Distributed Size-Constrained Clustering Algorithm for Modular Robot-based Programmable Matter

JAD BASSIL, ABDALLAH MAKHOUL, BENOÎT PIRANDA, and JULIEN BOURGEOIS, Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, France

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-reconfiguration process that focuses on finding a sequence of reconfiguration actions to convert robots from an initial shape to a goal one. The main idea of clustering is to divide the modules in an initial shape into a number of groups based on the final goal shape in order to enhance the self-reconfiguration process by allowing clusters to reconfigure in parallel. In this work, we prove that the size-constrained clustering problem is NP-complete and we propose a new tree-based size-constrained clustering algorithm called "SC-Clust". The idea is to divide a network into a predefined number of clusters constrained by a given number of modules in each cluster based on the final goal shape. The result is an efficient algorithm that scales to large modular robot systems. To show the efficiency of our approach, we implement and demonstrate our algorithm in simulation on networks of up to 30,000 modules and on the *Blinky Blocks* hardware with up to 144 modules.

CCS Concepts: • **Theory of computation → Distributed algorithms**; • **Computer systems organization → Robotics**.

Additional Key Words and Phrases: programmable matter, modular robots, clustering algorithms, distributed algorithms

ACM Reference Format:

Jad Bassil, Abdallah Makhoul, Benoît Piranda, and Julien Bourgeois. 2022. Distributed Size-Constrained Clustering Algorithm for Modular Robot-based Programmable Matter. 1, 1 (September 2022), 23 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Programmable matter is matter that can be programmed to change its physical properties on demand or due to internal or external events [32]. It can be achieved using modular self-reconfigurable robots (MSR) composed of thousands or millions of homogeneous micro-modules. The shape of the micro-modules differs depending on the modular robotic system used. They can communicate by exchanging messages and move around each other to reconfigure from their initial shape to a goal one in order to adapt to their task-environment, accommodate different conditions and cover failure. Figure 1 shows a self-reconfiguration example of an initial mug shape made of tiny spherical modules into a goal plate shape. Such matter can have many applications and can be deployed in a large variety of domains including surgery, space exploration, environmental science, construction, etc [4, 61]. Examples of future applications include

Authors' address: Jad Bassil, jad.bassil@femto-st.fr; Abdallah Makhoul, abdallah.makhoul@femto-st.fr; Benoît Piranda, benoit.piranda@femto-st.fr; Julien Bourgeois, julien.bourgeois@femto-st.fr, Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, 1 cours Leprince-Ringuet, Montbéliard, France, 25200.

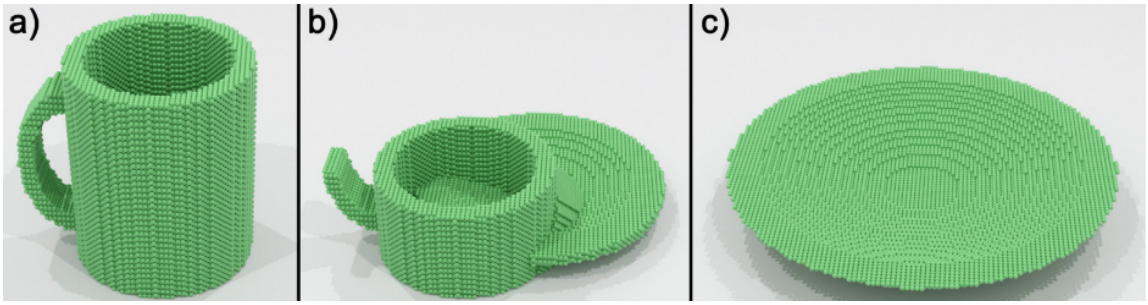


Fig. 1. A self-reconfiguration of a programmable matter made with tiny spherical modules [58]. a) initial configuration. b) intermediate configuration during self-reconfiguration. c) goal configuration.

delivering drugs in the human body, monitoring hostile environments, educational robots, new sets of robotics toys, etc [4, 13, 50].

Planning for self-reconfiguration which consists of finding the sequences of movements to be carried out by the modules to change the shape of the MSR is a difficult process. The number of possible configurations increases exponentially when the number of modules in the system increases. The self-reconfiguration planning problem has been shown and proven to be NP-complete for chain-type MSR where modules are arranged in a chain and is expected to be at least NP-complete for lattice MSR [58] where modules are arranged in a regular lattice structure. Therefore, the self-reconfiguration problem stands as a major challenge for achieving programmable matter. Clustering the modular robot can help reduce the search space thus enhancing the self-reconfiguration process. Accomplishing tasks in cluster-based approaches allows parallelization and increases the efficiency in terms of execution time, communication load and energy consumption. In fact, a cluster head (CH) will be designated in each cluster to schedule tasks and activities in its cluster and to coordinate intra-clusters operations with other CHs. Cluster's members will only communicate with their CH reducing the communication scope to inter-cluster only thus avoiding passing of redundant messages.

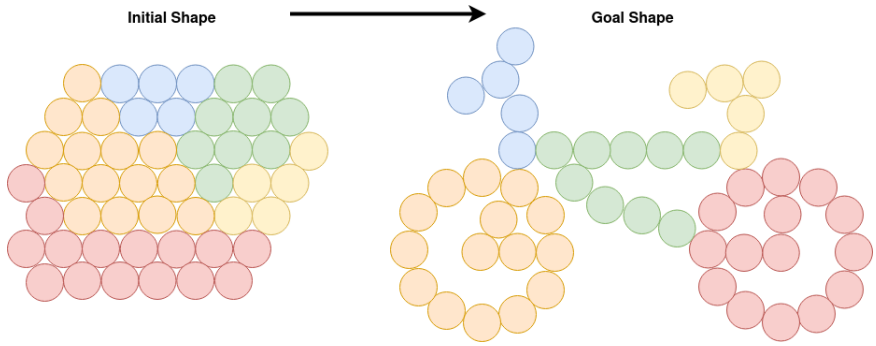


Fig. 2. Clustering motivation

Figure 2 shows the benefit that clustering can yield to the self-reconfiguration process. The modules in the initial shape are clustered into 5 clusters with sizes chosen according to the goal shape. The formed clusters can then reconfigure in parallel to form their corresponding part of the goal shape thus, reducing the time and communication required for transforming the configuration from the initial shape to the goal one. For instance, in [40], the authors proposed a

cluster-based self-reconfiguration algorithm where clusters of modules in the initial shape reconfigure in parallel to form the goal shape. To show the advantage that clustering can yield to self-reconfiguration, they compared the execution time and communication load while varying the number of clusters. The results showed that both the execution time and the number of exchanged messages decrease by a factor of k where k is the number of clusters. However, they suppose that the clusters are given initially and do not propose a clustering method.

Our objective is to propose an efficient distributed clustering algorithm to partition the modules in the initial shape given the number of clusters and the size of each cluster according to the goal shape in order to enhance the self-reconfiguration process. A tree-based density-cut algorithm was proposed in [6] for the same purpose. However, it resulted in arbitrary sized clusters so, we aim to propose a new algorithm to control the number of modules in each cluster which is crucial for self-reconfiguration since a cluster of modules in the initial shape needs to reconfigure into a specific part of the goal shape requiring a fixed number of modules.

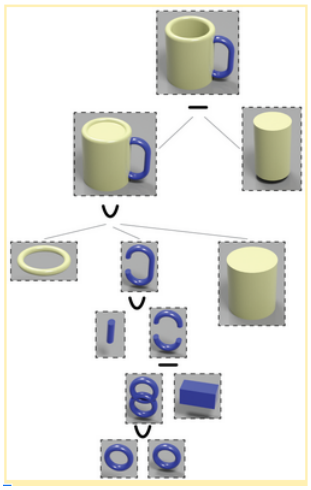


Fig. 3. A mug represented in CSG tree

Prior to self-reconfiguring, modules can be assembled randomly so they are not aware of their initial configuration. But, an efficient encoding of the goal configuration is required for self-reconfiguration since a module needs to know its position according to the goal map. A solution based on Constructive Solid Geometry (CSG)[52] is proposed in [60]. It can be used to determine the number and sizes of clusters. It defines the goal shape as a tree made of basic geometrical objects and transformations (union, intersection, difference) that when combined form the final scene as shown in Figure 3. First, the 3D object to be formed is discretized and encoded into the CSG tree via centralized computations. During this process, the number of clusters and the size of each cluster can also be calculated according to the goal shape. Then, the CSG tree can be transmitted along with the number of clusters and clusters' sizes to a master module to be then flooded and stored in all modules.

In this paper, we present SC-Clust, a distributed algorithm that partition the modules of a modular robot in a predefined number of clusters with predefined cluster sizes. This paper is organized as follow. Section 2 presents the size-constrained clustering problem and lists the system assumptions. Section 3 gives an overview of the related works existing in the literature. In section 4, our proposed solution, the SC-Clust algorithm is described. Section 5 gives a

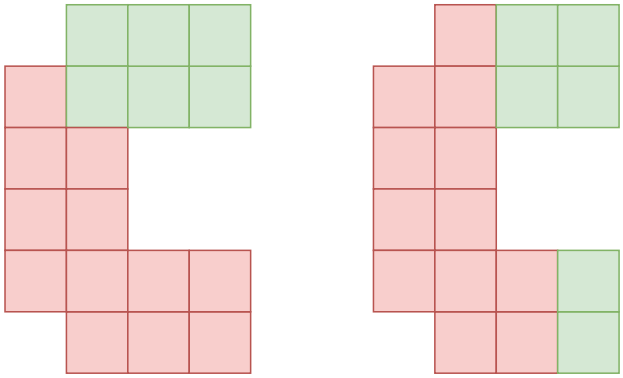


Fig. 4. An example of two possible size-constrained k-partitioning where $k = 2$, $s_1 = 12$ (Red) and $s_2 = 6$ (Green). (a) shows a correct size-constrained k-partitioning. (b) shows an incorrect size-constrained k-partitioning because the second partition shown in green is disconnected.

