Cluster-Based Distributed Self-Reconfiguration Algorithm for Modular Robots

Mohamad Moussa, Benoit Piranda, Abdallah Makhoul and Julien Bourgeois

Abstract Modular robots are automated modules that can change their morphology self-sufficiently and progressively for control or reconfiguration purposes. Self-reconfiguration is a very challenging problem in modular robots systems. Existing algorithms are complex and not suitable for low resources devices. In this paper, we propose a parallel and fully distributed cluster-based algorithm to convert a set of blocks/modules in a new geometrical configuration. We proposed a cluster based self-reconfiguration algorithm. The main idea is to study the impact of clustering on the self-reconfiguration problem. The modules in each cluster remain together and try to move in order to find the final configuration. Based on this concept, our algorithm operates in parallel in each cluster to fasten reconfiguration process and ultimately the set of blocks will reach the desirable shape. We evaluate our algorithm on the centimeter-scale sliding blocks, developed in the Smart Blocks project in a simulator showing the entire reconfiguration process in real-time. We show the effectiveness of our algorithm, especially the benefits of clustering in self-reconfiguration task by comparing performance of both approaches (with and without clustering) while varying the number of clusters.

1 INTRODUCTION

Programmable matter consists of tactile materials that have physical form dynamically modifiable. They are able to transform on its own, depending on program controlled. It is essentially composed from materials and components that have capabilities to be programmed and change their shape property on demand. The main approach to implement programmable matter is to build a huge self-reconfigurable robot consisting of a bunch of modules. Each of these modules has some com-

Mohamad Moussa, Benoit Piranda, Abdallah Makhoul and Julien Bourgeois
FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS, Montbéliard, France. e-mail: firstname.lastname@univ-fcomte.fr

putation, sensing and communication capabilities. The modules can detach, move independently and connect to each other to form new configurations.

Modular robots are distributed systems made up of identical connected modules which are able to coordinate together in order to perform complex tasks that can revolutionize everything starting from surgical applications, to transportation applications or space exploration [1]. The main challenge in modular robots is self-reconfiguration problem which enables the modules to modify their structure from one arbitrary initial shape to another (goal shape) [15]. This problem is very challenging because of the very high number of possible configurations and a fixed set of rules does not work for all possible situations. It poses several challenges. Firstly, this process is very slow due to the limited number of modules which can move simultaneously without interfering which results in limited parallel possible motions and other system constraints. Secondly, self-reconfiguration requires a coordination between modules during motion in order to avoid any collision by adapting a set of rules that does not work for all situations since some motions requires cooperation outfaced by motion algorithms [11]. However,existing solutions for the self-reconfiguration problem propose several algorithms using search-based and control-based techniques [14, 5].The basic concept is to search for a goal configuration in the modular robots configuration space. These solutions are still very complex and not suitable for modular robots with low computation and energetic resources. One solution to fasten self-reconfiguration process is clustering. Grouping modules into clusters and carrying out some tasks in cluster-based approaches can improve the effectiveness in terms of execution time and communication cost. Therefore, the partitioning of the modules into clusters and the parallel displacement of the modules in each partition will accelerate the convergence of the modules to the goal shape [2]. In fact, nodes would care only for executing tasks at intra-cluster level and will not be affected by inter-cluster changes, thus reducing scope of inter-cluster interactions. Clustering can yield improvements to the execution of tasks related to modular robots' programmable matter such as self-reconfiguration where modules forming specific parts of the initial shape can form one cluster. Therefore, it requires less reconfiguration adjustments to attain a more or less similar parts of the goal shape. Hence, here lies the importance and effectiveness of clustering.

