# Distributed clustering and self-reconfiguration for modular robots based programmable matter

Jad Bassil

**THÈSE DE DOCTORAT**

**DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE-FRANCHE-COMTÉ**

**PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ**

École doctorale n°37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

par

JAD BASSIL

# Distributed Clustering and Self-Reconfiguration for Modular Robots Based Programmable Matter

## Clustérisation distribuée et auto-reconfiguration pour la matière programmable basée sur des robots modulaires

Thèse présentée et soutenue à Montbéliard, le 09 Novembre 2023

Composition du Jury :

| | | |
|---|---|---|
| MITTON NATHALIE | Directrice de recherche à l'Inria Lille | Présidente |
| SIMONIN OLIVIER | Professeur à l'INSA de Lyon | Rapporteur |
| MARTINOLI ALCHERIO | Professeur Associé à l'École Polytechnique Fédérale de Lausanne, Suisse | Rapporteur |
| BOURGEOIS JULIEN | Professeur à l'Université de Franche-Comté | Examinateur |
| MAKHOUL ABDALLAH | Professeur à l'Université de Franche-Comté | Directeur de thèse |
| PIRANDA BENOÎT | Professeur à l'Université de Franche-Comté | Codirecteur de thèse |

N° | X | X | X |

# ACKNOWLEDGEMENTS

I'm grateful to my thesis advisors Prof. Abdallah Makhoul and Dr. Benoit Piranda for their constant support and mentoring during my academic journey.

I am immensely grateful to Prof. Abdallah for granting me the opportunity to come to France, pursue my master's and doctoral studies under his guidance. His encouragement and expertise have been instrumental in shaping my research career.

I extend my heartfelt thanks to Dr. Benoît for his invaluable contributions to this work as well as his meticulous attention to detail. His dedication to editing the videos and figures has greatly enriched the quality and presentation of this thesis. Additionally, I am grateful to him for leading the development of the *VisibleSim* simulator and to all individuals who have contributed to its creation. The remarkable work of the *VisibleSim* developers has greatly facilitated my workflow during my PhD, allowing me to focus more effectively on the core problems at hand.

I also acknowledge and express my gratitude to Prof. Julien Bourgeois, who is leading the Programmable Matter Project. I am truly privileged to have had the opportunity to work on such an innovative project. His vision, expertise, and his involvement sometimes in the discussions surrounding my work have been pivotal in broadening my research horizons and pushing me to excel.

I would like to extend my sincere gratitude to Dr. Irina Kostitsyna for providing me with the opportunity to join her lab at TU Eindhoven for a three-week period. The experience I gained during my time in her lab was invaluable, and I am grateful for the knowledge and skills I acquired through our discussions and collaborations. I would also like to express my gratitude to Prof. Jacques Demerjian for encouraging me to pursue research in France.

Moreover, I am grateful to the National Agency for Research (ANR) for funding my PhD research as part of the EIPHI Graduate School (contract "ANR-17-EURE-0002").

I would also like to express my appreciation to my friends in Montbéliard, with whom I shared unforgettable moments. I am especially grateful to Perla, whose presence in my life has brought joy and balance. Her unwavering support and motivation have been instrumental in my academic journey. Additionally, I extend my best wishes to my fellow PhD student office mates for the rest of their doctoral journey.

But above all, I am deeply grateful for the unwavering support and boundless love of my

family throughout my academic journey and life. My mother's unwavering determination to ensure my success has left an everlasting imprint on my heart. Equally deserving of my deepest gratitude is my father, whose unwavering love and invaluable life lessons have shaped me into the person I am today. Furthermore, I would like to express my heartfelt appreciation to my older brothers Joseph and Fady, who have set the bar high and serve as role models in my life. I am also immensely grateful for my brothers' financial support when I had to make the pivotal decision to come to France. The relentless commitment of my family to support me has been nothing short of extraordinary.

Thank You.

# CONTENTS

# List of Abbreviations

**CAD:** Computer-aided design

**WSN:** Wireless Sensor Network

**CH:** Cluster Head

**DCT:** Density-Connected Tree

**MSR:** Modular Self-Reconfigurable Robot

**PM:** Programmable Matter

**TUI:** Tangible User Interface

**FCC:** Face-Centered Cubic

**GUI:** Graphical User Interface

**CSG:** Constructive Solid Geometry

**SC-Clust:** Size-Constrained Clustering

**MST:** Minimum Spanning Tree

**GC:** Global Coordinator

**OPC:** Operation Coordinator

**MML:** Meta-Module Leader

**FM:** Flowing Module

**MRTP:** Modular Robot Time Protocol

**ARS:** Adaptive Rate Search

**LR+:** Enhanced Linear Regression

# Introduction

## Contents



## Programmable Matter

From the dawn of human history, we have been creating and manipulating matter to suit our changing needs and facilitate our daily life. From the prehistoric stone tools of our ancestors to modern materials and technologies, we continuously push the boundaries of what is feasible. We have developed new technologies over the ages to harness the power of nature and turn raw materials into useful objects that evolve over time to satisfy our evolving needs and desires.

Early humans used stone as a tool to grind, scrape, and cut. Then the agricultural revolution and the domestication of animals led to the creation of more sophisticated tools, such as plows, wagons, and other farming tools. As societies became more complex, we came up with new innovations and technologies such as the printing press, telegraphs, and steam engines, which sparked the industrial revolution accompanied by the mass manufacturing of products. During this period, new innovations such as plastic and synthetic fibers enhanced our ability to create more innovative objects.

Today, advanced electronics and robotics have remarkably accelerated technological progress and revolutionized how we live and work. The spread of 3D printing in recent years has given everyone the ability to instantly create custom complex objects, which was impracticable and prohibitively expensive with traditional manufacturing methods. 3D printed objects are now present in many fields, from fashion accessories to medical implants and spare parts or tools in space. In addition, virtual reality and augmented reality allow the creation of virtual objects that have a sense of presence, allowing users to interact with them as if they were real, which enhanced our immersion in the digital world. Unlike real 3D printed objects, virtual objects can be easily modified and adjusted. They can be replicated with different shapes and sizes, and they allow for collaboration where multiple people, at different locations, can work on the same virtual object simultaneously.

Programmable Matter (PM) represents the next step in the evolution of object creation that bridges the gap between a digital representation of an object and the physical world. It allows the creation of tangible shape-shifting objects that can be reshaped at will and change their functionality to accommodate different tasks or to adapt to their environment. Unlike virtual objects, objects formed from PM can persist. They have mass, can be touched, interacted with, and used as a tool while sharing the interactivity, adaptability, flexibility, and replicability with virtual objects. This has the potential to revolutionize many areas of science and engineering, from medicine and manufacturing to entertainment and architecture.

