2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Linear Distributed Clustering Algorithm for Modular Robots Based Programmable Matter

Jad Bassil†, Mohamad Moussa†, Abdallah Makhoul†, Benoît Piranda† and Julien Bourgeois†

***Abstract*— Modular robots are defined as autonomous kinematic machines with variable morphology. They are composed of several thousands or even millions of modules which are able to coordinate in order to behave intelligently. Clustering the modules in modular robots has many benefits, including scalability, energy-efficiency, reducing communication delay and improving the self-configuration processes that focuses on finding a sequence of reconfiguration actions to convert robots from an initial configuration to a goal one. The main idea is to divide the nodes in an initial shape into some clusters based on the final goal shape in order to reduce the time complexity and enhance the self-reconfiguration tasks. In this paper, we propose a robust clustering approach based on a distributed density-cut graph algorithm to divide the networks into a predefined number of clusters based on the final goal shape. The result is an algorithm with linear complexity that scales to large modular robot systems. We implement and demonstrate our algorithm on a real *Blinky Blocks* system and evaluate it in simulation on networks of up to 30,000 modules.**

I. INTRODUCTION

The vision for programmable matter or modular robotic systems, is to create a material which can be reprogrammed to have different shapes and to change its physical properties on demand [7]. Programmable matter could be deployed in different domains while promising to have a variety of applications in construction, surgery, environmental science, space exploration, etc. [1, 15]. Examples of exciting future applications are robotic ensembles monitoring hostile environments (e.g., nuclear), delivering drugs in the human body, educational robots and a new set of robotic toys [11, 3, 1].

There are various ways to implement programmable matter. One is to build it as a huge modular self-reconfigurable robot composed of a large set of micro-robots (particles). These particles can have different forms (spherical, cubic, etc.). They must be able to stick to each other and move around their neighbours. The main task of a modular robots system is to reconfigure its shape in order to accommodate for variable conditions that need to be met in order to complete a given final goal. For example, one can program the robots so that, starting from an initial configuration without any holes, they can self-reconfigure into a line containing all the robots, without ever breaking the connectivity of the system [17, 6]. However, it has been shown and proven [7, 5] that the number of all possible configurations of modular robots increases drastically as the number of modules increases, so it is challenging to provide an efficient algorithm for self-reconfiguration. One of the solutions that may simplify the self-reconfiguration problem is to cluster the modular robots system. Assembling the particles into clusters and accomplishing tasks in cluster-based approaches can increase the efficiency in term of execution time and energy consumption. Therefore, clustering the particles in modular robots has many benefits, including scalability, energy-efficiency, and enhancing routing and communications. In fact, a group of nodes form the cluster and the local interactions between cluster members are handled by a leader cluster head (CH). The role of a CH is to schedule activities in the cluster. Modules would care only for selecting its CH and would not be affected by changes at inter-cluster level, so cluster members communicate with CH and data is collected and aggregated by CH, thus reducing the scope of inter-cluster interactions to CHs only and avoid passing of redundant messages among the modules.

Our objective in this paper is to propose a linear distributed clustering algorithm in order to simplify the transformation of the modular robots system from an initial shape to a goal shape as shown in Figure 1. The number of clusters is defined based on the goal shape then a clustering approach based on the density-cut graph algorithm is proposed in order to divide the particles into clusters. For example, in Figure 1 the initial filled shape is divided into 5 clusters, then each cluster, given its goal shape, is self-reconfigured to form its part of the overall goal shape.

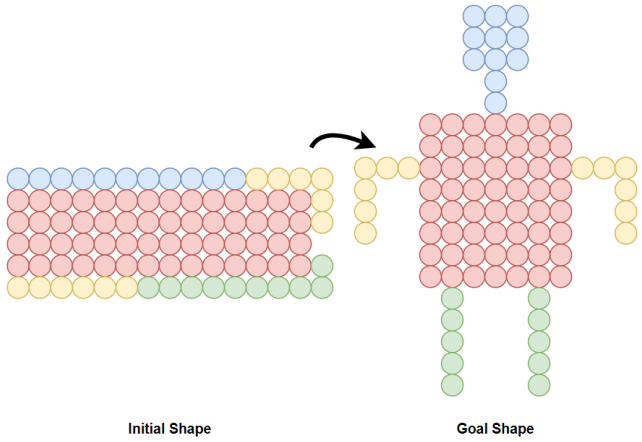


Fig. 1: Clustering and self-reconfiguration.