Our objective in this paper is to show the impact and the efficiency of clustering and parallelization on the self-reconfiguration problem. We propose a cluster-based distributed and parallel self-reconfiguration algorithm that scales to large modular robot systems in order to speed up the reconfiguration of the modular robot systems from an initial shape to a goal one. Our aim in this paper is not to propose a clustering technique, but to provide a new cluster-based distributed self-reconfiguration algorithm while using our clustering techniques as described in [2]. The number of clusters is specified based on the goal configuration and based on the density cut algorithm [13]. The proposed algorithm consists in sliding the not well-placed robots in the interior the current configuration in order to fill target free cells. For each not well-placed robot, the method compute a gradient distance from free goal cells, then construct a path for each not well-placed robot to a free cell which does not cross the path of another robot, and then moves successively horizontal and vertical

lines of robots from the end to the begin of the path. These steps are repeated until all the goal cells are filled. The remainder of this paper is organized as follows. In Section 2 we provide a short description of the related work. Section 3 presents the system model and some assumptions. Section 4 provides the details of our self-reconfiguration algorithm. The experimental results are presented in Section 5 and Section 6 concludes our paper.

2 RELATED WORKS

In this paper, we consider modular self-reconfigurable robot made from centimeter-size cubes (blocks) able to move [10]. Works with sliding cubes was widely used in the literature [6, 7, 8]. In [6], it is showed that using homogeneous Limited Sliding-Cube modules could be performed in quadratic time using meta-modules and tunneling motions. The method's advantage is a high degree of motion parallelism. It was then further investigated in [7] and a novel approach was proposed for heterogeneous modules. Moreover, a tunneling-based reconfiguration algorithm was later proposed in [8] to remove the limitation on the arrangement of start and goal configurations by employing tunneling techniques.

Self-reconfiguration is one of the most critical challenges for modular robots. It may seem easy to solve, however, it is one of the most challenging issues. Several solutions have been proposed in the literature and there is no general solution has been appeared yet. For instance, in [4] the authors propose a self-reconfiguration in two dimensions. They propose the idea of module substraction, where unwanted modules remove themselves from the system without external intervention to attend the desired configuration. The algorithm is composed of two parts, sequential one for unwanted modules selection and parallel one for modules movements. The algorithm has quadratic complexity and does not work in all situations especially in hollow structures. In [11], the authors propose a distributed algorithm for reconfiguration of lattice-based modular robots systems made from the same cubic modules used in this paper. This iterative algorithm can transform various large modular robots, beginning from a filled shape towards a target one by using motion rules and creating a train of modules made up of the border of the configuration, which will be moved gradually in order to reach the target shape. In cite [3] the authors introduced geometric algorithms for copying,rotating, scaling and reflecting a given polyomino and they also provided an algorithm for constructing a bounding box surrounding a polyomino.

Most of existing Self-reconfiguration algorithms are based on sequential movements which results in dramatically increasing the reconfiguration process's duration. In fact, there exists an obvious trade-off between a high degree of motion parallelism and full convergence to the goal shape due to collisions and deadlocks that may appear with increasing number of modules moving simultaneously. In the literature, existing methods with parallel motions suffer from a limitation in the number of parallel motions depending on the approach used. In [9],a scaffolding technique

is proposed which allows this parallelism while avoiding collisions. The contribution of this paper is to propose a fully parallel and distributed self-reconfiguration algorithm using clustering approaches with increase in motion parallelism for accelerating self-reconfiguration that can manage almost to build any goal shape given the initial shape divided into a pre-defined number of clusters.

3 SYSTEM MODEL AND ASSUMPTIONS

In this paper, we consider modular self-reconfigurable robot made from centimeter-size cubes (blocks) able to move [10]. Blocks are equipped wih a micro-electro-mechanical-system (MEMS) actuator array in the upper face to move the objects and they are powered externally. A block can slide along the border of one of its neighbors. Each block can connect to up to 4 neighbors positioned on each of its faces. They can communicate together by using neighbor-to-neighbor message passing.

We assume that these blocks are placed in a simple 2D Cartesian coordinate system along $\vec{x}$ and $\vec{y}$ axes. Each cell can either be occupied by a block or empty. It is assumed that blocks can recognize their attached neighbors at any given time and moving blocks cannot communicate with any other ones. We assume that, at any time, a block can detect if there exists a surrounding cell $\exists\, f \in \{East; West; North; South\}$ that should be filled.

Moreover, each cluster is given its goal shape G, also know by cluster members. We consider that this target shape is fixed during self-reconfiguration process, so no new modules can join or quit any cluster.

