders to be then transmitted to the modules in order to enable them to collectively discover an efficient global representation of their current configuration. To assess the performance of the proposed algorithm, we conducted simulations on three different configurations and compared it to a box-based approach. The results highlight significant reductions in communication complexity across all configurations. Additionally, our algorithm demonstrated improved memory efficiency in two out of three configurations.

1 Introduction

Self-reconfigurable modular robots are composed of multiple autonomous modules that can physically attach, detach, and rearrange themselves to change the global shape of the set in order to transform into different objects or to adapt to various tasks or environments. These robots offer high versatility and scalability, allowing them to change their shape and function based on the requirements of specific applications, such as exploration, rescue missions, and assembly tasks.

One potential application of self-reconfigurable modular robots is programmable matter [4]. Modular robot-based programmable matter envisions materials built as an assembly of thousands of micro-scale robotic modules. By leveraging the modularity and self-reconfigurability of these modules, programmable matter offers a new level of flexibility, enabling the creation of smart, responsive materials that

Université Marie et Louis Pasteur, CNRS, institut FEMTO-ST, France e-mail: fname.lname@univ-fcomte.fr

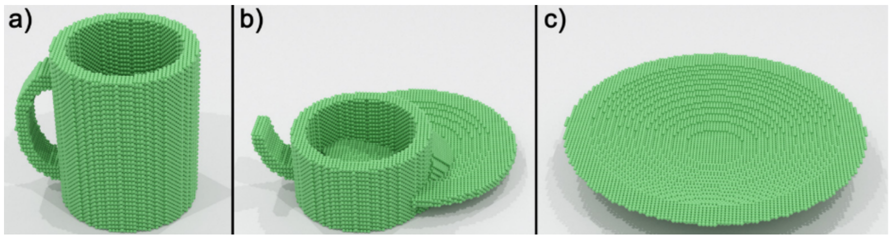


Fig. 1: A programmable matter system based on self-reconfigurable modular robots transforms from a mug shape (a) to a plate shape (c) passing through an intermediate configuration (b).

can reshape themselves to meet specific needs, whether for industrial, medical, or exploratory purposes. Objects built with programmable matter can possess computation at the core of the object itself, allowing the object to not only change its shape but also to perform computations and make decisions autonomously. Fig. 1 shows an example of an object made by tiny spherical module reconfiguring itself from a mug shaped configuration to a plate shaped configuration.

However, for these robots to operate effectively in a self-organizing manner, they must be able to recognize and understand their current configuration. Configuration recognition is essential for tasks like self-reconfiguration, fault tolerance, and global coordination. Configuration recognition consists of giving every individual module the awareness about the arrangement of the modules in the whole configuration. A module should be able to efficiently localize itself and determine whether another module, not directly connected to it, is present at a given position or not. This allow the modules to make informed decisions and collaborate efficiently with one another. For instance, during distributed self-reconfiguration planning, being aware of the current configuration allows a local calculation of the difference between the current and goal configurations which facilitates planning.

This paper presents a distributed algorithm for large-scale, 2D lattice-based modular robots to recognize their current configuration. Our approach enables modules to collaboratively determine their configuration by detecting and sharing border information. Once borders are identified and stored, modules can employ the even-odd ray casting method [8] to verify the presence of modules at specific positions. The algorithm involves a message-passing border traversal process to establish an efficient representation, which is then shared with all modules.

The paper is organized as follows: Section 2 reviews related work. Section 3 details the distributed border search algorithm, including system assumptions and complexity analysis. Section 4 presents a simulation-based evaluation and comparison with a box-based representation algorithm. Finally, Section 5 concludes the paper and outlines future work.

2 Related Work

Matching and mapping a given configuration to a library of known configurations, has been studied in [2, 9, 7]. A common approach involves a discovery phase where the current configuration is represented as a connectivity graph. Nodes in this graph correspond to individual modules, and edges represent connections between them. This graph is then matched against a known configuration to identify corresponding modules.

Butler et al. [5] address the matching problem using a distributed goal recognition algorithm that determines whether a configuration matches a target shape without requiring full discovery. Similarly, Baca et al. [2] present a distributed real-time algorithm for configuration discovery, enabling modules to detect each other and construct a connectivity graph using wireless infrared communication.

However, connectivity graphs face scalability issues as their size increases proportionally to the number of modules. This becomes particularly problematic in high-density programmable matter systems composed of a large number of microscale robots. Lattice-based modular robots can alleviate these challenges by exploiting geometric information to construct more compact shape representations.