It is important to have an efficient encoding of the goal shape description, and which will ultimately be useful in order to determine the number of clusters. Tucci et al. (2017),

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

†All authors are with Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, 1 cours Leprince-Ringuet, 25200, Montbéliard, France. {first}.{last}@femto-st.fr

proposed in [14] an efficient scene encoding method based on Constructive Solid Geometry (CSG) [12] that defines a tree of objects that can be combined to form the final scene as shown in Figure 2. The tree can be represented in a compact form and can be efficiently stored in each module. The number of clusters can be determined by the number of objects on a given level of the CSG tree. In the example of the mug shown in Figure 2, the number of clusters can be two which is the number of objects at level 1 of the tree.

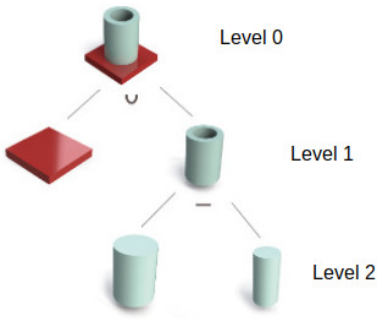


Fig. 2: A Mug represented in CSG tree [14]

To the best of our knowledge, we are the first that propose a clustering approach for modular robots. Similar strategies have been studied and many clustering protocols have been proposed in the wireless sensor networks domain [16, 8, 2, 9]. However, these approaches are not suitable for modular robots due to their specific constraints. In this paper, we propose a clustering algorithm based on the graph clustering algorithm density cut (DCut) [13]. The DCut algorithm allows partitioning a graph into multiple densely tight-knit clusters directly. We adapt this algoritm and propose a distributed version in order to fit the case of distributed modular robots and by modifying the conditions of the similarity function and the edge weight. To show the effectiveness of our approach, we implement and demonstrate it on a real Blinky Blocks system and evaluate it in simulation on networks of up to 30,000 modules.

The remainder of this paper is organized as follows. The description of the DCut algorithm and the problem formulation are presented in Section II. Section III describes the clustering algorithm for modular robots. The simulation results are presented in Section IV. Section V concludes the paper and gives some directions for future work.

II. Density cut based clustering

A. Background and DCut Algorithm

Shao et al. (2016), proposed the DCut algorithm in [13], an approach for graph clustering based on density, the idea is to consider the graph clustering as a density-cut problem such that the nodes in same cluster are densely connected and those between clusters are sparsely connected. In fact, their proposal was a centralized graph clustering algorithm applied on graph data, while in this work we seek to adopt it in a fully distributed fashion on the modular robots based programmable matter.

A good partitioning should consider the similarity of nodes in groups, so their proposal is to compute similarities of nodes inside and between graph clusters by building the density-connected tree (DCT), where any two close nodes with high similarity are densely linked together. The built tree captures the density connectivity of nodes so nodes with same topological attributes are strongly linked together while the weight of an edge that links two nodes of two lightly connected component is low. So, the idea is to cut such edge to form two distinct components hence two clusters. An example is shown in Figure 3 where the cut is made on the weak edge that connects node 6 with node 13 forming two dense clusters.

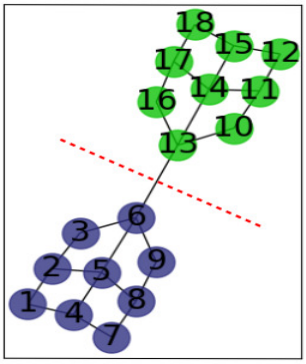


Fig. 3: Two clusters formed by the DCut algorithm

B. Problem formulation

Similarly to the algorithm proposed in [13], we consider that the system is modeled as an undirected weighted graph $G = (V, E, W)$ where V is the set of nodes representing the modules and E is the set of edges representing connections between modules. $e = \{u, v\} \in E$ indicates a connection between module u and module v. W is the set of weights associated to edges. Before defining the similarity measure, some relevant definitions are needed:

Definition 1. (Neighborhood of a node) Given an undirected unweighted graph $G = (V, E, W)$ the neighborhood of a node u is the set:

$$\Gamma(u) = \{v \in V | \{u, v\} \in E\} \cup \{u\}$$

Definition 2. (Jaccard coefficient) Given a graph $G = (V, E, W)$, the Jaccard coefficient of two adjacent nodes u and v is defined as:

$$\rho(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

The Jaccard coefficient is used to quantify two nodes' local topology similarity, the more common neighbors two nodes have, the more similar they are, as the Jaccard coefficient normalizes the number of common neighbors by the sum of the size of the two neighborhoods, it captures the local connectivity density of any two adjacent nodes in a graph.

Definition 3. (Anchors) Given a geometrical shape I, the minimum bounding box B is the box surrounding I with

minimum volume. The set of anchors A is defined as the set of coordinates of the minimum bounding box's corners C_B:

$$A = \{(x_c, y_c, z_c) \mid (x_c, y_c, z_c) \in \mathbb{Z} \text{ and } c \in C_B\}$$

A can be calculated by selecting the different minimum and maximum combinations while varying on the three axes x, y and z so a total of 8 points at the corners of B are defined, i.e. (min_x, min_y, min_z), (min_x, min_y, max_z), ...,(max_x, max_y, max_z).

Definition 4. (Edge weight) Given two neighboring modules u and v, the similarity of the two modules u and v in the graph G, is defined as:

$$w(u,v) = \frac{\rho(u,v)}{min(dist(u,A), dist(v,A))}$$

s.t:

$$dist(u,A) = min\{dist(u,a) \mid a \in A\},$$

where *dist* represents the euclidean distance.

C. DCT Partitioning

In order to find a good graph partitioning, as already mentioned, we consider graph clustering as a density-cut problem. With DCT capturing node density connectivity in a graph where nodes with similar topological attributes and intense connections are densely linked together, whereas component-to-component connections are sparsely connected (the similarity of the two nodes connecting the two components is relatively small), a solution is to directly cut the edges of the tree to obtain dense-based clusters. Formally, a modern density-based model is proposed, rather than computing the value of total (or normalized) weights of the edges linking the two components, the following measure computes its density connection between the two partitions according to the DCT tree. This measure is called DCut:

$$DCut(C1, C2) = \frac{d(C1,C2)}{min(|C1|, |C2|)}$$

Where $C1$, $C2$ are the two partitions, $d(C1,C2)$ means the corresponding weight of the edge connecting the two partitions. The term of $min(|C1|, |C2|)$ is used to avoid the bias towards splitting small sets of nodes. Since connection between nodes in the DCT is acyclic, each edge links two components of a graph. Hence, the bi-partitioning of the graph in terms of density can be effectively accomplished by cutting the weakest edge within the DCT. To obtain k clusters, the DCut recursively performs $k-1$ cuts on the DCT. The number of clusters k can be arbitrary. However, to obtain balanced clusters, k should be a power of 2 number.

III. APPLICATION TO MODULAR ROBOTS

The proposed algorithm is convenient to dense modular robots programmable matter since it characterizes the density of any two adjacent nodes in local fashion so no global view of the system is required. It creates a spanning-tree which can be used in tasks such as inter-cluster communication, intra-cluster communication, data aggregation, moving modules from one cluster to another, etc. Furthermore, it is easy to implement and it can be suitable for distributed systems such as programmable matter's modular robots.

A. System Assumptions

Several assumptions shall be presented before starting the development of the algorithm:

- The goal shape is known and can be efficiently encoded and stored in each module as explained in [14].
- All nodes possess unique identifiers.
- Modules are placed in cells of a regular lattice.
- Only neighbor-to-neighbor communications are possible. A module sends a message to its adjacent neighbors via one of its connectors. The receiver can reply by sending a message via the connector that received the message.
- A module is aware of its direct connections (i.e which borders are connected to other modules and which ones are not).
- We consider that the configuration is fixed and always connected during the process, i.e. no new modules are connected or disconnected during the execution of the algorithm.