Furthermore, the number of blocks in each cluster must be sufficient enough (greater or at least equal) in order to be able to build the cluster's target shape.

4 CLUSTER BASED SELF-RECONFIGURATION

4.1 Agents

Throughout this paper, we consider the *Cluster Head* agent for explaining the algorithm. The *Cluster Head* is a special treatment to manage a cluster, the module with this agent acts as a local coordinator guiding the transformation process to form the target shape of its own cluster. The *Cluster Head* agent is dynamic and can change throughout the algorithm. A new *Cluster Head* module can be elected in each cluster during reconfiguration, it is defined as:

Definition 1. (Cluster Head Coordinator)

Let $N(M, D)$ be the neighbor cell of M in the direction D, and G be the goal configuration. M is a *Cluster Head* coordinator, if it exists: D verifying $(N(M, D) \in G) \wedge (N(M, D)$ is empty$)$.

4.2 Principle

In previous work [10], a module could block all the others by the path it defined to move to an empty cell. Our algorithm allows reconfiguring a set of connected blocks divided into clusters considering that each cluster is self-reconfigured to form its corresponding part of the overall goal shape. This algorithm consists in exchanging messages with data transmission in the network made up of connected blocks. Each block is considered as an independent robot equipped with its own processing unit and memory able to execute the program instructions. A first stage affects a position to each each block in the grid and the goal shape for its cluster. Algorithm 2 illustrates how cluster-based self-reconfiguration is handled by the *Cluster Head* block.

At each iteration, one *Cluster Head* module is elected in each cluster among potential ones. This block will trigger cluster building (see Algorithm 2) and then propagate useful information, i.e. distance, to cluster member blocks in order to start the motion and self-reconfigure into cluster's goal shape. Also, it will schedule activities in the cluster and will be in charge of handling intra-cluster interactions. This *Cluster Head* block is not static during the self-reconfiguration algorithm. It can change dynamically at the end of each blocks' movement. It is the case when the previous *Cluster Head* module does not verify the Definition 1 anymore, in other words it no longer has any empty surrounding cell to fill. *Cluster Head* block will take in charge of finding the motion path starting from the block not belonging to goal shape with maximum distance in order to start self-reconfiguration into cluster's target configuration.

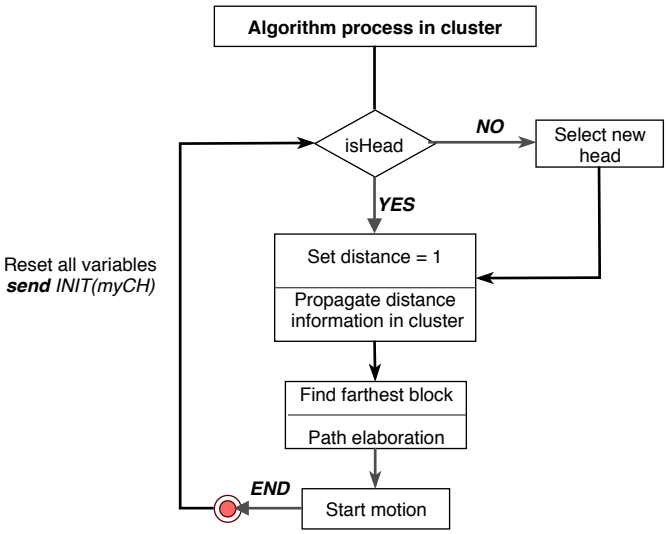


Fig. 1: Algorithm of cluster-based reconfiguration.