In [10, 6] the authors propose transforming a CAD model into overlapping ”bricks” to simplify the process for modules to determine their positions relative to the goal configuration, facilitating self-reconfiguration. Meanwhile,[12] introduces the use of Constructive Solid Geometry (CSG) trees. This model, derived from the field of image synthesis, represents shapes using a hierarchical tree structure: leaves contain basic geometric primitives, and internal nodes define geometric transformations and combination operations (e.g., union, intersection, or difference), with the root representing the final shape. These methods rely on a centralized computation to encode the goal shape and transmit it to the modules for self-reconfiguration.

In [3] a distributed 3D shape recognition algorithm for lattice-based modular robots is proposed. It consists of finding a set of overlapping boxes that cover the whole configuration through message-passing between directly attached modules. While this method provides an effective representation of the current shape, its memory requirements grow for configurations with many irregular border features. Each stair-like corner along the boundary requires an additional box, resulting in an increase in memory usage which can become prohibitive in resource-constrained modules.

In this paper, we propose a distributed algorithm designed for large-scale 2D lattice-based modular robots. Our approach enables modules to collectively discover the overall shape of their configuration represented as a set of borders. The goal is to reduce memory usage relative to the boxes approach [3] in the case of irregular configurations by allowing non-straight segments in the border representations.

3 Algorithm Description

Our robots are placed in a square 2D grid, so each cell has 4 neighbors connected by its border. Since we're considering 4 connections, modules are part of the border if they have at least one empty direct neighbor cell. Two types of border can appear: external borders and internal borders, which indicate a hole in the configuration.

In this section, we describe the distributed algorithm for determining the set of borders in a given configuration. After the borders are identified, their representation can be compressed and stored in each module's memory. This data allows each module to locally verify whether a given position is part of the current configuration. The verification is performed using the ray-casting algorithm [8], which counts how many times a ray, originating from the given position and extending in a predetermined direction, intersects with the borders. A position outside the configuration will result in an even number of ray-border intersections. A position inside the configuration will result in an odd number of intersections.

In the next sections, a distributed algorithm is described to search for borders. The system assumptions are given in section 3.1. The algorithm description consisting of messages tracing each border to determine an efficient border representation is provided in section 3.2. A complexity analysis in terms of time and communication is provided in section 3.3.

3.1 System assumptions

In this work, it is assumed that the modular robot system has the following properties:

- Modules are positioned on the cells of a regular 2D square lattice, where each module has local knowledge of its coordinates and orientation.
- Communication is restricted to neighbor-to-neighbor message-passing. A module can send messages to its adjacent neighbors via one of its connectors, and the receiving module can identify the direction of the sender.
- Each module's perspective is limited to its immediate neighborhood, with computations performed locally based solely on information obtained through message-passing within the neighborhood.
- Each module is aware of its direct connections, specifically identifying which of its borders are connected to neighboring modules and which are not.
- The configuration is assumed to remain fixed and fully connected throughout the process. No modules are added or removed during the algorithm's execution.

Algorithm 1: Borders search

```
input : $P$  // Module's position
    $directions \leftarrow \{FRONT, RIGHT, BACK, LEFT\}$ // Possible movement directions
    $dirInit$ // Key-value pairs where keys corresponds to received direction and values to initiators positions
output: $border$ // The border's information. A set of corners' coordinates.
Function $isInitiator()$:
    return $empty(x, y-1) \wedge (empty(x-1, y) \vee \neg empty(x+1, y-1)))$
Function $getNextDir(prevDir \in directions)$:
    $k \leftarrow (prevDir + 3) \bmod 4$
    for $i \in [0, 3]$ do
        $Q \leftarrow P + nextPositionInDir(P, k)$ // Q gets the position of the neighbor in direction k
        if $(\neg empty(Q))$ then
            return $k$
        $k \leftarrow (k+1) \bmod 4$
Initialization:
    if $isInitiator()$ then
        $nextDir \leftarrow getNextDir(FRONT)$
        send BORDER_SEARCH_MSG($nextDir, border \cup P, P$) to $nextDir$
Msg Handler $BORDER\_SEARCH\_MSG(prevDir, border, initiatorPos)$:
    if $(P = initiatorPos)$ then
        // border found
        return
    if $(prevDir \in dirInit \wedge dirInit[prevDir] < initiatorPos)$ then
        return // already received a message from $prevDir$
    $nextDir \leftarrow getNextDir(prevDir)$
    if $(nextDir \neq prevDir \wedge isOnBorder())$ then
        if $(|border| > 2)$ then
            $alt \leftarrow$ altitude to the hypotenuse of the triangle formed by $P$, $border[|border|-1]$ and $border[|border|-2]$
            if $(collinear(P, border[|border|-2], border[|border|-3]) \wedge alt < \sqrt{2})$ then
                $border \leftarrow border \setminus \{border[|border|-1], border[|border|-2]\}$
        // Is a corner
        $border \leftarrow border \cup P$
    $dirInit[prevDir] \leftarrow initiatorPos$
    send BORDER_SEARCH_MSG($nextDir$, $border$, $initiatorPos$) to $nextDir$
```