Various technologies have been studied to achieve PM such as 4D printing Ahmed et al. (2021), folding structure Hawkes et al. (2010), and DNA structures Kim et al. (2011b). A particularly promising technology is based on Modular Self-Reconfigurable Robot (MSR), which involves creating robotic modules that can assemble themselves into complex structures and then rearrange the connections of the modules to morph into different shapes. Unlike the atoms of real matter that are strongly linked to each other, the modules that form an MSR have displacement capabilities that allow matter to be reshaped by local movements. This technology has gained significant attention, having been popularized by the Claytronics project Goldstein and Mowry (2004) and is now being furthered by research at FEMTO-ST Bourgeois et al. (2016a). Its main objective is to create 3D interactive synthetic objects that bridge the gap between virtual and augmenting reality and the physical world.

MSRs have interesting properties that make them suitable for achieving programmable matter. This includes programmability, evolutivity, and autonomy. The ensemble forms a distributed system where each module has computational and communication capabilities, so they can be programmed to cooperate to achieve a common goal, such as rearranging their connections to change the whole structure. They possess the ability to alter their shape using motion and docking actions and can be controlled externally by a computer or run autonomously executing a distributed program.

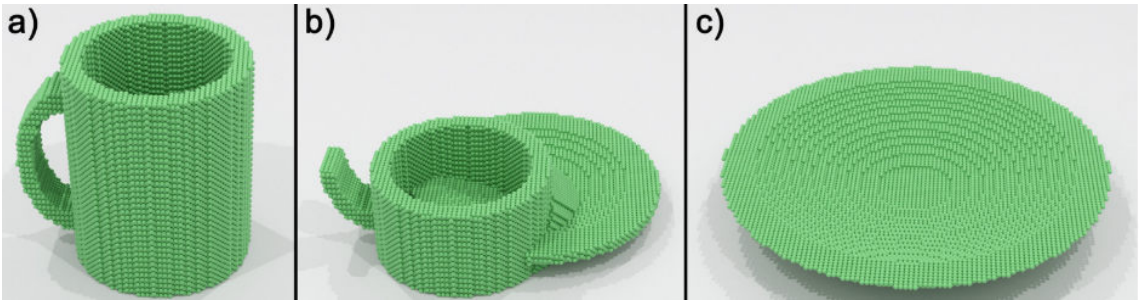



Figure 1: An example of self-reconfiguration of an object made with tiny spherical modules. a) the initial mug shape. b) an intermediary configuration. c) the goal plate shape.

Self-reconfiguration is the process that an MSR performs to transform itself from an initial

configuration to a goal one. Reconfiguration speed is crucial, especially for applications that require real-time interactions with the matter. An example of self-reconfiguration is shown in Figure 1 where a mug made of tiny spherical modules is reconfigured into a plate. Planning for self-reconfiguration is a challenging process. It has been shown in Hou and Shen (2014), that optimal self-reconfiguration planning is NP-complete for chain-type MSR. The exploration space between two random configurations increases exponentially with the number of modules: It has been shown in Park et al. (2008) that the number of unique possible configurations is $(c \times w)^n$ where $n$ is the number of modules, $c$ the number of possible connections per module, and $w$ the different ways of connecting the modules. We expect to have ensembles made of thousands or even millions of micro-scale modules to build objects with high resolution, which compounds the complexity of the problem. Additionally, distributed coordination of a large number of concurrent mobile modules while avoiding collisions and blocking is also challenging.

Clustering the modules can help mitigate the complexity of self-reconfiguration. By grouping the modules together, it may be possible to exploit the parallelism of motion where modules within a cluster can move independently and in parallel to achieve the desired configuration. This can improve the speed and efficiency of the self-reconfiguration process as multiple clusters can work concurrently to achieve the goal configuration. Parallelism of motions is particularly beneficial for large-scale ensembles, where the number of modules and the complexity of the problem can make it difficult to achieve real-time self-reconfiguration without parallelizing the motion of the modules.

## CONTRIBUTION

The goal of this thesis is to develop distributed algorithms that can improve the efficiency of self-reconfiguration in terms of both energy and time. In light of this objective, the central research question is: What is the most efficient way to reconfigure an MSR from a starting shape $\mathcal{I}$ to a goal shape $\mathcal{G}$? In response to this question, this work presents two key contributions that aim to address the challenge.

The first involves using a clustering algorithm with a constraint on the size of each cluster to reduce the search space for self-reconfiguration planning and enhance parallelization by allowing each cluster to reconfigure in parallel. It consists of building a spanning tree and then cutting it into branches to form the clusters. Another method consists of clustering modules into cubic-shaped clusters whose union forms a representation of the current shape. Recognizing the current shape is useful as input to a self-configuration planner or to efficiently report the shape to an external computer connected to the ensemble.

The second involves constructing the object using a porous structure that contains

enough empty space to allow for a concurrent flow of modules inside it. The porous structure also doubles as a storage space for modules within its empty volume, which relaxes the constraint of requiring the same number of modules in the initial and goal configurations. To create this structure, we use meta-modules, which are small clusters of modules locked together in a regular lattice pattern. Two self-reconfiguration algorithms are proposed for this structure: *RePoSt* and *ASAPs*. *RePoSt* is a round-based fully distributed algorithm where in each round a set of disjoint paths are determined to guide modules' flow towards their goal positions. *ASAPs* follows a hybrid approach where an initial centralized planner calculates all the flowing paths to reach the goal configuration and then the modules flow asynchronously in a single round to reach their goal position.

## OUTLINE

The organization of this thesis is divided into four main parts. The first part sets the context and state-of-the-art, which is covered in three chapters. Chapter 1 provides an introduction to modular robot-based programmable matter and its potential applications. Chapter 2 discusses the software challenges tackled during this thesis, as well as the state-of-the-art and recent advances. The third chapter 3 presents the experimentation and simulation tools used in this work.

The second part of the thesis focuses on clustering algorithms and is composed of two chapters. The first chapter 4 presents a Size-Constrained Clustering (SC-Clust) algorithm. The second chapter 5 proposes a distributed shape recognition algorithm.