.

complexity analysis in terms of communication load and execution time. Section 6 shows the conducted simulations and evaluates the performance of the SC-Clust algorithm in terms of execution time and communication load on different configurations and cluster distributions. In section 7, we conclude this paper and mention the intended future works.

2 PROBLEM DEFINITION AND SYSTEM ASSUMPTIONS

The modular robot ensemble can be modeled as an undirected graph $G(V, E, W)$ where V represents the set of modules, E represents the set of edges such that for each pair of modules $(u, v) \in V^2$, $e(u, v) \in E$ denotes a connection between u and v. Therefore, two nodes u and v are neighbors if $\exists e(u, v) \in E$. For each edge $e \in E$, a non-negative weight $w \in W$ is associated, $w : E \longrightarrow \mathbb{R}^{+*}$.

Modules are homogeneous, placed in a regular lattice and they are attached border-to-border. Since they can only communicate with their direct connected neighbors in their adjacent cells, they form a sparse communication graph with large network diameter [43].

Definition 1. Size-constrained partition: A size-constrained partition $G_i(V_i, E_i, W_i)$ is a connected sub-graph of G that have a predefined number of nodes s_i i.e. $|V_i| = s_i$.

Definition 2. Size-constrained k-partitioning: partitions the graph G into k size-constrained partitions (Definition 1) such as:

(1) Partitions are exhaustive, each node must belong to a partition: $V_1 \cup V_2 \cup ... \cup V_k = V$
(2) Each node belongs to only one partition, such as: $\forall i \neq j, V_i \cap V_j = \emptyset$
(3) The size of each size-constrained partition G_i is predefined before partitioning, such as, $\sum_{i=1}^{k} s_i = |V|$

Figure 4 (a) shows a correct size-constrained k-partitioning. Figure 4 (b) shows an incorrect solution since the green cluster is disconnected. The objective of this work is to propose a distributed algorithm that clusters the modular robot ensemble into k clusters by performing size-constrained k-partitioning (Definition 2) on G given the number of partitions k and the desired size of each partition s_i and considering the following assumptions:

- The goal shape is known and can be efficiently encoded and stored in each module, as explained in [60].
- Each module is identified by an unique number (ID).
- Modules are placed in the cells of a regular 3D lattice and they store locally their coordinates and orientation.
- Only neighbor-to-neighbor communications are possible. A module may send a message to its adjacent neighbors through one of its connectors. The receiver can respond by sending a message through the connector that received the message.
- No global view of the modular robot network is available. The view of each module is limited to its direct neighborhood. Modules perform their computations locally, and they can only access local information in their neighborhood via message-passing.
- A module is aware of its direct connections (i.e., which borders are connected to other modules and which ones are not).
- We consider the configuration to be fixed and always connected during the process, that is, no new modules are connected or disconnected during the execution of the algorithm.

2.1 Size-constrained k-partitioning is NP-complete

In this section, we first define the k-balanced clustering problem and then prove that the size-constrained k-partitioning problem is NP-complete. To do so, we prove that it is NP-hard by restriction from the k-balanced clustering problem. NP-completeness follows since it is simple to verify a given solution with a linear algorithm.

Definition 3. **k-balanced clustering Problem:**
INSTANCE: A connected lattice graph $G(V, E)$ the number of wanted clusters k.
QUESTION: Does there exist k equal sized partitions $V_1, ..., V_k$ such that $|V_i| = \frac{|V|}{k}$, $V_1 \cup V_2 \cup ... \cup V_k = V$ and $\forall i \neq j, V_i \cap V_j = \emptyset$?

The k-balanced partitioning problem defined in Definition 3 is proved to be NP-hard on 2D lattice graphs by reduction from Hamiltonian path in [7] and 3-partition in [20]. The size-constrained k-partitioning problem contains the k-balanced clustering problem as a special case where all clusters are equal in size. Therefore, by restriction [22], The size-constrained k-partitioning problem is NP-hard on 2D lattice graphs and therefore, it is at least NP-hard on 3D lattice graphs representing module connections in lattice-based modular robots.

3 RELATED WORKS

The problem we consider is related to graph clustering or graph partitioning. The graph partitioning problem has been widely studied in the literature and it is known to be a NP-hard problem [1, 11, 54]. Existing graph partitioning methods rely on two search techniques: global and local. They aim to partition a given graph into k disjoint balanced dense partitions. Global search algorithms work on the entire graph to find a direct solution. They include solutions based on linear programming [19, 26], spectral clustering [28, 33, 35] and geometrical clustering [5, 23, 57]. Local search algorithms use heuristic and metaheuristic methods to iteratively improve an initial solution based on an optimization function. They use techniques such as node swapping [44], tabu search [53], random walk [64], graph growing [15, 49], genetic algorithms [31], multilevel approach [38, 39] ... The aforementioned methods are used for graph structured data and are not suitable for modular robots, as they require global knowledge of the graph.

Capacitated clustering problem (CCP) [42] is a problem closely related to the graph partitioning problem. Its objective is to partition the weighted nodes of a graph into a set of disjoint clusters where the sum of the nodes' weights in

each cluster is constrained by an upper and lower capacity limit while maximizing the edges' weights of each cluster. Existing CCP solutions use centralized heuristic approaches [24, 34, 55, 66]. For example, in [66], two heuristics are used: tabu search and mimetic algorithms. They can be applied to find size-constrained partitions but they require global knowledge of the graph and do not scale to thousands of resource constrained modules since they require thousands of iterations to find an acceptable solution.

Distributed partitioning methods were developed to overcome the high computation cost when the graph size becomes very large. The distributed partitioning model distributes the partitioning task across a network of computers. For example, in [51] the author proposed JA-BE-JA, a fully distributed iterative method that can find balanced partitions while reducing the number of cut-edges using local search and simulated annealing. It requires thousands of iterations and uses multi-start strategies to converge towards an optimal solution resulting in a huge communication load, especially in a sparse graph such as the one representing modules connections. In a more recent work, Adoni et al. presented DHPV [2], a distributed algorithm that outperforms JA-BE-JA. It is more suitable to the master-slave distributed architecture where a master node coordinates the partitioning process and the partitioning task is distributed to slave nodes that operates in parallel to add a new vertex to their partition's subgraph. These methods are suitable for balanced partitioning of graph structured data and can't be applied to distributedly partition the modules of a modular robot with communication limited to neighbor-to-neighbor and no centralized global control.

The multi-robot task allocation problem [30] is about assigning a group of robots to a set of tasks in the most optimal way based on a utility function. The utility function measures how well a robot can perform a task. Some tasks require multiple homogeneous robots or heterogeneous robots with different capabilities to be accomplished. So, robots are partitioned to form k coalitions based on the utility function. Then, tasks are assigned to coalitions to be executed simultaneously [18, 37, 65]. The problem we are tackling in this paper is different from the multi-robot task allocation problem since we consider the partitioning problem independently of the task to be performed which is the self-reconfiguration. Therefore, these methods are not applicable to solve our problem.

Partitioning the set of modules for configuration generation in modular robots has been studied in [16, 17]. In [16], an algorithm based on a coalition search graph is proposed for partitioning a set of modules. It aims for an efficient shape configuration of scattered modules by partitioning-based coalition formation constrained by the maximum number of modules required to form the configuration. It finds the best coalition structure of separated modules based on a utility function. The modules forming a coalition are then docked together to form the goal configuration. Another method for the same purpose is proposed in [17] where a minimum spanning tree is built to minimize docking cost. Then, the best coalition or configuration is found by partitioning the built tree taking into consideration the size, communication and battery constraints. These methods focus on configuring small sets of separated modules scattered in their environment. Hence, they are not applicable to solve our problem.

Clustering has been studied for robotic swarms. The purpose is to split the swarm into clusters for pattern formation and for better problem solving efficiency by dividing the problem into sub-problems and allocating different tasks to each cluster. Mostly, the existing methods rely on robot mobility directed by external stimuli in the environment, so they are not suitable for modular robot's based programmable matter, to cite a few [27, 29, 47, 62]. Other methods based on token clustering were proposed. In [12], a fully distributed algorithm is proposed based on consensus and load balancing to partition the robots with wireless communication into two spatially separated clusters. Then it was extended in [10] to spatially partition the set of robots into multiple clusters. However, the experimental results show that the time required for convergence is high for a small number of robots and a small number of clusters. The experiments showed

Table 1. Comparative table.

Work	distributed	local knowledge	size-constraint
Global Search[5, 26, 28, 31, 33, 38]	×	×	×
Local Search[44, 49, 53, 64]	×	✓	×
CCP [24, 34, 42, 55, 66]	×	×	✓
Distributed Methods[2, 51]	✓	✓	×
SWARM clustering [10]	✓	✓	×
WSN clustering [8, 41, 46, 63]	✓	✓	×
DCut [6]	✓	✓	×
SC_Clust	✓	✓	✓

that it can take minutes to cluster 20 robots into 4 classes. The convergence time is expected to increase immensely for large scale modular robots with communication limited to neighbor-to-neighbor.

Clustering for wireless sensor networks (*WSN*) and mobile ad-hoc networks is related to our problem in which sensors are grouped into clusters to achieve network scalability by creating a hierarchical structure. For each cluster, a cluster-head (*CH*) plays significant roles such as scheduling tasks and aggregating and relaying data generated by its cluster members to limit inter-clusters communications to *CHs* only thus reducing communication load [3]. Many clustering algorithms have been proposed for *WSN* [8, 41, 46, 63] but they are not suitable to modular robots due to their specific constraints which make them inapplicable on modular robots: wireless communication, existence of a base station, pre-election of cluster heads....