Algorithm 1: Distributed algorithm for the construction of a *cluster's tree*.

```
   myCH // Cluster Id
   parentCluster
   subTreeSize // per cluster
   neighbors // set of neighbours
   childrenCluster // set of children
   nbExpectedAnswers
   initialized // boolean value
 1 if isHead() then
 2 |  TriggerCluster()
 3 Msg Handler CLUSTER_GO(msgCH, msgDistance):
 4 |  initialized ← false
 5 |  if myCH = msgCH then
 6 |  |  distance ← msgDistance + 1
 7 |  |  if parentCluster = null then
 8 |  |  |  parentCluster ← sender
 9 |  |  |  nbExpectedAnswers ← neighbours.size − 1
10 |  |  |  if nbExpectedAnswers = 0 then
11 |  |  |  |  subTreeSize ← 1
12 |  |  |  |  send CLUSTER_BACK(myCH,1) to sender
13 |  |  |  else
14 |  |  |  |  foreach nId ∈ neighbours do
15 |  |  |  |  |  if nid ≠ Mj.id then
16 |  |  |  |  |  |  send CLUSTER_GO(myCH) to nId
17 |  |  else
18 |  |  |  send CLUSTER_BACK(myCH,−1) to sender
19 |  else
20 |  |  send CLUSTER_BACK(myCH,−1) to sender
21 Msg Handler CLUSTER_BACK(msgCH, msgSize):
22 |  nbExpectedAnswers ← nbExpectedAnswers − 1
23 |  if msgSize ≠ −1 and myCH = msgCH then
24 |  |  childrenCluster.add(sender)
25 |  |  subTreeSize ← subTreeSize + msgSize
26 |  if nbExpectedAnswers = 0 then
27 |  |  subTreeSize ← subTreeSize + 1
28 |  |  if parentCluster ≠ null then
29 |  |  |  send CLUSTER_BACK(myCH,subTreeSize) to sender
30 |  |  else
   |  |  |  // head
31 |  |  |  foreach child ∈ childrenCluster do
32 |  |  |  |  send REQUEST_MAX(myCH) to child
```

The algorithm is detailed in the following. It allows to start the motion events once all the computation is done on all blocks. Mainly, our algorithm consists of 3 major steps. As shown in Figure 1, these three steps will be executed in parallel and repeated in each cluster until the cluster's goal shape is reached.

1. Set blocks distance in each cluster.
2. Find the block B_{max} with maximum distance not in the goal shape.

3. Displacement of blocks by establishing motion path following decreasing distance order starting from B_{max}.

4.3 Self-reconfiguration algorithm

We use message passing system to communicate and transmit data between connected blocks. As aforementioned, our reconfiguration approach follows three main steps we detail bellow.

Algorithm 2: Distributed control algorithm for the *Cluster Head* agent.

$neighbours$ // maximum 4 neighbours
$head \leftarrow true$
$bmaxId$ // farthest block's id not in goal shape
$bmaxDistance$ // bmaxId's distance
$myCH$ // Cluster Id
1 **while** *head* **do**
2 | $TriggerCluster()$
3 | $isHead()$
4 **Function** *TriggerCluster()*:
5 | setNeighbours()
6 | $distance \leftarrow 1$
7 | $nbExpectedAnswers \leftarrow neighbours.size()$
8 | **foreach** $nId \in neighbours$ **do**
9 | | **send** CLUSTER_GO($myCH$,$distance$) to nId
10 **Event Handler** *TRANSLATION_END*:
11 | setNeighbours()
12 | $nbExpectedAnswers \leftarrow neighbours.size$
13 | **foreach** $nId \in neighbours$ **do**
14 | | **reset** all variables
15 | | **send** INIT($myCH$) to nId
16 **Function** *isHead()*:
17 | return $(\exists emptyCell \in \{West, North, East, South\} \wedge emptyCell \in goal)$

4.3.1 Defining distances to blocks in each cluster

- *CLUSTER_GO*: Sent by *Cluster Head* block to its connected neighbors to start construction of its own cluster's tree while distributing distance. When received by a cluster member blocks, it defines its distance as being that received in the message plus 1.

- *CLUSTER_BACK*: Response to *CLUSTER_GO* sent by a block to either acknowledge receipt of distance and that all its neighbors have successfully set their distance, or to notify the sender that it does not belong to the same cluster.

Once cluster tree is built, the block with maximum distance and not in goal shape must be found, then the motion starts by following the decreasing path order of distance. Algorithm 1 illustrates the first step in details. Cluster tree construction is done by calling the helper function *TriggerCluster*. Starting from a *Cluster Head* module in each cluster, a structure tree rooted at the *Cluster Head* node is built with distance assignment using two messages *CLUSTER_GO* and *CLUSTER_BACK*.