B. Clustering Algorithm

In this section, we present the distributed clustering algorithm. Basically the algorithm 1 follows three main steps:

1) Compute the edges data.
2) Build the DCT tree [4] using a distributed algorithm.
3) Find the minimum edge where the cut will take place.
4) Repartition the segmented DCT recursively until k clusters are obtained.

Consider $G = (V, E, W)$ as the initial graph. The first step consists of each node computing the edges weights (see Algorithm 1 , lines 1-3). In order to compute it two messages are required: *REQUEST_DENSITY* and *SEND_DENSITY*, noting that whenever there is a connection between two nodes the computation is done on one node having the lower *id* than the one requesting, then sent back within the reply message, thus reducing the computation complexity on the nodes. GHS [4] algorithm (see Algorithm 1 , line 4) will end up by creating a tree structure rooted at one node in a fully distributed fashion, then the algorithm will operate by finding the minimum edge where the cut will take place in order to partition the graph into two clusters, and recursively repartition the segmented tree until k components of the graph are obtained. Firstly, the root will compute the number of nodes in the system. Once computed, it assigns itself as a cluster head and decrements the number of clusters by one considering that the complete tree represents a single cluster and triggers the final step (see Algorithm 1 , lines 6-8) by calling the $Cut()$ function. Algorithm 2 presents the message handler, where the messages are represented as: **MESSAGE_NAME** (**DATA**) considering that each message is received by module M_i from M_j.

In this approach, the *GHS* distributed algorithm for minimum weight spanning trees is used to build the DCT, where each module represents a node of the graph and the edge

Algorithm 1: Clustering algorithm pseudo-code

```
   CHid
   subTreeSize
   neighbours ← neighbours ∪ {node's id}
   anchors // nodes at the extremities
   p_Mi // position
   sim // similarity measure
   nbWaitedAnswers
   isCH // cluster head or not
   edges // set of edges
   children // set of children
   k // number of clusters
   DCut // temporary variable to store
      the received cut value
   cutAt // node where the cut will occur
   T // DCT Tree
1  foreach Nid in neighbours do
2  |  if Nid < id then
3  |  |  send REQUEST_DENSITY(neighbours) to Nid
4  T ← GHS(G)
5  Cut()
6  Function Cut():
7  |  foreach child in children do
8  |  |  DCut_Data ← (CHid, subTreeSize)
9  |  |  send REQUEST_DCUT(DCut_Data) to child
```

between nodes of the graph represent a communication link between two modules. When GHS terminates, the resulted tree is rooted at two core nodes. One can choose the node with minimum id to be the root of the DCT. The root will then launch the search for the minimum weight edge where the cut will happen. To do so, each module calculate its *DCut* value and maintain the minimum which will be returned back to the root, then the root will initiate the clustering by cutting the graph at the *cutAt* module to form the first two clusters with the root and the *cutAt* modules as CHs. Then the number of the remaining cuts is calculated by dividing the number of clusters by two and the ceil is given to the CH having the more modules in its cluster and the floor to the other (see Algorithm 2, lines 32-37). If the number of remaining cuts is bigger than 0 the CH will recursively relaunch the search for a new cut until k clusters are obtained (see Algorithm 2, lines 42-45).

C. Complexity Analysis

We note $m = |E|$ as the number of edges connections in the graph and n the number of modules. The number of messages exchanged depends on the topology. However we assume that all modules have m connections. During the first step and in order to compute all the weights of the edges, the number of messages exchanged is $O(2m)$. The second step of building the tree using the GHS algorithm requires $O(n\log n)$ in terms of time complexity and a message complexity of $O(m\log n)$. The final step consisting of finding the minimum edge, cutting the graph and distributing the different clusters requires $O(k\log n)$ messages since after a cut, the next cuts are searched in the resulted partitions. Therefore, the total number of messages exchanged during the execution is $O(2m+m\log n+k\log n)$.

IV. Simulation and Results

In order to validate our algorithm, we make two kinds of experiments on different modular robots, one on real robots named *Blinky Blocks* and one using *VisibleSim* [10]. *VisibleSim* is a discrete-event 3D simulator for modular robots. This simulator supports large-scale ensembles and different robotic systems including *3D Catoms* used in our simulations (cf. Figure 4).