In [6] we proposed a fully distributed and adapted version of the DCut algorithm originally proposed by Shao et al. (2018) [56] in the context of modular robots.

It takes into consideration the geometrical aspect of the ensemble and captures the density between adjacent modules locally using Jaccard Coefficient. The idea is to build a density-connected tree (DCT) that captures the topological similarities between modules relative to fixed points on the extremities of the geometry bounding box. Since the DCT forms an acyclic graph, an edge connects two partitions. So, instead of partitioning the whole graph representing all connections between modules, it partitions the DCT by recursively finding and removing cut edges until k clusters are obtained. 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 distributed and efficient. However, it does not take into consideration the size-constraint which is crucial for transforming clusters of the initial shape to specific parts of the goal shape requiring a fixed number of modules.

The existing work aforementioned in this section fails to satisfy the requirements to solve the size-constrained k-partitioning problem for modular robots described in Section 2. The solution must be distributed, based on the limited local knowledge of each module about its neighborhood, and satisfies the size-constraint. Therefore, in this work we present SC-Clust, a distributed solution for the size-constrained k-partitioning problem for modular robots that uses the local knowledge of modules to cluster the ensemble. Table 1 shows which requirements are met by the existing solutions. We excluded from the table the above-mentioned solutions for the multi-robot task allocation problem and the configuration generation problem because partitioning is not their primary focus and they address a different problem than ours.

4 ALGORITHM DESCRIPTION

In this section, we propose the SC-Clust algorithm, a solution to the size-constrained clustering for lattice graphs representing module connections in modular robots. It identifies k size-constrained partitions in $O(nlogn)$ time and communication complexity. The SC-Clust algorithm operates in three phases. First, we define the edge weights and how they are calculated and stored in each module (Section 4.1). Second, a minimum spanning tree (MST) is built. A fully distributed and asynchronous algorithm [21] is used for this purpose. Third, the MST is partitioned. Initially, all modules form the initial cluster; then the MST is sequentially partitioned by finding, adjusting, and separating branches having the desired number of modules (Section 4.3).

4.1 Weight Calculation

In this section, an edge weight measure is defined that captures the geometric aspects of the ensemble. We start with the following definitions:

Definition 4. Anchors: Given a geometrical shape I, the minimum bounding box B is the box surrounding I aligned with the coordinate axes with the minimum volume. The set of anchors A is defined as the set of coordinates of the corners of the minimum bounding box.

Since the modules in a modular robot are placed in a regular lattice, A can be easily and efficiently 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 are defined at the corners of B, that is, all possible combinations of $(\{min_x, max_x\}, \{min_y, max_y\}, \{min_z, max_z\})$.

Definition 5. Edge weight: Given two neighboring modules u and v, the weight $w(u, v)$ of the edge $e(u, v)$ connecting u and v in the graph G, is defined as:

$$w(u, v) = min(dist(u, A), dist(v, A))$$

s.t:

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

where $dist$ represents the Euclidean distance.

The weight measure defined in definition 5 captures the geometrical aspects of the ensemble in a way that edges connecting modules near the borders of the configuration will have lower weights. This will later results in having clusters positioned near borders which facilitate modules movements for self-reconfiguration.

Anchors positions are calculated by building a spanning tree rooted at a randomly chosen module. During the building process, the values of min_x, min_y, min_z, max_x, max_y and max_z are returned to the root then broadcasted to all modules via the built tree. Upon reception, modules can calculate and store the distance to their nearest anchor then, store their adjacent edges weights.

4.2 Tree Construction

After all modules have stored their adjacent edge weights, a Minimum Spanning Tree (MST) is built. It minimizes the $\sum_{(u,v)\in V_{MST}} w(u, v)$. Any distributed algorithm to find a MST can be used. We use a fully distributed asynchronous algorithm called GHS proposed in [21]. GHS is known to have an optimal communication complexity of $O(m + nlog(n))$ messages. Its time complexity is $O(nlog(n))$ which is not optimal. Existing distributed algorithms solve the minimum spanning tree problem with better time complexity at the cost of increasing the communication load [9, 25, 36, 45], which is not suitable for modular robots, as sending messages consumes the limited energy resources of the modules.

The GHS algorithm requires that each edge has an unique weight. In case the weights are not distinct, which is our case, one can simply append the identities of the edge's adjacent nodes starting by lower order first. Initially, each node forms a fragment. Nodes wake up to start the GHS algorithm execution asynchronously, so there are no restrictions on the wake-up process, thus, all nodes can wake up at the same time or only one node can wake up and the tree is formed which is suitable for our case.

The GHS algorithm operates in phases. During each phase, fragments are extended by merging with other fragments. Nodes in each fragment are connected with edges to form a rooted MST. Each node holds a pointer to the next node in the tree that leads to the fragment's root. Fragments are merged through their minimum outgoing edge. To find the minimum outgoing edge of a fragment, a message is broadcasted asking all the fragment's nodes about their minimum outgoing edge. Each node waits for the answers of all its children in the tree before sending it upwards on the tree to reach the fragment's root. Once the minimum outgoing edge is found, a message is sent over that edge to the fragment on the other side. The two fragments will then merge into a larger fragment. If the two fragments chose the same minimum outgoing edge they agree to merge and the edge chosen by the two fragments is called *core edge*.

During the last phase, two fragments will be merged via a *core edge* into one large fragment forming the MST. We refer the reader to [21] for a complete description of the algorithm. Once the MST is formed, we can proceed to its partitioning. One can choose one of the core nodes adjacent to the core edge as the root of the tree. However, to have clusters distributed closer to borders as much as possible, we choose the root to be the node with minimum distance to one of the anchors (Definition 4) at the extremities of the initial configuration. Ties are broken randomly. To do so, after the root is found, it broadcasts a message through the tree. The receiving nodes set the sender as a parent leading to the root and save the edges leading to their children in the MST. The resulting tree on a 2D regular lattice is shown in Figure 5.

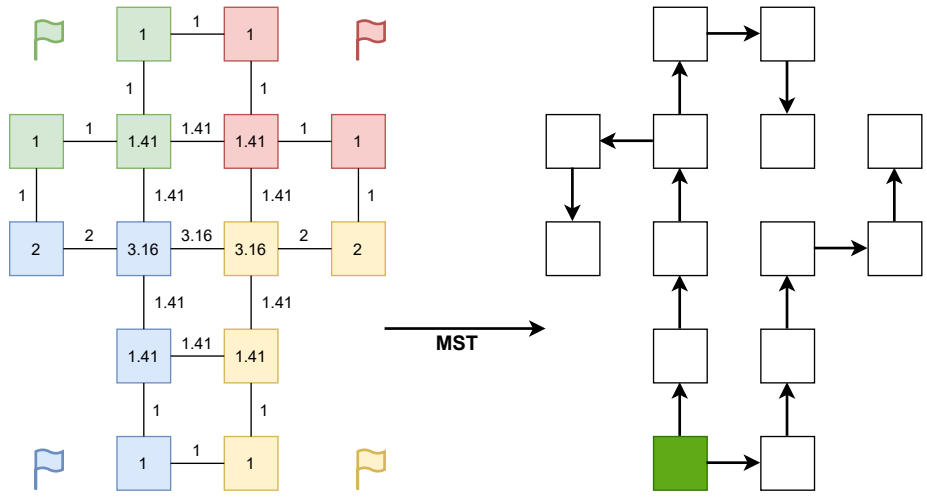


Fig. 5. An example of MST construction. On the left the weight distributed according to the distance to the nearest anchor (The flag of the same color). On the right, the MST is constructed and the root is colored in green.

4.3 Tree Partitioning

In this phase, given the set of desired cluster sizes S, the MST is partitioned in order to obtain $k = |S|$ size-constrained clusters. The idea is to find the cut-edge that results in a branch to form the cluster in a way to minimize the difference between the number of modules in the branch and the desired number of modules in the cluster. To do so, we define the cut-edge as follows:

Definition 6. cut-edge: A cut-edge c_i is an edge $e(u, v)$ that separates the partition containing u and v in which c_i is searched from the new partition. The nodes V_i of the new partition G_i are the nodes in the branch of the MST rooted at $cutAt = v$: the node in V_i adjacent to c_i. Given the set of desired partitions' sizes S, in order to satisfy the size constraint described in Section 2, $|V_i|$ should be equal to s_i. However, a cut-edge that satisfies this constraint may not exist since a branch in the MST having exactly s_i nodes could not be found. Therefore, the cut-edge c_i is found in a way to minimize the difference $Diff_{c_i}$ between the size of the sub-tree rooted at $cutAt$ and s_i. Therefore:

$$c_i = e(u, v) \in E \mid Diff_{c_i} = \min_{e(u,v)\in E} |Diff_e|$$

s.t.

$$Diff_{e(u,v)} = s_i - subtreesize(v)$$

After removing a cut-edge c_i, the difference between the resulting cluster size and the desired size $Diff_{c_i}$ may not be null if a branch containing the desired number of modules did not exist in the MST. To fix this issue two methods are presented in the following sections. The first in section 4.4 is a naive method that builds and exchanges a chain of modules to fix the erroneous cluster's size. The second in section 4.5, makes additional cuts and associates the resulting branches to the erroneous cluster until having the desired size.