4.3.2 Finding B_{max}

B_{max} is defined as the block with maximum distance and not belonging to cluster's goal shape. The root (*Cluster Head*) in each cluster will then launch the search for B_{max} by sending *REQUEST_MAX* to its children through its cluster tree. This message will be transmitted locally across all cluster members. When received by a block already in the cluster's goal shape this message is propagated to children blocks until leaf ones, otherwise block is not in cluster's goal shape and should send its id and $distance$ back to tree's root using *SEND_MAX* message which is progressively returned to the parent until it reaches the root.Thus, the root will maintain the block's id ($bmaxId$) not in goal shape having the maximum distance ($bmaxDistance$) value.

4.3.3 Blocks displacement

The motion path can be easily established. Starting from B_{max} block and following decreasing order distance by gradually crossing $parentCluster$ blocks benefiting from the tree structure. This way back leads us to the *Cluster Head* block that has empty spot to fill, once *Cluster Head* is reached the displacement of the blocks starts where each block has to replace and move to its $parentCluster$'s position.

Using programming, the blocks are able to deal with state changes locally and to handle different events, such as connection/disconnection of a new block, or the end of a motion. When a *Cluster Head* block finishes motion, i.e, all the blocks on the path succeeded in moving to its $parentCluster$'s position, it resets all its variables and sends a message of type *INIT(myCH)* to all neighbors in order to notify the blocks to reinitialize all their variables with a new iteration taking place. Then, this *Cluster Head* must verify if it can still acts as *Cluster Head* block or not. If it is the case, old *Cluster Head* triggers a new iteration and repeat the process, otherwise a new *Cluster Head* takes over.This reconfiguration algorithm is being repeated in each cluster, until all cluster's blocks converge towards goal shape, hence cluster's target shape is well formed.

Algorithm 3: Message Handler for any module M_i.

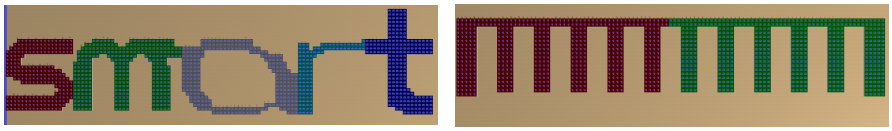
```
   wellPlaced // module ∈ goal shape or not
1  Msg Handler REQUEST_MAX(msgCH):
2  | if ¬wellPlaced and myCH = msgCH then
3  | | send SEND_MAX(M_i.id,distance) to parentCluster
4  | foreach child ∈ childrenCluster do
5  | | send REQUEST_MAX(msgCH) to child
6  Msg Handler SEND_MAX(msgId,msgDistance):
7  | if parentCluster ≠ null then
8  | | send SEND_MAX (msgId,msgDistance) to parentCluster
9  | else
10 | | if bmaxDistance < msgDistance then
11 | | | bmaxDistance ← msgDistance
12 | | | bmaxId ← msgId
13 | | | start motion
14 Msg Handler INIT(msgCH):
15 | if myCH ≠ msgCH then
16 | | send INIT_ACK() to sender
17 | else
18 | | if !initialized then
19 | | | reset all variables
20 | | | toAnswer ← sender
21 | | | nbExpectedAnswers ← neighbours.size − 1
22 | | | if nbExpectedAnswers = 0 then
23 | | | | send INIT_ACK () to sender
24 | | | else
25 | | | | foreach nId ∈ neighbours do
26 | | | | | send INIT(myCH) to nId
27 | | else
28 | | | send INIT_ACK () to sender
29 Msg Handler INIT_ACK():
30 | nbExpectedAnswers ← nbExpectedAnswers − 1
31 | if nbExpectedAnswers = 0 and toAnswer ≠ null then
32 | | send INIT_ACK () to toAnswer
33 | else
34 | | if isHead() then
35 | | | TriggerCluster()
36 | | else
37 | | | foreach nid in neighbours do
38 | | | | send NEW_HEAD (id) to nid
39 Msg Handler NEW_HEAD(msgId):
40 | if M_i.id = msgId then
41 | | TriggerCluster(); // new Head
42 | foreach nid ∈ neighbours do
43 | | if nid ≠ M_j.id then
44 | | | send NEW_HEAD (msgId) to nid; // except sender
```