3.2 Borders Definition

Algorithm 1 describes a distributed algorithm to search for borders. The algorithm is initiated by modules satisfying the following condition (cf. Algorithm 1 l. 1-2):

$$initiator(x, y) = empty(x, y-1) \wedge [empty(x-1, y) \vee \neg empty(x+1, y-1)] \quad (1)$$

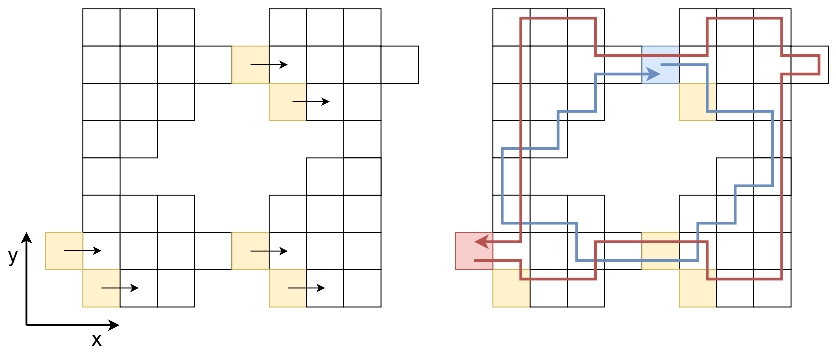


Fig. 2: An example of initiators and two paths following borders. Initiators are shown in yellow. The red and blue arrows shows the only two completed paths starting and ending at the red and blue initiators.

This condition ensures that all borders internal and external will have at least one initiator. All initiators will initiate a message (*BORDER_SEARCH_MSG*) that will follow the border (cf. Algorithm 1 lines 11-13). Fig. 2 (left) shows all initiators modules in yellow. The following of the border is done according to the *getNextDir* functions that returns the next direction to follow (cf. Algorithm 1 lines 3-9). Each module records the initiator's position for every received direction to track ongoing border searches and to allow participation in multiple borders.

The message will proceed along its path if the receiving module has not yet received the message from the sender's direction, or if the initiator's position is smaller than the previously recorded initiator position. A position (x_1, y_1) is defined as less than another position (x_2, y_2) if and only if the eq.2 is verified.

$$(x_1, y_1) < (x_2, y_2) \iff [(x_1 < x_2) \vee ((x_1 = x_2) \wedge (y_1 < y_2))] \tag{2}$$

This ordering prioritizes the $x-$coordinates and uses the y-coordinates as a tiebreaker when x-coordinates are equal. This guarantees that only the initiator with the minimum position receives the message after a complete traversal of the border. Consequently, this marks the termination of the border-following process, ensuring that all border information is collected by that module (cf. Algorithm 1 lines 15-18). The initiator can then broadcast the border representation to all the system.

The border information is expressed by the sequence of corners coordinates, each encoded in two bytes, as shown in Algorithm 1 lines 20-24. These corners are found through the message-passing border traversal. At each change in message direction, the position P_m of module m is added to the border if the following conditions are satisfied (cf. Algorithm 1 line 22):

1. P_m and the two positions before the last added one P_{m-2} and P_{m-3} are not collinear.

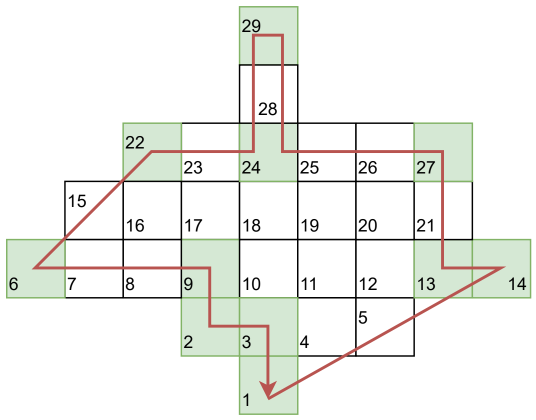


Fig. 3: An example of a border found using Algorithm 1. The green cells represent modules whose coordinates are border's corners. The border segments are shown in red. The two Segments formed by modules 1,5 and 14 and 22, 15 and 6 satisfies the collinearity conditions. Note that a segment is not formed between 9,2,3 and 1 since one more corner is required.