The third part of the thesis is dedicated to self-reconfiguration and consists of three chapters. The first chapter 6 presents the proposed porous structure. The second chapter 7 presents *RePoSt*: a fully distributed self-reconfiguration algorithm for the porous structure. The third chapter 8 presents *ASAPs*: a hybrid self-reconfiguration algorithm, which combines centralized and distributed approaches to achieve faster and more efficient reconfiguration.

Finally, Chapter 9 offers a summary and a discussion of possible avenues for future research.

# I

# Context and State of the Art

# 1

# Modular Robot-Based Programmable Matter

## Contents



## 1.1/ Introduction

A Modular Self-Reconfigurable Robot (MSR) is a robot composed of an ensemble of interchangeable, autonomous, and communicating robotic modules. It can be used to achieve a Programmable Matter (PM) where each module of the MSRs can be thought of as the building block of such a matter. Furthermore, the modules that form the MSR have displacement capabilities that allow, by local movements, the rearrangement of the modules' connections in order for the matter to self-organize in response to an external stimulus or user input. The modules' embedded computation, their communication and motion capabilities offer programmability, reconfigurability, and interacivity, provide essential features for a programmable matter. Programmbality enables the modules to be programmed and reprogrammed to adapt to various tasks and requirements. Reconfigurability gives the modules the ability to move and change their interconnections to be arranged and rearranged as needed allowing for flexible configurations. And, interactivity allows the user to interact with the modular robot by manipulating the modules.

This chapter provides a contextual overview of modular robotic systems for building PM. It is organized into several sections that cover different aspects of the topic. In Section 1.2, we discuss the various applications of these systems. Section 1.3 presents the

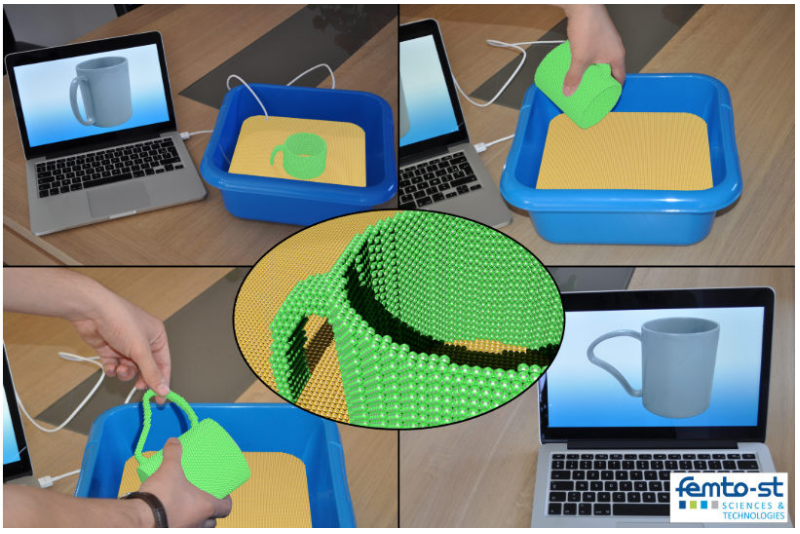



Figure 1.1: An illustration of a system where a PM composed of a large number of tiny spherical modules is used to replicate the object designed using the CAD software (from Bourgeois et al. (2016b)). The object is then manually manipulated and the virtual version reflects the changes and remains consistent with the physical object.

PM project, which is the context under which this thesis is conducted. Finally, in Section 1.4, we provide an overview of the existing modular robotic hardware.

## 1.2/ Practical Applications

In this section, we provide some examples of potential applications of programmable matter based on MSRs.

**Interactive CAD Design** Programmable matter can be used to enhance the computer-aided design process. The idea is to connect the matter in real time to a computer running a Computer-aided design (CAD) software. Using the CAD software, a shape is described and then transmitted to the matter. On reception, the matter will reconfigure itself into the given shape autonomously, or by responding to physical user interaction. These transformations in physical matter are reflected by the digital object in the CAD software, creating an interactive design tool, as shown in Figure 1.1.

**Multi Purpose Objects** Multi-purpose objects can be developed using modular robotic programmable matter to create objects that can change shape, size, and functionality to perform a variety of tasks, reducing the need for multiple specialized tools. In space, volume and weight constraints are critical factors, as spacecraft and habitats have limited space for equipment and tools. To overcome this constraint, multi-purpose objects can

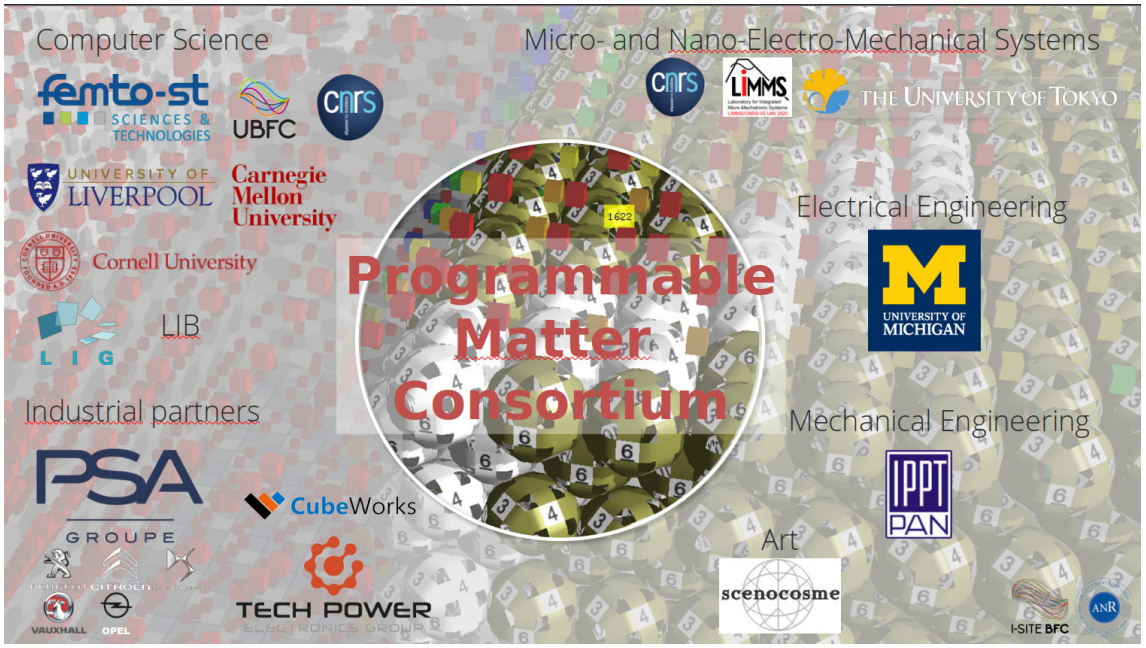



Figure 1.2: The Programmable Matter consortium.

be used.