3D Catoms are quasi-spherical modules placed in a FCC lattice which provides to connect a module up to 12 neighbors. In the simulator it is possible to light a *3D Catom* in colors to show a status, for example the color of the cluster of which it is a part.

Blinky Blocks are 4 cm large robots placed in a regular cubic lattice. They communicate with up to 6 neighbors using a serial connector. They lights in color in order to show a status.

To provide an objective evaluation of the proposed clustering algorithm, we carried out several clustering scenes on different shapes consisting of thousands of modules as shown in Figure 4. The first example (Figure 4a) shows 4 clusters of a large set of 20 000 *3D Catoms* randomly assembled to create an irregular dense cloud. We note that in this case clusters define 4 main quarters with some irregularities on their borders. In the other hand, in the next experiments on Figures 4b, 4c and 4d presenting very regular volumes with symmetries, the clusters follow the coordinate axes to regularly divide the space.

A short video[1] shows our program running on *Blinky Blocks*. These experiments propose 5 different 3D shapes made of 20 to 28 connected robots. For each experiment *Blinky Blocks* change their color depending on the steps of the code during a couple of seconds and then light with the color of their associated cluster. Moreover during this final stage the cluster leader is blinking in order to be easily localized in the cluster.

In Figure 5, we studied the number of messages exchanged while varying the number of modules on two and ten clusters. The messages in the system are used to compute the weights, build the tree and assign nodes to the clusters. Exchanging messages is the more time consuming activity in distributed algorithm on modular robots. As a consequence the number of messages will directly affect the calculation time of the algorithm.

The number of messages increases linearly by increasing the number of modules and that is valid on different shapes. We note that the random shape in Figure 4a requires the highest number of messages exchanged due to the high number of connections between modules. It is obvious that number of messages are increasing with the increase of

[1] Youtube video: https://youtu.be/Q7R4c0MsyIM

Algorithm 2: Message Handler for any module M_i

Msg Handler *REQUEST_DENSITY(m)*:
- $minDist \leftarrow Minimum(p_{M_i}, anchors)$
- $sim \leftarrow \rho(M_i, M_j) \div minDist$
- $e \leftarrow Edge(M_i.id, M_j.id, sim)$
- $edges \leftarrow edges \cup \{e\}$
- **send** SEND_DENSITY(sim) to *sender*

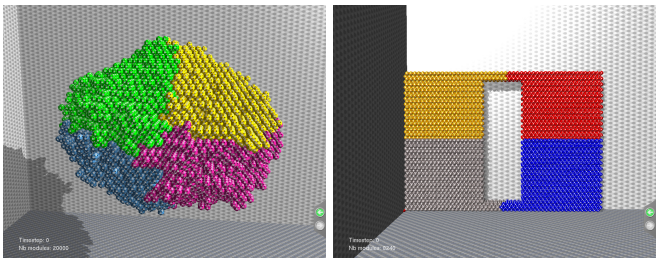
Msg Handler *SEND_DENSITY(d(sim))*:
- $sim \leftarrow d.sim$
- $e \leftarrow Edge(M_i.id, M_j.id, sim)$
- $edges \leftarrow edges \cup \{e\}$
- **if** $(edges.size = neighbours.size\ )$ **then**
  - *wakeup*() `// start of GHS`

Msg Handler *REQUEST_DCUT(d)*:
- **if** *isCH* **then**
  - *return*
- **if** $children.size = 0$ **then**
  - $DCut\_Data \leftarrow$ **compute** my current DCut value
  - **send** RESPONSE_DCUT($DCut\_Data$) to *parent*
- **else**
  - **foreach** *child in children* **do**
    - **send** REQUEST_DCUT(d) to *child*
    - $nbWaitedAnswers \leftarrow nbWaitedAnswers + 1$