4.4 Naive Solution Based On Modules Exchange

In this section, a way to satisfy the size-constraint for a cluster V_i resulted after the i^th cut is described. First, all modules belong to cluster V_0. The root of V_0 initiates $k - 1$ cuts to obtain k clusters. After each cut, if $Diff_{c_i}$ is not null, $Diff_{c_i}$ modules are exchanged between V_0 and V_i. To do so, the furthest module in V_i from the root of V_i having at least one neighbor in V_0 is chosen as chain source. Then, a chain consisting of a sequence of modules starting from the chain source is built. In case $Diff_{c_i} > 0$, the chain is built in V_i consisting of a maximum $Diff_{c_i}$ module and exchanged with V_0. In case of $Diff_{c_i} < 0$, the chain is built in V_0 starting from the chain source and exchanged with V_i. If after an exchange, $Diff_{c_i}$ is still not null, the exchange process is repeated. An example of module exchange is shown in Figure 6 where four clusters of equal size are sequentially formed on a 2D humanoid shape starting from the left figure. The clusters roots are colored in brown, the chain source is colored in grey, and the exchanged chain is colored in white.

To build the chain, the last module added to the chain must choose the next one to add. A strategy is required to make this choice. Three strategies have been studied, they are presented in Figure 7 in an example where four equal sized clusters are formed on a 3D mug shape.

The first strategy in Figure 7a consists in adding the module with minimum distance to the centroid of the cluster. The second strategy in Figure 7b adds the module with the minimum Euclidean distance to the centroid. The third in Figure 7c builds a chain with modules on the border between V_i and V_0. As it can be seen the cluster shapes differ according to the exchange strategy.

4.4.1 Exchanging Modules Problem. A problem that can occur is that an exchanged chain can possibly disconnect a cluster. As shown in Figure 8, the cluster colored in red becomes disconnected : modules circled in red are not accessible

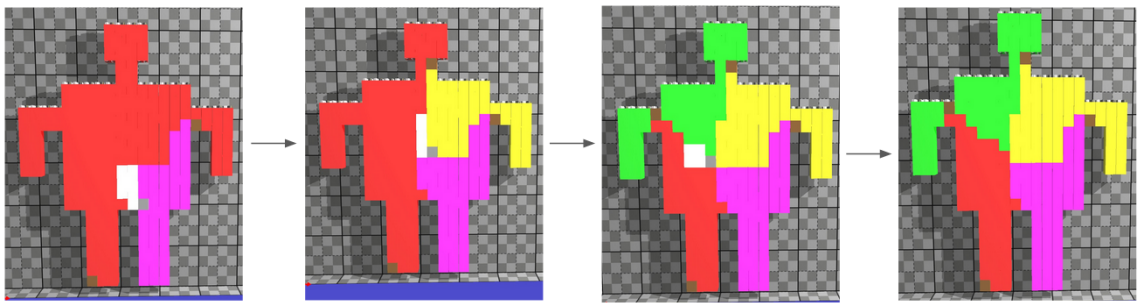
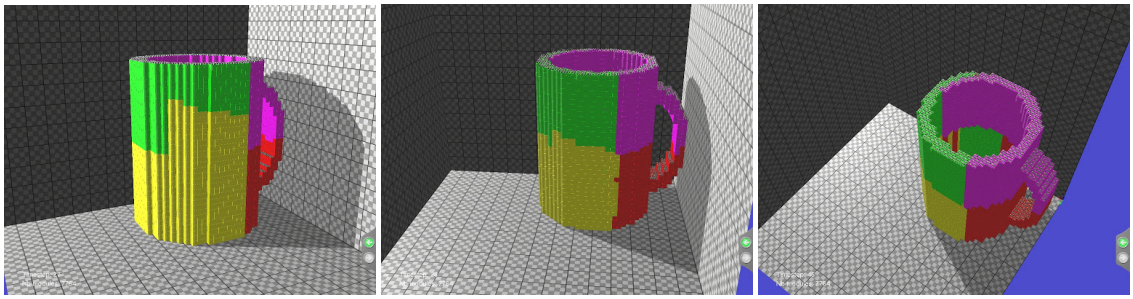


Fig. 6. Modules exchange example



(a) Distance to centroid (b) Distance to center of gravity (c) Exchange on borders

Fig. 7. Modules exchange strategies

by the root of their partition via a path of red modules only. This issue can be solved by checking, while building the exchange chain, if the module to be added causes a disconnection. Or after building the chain, if exchanging the built chain causes a disconnection, then find another chain. However, checking if exchanging a module or a chain causes a disconnection is a heavy time consuming process that requires an additional communication load. In addition, choosing the best strategy is not evident since it depends on the geometry of the ensemble. Therefore, another disconnection-less method to deal with the size difference resulted after a cut is described in the next section and it is considered in future sections.

4.5 Additional Cuts

In this section, a new method to deal with size difference after a cut is presented. It consists in performing additional cuts until the size constraint for cluster i is satisfied i.e. $Diff_{c_i} = 0$. Initially, all modules belong to V_0. If after a cut c_i, $|V_i| \neq s_i$, an additional cut is made to find an adjacent branch with size equal to $Diff_{c_i}$ that contains at least one module that has a neighbor in V_i and the resulting branch is joined with or cut off the erroneous cluster. The flow chart for creating a partition V_i is depicted in Figure 9. Three cases are presented after an initial cut:

(1) If $Diff_{c_i} > 0$, the root of V_i in the *MST* initiates the search for a new cut-edge $c_{ij}(u, v)$ in its partition that minimizes: $|Diff_{c_i} - subtreesize(v)|$, the resulting branch is added to V_0.
(2) If $Diff_{c_i} < 0$, the root of the *MST* initiates the search for a new cut-edge $c_{ij}(u, v)$ in its partition V_0 that minimizes: $|Diff_{c_i} - subtreesize(v)|$. The resulting branch is added to V_i.

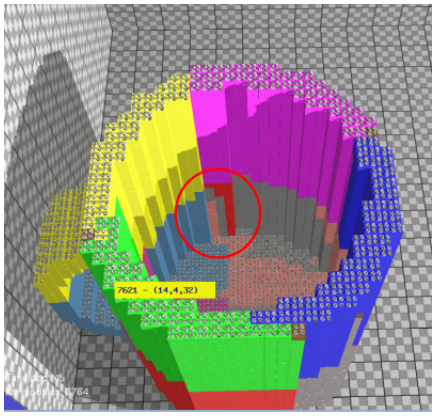


Fig. 8. Disconnection problem

(3) If $Diff_{c_i} = 0$, the size-constraint is satisfied. The root start the search for a cut-edge c_{i+1} for the V_{i+1} partition.

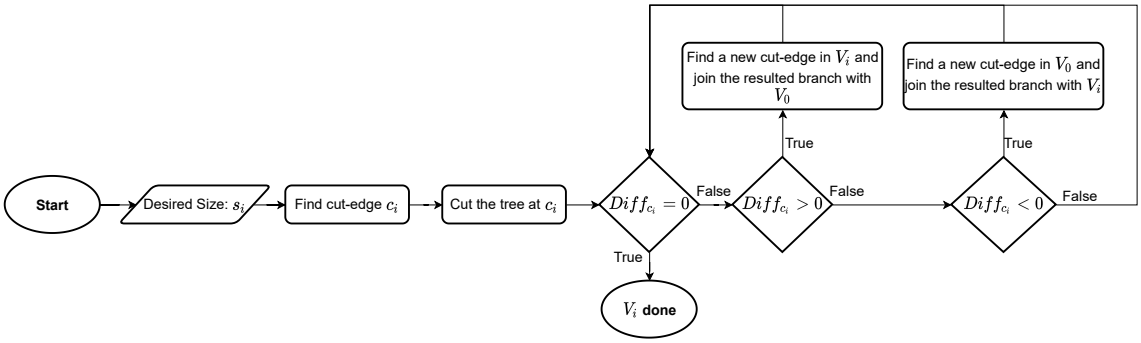


Fig. 9. Flow chart for the i^{th} partition

This method guarantees that the resulted clusters are always connected since the structure of the MST is maintained. In addition, the size-constraint can always be satisfied because in a worst case scenario where both $|Diff_{c_i}|$ and $\forall v \in V$, $|Diff_{c_i} - subtreesize(v)|$ are large, $|Diff_{c_i}|$ additional cuts resulting in partitions containing 1 module each can be made.

Finding a cut-edge is initiated by calling the *cut* procedure (see Algorithms 1, 2, 3) with three parameters:

(1) $recut$: A boolean that indicates if the cut-edge to be found is an additional cut to deal with a previous partition's size difference.
(2) $desiredSize$: The desired size of the partition.
(3) adj: In case of an erroneous size partition i ($Diff_{c_i} \neq 0$), adj takes the value of the partition id i to which the resulted partition needs to be joined. Otherwise it takes the value 0.

Initially, all nodes belong to partition V_0 with $|V_0| = |V|$. For $i \in [1, k-1]$, a partition V_i is obtained after removing a cut-edge c_i. Algorithms 1, 2 and 3 describe partitioning. The root of the *MST* first executes the *cut* procedure that initiate

Algorithm 1: Partitioning algorithm (Part 1)

```
nbModules // Number of modules in the system
subTreeSize // sub-tree size of the module
cutAt // a boolean indicating if the module is the root of the cut branch
S // set containing the desired cluster sizes
MST // the minimum spanning tree built in phase 2
isMSTRoot // a boolean indicating if module is the root of the MST
recut // a boolean indicating if an additional cut is being found
toBestCut // the interface to reach the cut edge
Cluster // cluster identifier
minDiff // minimum Diff found
maxNbAdj // maximum number of modules adjacent to the erroneous cluster
toLastCut // root of the latest identified cluster
children // set containing child modules in the MST
nbWaitedAnswers
1  if isMSTRoot then
2  |  isRoot ← true; i ← 1; desiredSize ← S[i]
3  |  cut(false, desiredSize, 0)
4  Procedure Cut(recut, desiredSize, adj):
5  |  nbWaitedAnswers ← 0
6  |  foreach child in children do
7  |  |  send FIND_CUT(recut, desiredSize, adj) to child
8  |  |  nbWaitedAnswers ← nbWaitedAnwers + 1
9  Msg Handler FIND_CUT(recut, d, adj):
10 |  minDiff ← ∞; subTreeSize ← 0; desiredSize ← d
11 |  if |children| = 0 then
   |  |  // Leaf
12 |  |  subTreeSize ← 1; nbAdj ← nb of neighbors in adj
13 |  |  minDiff ← |subTreeSize − desiredSize|; maxNbAdj ← nbAdj
14 |  |  send RESP_CUT(subTreeSize, minDiff, maxNbAdj) to parent
15 |  else
16 |  |  nbWaitedAnswers ← 0
17 |  |  foreach child in children do
18 |  |  |  send FIND_CUT(recut, desiredSize, adj) to child
19 |  |  |  nbWaitedAnswers ← nbWaitedAnwers + 1
```

the search for the first cut-edge. **FIND_CUT** message is sent in broadcast and **RESP_CUT** is sent using convergecast as described in algorithm 1. During this process, each module calculates the difference between its sub-tree size and the desired cluster size. In case of an additional cut ($recut = true$), the branch to join with partition V_i should have at least one neighbor in V_i to avoid having disconnected partitions (algorithm 2, lines 32, 35). The minimum difference of a branch size with the maximum number of neighbors possible in adj is returned to the root, and the module interface to reach the *cutAt* module is saved in *toBestCut*. The root will then send a **CUT** message to the *cutAt* module connected to the cut-edge which will become the root of the new partition.