**Flexible Tangible Interfaces** PM can be used to create shape-changing Tangible User Interface (TUI) that combine the flexibility of graphical user interfaces and the tangibility of physical ones Pruszko et al. (2021). For example, it can be used to create a personalized control panel that can be dynamically reconfigured to adapt to the task at hand or to support different applications.

**Art and Design** Programmable matter offers new possibilities for artists and designers to create interactive and dynamic sculptures, or immersive environments that can transform or respond to viewer interaction. An illustrative instance is reactive matter ReactiveMatter that constructs interactive sculptures capable of responding to viewers' touch and sound input, resulting in diverse sound patterns, rhythms, and fluctuations in light intensity.

## 1.3/ PROGRAMMABLE MATTER PROJECT

This thesis is part of the Programmable Matter (PM) project. I became acquainted with this project during my master 2 in "Internet of Things" at the University of Bourgogne Franche-Comté. During my Master's internship, I joined FEMTO-ST to start working on developing clustering algorithms specifically designed for large-scale modular robots

where I proposed DCut Bassil et al. (2020): an algorithm that creates clusters by cutting branches of a spanning tree. Encouraged by my internship results, I started my PhD thesis to further advance my research.

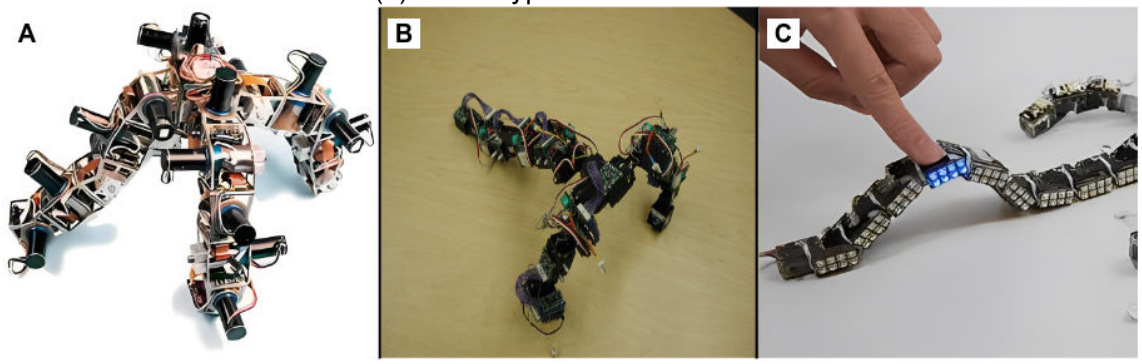
The Programmable Matter (PM) project is a continuation of the Claytronics project (now discontinued) Goldstein and Mowry (2004) that started at Carnegie Mellon University with the vision to build 3D programmable matter using nanoscale robots called Catoms (Claytronic Atom). Today, FEMTO-ST leads a consortium of academic and industrial partners with a wide range of expertise that are working to bring the vision of PM to reality. The current partners in the consortium are shown in 1.2. On the hardware level, efforts are being made in the fields of micro-electro-mechanical systems and electrical engineering to design and develop reliable millimeter-scale robotic modules. On the software level, where this thesis fits, the focus is on developing an algorithmic foundation for MSR based Programmable Matter to solve problems such as: self-reconfiguration to transform the shape of modular robot, time synchronization to compensate for the skew of modules internal clocks, shape representations to find an efficient encoding of a geometrical shape, leader election to break the symmetry by electing one of the modules to be the leader of the ensemble, etc. and to better understand the capability and complexity of PM systems on the theoretical level.

## 1.4/ Existing Modular Robotic Systems

MSRs differ from classic robots in their ability to change their physical structure and functionality by adding, removing, or reconfiguring modules. Classic monolithic robots, on the other hand, have a fixed structure and functionality that is pre-determined and cannot be altered without significant re-engineering. Modular robots have a higher degree of versatility and flexibility in their design and functionality.

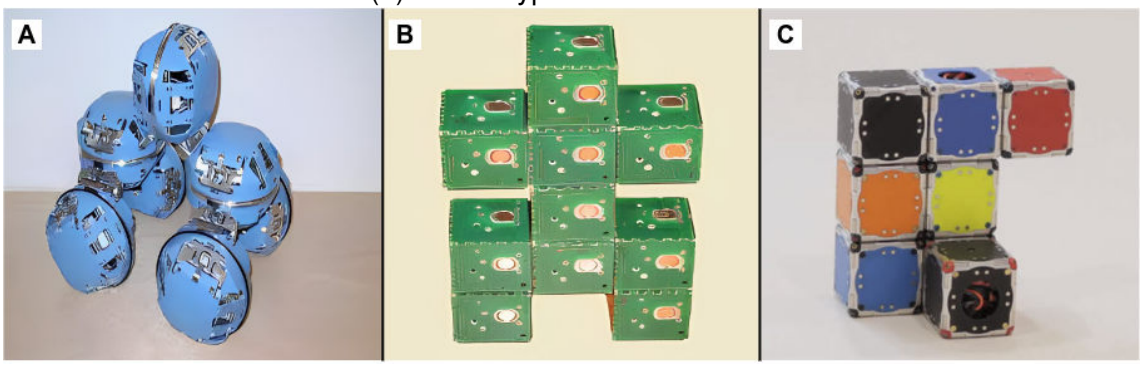
An exhaustive overview of existing MSR systems can be found in the surveys Dokuyucu and Özmen (2022); Chennareddy et al. (2017); Yim et al. (2007). Modular robots can be classified into multiple types of structural formation. Chain-type formation consists of modules arranged in a tree-like fashion. They are able to perform locomotion gait on rough terrains. In a lattice-type formation, the modules reside in a regular 2D or 3D lattice structure. Some modular robots exhibit a combination of chain-type and lattice-type formations; combining the characteristics of both types, they form the hybrid-type. Some of the existing hardware with different formation types are shown in Figure 1.3. In addition, some modular robotic systems have mobile architectures where modules are free to move in their environment and can dock with each other to form a chain or/and a lattice structural formation. Their mobility can be achieved through self-locomotion, as demonstrated in Parrott et al. (2018); Liu et al. (2023), or by being propelled through

(a) Chain-type modular robots.