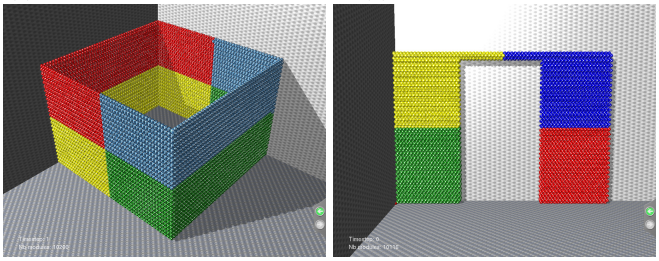
Msg Handler *RESPONSE_DCUT(d)*:
- $nbWaitedAnswers \leftarrow nbWaitedAnswers - 1$
- **if** $d.DCut < DCut$ **then**
  - $DCut \leftarrow d.DCut$
- **if** $nbWaitedAnswers = 0$ **then**
  - **if** $!isCH$ **then**
    - $myDCut \leftarrow$ **compute** my current DCut value
    - $d \leftarrow Minimum(d, myDCut)$
  - **else**
    - `// Cluster Head`
    - $x \leftarrow$ nbr of nodes in the cluster on cutAt
    - $y \leftarrow$ nbr of nodes in the resulting partition
    - **if** $x > y$ **then**
      - $cutAt.k \leftarrow \lceil k/2 \rceil$
      - $k \leftarrow \lfloor k/2 \rfloor$
    - **else**
      - $cutAt.k \leftarrow \lfloor k/2 \rfloor$
      - $k \leftarrow \lceil k/2 \rceil$
    - **send** CUT($cutAt, k$) to *cutAt*

Msg Handler *CUT(cutAt, k)*:
- **if** *id = cutAt* **then**
  - $isCH = true$
  - **if** $k > 0$ **then**
    - *cut*()
- **else**
  - **send** CUT($cutAt, k$) to *cutAt*

number of clusters, since it will require more messages to execute the third step of the algorithm to cut the graph in several partitions and then distribute the nodes on the different clusters. Finally, after conducting different test cases and scenarios, and based on the results shown in Figures 4 and 5, we can say that the efficiency of the proposed algorithm is highly affected by the number of modules and the shape of the ensemble.



(a) Random: 20000 *3D Catoms* (b) Hole: 8240 *3D Catoms*



(c) Cube: 10200 *3D Catoms* (d) Arc: 10000 *3D Catoms*

Fig. 4: Screenshots of VisibleSim - 4 clusters on different shapes

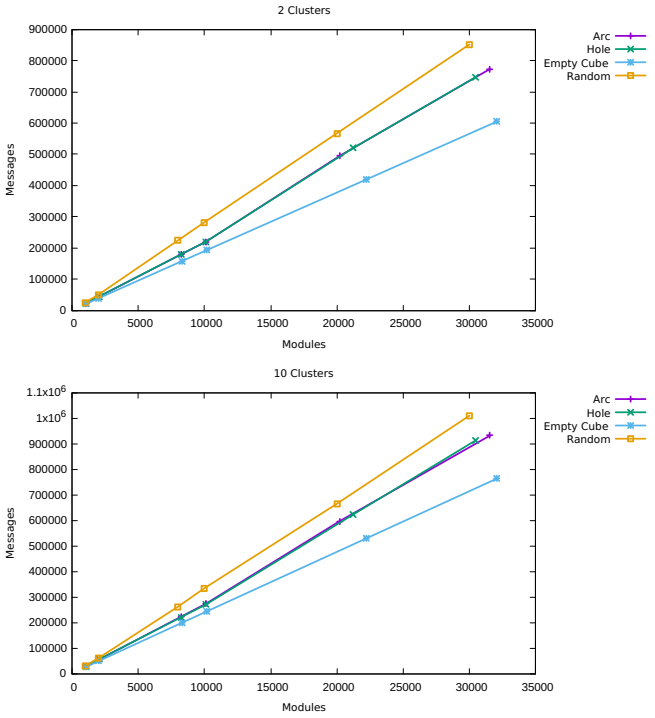


Fig. 5: Number of messages exchanged depending on the number of modules for a dividing into 2 and 10 clusters.