2. There should be no empty cell between the occupied cells in a non-straight border segment. This condition holds if an empty cell cannot fit in the right triangle formed by P_m, P_{m-1} and P_{m-2}. It can be verified by checking that the length of the altitude to the hypotenuse is less than $\sqrt{2}$. The altitude can be calculated using the formula $\frac{a \times b}{c}$ where a and b are the triangle's legs, and c is the hypotenuse length.

A border example found by the described algorithm can be seen in Fig. 3.

3.3 Complexity analysis

The number of messages needed to follow and identify the borders is $O(ib)$, where i represents the number of initiators, and b denotes the number of modules on the borders. Additionally, $O(n)$ messages will be required to broadcast the borders representations to all the modules in the configuration. Since both i and n are bounded by n, the communication complexity can be expressed as $O(n^2)$.

As for the time complexity, since the messages for each of the borders searches are sent in parallel, the time complexity of finding the borders representation is $O(n)$ where n is the largest possible border length. In order to broadcast the representation to all modules, $O(d)$ time where d is the diameter of the configuration is required. Since $d \leq n$, the overall time complexity can be expressed as $O(n)$.

4 Evaluation

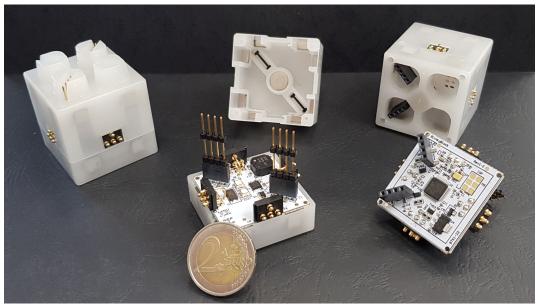
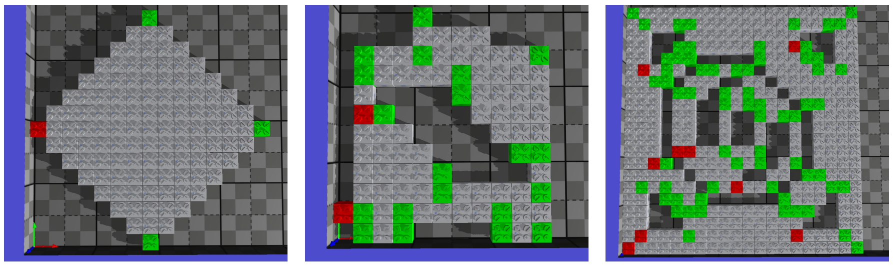


Fig. 4: *Blinky Blocks*

We implemented the algorithm in *VisibleSim* [11], a discrete-event simulator designed for distributed modular robotic systems, with support for *Blinky Blocks* [1]. The *Blinky Blocks* system comprises centimeter-scale blocks connected via magnets in a square cubic lattice configuration, as illustrated in Fig. 4. Each block is a cube approximately 40 mm in size, equipped with processing, storage, and communication capabilities. Communication between *Blinky Blocks* is facilitated through serial links with directly connected neighbors, using packets with a payload size of 227 B.



(a) Diamond (b) One hole (c) Multiple holes

Fig. 5: The three configurations captured from *VisibleSim*. Green modules represent corners and the red modules are the terminal borders initiators.

We have done the comparison on three different configurations shown in Fig. 5 with different characteristics:

1. Diamond: A diamond-shaped configuration composed of 133 modules, featuring stair-like borders on all sides.

2. Single hole: An 82-module configuration with a single hole that is not aligned with the axes. The external borders of this configuration exhibit some irregularities.
3. Multiple Holes: A larger configuration comprising 337 modules. It has a regular external border that is aligned with the axes and multiple internal holes of various sizes and shapes.