4.4 Message evaluation

The proposed cluster-based self-reconfiguration algorithm operates by using a message–passing model. Our algorithm involves only local and intra-cluster interactions between blocks. The messages are used in the system for communication purposes, where a block sends a message to a neighbor one asking for data. We note m the maximum number of connections a module can have, n the number of blocks, and k the number of clusters. During the first step, except for the *Cluster Head*, when a block receives a *CLUSTER_GO* message, it will send it back to all its neighbours except the sender hence the number of *CLUSTER_GO* messages exchanged is $O(k \times n \times (m-1))$. Each reception of a *CLUSTER_GO* is followed by a response of *CLUSTER_BACK* message. Therefore, the total number of messages exchanged during step 1 is $O(2k \times n(m-1))$. The total number of messages exchanged during the last two steps, by using $\langle$*REQUEST_MAX, SEND_MAX*$\rangle$ and $\langle$*INIT, INIT_ACK*$\rangle$ is $O(2k \times m)$ at each step. Then, the new *Cluster Head* selection requires $O(k \times m)$ messages. Therefore, the total number of messages exchanged at each iteration's execution is $O(k \times n \times m)$.

5 SIMULATION AND RESULTS

We implemented our proposed algorithm in *VisibleSim* [12], a simulation and development environment that supports multi-target large scale modular robot systems (*Smart Blocks*, *Blinky Blocks*, *Claytronics*...). We conducted extensive simulations to evaluate the performance of the proposed algorithm. In order to visualize clustering impact, we focus on comparing cluster-based and without clustering algorithms with varying number of clusters and ensemble sizes. Two types of complex reconfiguration scenarios have been carried out on several shapes while varying the number of modules in the ensemble. For each scenario, we generated different versions of the goal configurations using different scales ranging from hundreds to ten thousands of blocks. The first scenario formed by the text: 'smart' with various number of blocks. 'Smart' word is composed from five letters, one idea is to let each cluster form one letter as shown in Figure 2a. In fact, we have five clusters, each corresponding to one letter with a specific color. We consider that blocks are initially divided into 5 clusters using [2]. Each block belongs to a cluster and the cluster's goal shape is stored in all cluster members.



(a) Smart word formed by 5 clusters.

(b) Large comb formed by 2 clusters.

The second scenario is to build a large comb. We test our algorithm to self-reconfigure large networks of more than 10,000 blocks. In this experiment (cf. Figure 2a), a large set of *Smart Blocks* is assembled presenting very regular volumes of blocks with symmetries. For each experiment, blocks can glow in different colors depending on running program in order to show a state upon receipt of a message, detection of an event or to indicate that it has successfully reached the goal shape. They are also able to display value upon their front face, used in our case to display useful information such as distance and cluster identifiers. In Figure 3, we studied the number of messages exchanged in the two scenarios according to the size of the goal shape. Firstly, using five clusters to form the smart' word with various network sizes (up to 4,800 blocks) and compare it with results obtained without clustering. The total messages exchanged by both algorithms linearly increases with the number of blocks. Obviously, the number of messages exchanged using 5 clusters is much lower without clustering. Secondly, Figure 3 also shows the total number of messages sent to construct a comb. It displays the messages exchanged for a dividing into 2,4 and 8 clusters, versus those exchanged without clustering. Without clustering approach requires the highest number of messages exchanged due to the fact that the whole system is considered as a single cluster, hence a higher number of connections and messages exchanged between blocks. When no cluster is used, the number of messages increases exponentially, while it decreases gradually and becomes linear with the increase in the number of clusters. Hence, in each cluster a lot of messages will be dropped, especially by modules at the boundaries between clusters. As a consequence, this confirms that our algorithm only involves local intra-cluster interactions as announced in previous sections. Figure 3 enables one to easily notice that with the increase of both number of clusters and ensemble size, also the gap between the two approaches is largely increasing.