Algorithm 2: Partitioning algorithm (Part 2)

```
20 Msg Handler RESP_CUT(s, e, m):
21   nbWaitedAnswers ← nbWaitedAnswers − 1; subTreeSize ← subTreeSize + s
22   if |e| < minDiff then
23     minDiff ← e; toBestCut ← sender
24   if recut = true and |e| = minDiff and m > maxNbAdj then
25     maxNbAdj ← m; toBestCut ← sender
26   if nbWaitedAnswers = 0 then
27     subTreeSize ← subTreeSize + 1
28     myDiff ← subTreeSize − desiredSize
29     if cutAt = false and isRoot = false then
30       if |myDiff| < minDiff then
31         minDiff ← myDiff; toBestCut ← NULL
32       if recut = true then
           // Count nb of modules adjacent to cluster adj in current branch
33         maxNbAdj ← m + nbAdj
34         if maxNbAdj = 0 then
             // Do not consider the branch
35           minDiff ← ∞
36       send RESP_CUT(subTreeSize, minDiff, maxNbAdj) to parent
37     else
38       if isRoot = true and (recut = false or desiredSize > 0) then
           // Cluster i is found
39         send CUT(i) to toBestCut
40       else
           // cutAt performs an additional cut and join the resulted branch to cluster 0
41         send CUT(0) to toBestCut
42 Msg Handler CUT(i):
43   if recut = false then
44     toLastcutAt ← toBestCut
45   if toBestCut = NULL then
46     cutAt ← true
47     Cluster ← i
48     myDiff ← subTreeSize − desiredSize
49     assign sub-tree to cluster i
50     if myDiff > 0 then
         // Cluster i has an excess of modules. Must find a new cut to join the resulting branch
            to cluster 0
51       execute cut(true, myDiff, 0)
52     else
         // Cluster i has a deficit of modules. Report the difference to the root
53       send REPORT_CUT(−myDiff)
54         to parent
55   else
56     send CUT(i) to toBestCut
```

Algorithm 3: Partitioning algorithm (Part 3)

```
57 Msg Handler REPORT_CUT(diff):
58   if isRoot then
59     if recut = false then
60       toLastcutAt ← sender
61     if diff > 0 then
         // Find a new branch with size diff to join it with cluster i
62       recut ← true
63       execute cut(true, diff, i)
64     else
65       if diff < 0 then
           // Send report to the last cutAt so it can find a new branch with size diff and join
              it to cluster 0
66         send REPORT_CUT(|diff|) to toLastcutAt
67       else
           // diff = 0
68         updateTree()
           // Initiate the search for the next cluster
69         i ← i + 1
70         execute cut(false, s_i, 0)
71   else
72     if cutAt = true then
73       if diff > 0 then
74         desiredSize ← diff
75         if recut = true then
76           updateTree()
77         recut ← true
           // Excess of modules. Must find a new cut of size diff and join the resulting branch
              to cluster 0
78         execute cut(true, desiredSize, 0)
79       else
80         send REPORT_CUT(|diff|) to parent
81     else
82       if sender = parent then
83         send REPORT_CUT(diff) to toLastcutAt
84       else
85         send REPORT_CUT(diff) to parent
```

After a cut c_i, the *cutAt* module is aware of the size difference $Diff_{c_i}$ of its partition. If $Diff_{c_i} > 0$ (the resulted cluster has modules in excess), it calls $cut(true, Diff, 0)$ to find a new *cut-edge* inside its partition and the resulted branch is rejoined with the initial partition V_0 to minimize the difference (algorithm 2, line 45,51). Otherwise, it sends a **REPORT_CUT** message with the value of $Diff_{c_i}$ to the root of partition V_0 (algorithm 2, line 53). When the root receives the message, if the received value of $Diff_{c_i}$ is not null, it executes $cut(true, Diff_{c_i}, i)$ to find a cut within its partition and join the resulted branch to the partition V_i (algorithm 3, line 61,63). If after joining a branch to V_i the size of V_i becomes larger than the desired size s_i, the root re-sends **REPORT_CUT** message containing $Diff_{c_i}$ to the last

cutAt module, which is the root of V_i to deal with this difference (algorithm 3, lines 66, 67). Otherwise, if the root receives **REPORT_CUT** message with the value of $Diff_{c_i} = 0$, it updates its cluster tree and then executes $cut(false, s_{i+1}, 0)$ to find the partition V_{i+1} (algorithm 3, lines 67, 70). The tree must be updated to join the resulted branches after additional cuts with their corresponding partitions. The $updateTree()$ procedure depends on the algorithm used to build the MST. After considering nodes in additional branches as disconnected nodes, we use the tree maintenance algorithm described in [14] where the GHS algorithm for building the MST is relaunched inside the partition to join an additional disconnected branch.

5 COMPLEXITY ANALYSIS

In this section, we give a complexity analysis by phase in terms of communication load and execution time. We note $n = |V|$ the number of modules and $m = |E|$ the number of connections between modules.

5.1 Communication Load

In the first phase, the anchor positions are found, and all edges' weights are calculated. It requires $O(n)$ messages to find and store anchors through tree traversal. In addition, to calculate and store an edge weight, two messages are exchanged between the edge's adjacent modules. Therefore, the communication complexity of the first phase is $O(n + m)$.

The second phase consists in building a minimum spanning tree. We use the GHS algorithm described in [21] which has a complexity of $O(m + n \log n)$ in addition to $O(n)$ for finding the root and redirecting edges towards it.

During the third phase, the tree is partitioned to obtain k partitions. The SC-Clust requires $k - 1$ cuts plus a number a of additional cuts used to fix size differences. Therefore, the number of messages required is $O((k - 1 + a) \log n)$ since after each cut the search space for the next cut is reduced. The number of additional cuts a will be discussed in the next section. Moreover, clusters' trees are updated after each cut to join additional branches resulted by additional cuts which requires $O(k \log n)$ messages.

Overall, by summing the complexities of the three phases, the communication complexity is equal to: $O(n + m) + O(n + m + m \log n) + O((k + a) \log n) = O(n + m) + O((m + k + a) \log n)$. In a filled cubic geometry the maximum number of connections m is equal to $3n$. Also, in all practical cases $k \ll n$ and $a \ll n$ unless $s_i = 1$ for $i \in [1, n]$. Therefore, the overall communication complexity can be expressed with the number of modules in the system n and it is equal to $O(n) + O(n \log n) = O(n \log n)$.

5.2 Execution Time

The time required for the first phase in which anchor positions are found and edges weights are calculated depends on the diameter d of the network since the maximum tree length is bounded by d. Three tree traversals are required. Thus, the time complexity of the first phase is $O(d)$.

The time complexity of building the tree in the second phase is $O(n \log n)$ [21]. Redirecting all edges towards the root requires a tree traversal. The time taken for tree traversal is $O(n)$ since the maximum possible diameter of the MST can be equal to n. Therefore, the time required for the second phase is $O(n) + O(n \log n) = O(n \log n)$.

As for the third phase, the time required for finding a cut is $O(n)$. $k + a$ cuts need to be found. Therefore, the time complexity for partitioning the MST is $O((k + a).n)$ in addition to the time required for joining additional cuts and updating clusters' tree which is $O(k. \log n)$. Therefore, the overall time complexity of the third phase is $O(n) + O(k. \log n) = O(n)$.

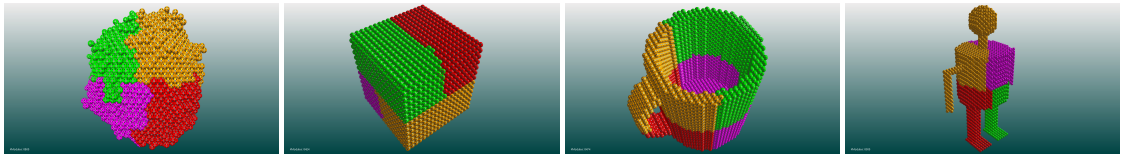
The overall complexity of the three phases is $O(d) + O(n \log n) + O(n) = O(n \log n)$. This complexity is mostly due to the construction of the MST.

6 SIMULATIONS AND RESULTS

We evaluated our algorithm in simulation using *VisibleSim* [59], a discrete-event 3D simulator for modular robots that supports thousands of modules that form large-scale ensembles. It supports different modular robotic systems including *3D Catoms* [48] used in our simulations.