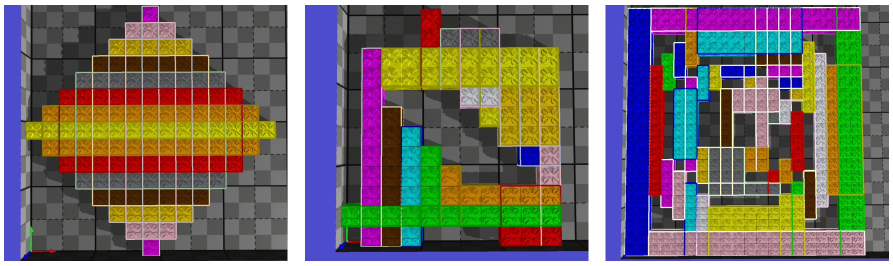

(a) Diamond (b) One hole (c) Multiple holes

Fig. 6: The three configurations captured from *VisibleSim* showing the found boxes in different colors.

Next, we compare our border-search algorithm with the axis-aligned overlapping boxes algorithm presented in [3]. This algorithm involves:

1. Projecting messages backward to determine the length of each segment.
2. Expanding segments as much as possible in the appropriate directions to form maximal rectangles.
3. Stacking rectangles vertically to create overlapping 3D boxes, provided the bottom rectangle can fit within the one above it. For a more in-depth explanation, please refer to the original paper.

Fig. 6 shows the boxes found on the tested configurations. Each box can be encoded in 4 bytes to store the coordinates of two diagonally opposite corners.

Fig. 7 compares both methods in terms of communication, time and memory needed to store the representation. A comparison of the two methods in Fig. 7a reveals that the border method requires significantly fewer messages than the boxes method to determine the configuration representation. This is because the border method only exchanges messages along the borders, while the boxes method requires message exchange across the entire configuration to find boxes dimensions.

However, as shown in Fig. 7b, the border method is more time-consuming. This is due to the sequential nature of border tracing, where messages propagate sequentially from an initiator until they return to the same initiator. In contrast, the boxes method uses parallel messages, enabling the simultaneous identification and expansion of segments to determine box dimensions. The difference is particularly pronounced in diamond configurations due to the stair-like borders, which increase the

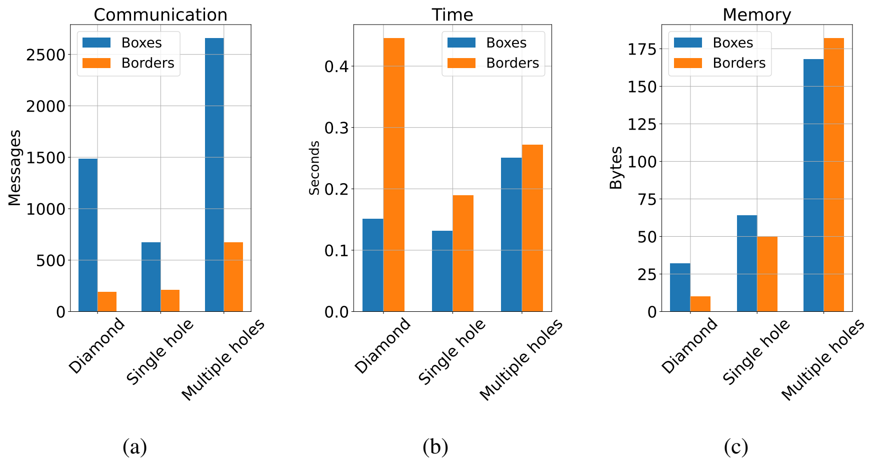


Fig. 7: A comparison between the border and the boxes for configuration recognition. (a) compares the number of exchanged messages, (b) compares the time for the algorithm execution and (c) compares the memory needed for the representation.

distance messages must travel as modules can only communicate with directly attached neighbors.

As for the memory needed to store the configuration representation, it can be seen in Fig. 7c that the border method is more memory efficient in the diamond and single border configurations. However, the box method is more memory efficient for the multiple holes configuration. The reason is that a single box, encoded using 4 bytes, can effectively cover a region between two borders. Conversely, the border-based method might requires multiple corner points to represent the segments of these borders, resulting in increased memory usage. For example, the leftmost blue box in Fig. 6c encompasses 6 border corners requiring 12 bytes, as seen in Fig. 5c. Therefore, in configurations with fewer aligned holes, the border method demonstrates better memory efficiency. However, for configurations with numerous aligned holes where boxes can cover regions connected to multiple borders, the box method exhibits better efficiency.

5 Conclusion

This paper presents a distributed algorithm for 2D lattice-based modular robots to efficiently recognize their current configuration. The algorithm enables modules to collaboratively determine a compact representation of the configuration, allowing them to locate themselves and verify the presence of distant modules at any cells positions. By identifying internal and external borders, modules can collectively construct a global configuration representation.