V. Conclusion and future work

In this paper, we presented a linear distributed clustering algorithm based on graph cut for large modular robots ensembles with neighbor-to-neighbor communication that can be applied to programmable matter. To the best of our knowledge it is the first one in the literature. We evaluated the algorithm while considering different shapes, varying the number of clusters and the number of modules in the ensemble. We showed with the obtained results that the performance of the algorithm is affected by the number of modules, the shape of the ensembles and the number of clusters.

In future works, we intend to alter the algorithm so we can control the number of modules in each cluster which is crucial for cluster based self-reconfiguration. In addition, we aim to study the effect of the positions of anchors on the resulting clusters shapes. Then, since modules are able to move and change their connections which can cause an unwanted disconnection in the built tree, we aim to extend the algorithm to support dynamic clusters while maintaining balance between clusters and intra-clusters connectivity. Next, we seek to show the improvement that clustering can yield to the execution of tasks related to modular robots' programmable matter such as self-reconfiguration where modules forming specific parts of the current shape (e.g. a hand in a humanoid shape) can form one cluster hence requires less reconfiguration actions to attain a more or less similar parts of the goal shape (e.g a hand in another humanoid shape).

Acknowledgement

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

References

[1] Reem J Alattas, Sarosh Patel, and Tarek M Sobh. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*, 95(3-4):815–828, 2019.

[2] Jyoti Bhola, Surender Soni, and Gagandeep Kaur Cheema. Genetic algorithm based optimized leach protocol for energy efficient wireless sensor networks. *Journal of Ambient Intelligence and Humanized Computing*, 11(3):1281–1288, 2020.

[3] Giuseppe A Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33(1):69–101, 2020.

[4] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

[5] S Hauser, M Mutlu, P-A Léziart, H Khodr, A Bernardino, and AJ Ijspeert. Roombots extended: Challenges in the next generation of self-reconfigurable modular robots and their application in adaptive and assistive furniture. *Robotics and Autonomous Systems*, page 103467, 2020.

[6] Andrew B Jones, Thomas Cameron, Benjamin Eichholz, David Loegering, Taylor Kray, and Jeremy Straub. Self-reconfiguring modular robot learning for lower-cost space applications. In *2019 IEEE Aerospace Conference*, pages 1–6. IEEE, 2019.

[7] Chao Liu and Mark Yim. Configuration recognition with distributed information for modular robots. In *Robotics Research*, pages 967–983. Springer, 2020.

[8] Proshikshya Mukherjee. Leach-vd: A hybrid and energy-saving approach for wireless cooperative sensor networks. In *IoT and WSN Applications for Modern Agricultural Advancements: Emerging Research and Opportunities*, pages 77–85. IGI Global, 2020.

[9] A. Pietrabissa and F. Liberati. Dynamic distributed clustering in wireless sensor networks via voronoi tessellation control. *International Journal of Control*, 92(5):1001–1014, 2019.

[10] Benoit Piranda, S Fekete, A Richa, K Römer, and C Scheideler. Visiblesim: Your simulator for programmable matter. In *Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271)*. Jul, 2016.

[11] Song Qi, Hengyu Guo, Jie Fu, Yuanpeng Xie, Mi Zhu, and Miao Yu. 3d printed shape-programmable magneto-active soft matter for biomimetic applications. *Composites Science and Technology*, 188:107973, 2020.

[12] Aristides AG Requicha and Herbert B Voelcker. Constructive solid geometry. 1977.

[13] Junming Shao, Qinli Yang, Jinhu Liu, and Stefan Kramer. Graph Clustering with Density-Cut. *arXiv preprint arXiv:1606.00950*, 2016.

[14] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. Efficient scene encoding for programmable matter self-reconfiguration algorithms. *Proceedings of the ACM Symposium on Applied Computing*, Part F1280:256–261, 2017.

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

[16] Fan Xiangning and Song Yulin. Improvement on leach protocol of wireless sensor network. In *2007 international conference on sensor technologies and applications (SENSORCOMM 2007)*, pages 260–264. IEEE, 2007.

[17] Meibao Yao, Christoph H Belke, Hutao Cui, and Jamie Paik. A reconfiguration strategy for modular robots using origami folding. *The International Journal of Robotics Research*, 38(1):73–89, 2019.