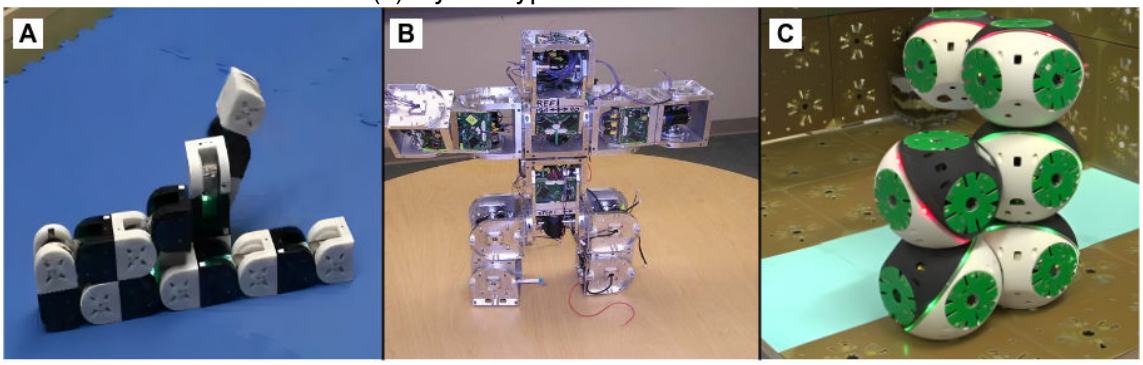

A) PolyBot Yim et al. (2000). B) Conro Castano et al. (2000). C) ChainForm Nakagaki et al. (2016)

(b) Lattice-type modular robots.




A) Atron Østergaard et al. (2006). B) Miche Gilpin et al. (2008). C) 3D M-Blocks Romanishin et al. (2015)

(c) Hybrid-type modular robots.




A) Mtran III Kurokawa et al. (2008). B) Superbot Salemi et al. (2006). C) Roombots Spröwitz et al. (2010)

Figure 1.3: Hardware examples with different modular robotic formation.

external forces from their environment, as exemplified in White et al. (2005); Haghighat et al. (2015).

Furthermore, modular robotic systems differ in their communication model. They either use a neighbor-to-neighbor communication model or a global one in which all modules can communicate with each other via a global bus. They also differ in their geometrical properties, such as their shape (spherical, cubic, triangular) and size, their coupling

mechanism Saab et al. (2019) (magnetic, mechanical, and electrostatic), and their actuating and sensing capabilities. A MSR is homogeneous if all its modules are identical and autonomous. Otherwise, it is specified as heterogeneous. The complexity of coordinating these modular robotic systems is affected by all the hardware parameters mentioned above.

The following are the hardware properties of the MSRs considered in this work to achieve PM:

- **Lattice-based Structure:** Lattice-type modular robots are more suitable for creating programmable matter, as they allow more flexibility to approximate a given shape. In addition, modules are assigned a cell position with a unique coordinate value that a planner can exploit for efficient self-reconfiguration. A network characterization of lattice-based MSR can be found in Naz et al. (2018a).

- **Geometry:** Different module shapes allow for different numbers of actuators and neighbors' dispositions around a module which gives it a local knowledge about its neighborhood. The number of neighbors corresponds to the degree of the interconnections graph; a larger degree gives access to more information in fewer communication hops. This can potentially improve the efficiency of algorithms by reducing the number of steps required to exchange information. Also, the geometry of the surface affects the motion capabilities. For example, translation on square surfaces is easier, while rotation on rounded surfaces is easier. Furthermore, the resolution of a programmable matter object is strongly affected by the geometry of the modules. Smaller modules can allow the creation of objects with higher resolution and more intricate features, thus creating programmable matter objects with greater fidelity to their counterparts. So, we envision building a programmable matter object with a large number of tiny modules.

- **Neighbor-to-neighbor communication:** Communication between modules is essential as it allows information exchange and distributed computation to plan and coordinate global tasks. An MSR based Programmable Matter can be made up of thousands of tiny communicating modules. Therefore, the use of a global communication model where all modules communicate through a global communication medium is limited in terms of scalability and packet collisions. In addition, using wireless communication can be problematic on a micro-scale due to interference. Therefore, we consider local neighbor-to-neighbor communication.

- **Homogeneous modules:** Homogeneous MSR are made of identical modules, allowing low-cost mass production and easy replacement. They are simple to scale in size by adding more modules. And they can reduce the complexity and computational cost of finding feasible configurations due to the fact that all modules can

interchangeably change their positions. Consequently, a self-reconfiguration planner does not need to account for the constraint of having a specific module type in a particular position.

- **Limited resources:** Modules are usually small electronic devices that have limited capabilities, such as insufficient computing power and memory capacity. In addition, they have limited energy resources. They can be powered through an external source or by utilizing energy harvesting techniques from their environment, such as ambient light or electromagnetic sources. Optimizing energy consumption, employing energy-efficient algorithms and distributed power management techniques are essential for the functionality of the MSR.

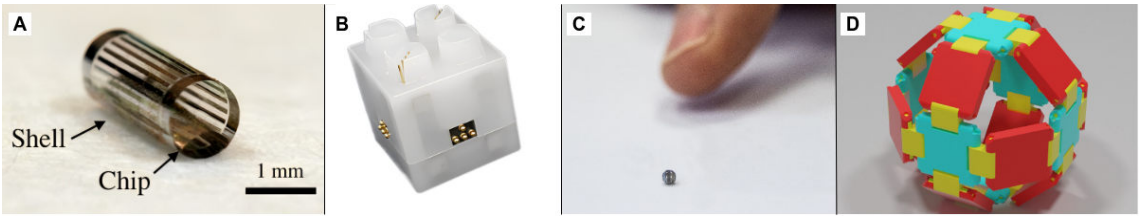



Figure 1.4: Modular Robotic systems developed as part of the Programmable Matter Project. A) *2D Catom*. B) *BlinkyBlock*. C) *3D Catom*. D) *Datom*.

The following are the MSR systems developed as part of the Programmable Matter project and are shown in Figure 1.4:

- 2D Catoms Kirby et al. (2007): are cylindrical millimeter-scale modules of $6$ mm long and $1$ mm diameter. These real tiny robots move by rolling on their fixed neighbors in a clockwise or counter-clockwise direction by activating electrodes on their surface. They are organized in an hexagonal lattice.

- *BlinkyBlocks* Kirby et al. (2011): are $4$ cm cubic robots organized in a cubic-lattice. They connect to up to six neighbors using magnets. They communicate with their directly attached neighbors using serial connections. They can be docked and undocked manually. The second version developed in the Programmable Matter consortium is equipped with LED light and acoustic actuators, so modules can be programmed to change colors and emit sounds. They are mainly used to create interactive art, education, and research on distributed systems.