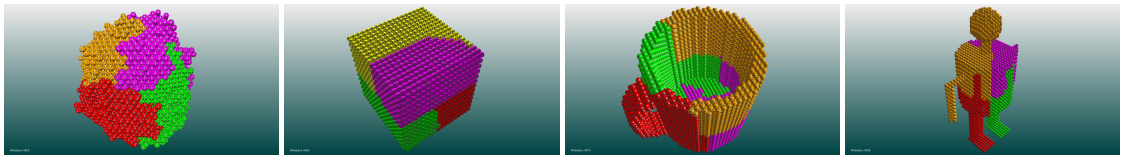
We also validated the SC-Clust algorithm on real robotic systems called *Blinky Blocks*. The video[1] shows 6 different experiments on 144 real *Blinky Blocks* consisting of subdividing 3 different shapes (a square, a cube and a double F shape) into 4 clusters. For each shape, we run the code one time to create clusters with the same number of *Blinky Blocks* and another time to create heterogeneous clusters with 10 %, 20 %, 30 % and 40 % of the set.

3D Catoms are quasi-spherical modules placed in a FCC lattice where a module can connect to up to 12 neighbors. *VisibleSim* allows to light up a module with a certain color to show its status. We use this feature to distinguish clusters by coloring each cluster with a different color.



(a) Random shape (b) Cubic shape (c) Mug shape (d) Humanoid shape

Fig. 10. DCut results on 4 different shapes with 4 clusters



(a) Random shape (b) Cubic shape (c) Mug shape (d) Humanoid shape

Fig. 11. SC-Clust results on 4 different shapes with 4 equal size clusters

Figure 10 shows 4 clusters created by DCut, our previously proposed partitioning algorithm that results in arbitrary sized clusters on 4 different shapes: a randomly generated shape forming an irregular dense cloud with 8,500 modules, a cubic shape formed by 7,225 modules with a densely filled volume, a mug shape formed by 8,584 modules and a humanoid shape formed by 8,291 modules with components of different densities. The 4 clusters are distributed regularly along the borders of the configuration. Figure 11 shows 4 clusters created by SC-Clust of equal sizes on the same shapes as in Figure 10. The created clusters shapes differ from the shapes created by the DCut algorithm since they partition the tree differently. The clusters created by SC-Clust show some irregularities on their borders due to additional cuts that attach or remove modules on the borders to satisfy the size constraint as explained in Section 4.5.

[1] YouTube video: https://youtu.be/niYHGoqWbQs

6.1 Evaluating SC-Clust

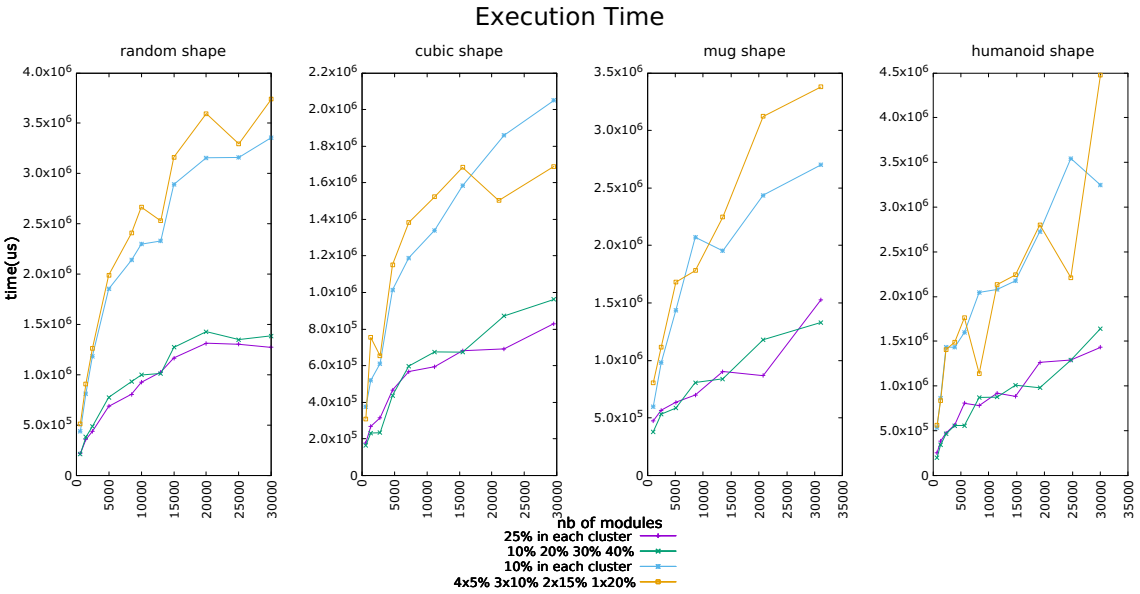


Fig. 12. SC-Clust execution time evaluation

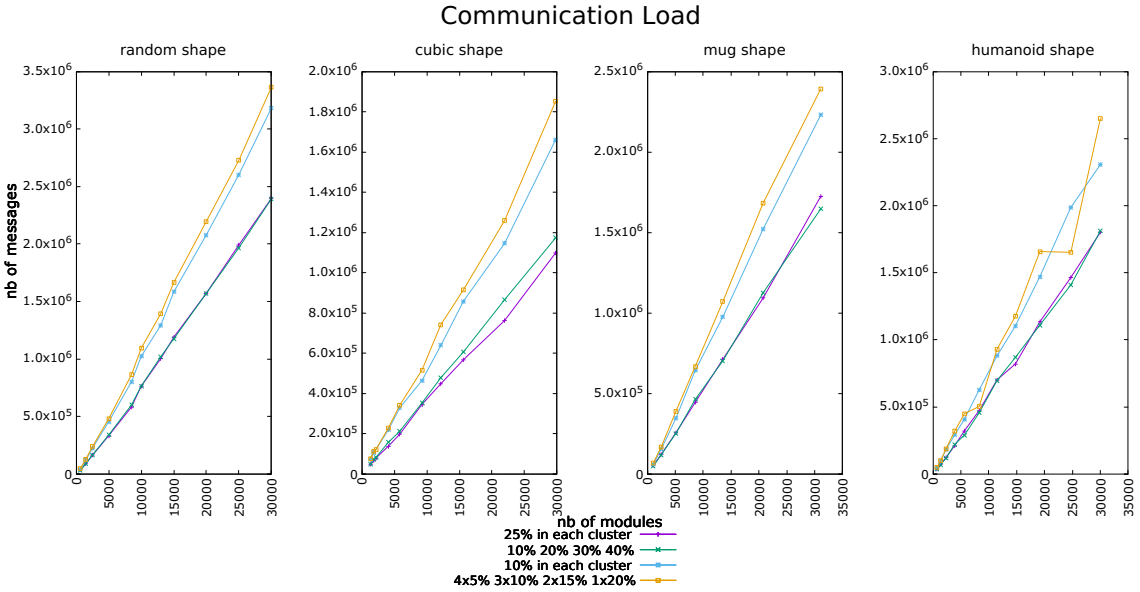


Fig. 13. SC-Clust communication load evaluation

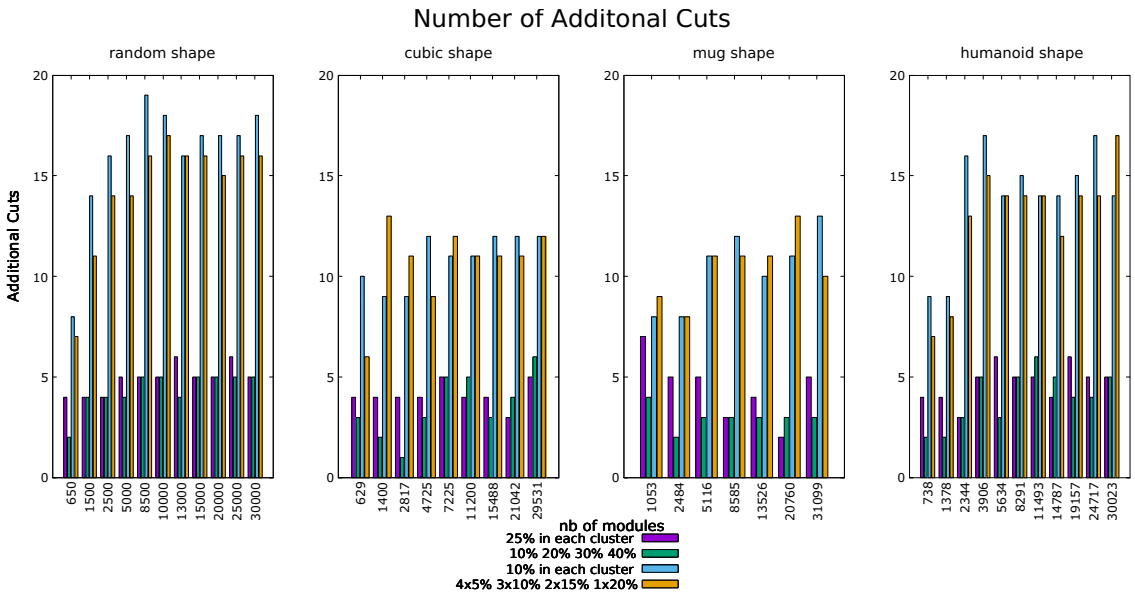


Fig. 14. Number of additional cuts

To provide an objective evaluation, we carried out different simulations with different shapes consisting of up to 30,000 *3D Catoms*. Each shape has different geometrical properties to show that the proposed algorithm find a solution independently from the geometrical shape. For each shape, we conducted simulations with the following cluster distributions:

- 4 clusters with 25 % in each cluster.
- 4 clusters with 10 % 20 % 30 % 40 %.
- 10 clusters with 10 % in each cluster.
- 10 clusters with 4 clusters containing 5 % each, 3 clusters containing 10 % each, 2 clusters containing 15 % each and 1 cluster containing 20 %.