6 Conclusion and future work

In this paper, we propose a cluster based self-reconfiguration algorithm in a fully distributed fashion. We conducted two types of complex reconfiguration experiments, and we evaluated the algorithm on several shapes while varying the number of clusters and the number of modules in the ensemble. Our proposal has shown to be effective, i.e, self-reconfiguration algorithms can be aided by clustering approaches to speed up reconfiguration process. From obtained results, one can easily notice that the speed of algorithm's convergence with clustering is much way better than without clustering. The results also show that clustering has many benefits including energy-efficiency by reducing communication load (messages exchanged), scalability. As a future work, we aim to improve the proposed algorithm by making many possible motions within each cluster while avoiding collisions, thus achieving a high degree of motion parallelism per cluster. This will surely affect the reconfiguration speed and reduce complexity, but it is a complex task since it involves more interactions at the inter-cluster level.

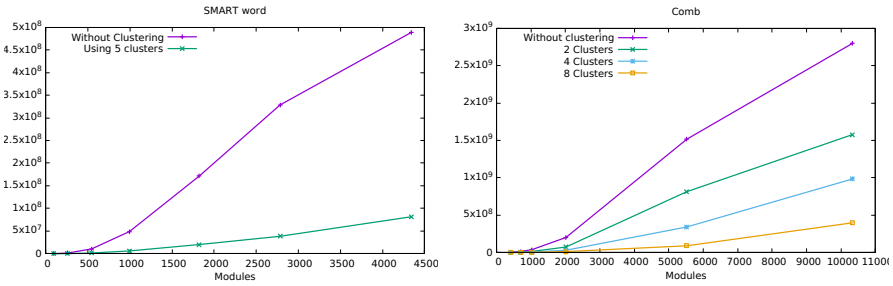


Fig. 3: Number of messages exchanged for 'smart' word using 5 clusters and to build a comb using 2,4 and 8 clusters.

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. Jad Bassil, Mohamad Moussa, Abdallah Makhoul, Benoît Piranda, and Julien Bourgeois. Linear distributed clustering algorithm for modular robots basedprogrammable matter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*.
3. Sándor P Fekete, Robert Gmyr, Sabrina Hugo, Phillip Keldenich, Christian Scheffer, and Arne Schmidt. Cadbots: Algorithmic aspects of manipulating programmable matter with finite automata. *Algorithmica*, pages 1–26, 2020.
4. Matthew D Hall, Anıl Ozdemir, and Roderich Groß. Self-reconfiguration in two-dimensions via active subtraction with modular robots. In *Robotics: Science and System XVI*, 2020.
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*, 127:103467, 2020.
6. H. Kawano. Complete reconfiguration algorithm for sliding cube-shaped modular robots with only sliding motion primitive. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 3276–3283, 2015.
7. Hiroshi Kawano. Tunneling-based self-reconfiguration of heterogeneous sliding cube-shaped modular robots in environments with obstacles. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 825–832, May 2017.
8. Hiroshi Kawano. Distributed Tunneling Reconfiguration of Sliding Cubic Modular Robots in Severe Space Requirements. page 14, October 2018.
9. Jakub Lengiewicz and Paweł Hołobut. Efficient collective shape shifting and locomotion of massively-modular robotic structures. *Autonomous Robots*, 43(1):97–122, 2019.
10. B Piranda, G Laurent, J Bourgeois, and al. A new concept of planar self-reconfigurable modular robot for conveying microparts. *Mechatronics*, 23(7):906–915, 2013.
11. Benoit Piranda and Julien Bourgeois. A distributed algorithm for reconfiguration of lattice-based modular self-reconfigurable robots. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 1–9. IEEE, 2016.
12. Benoit Piranda, S Fekete, A Richa, K Römer, and C Scheideler. Visiblesim: Your simulator for programmable matter. In *Algorithmic Foundations of Programmable Matter*, 2016.
13. J Shao, Q Yang, J Liu, and S Kramer. Graph clustering with density-cut. *arXiv preprint*, 2016.
14. P Thalamy, B Piranda, and J Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, 2019.
15. Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.