The algorithm is evaluated in simulation on *Blinky Blocks* modules and compared with another algorithm that builds a representation based on overlapping boxes. The results indicate significant improvements in communication complexity for the three tested configuration and memory usage for two out of three tested configurations. Nevertheless, the box-based method presents faster execution times due to its use of a higher degree of message concurrency.

Future research directions include extending the border-based representation algorithm to 3D configurations and conducting a thorough comparative analysis with the box-based approach. For instance, new methods combining the border-based and boxes-based representations can be explored to find boxes that are not necessarily aligned with the axes or even polyhedrons. Additionally, considering the dynamic nature of self-reconfigurable modular robots, where modules can change positions and consequently alter the configuration, it is essential to develop dynamic configuration recognition algorithms capable of maintaining real-time representations of the configuration during self-reconfiguration processes.

Acknowledgment

This work has been supported by the EIPHI Graduate School (contract ”ANR-17-EURE-0002”).

References

1. Blinky blocks: Programmable matter. https://www.programmable-matter.com/technology/blinky-blocks
2. Baca, J., Woosley, B., Dasgupta, P., Nelson, C.: Real-time distributed configuration discovery of modular self-reconfigurable robots. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 1919–1924. No. June, IEEE (may 2015). https://doi.org/10.1109/ICRA.2015.7139449, http://ieeexplore.ieee.org/document/7139449/
3. Bassil, J., Yaacoub, J.P.A., Piranda, B., Makhoul, A., Bourgeois, J.: Distributed shape recognition algorithm for lattice-based modular robots. In: 2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). pp. 85–91 (2023). https://doi.org/10.1109/MRS60187.2023.10416786
4. Bourgeois, J., Piranda, B., Naz, A., Boillot, N., Mabed, H., Dhoutaut, D., Tucci, T., Lakhlef, H.: Programmable matter as a cyber-physical conjugation. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 002942–002947. IEEE (10 2016). https://doi.org/10.1109/SMC.2016.7844687, https://hal.archives-ouvertes.fr/hal-02140347 http://ieeexplore.ieee.org/document/7844687/
5. Butler, Z., Fitch, R., Rus, D., Yuhang Wang: Distributed goal recognition algorithms for modular robots. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). vol. 1, pp. 110–116. IEEE (2002). https://doi.org/10.1109/ROBOT.2002.1013347, http://ieeexplore.ieee.org/document/1013347/

6. Fitch, R., Butler, Z.: Million module march: Scalable locomotion for large self-reconfiguring robots. International Journal of Robotics Research **27**(3-4), 331–343 (2008). https://doi.org/10.1177/0278364907085097
7. Liu, C., Yim, M.: Configuration Recognition with Distributed Information for Modular Robots. In: Springer Proceedings in Advanced Robotics, vol. 10, pp. 967–983. springer (2020). https://doi.org/10.1007/978-3-030-28619-4-65, http://link.springer.com/10.1007/978-3-030-28619-4-65
8. Shimrat, M.: Algorithm 112: Position of point relative to polygon. Commun. ACM **5**(8), 434 (Aug 1962). https://doi.org/10.1145/368637.368653, https://doi.org/10.1145/368637.368653
9. Shiu, M.C., Fu, L.C., Chia, Y.J.: Graph isomorphism testing method in a self-recognition velcro strap modular robot. In: 2010 5th IEEE Conference on Industrial Electronics and Applications. pp. 222–227. IEEE (2010)
10. Stoy, K., Nagpal, R.: Self-Reconfiguration Using Directed Growth. In: Distributed Autonomous Robotic Systems 6, pp. 3–12. Springer Japan, Tokyo (2008). https://doi.org/10.1007/978-4-431-35873-2-1, http://link.springer.com/10.1007/978-4-431-35873-2-1
11. Thalamy, P., Piranda, B., Naz, A., Bourgeois, J.: Visiblesim: A behavioral simulation framework for lattice modular robots. Robotics and Autonomous Systems p. 103913 (2021). https://doi.org/https://doi.org/10.1016/j.robot.2021.103913, https://www.sciencedirect.com/science/article/pii/S0921889021001986
12. Tucci, T., Piranda, B., Bourgeois, J.: Efficient scene encoding for programmable matter self-reconfiguration algorithms. Proceedings of the ACM Symposium on Applied Computing **Part F1280**, 256–261 (2017). https://doi.org/10.1145/3019612.3019706