- *3D Catoms* Piranda and Bourgeois (2018): are millimeter-scale quasi-spherical modules placed in a Face-Centered Cubic (FCC) lattice. They use electrostatic forces to latch on and move around their neighbors. They can communicate with their neighbors through their latching interfaces. *3D Catoms* do not yet exist in numbers and are being developed by actors in the PM consortium. The startup *Phigi* [1]

[1] Phigi: https://www.phigi.io/

is focused on the industrialization of *3D Catoms*.

- Datoms Piranda and Bourgeois (2022): are deformable modules that resemble the *3D Catoms* in their geometry and communication model. A Datom is able to deform by compressing one of its sides to facilitate the modules' motions by giving space for other modules in its neighborhood to move. Although it is still a theoretical model that requires further study to implement, it can be a leap forward in reducing the complexity of self-reconfiguration planning by relaxing motion constraints.

## 1.5/ CONCLUSION

This chapter introduced programmable matter based on modular robots and explored its potential applications. Additionally, it discusses the hardware characteristics of these modular robot systems, highlighting their key properties. Moreover, we presented the modules developed within the programmable matter project, providing insight into their design and functionality.

With the exception of *BlinkyBlock*, existing hardware models remain theoretical, and operating modules with motion capabilities are not yet available in quantity. As a result, the validation of self-reconfiguration algorithms on a real large scale MSR is currently not feasible. However, *BlinkyBlocks* are accessible and can be utilized to validate algorithms that do not rely on motion.

The inaccessibility of hardware makes it difficult to show the effectiveness of our proposed algorithms in real-world scenarios or even to create an accurate physical model in simulation. Therefore, the work conducted during this thesis was validated on real *BlinkyBlocks* whenever possible. Regarding self-reconfiguration, we use the behavioral simulation of *3D Catoms* to assess performance and visualize their behavior in a physically unconstrained simulated environment. The experimentation and simulation environment under which this work is conducted is detailed in Chapter 3.

# 2

# CHALLENGES AND STATE OF THE ART

## Contents



## 2.1/ INTRODUCTION

To make Programmable Matter (PM) a reality, challenges must be overcome at the hardware and software levels. At the hardware level, miniaturizing components to create small, lightweight modules that can connect, compute and communicate reliably is a crucial task. Additionally, the development of reliable actuators is crucial. While electro-mechanical actuators can be effective on larger scales, electrostatic actuators are better suited on smaller scales due to their compact size, low power consumption, micro-fabrication compatibility, and high precision and control.

Power transfer and management is also essential, as each module may require a power source to operate. It might also need a power storage that keeps modules operating when moving, which contributes to the module's weight and consequently increases the energy demand for movement.

The mechanical stability and durability of the module connections in addition to motion actuation need to be addressed. For instance, the electrostatic actuation of *3D Catoms*

requires a sequence of repulsing/attachement/detachements which is error-prone and can cause the disconnection of a module. Therefore, the development of *Datoms* aims to maintain a linked connection between the moving module and its pivot. Finally, cost-effective manufacturing and assembly techniques are necessary to produce these robots at scale. Due to their tiny scale, it is difficult to manipulate the components by hand, hence the need of a special machine for the assembly of the micro components using micro-clamps and cameras to align the components with micrometer precision.

Furthermore, significant challenges must also be addressed at the software level. One of the most important aspects is the development of non-complex distributed algorithms and control systems that enable efficient and autonomous operation of modular robots. These algorithms should support distributed decision making to allow individual modules to make intelligent decisions and take actions while cooperating with other modules. Exploring and addressing algorithmic challenges that cover leader selection, localization, fault tolerance, self-reconfiguration, self-assembly, clustering, and similar problems are imperative areas of research.

The difficulty of distributed programming lies in the locality of information, hence the need to exchange data to accomplish a global behavior. The emergence of this behavior is based on the processing of the exchanged data. Therefore, when the goal is to achieve a complex behavior, the code can quickly become complex and convoluted, making it difficult to understand and maintain. The system can be modeled as a multi-agent system where each agent represents a module's role with its own knowledge and goals. This provides a higher level of abstraction and allows us to design and reason about the system's behavior at a conceptual level.

In this chapter, the software challenges that were tackled during the three years of the thesis are presented in the following sections. While most of the work focused on self-reconfiguration and clustering, which are the main topic of this manuscript, other challenges such as fault tolerance, time synchronization, and shape recognition were tackled and solutions were proposed. The following sections of this chapter detail each of these challenges.

## 2.2/ SELF-RECONFIGURATION

Self-reconfiguration is the main and most crucial task of MSR. It aims to transform the initial shape of a modular robot into a given goal shape using communication-coordinated motions.

The difficulty of the self-reconfiguration problem lies in the properties of a limited memory space and the locality of the information. In fact, because of their compact size, individual

modules or particles have very low computational and energetic resources. Furthermore, due to the locality of information, a module does not have the information about the global configuration and state of the modular robot system, and then it cannot take decisions individually. Moreover, having mobile and connected robots is insufficient to obtain self-reconfigurable programmable matter. If we consider that robots initially build a shape $\mathcal{I}$ and must at the end self-reorganize to build a goal shape $\mathcal{G}$, a subset of these robots has to move while avoiding collisions and maintaining the connectivity of the assembly. As a result, the graph representing the interconnections, which defines the network neighborhood and physical connections, becomes dynamic in both space and time. In addition, planning self-reconfiguration is a very complex problem. It has been shown to be NP-complete for chain-type modular robots Hou and Shen (2014). Optimal solutions are impossible to find because the number of possible configurations increases exponentially as the size of the system (number of modules) increases Bourgeois et al. (2016a) and we expect the matter to be made up of thousands or even millions of tiny modules.

The self-reconfiguration time, that is, the time required to transform an initial shape into a goal shape, is an important parameter that must be optimized. A self-reconfiguring solution that uses sequential movements to achieve a target shape simplifies planning by eliminating potential problems, such as dealing with deadlocks and collision uncertainties, such as in Hourany et al. (2021); Fitch et al. (2003). However, sequential motions are highly restrictive in medium to large self-reconfiguring modular robots, as they tend to significantly lengthen the duration of the reconfiguration process. I propose utilizing two main optimizations to reduce the complexity of this problem, they consist of reducing the distance traveled by the modules and permitting the simultaneous (parallel) displacement of a large number of modules in the system while avoiding collisions.