6.1.1 Execution Time. Figure 12 shows the execution time of the SC-Clust algorithm. We can see that the execution time increases logarithmically when the number of modules increases. This is valid for all shapes and all cluster distributions. The reason is obvious. The increase in the number of clusters directly affects the execution time as explained in section 5.2 because as the number of clusters becomes greater, the number of cuts to be found increases. Moreover, the execution time is also affected by the shape and diameter of the system. When the diameter of the ensemble increases and its density decreases, the execution time increases; as can be seen in Figure 12, the humanoid shape requires more time than the other shapes. In addition, when the number of clusters is the same and the clusters sizes distribution differ, the execution time is affected due to the additional number of cuts (see in Figure 14) used to satisfy the size-constraint and the search space to find these cuts which vary according to the clusters sizes.

6.1.2 Communication Load. The communication load is shown in Figure 13. The number of exchanged messages for all shapes increases linearly when the number of modules in the system increases. It also increases when the number of clusters becomes larger due to the messages needed to find the cuts. The communication load complexity in Section

5.1 depends on the number of modules and the connections between modules. The random shape presents the largest number of connections between its modules; thus, it requires a larger number of exchanged messages. Moreover, when sizes distributions with the same number of clusters differ, it slightly affects the number of exchanged messages, which are needed to find additional cuts and join branches to satisfy the size constraint.

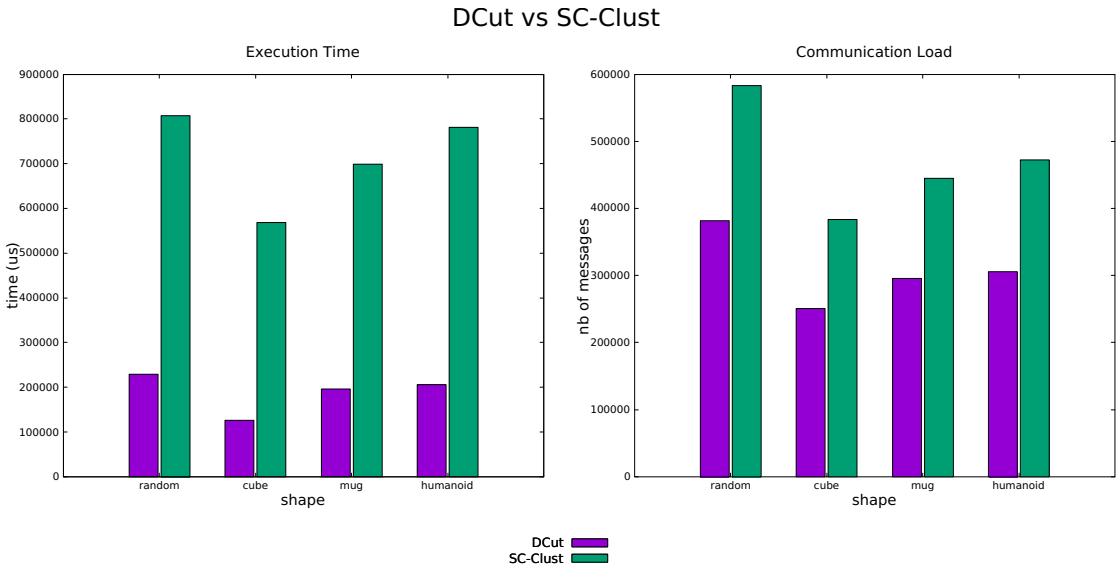


Fig. 15. Comparing DCut and SC-Clust

6.1.3 Additional Cuts. We recall that additional cuts are needed when the resulted cluster size after an initial cut does not satisfy the size constraint. So, additional cuts are performed until the cluster size is equal to the desired size. Figure 14 shows the number of additional cuts that have a direct impact on execution time and communication load. It can be seen that when the number of clusters becomes larger, the number of additional cuts needed increases. Furthermore, it is not affected by the number of modules in the system. It is directly affected by the formation of the MST, which in its turn affects by the geometrical aspects of the ensemble and not its size. Therefore, it can be arbitrary for the same number of clusters with different size distributions since finding a cut that results in a cluster with a size equal to the desired size depends on finding a cut module with a sub-tree size equal to the desired size, which highly depends on the structure of the MST.

6.2 Comparing DCut with SC-Clust

Here, we compare the DCut algorithm with SC-Clust. Figure 15 compares DCut with SC-Clust in terms of execution time and communication load on the shapes of Figures 10 and 11 with the same number of modules for each shape and 4 clusters. As seen in Figure 15, the SC-Clust requires more exchanged messages on all shapes since for each cut, additional cuts may be needed to satisfy the size constraint. As for the execution time, the amount needed by SC-Clust is significantly higher. The reason is that the DCut algorithm finds cuts in parallel in case of $k > 3$. On the other hand, finding cuts in SC-Clust is completely sequential: finding a cluster V_i cannot begin before the cluster V_{i-1} has been

found. In addition, SC-Clust requires $k-1+a$ cuts to obtain k clusters where a is the number of additional cuts. DCut requires $k-1$ cuts.

7 CONCLUSION AND FUTURE WORKS

In this work, we proposed SC-Clust, a fully distributed size-constrained clustering algorithm based on graph cuts. It assembles modules with neighbor-to-neighbor communication in a large scale modular robot into clusters of given sizes to enhance the self-reconfiguration of modular robot-based programmable matter using cluster-based methods to increase the parallelization of movements. To the best of our knowledge, it is the first distributed tree-based clustering algorithm with a size-constraint in the literature. We evaluated our algorithm on multiple shapes with different geometrical properties while varying the number of modules, the number of clusters, and the cluster sizes. The results show that our algorithm is scalable and efficient with $O(nlogn)$ time and communication complexity.

In the future, we aim to implement our algorithm on real large scale modular robotics ensembles. Then we intend to study the clusters' leaders positions to optimize inter and intra-cluster communication. In addition, we aim to control the shape and position of each cluster in the initial shape to reduce the number of modules in blocking positions in order to facilitate the transition to the goal shape. Furthermore, we seek to show the improvement that clustering can yield to the self-reconfiguration process and work on proposing cluster-based self-reconfiguration algorithms.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Hamilton Wilfried Yves Adoni, Tarik Nahhal, Moez Krichen, Brahim Aghezzaf, and Abdeltif Elbyed. 2020. A survey of current challenges in partitioning and processing of graph-structured data in parallel and distributed systems. , 495–530 pages. https://doi.org/10.1007/s10619-019-07276-9

[2] Wilfried Yves Hamilton Adoni, Tarik Nahhal, Moez Krichen, Abdeltif El byed, and Ismail Assayad. 2020. DHPV: a distributed algorithm for large-scale graph partitioning. *Journal of Big Data* 7, 1 (dec 2020), 1–25. https://doi.org/10.1186/s40537-020-00357-y

[3] M. Mehdi Afsar and Mohammad-H. Tayarani-N. 2014. Clustering in sensor networks: A literature survey. *Journal of Network and Computer Applications* 46 (2014), 198 – 226. https://doi.org/10.1016/j.jnca.2014.09.005

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

[5] Shahab U Ansari, Masroor Hussain, Suleman Mazhar, Tareq Manzoor, Khalid J Siddiqui, Muhammad Abid, and Habibullah Jamal. 2019. Mesh partitioning and efficient equation solving techniques by distributed finite element methods: A survey. *Archives of Computational Methods in Engineering* 26, 1 (2019), 1–16.

[6] Jad Bassil, Mouhamad Moussa, Abdallah Makhoul, Benoit Piranda, and Julien Bourgeois. 2020. Linear Distributed Clustering Algorithm for Modular Robots Based Programmable Matter. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[7] Cedric Berenger, Peter Niebert, and Kevin Perrot. 2018. Balanced connected partitioning of unweighted grid graphs. In *Leibniz International Proceedings in Informatics, LIPIcs*, Vol. 117. https://doi.org/10.4230/LIPIcs.MFCS.2018.39

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

[9] Lélia Blin and Franck Butelle. 2001. A very fast (linear time) distributed algorithm, on general graphs, for the minimum-weight spanning tree. In *OPODIS 2001*. SUGER, 113–124.

[10] Nicolas Bulla Cruz, Nadia Nedjah, and Luiza Mourelle. 2017. Robust distributed spatial clustering for swarm robotic based systems. *Applied Soft Computing Journal* 57 (2017), 727–737. https://doi.org/10.1016/j.asoc.2016.06.002

[11] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. *Recent Advances in Graph Partitioning*. Springer International Publishing, Cham, 117–158. https://doi.org/10.1007/978-3-319-49487-6_4

[12] Gianni Di Caro, Frederick Ducatelle, and Luca Maria Gambardella. 2012. A fully distributed communication-based approach for spatial clustering in robotic swarms. *Proceedings of the 2nd Autonomous Robots and Multirobot Systems Workshop (ARMS), affiliated with the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2012), 153–171.

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

[14] Sergio Diaz and Diego Mendez. 2019. Dynamic minimum spanning tree construction and maintenance for Wireless Sensor Networks. *Revista Facultad de IngenierÃa Universidad de Antioquia* (12 2019), 57 – 69. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0120-62302019000400057&nrm=iso

[15] Ralf Diekmann, Robert Preis, Frank Schlimbach, and Chris Walshaw. 2000. Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Parallel Comput.* 26, 12 (2000), 1555–1581.