### 2.2.1/ SELF-RECONFIGURATION ALGORITHMS

Self-reconfiguration consists of performing module-level movements to change an initial configuration into a goal one. The authors in Thalamy et al. (2019); Ahmadzadeh and Masehian (2015); Dokuyucu and Özmen (2022) provide surveys on self-reconfiguration algorithms. Algorithms differ in their control properties; they can be centralized or distributed, synchronous or asynchronous, and deterministic or stochastic. They are also hardware-dependent or generic.

Movements through the internal volume of the robot allow for a higher degree of parallelism and require a smaller number of movements to reach the goal configuration according to Rus and Vona (2001). Internal movements can be achieved using tunneling or/and scaffolding.

Many tunneling-based reconfiguration algorithms that use meta-modules have been pro-

posed in the literature Vassilvitskii et al. (2002); Kawano (2017, 2019, 2020); Lengiewicz and Hołobut (2019). Tunneling allows in-place reconfiguration where modules flow through the internal volume of the ensemble. They are specific to deformable modules that can contract and expand, which requires a specific hardware design. In most of them, modules are grouped into meta-modules seen as a unit to facilitate both planning and motion operations. Parada et. al Parada et al. (2021) described a meta-module design that simulates the contract/expand capability. It can be applied to a wide range of non-deformable modular robots to enable tunneling. Nevertheless, it is only suitable for modules placed in square cubic lattice.

Scaffolding first proposed in Kotay and Rus (2000) is another technique used to optimize the self-reconfiguration process. It consists of building the structure using hollow substructures or meta-modules leaving enough empty volume inside the structure that allows modules to navigate through it in parallel while avoiding blocking and collisions due to overcrowding at the cost of decreasing the granularity of the configuration. By approximating the target configuration with a porous representation, scaffolding is used to reduce the complexity of the reconfiguration of cubic modules that move by translation and rotation guided by cellular automata in Stoy (2006) or by gradient descent in Stoy and Nagpal (2007)

Dewey et al. Dewey et al. (2008) described a generalized model for meta-modules independent of the module design, which inspired the self-reconfiguration scheme proposed in this thesis. The aim is to achieve a holonomic system where modules are arranged in meta-module units that can be in two states: filled or empty. Modules flow from a filled meta-module to an empty one to reach their target position guided by a planner.

Lengiewicz and Holobut Lengiewicz and Hołobut (2019) presented a method to self-reconfigure large ensembles of cubic modules that form a porous scaffolding structure made of cubic meta-modules of seven modules and one empty space. They tackled the self-reconfiguration problem by decomposing it into two subproblems: determining how the boundary of the current configuration must evolve to reach the goal configuration and finding an optimal flow of modules between the boundaries of the current shape and through its volume using an asynchronous distributed max-flow search based on local memory and communication. Their proposed algorithm is efficient, with the number of movements of the modules proportional to the resolution of the robot. This method might be used with any other hardware system that has the ability to internally move modules through a scaffolding setup. We used the same max-flow planning approach in our proposed algorithms.

In the context of the PM consortium, many algorithms have been proposed for the different hardware:

**2D Catoms** In Naz et al. (2016), a distributed, asynchronous, and deterministic self-reconfiguration algorithm is proposed for *2D Catoms*. Initial and goal configurations must be formed by continuous horizontal layers. It consists of a continuous flow of modules that roll on fixed pivots in one direction from the initial configuration towards a goal configuration. To avoid deadlocks and collisions, modules flow in a stream while keeping at least one empty position between them by applying a message-passing traffic-light-like system. The proposed solution does not require a complex path planning for modules flow due to simple cylindrical geometry of *2D Catoms* that can flow by rotating on the surface following a single path in one direction to build the goal configuration out-place next to the initial configuration, which is not feasible in 3D. The traffic-light system applied to maintain enough space between the flowing modules was then used for *3D Catoms* in Thalamy et al. (2021a) and is also used in the self-reconfiguration solution proposed in this thesis.

**3D Catoms Self-Assembly** The shape shifting of a MSR requires assembly planning to specify the order of placement by assigning to each robot its final position in the goal configuration while avoiding blocking positions. In Pescher et al. (2020) the authors proposed GAPCoD: a generic assembly planner by constrained disassembly. GAPCoD outputs a directed assembly graph obtained through the disassembly of the goal configuration submitted to given constraints. Each vertex representing a module must be positioned before its child nodes. In Tucci et al. (2018) the authors proposed a distributed approach for the same problem where modules use a goal shape description to attract other modules to latch at empty positions in the goal configuration based on predefined rules to avoid blocked positions. These methods do not take into consideration the spatiotemporel scheduling of module motions towards the planned positions.

**3D Catoms Self-Reconfiguration** Thalamy et al. proposed a self-reconfiguration scheme for modular robotic programmable matter using the same hardware that we use in this work: *3D Catoms*. It envisions assembling the scaffold of a shape using multi-module tiles. The tiles are built with modules that flow upward from a reserve of modules placed beneath the shape called *sandbox*. The shape can then be coated with a thin layer of modules to better represent the target shape Thalamy et al. (2021a). Like in Naz et al. (2018b), flowing modules use a local message passing coordination algorithm inspired by the traffic-light system that forces modules to keep enough empty space between them to avoid collisions. The scaffold can then be coated by a thin layer of modules as described in Thalamy et al. (2020a). The difference from our work is that they describe only the construction of a scaffold for a given shape starting from a reserve of modules placed beneath it, not the self-reconfiguration of an initial shape into a goal one.