[16] Ayan Dutta, Prithviraj Dasgupta, and Carl Nelson. 2016. A bottom-up search algorithm for size-constrained partitioning of modules to generate configurations in modular robots. *Web Intelligence* 14, 1 (2016), 67–82. https://doi.org/10.3233/WEB-160332

[17] Ayan Dutta, Raj Dasgupta, José Baca, and Carl A. Nelson. 2015. Spanning Tree Partitioning Approach for Configuration Generation in Modular Robots. In *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida, USA, May 18-20, 2015*, Ingrid Russell and William Eberle (Eds.). AAAI Press, 360–365. http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10447

[18] Ayan Dutta, Vladimir Ufimtsev, Asai Asaithambi, and Emily Czarnecki. 2019. Coalition Formation for Multi-Robot Task Allocation via Correlation Clustering. *Cybernetics and Systems* 50, 8 (2019), 711–728. https://doi.org/10.1080/01969722.2019.1677334 arXiv:https://doi.org/10.1080/01969722.2019.1677334

[19] Neng Fan and Panos M Pardalos. 2010. Linear and quadratic programming approaches for the general graph partitioning problem. *Journal of Global Optimization* 48, 1 (2010), 57–71.

[20] Andreas Emil Feldmann. 2013. Fast balanced partitioning is hard even on grids and trees. *Theoretical Computer Science* 485 (may 2013), 61–68. https://doi.org/10.1016/j.tcs.2013.03.014 arXiv:1111.6745

[21] R. G. Gallager, P. A. Humblet, and P. M. Spira. 1983. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5, 1 (1983), 66–77. https://doi.org/10.1145/357195.357200

[22] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)* (first edition ed.). W. H. Freeman. http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455

[23] John R Gilbert, Gary L Miller, and Shang-Hua Teng. 1998. Geometric mesh partitioning: Implementation and experiments. *SIAM Journal on Scientific Computing* 19, 6 (1998), 2091–2110.

[24] Mario Gnägi and Philipp Baumann. 2021. A matheuristic for large-scale capacitated clustering. *Computers & Operations Research* 132 (2021), 105304.

[25] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. 2018. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. 119–128.

[26] William W Hager, Dzung T Phan, and Hongchao Zhang. 2013. An exact algorithm for graph partitioning. *Mathematical Programming* 137, 1 (2013), 531–556.

[27] Adam Hayes, A. Martinoli, and Rodney Goodman. 2003. Swarm Robotic Odor Localization: Off-Line Optimization and Validation with Real Robots. *Robotica* 21 (07 2003). https://doi.org/10.1017/S0263574703004946

[28] Dong Huang, Chang-Dong Wang, Jian-Sheng Wu, Jian-Huang Lai, and Chee-Keong Kwoh. 2019. Ultra-scalable spectral clustering and ensemble clustering. *IEEE Transactions on Knowledge and Data Engineering* 32, 6 (2019), 1212–1226.

[29] Sanza Kazadi, M. Chung, B. Lee, and R. Cho. 2004. On the dynamics of clustering systems. *Robotics and Autonomous Systems* 46 (01 2004), 1–27. https://doi.org/10.1016/j.robot.2003.10.001

[30] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. 2015. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative robots and sensor networks 2015* (2015), 31–51.

[31] Jin Kim, Inwook Hwang, Yong-Hyuk Kim, and Byung-Ro Moon. 2011. Genetic approaches for graph partitioning: a survey. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 473–480.

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

[33] Jialu Liu and Jiawei Han. 2018. Spectral clustering. In *Data Clustering*. Chapman and Hall/CRC, 177–200.

[34] Yaoyao Liu, Ping Guo, and Yi Zeng. 2022. MEACCP: A membrane evolutionary algorithm for capacitated clustering problem. *Information Sciences* 591 (2022), 319–343. https://doi.org/10.1016/j.ins.2022.01.032

[35] CH Martin. 2012. *Spectral clustering: a quick overview*. Ph.D. Dissertation. PhD thesis.

[36] Ali Mashreghi and Valerie King. 2021. Broadcast and minimum spanning tree with o (m) messages in the asynchronous CONGEST model. *Distributed Computing* 34, 4 (2021), 283–299.

[37] Petra Mazdin and Bernhard Rinner. 2021. Distributed and Communication-Aware Coalition Formation and Task Assignment in Multi-Robot Systems. *IEEE Access* 9 (2021), 35088–35100.

[38] Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2014. Partitioning complex networks via size-constrained clustering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8504 LNCS (2014), 351–363. https://doi.org/10.1007/978-3-319-07959-2_30 arXiv:1402.3281

[39] Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2017. Parallel graph partitioning for complex networks. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2625–2638.

[40] Mohamad Moussa, Benoit Piranda, Abdallah Makhoul, and Julien Bourgeois. 2021. Cluster-Based Distributed Self-reconfiguration Algorithm for Modular Robots. *Lecture Notes in Networks and Systems* 225 LNNS (2021), 332–344. https://doi.org/10.1007/978-3-030-75100-5_29

[41] Proshikshya Mukherjee. 2020. 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*. IGI Global, 77–85.

[42] John M Mulvey and Michael P Beck. 1984. Solving capacitated clustering problems. *European Journal of Operational Research* 18, 3 (1984), 339–348.

[43] André Naz, Benoît Piranda, Thadeu Tucci, Seth Copen Goldstein, and Julien Bourgeois. 2018. Network characterization of lattice-based modular robots with neighbor-to-neighbor communications. In *Distributed Autonomous Robotic Systems*. Springer, 415–429.

[44] Vitaly Osipov and Peter Sanders. 2010. n-Level Graph Partitioning. In *European Symposium on Algorithms*. Springer, 278–289.

[45] Gopal Pandurangan, Peter Robinson, Michele Scquizzato, et al. 2018. The distributed minimum spanning tree problem. *Bulletin of EATCS* 2, 125 (2018).

[46] A. Pietrabissa and F. Liberati. 2019. Dynamic distributed clustering in wireless sensor networks via Voronoi tessellation control. *Internat. J. Control* 92, 5 (2019), 1001–1014. https://doi.org/10.1080/00207179.2017.1378441 arXiv:https://doi.org/10.1080/00207179.2017.1378441

[47] C. Pinciroli, R. O'Grady, A. L. Christensen, and M. Dorigo. 2009. Self-organised recruitment in a heteregeneous swarm. In *2009 International Conference on Advanced Robotics*. 1–8.

[48] Benoît Piranda and Julien Bourgeois. 2018. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Auton. Robots* 42, 8 (2018), 1619–1633. https://doi.org/10.1007/s10514-018-9710-0

[49] Maria Predari and Aurélien Esnard. 2016. A k-way greedy graph partitioning with initial fixed vertices for parallel applications. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 280–287.

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

[51] Fatemeh Rahimian, Amir H Payberah, Sarunas Girdzijauskas, Mark Jelasity, and Seif Haridi. 2015. A distributed algorithm for large-scale graph partitioning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10, 2 (2015), 1–24.

[52] A.A.G. Requicha, H.B. Voelcker, and University of Rochester. Production Automation Project. 1977. *Constructive Solid Geometry*. Production Automation Project, University of Rochester. https://books.google.fr/books?id=hG2lngEACAAJ

[53] Erik Rolland, Hasan Pirkul, and Fred Glover. 1996. Tabu search for graph partitioning. *Annals of operations research* 63, 2 (1996), 209–232.

[54] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer Science Review* 1, 1 (2007), 27 – 64. https://doi.org/10.1016/j.cosrev.2007.05.001

[55] Stephan Scheuerer and Rolf Wendolsky. 2006. A scatter search heuristic for the capacitated clustering problem. *European Journal of Operational Research* 169, 2 (2006), 533–547.

[56] Junming Shao, Qinli Yang, Zhong Zhang, Jinhu Liu, and Stefan Kramer. 2018. Graph Clustering with Local Density-Cut. In *Database Systems for Advanced Applications*, Jian Pei, Yannis Manolopoulos, Shazia Sadiq, and Jianxin Li (Eds.). Springer International Publishing, Cham, 187–202.

[57] Horst D Simon. 1991. Partitioning of unstructured problems for parallel processing. *Computing systems in engineering* 2, 2-3 (1991), 135–148.

[58] Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. 2019. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems* 120 (2019), 103242. https://doi.org/10.1016/j.robot.2019.07.012

[59] Pierre Thalamy, Benoît Piranda, André Naz, and Julien Bourgeois. 2021. VisibleSim: A behavioral simulation framework for lattice modular robots. *Robotics and Autonomous Systems* (2021), 103913. https://doi.org/10.1016/j.robot.2021.103913

[60] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. 2017. Efficient scene encoding for programmable matter self-reconfiguration algorithms. *Proceedings of the ACM Symposium on Applied Computing* Part F1280 (2017), 256–261. https://doi.org/10.1145/3019612.3019706

[61] Thadeu Tucci, Benoît Piranda, and Julien Bourgeois. 2018. A Distributed Self-Assembly Planning Algorithm for Modular Robots. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018*. 550–558.

[62] Mostafa Wahby, Julian Petzold, Catriona Eschke, Thomas Schmickl, and Heiko Hamann. 2019. Collective Change Detection: Adaptivity to Dynamic Swarm Densities and Light Conditions in Robot Swarms. In *Artificial Life Conference Proceedings*. https://doi.org/10.1162/isal_a_00233

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

[64] Yaowei Yan, Yuchen Bian, Dongsheng Luo, Dongwon Lee, and Xiang Zhang. 2019. Constrained local graph clustering by colored random walk. In *The world wide web conference*. 2137–2146.

[65] Yu Zhang, Lynne E Parker, and Subbarao Kambhampati. 2014. Coalition coordination for tightly coupled multirobot tasks with sensor constraints. In *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 1090–1097.

[66] Qing Zhou, Una Benlic, Qinghua Wu, and Jin-Kao Hao. 2019. Heuristic search to the capacitated clustering problem. *European Journal of Operational Research* 273, 2 (2019), 464–487. https://doi.org/10.1016/j.ejor.2018.08.043