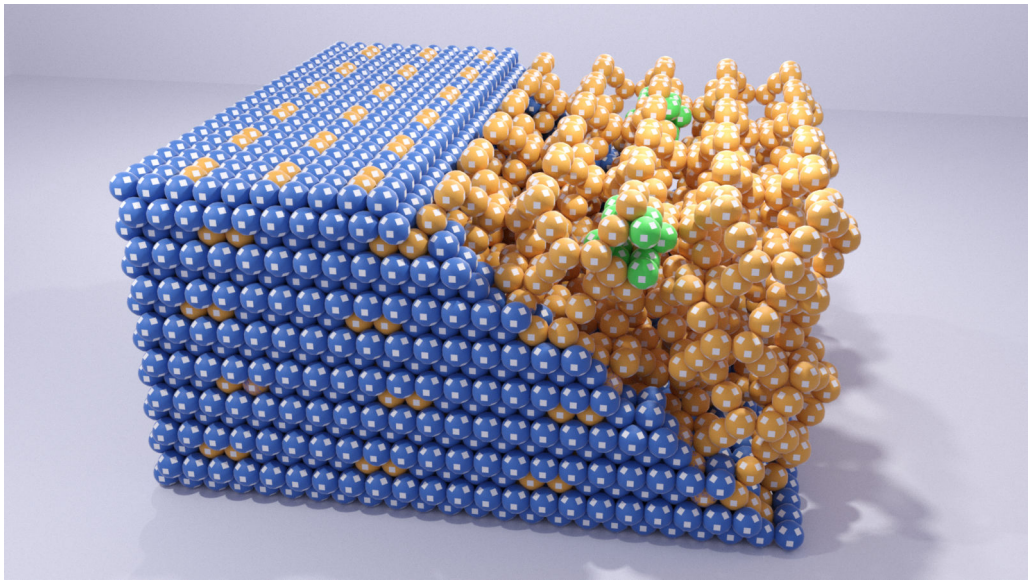


Figure 2.1: A box made of 5,914 3D Catoms, we remove top right border modules to better present the internal structure. The porous structure is made of meta-modules that construct a scaffold in orange, some reserves of modules are drawn in green and border meta-modules are drawn in blue.

**Contribution** My contribution in this thesis is to reduce the self-reconfiguration time by proposing, in Part III, self-reconfiguration scheme based on a porous structure formed using meta-modules that group several modules. Internal meta-modules construct a scaffold structure drawn in orange in Figure 2.1 that helps with the displacement of modules within the structure and, at the same time, allows modules to be stored in the structure. The structure will be coated with solid meta-modules (drawn in blue) to close the volume. The self-reconfiguration scheme proposed in this work combines self-assembly and self-reconfiguration. The two proposed planning methods in Chapters 7 and 8 find motion paths that start at an occupied position in the initial shape that must be emptied and end at an empty position in the goal configuration that must be occupied. The modules then flow along the paths to reach their final positions. The first is fully distributed, and the second is a hybrid centralized/distributed solution. Both exploit the properties of the porous structure to reduce the distances covered by the robots and parallelize their movements.

## 2.3/ CLUSTERING

### 2.3.1/ SIZE-CONSTRAINT CLUSTERING PROBLEM

Clustering the modules of a MSR ensemble can yield many benefits. It can help reduce the search space for self-reconfiguration planning and allow the parallelization of movements where each cluster reconfigures in parallel. For instance, in Moussa et al. (2021)

the authors proposed a cluster-based self-reconfiguration method. 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 assume that the clusters are given initially and do not propose a clustering method.

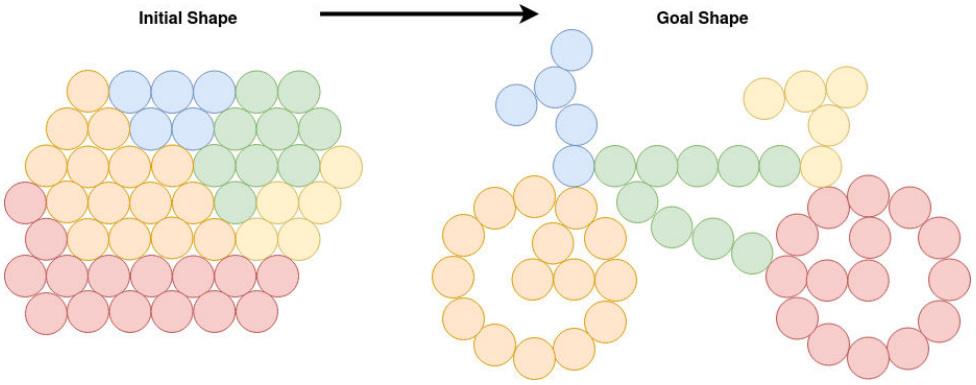



Figure 2.2: An example of self-reconfiguration showing the importance of the size-constraint clustering. Each cluster is colored differently. A cluster in the initial shape with a fixed number of modules must transform into a part of the goal shape with the same number of modules.

To perform a cluster-based self-reconfiguration, clusters of modules in the initial configuration must move to form a specific part of the goal configuration that requires a fixed number of modules, as can be seen in Figure 2.2. Therefore, clustering the initial configuration must be performed while taking a size-constraint into consideration that specifies the number of modules in each cluster based on the part of the goal configuration that the cluster's modules must reconfigure into. In the next section, we present a review of the literature related to size-constrained clustering.

#### 2.3.1.1/ CLUSTERING ALGORITHMS

In this section, we present existing clustering algorithms organized in different categories.

**Graph clustering** Clustering an ensemble of modules is related to graph clustering or graph partitioning. The graph partitioning problem has been widely studied in the literature and is known to be an NP-hard problem Schaeffer (2007); Buluç et al. (2016); Adoni et al. (2020a). Existing graph partitioning methods rely on two search techniques: global and local. They aim to partition a given graph into $k$ disjoint balanced densely connected partitions. Global search algorithms work on the entire graph to find a direct solution. They include solutions based on linear programming Fan and Pardalos (2010);

Hager et al. (2013), spectral clustering Huang et al. (2019); Martin (2012); Liu and Han (2018), and geometrical clustering Simon (1991); Gilbert et al. (1998); Ansari et al. (2019). 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 Osipov and Sanders (2010), tabu search Rolland et al. (1996), random walk Yan et al. (2019), graph growing Predari and Esnard (2016); Diekmann et al. (2000), genetic algorithms Kim et al. (2011a), and multilevel approach Meyerhenke et al. (2017, 2014). 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** The capacitated clustering problem (CCP) Mulvey and Beck (1984) 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 Zhou et al. (2019); Scheuerer and Wendolsky (2006); Liu et al. (2022); Gnägi and Baumann (2021). For example, in Zhou et al. (2019), 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 Clustering** Distributed partitioning methods were developed to overcome the high computational cost when the graph size becomes very large. The distributed partitioning model distributes the partitioning task across a network of computers. For example, in Rahimian et al. (2015) 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 Adoni et al. (2020b), a distributed algorithm that outperforms JA-BE-JA. It is more suitable for the master-slave distributed architecture, where a master node coordinates the partitioning process, and the partitioning task is distributed to slave nodes that operate in parallel to add a new vertex to their partition subgraph. These methods are suitable for balanced partitioning of graph structured data and cannot be applied for partitioning the nodes of a distributed system such as a modular robot where the communication is limited to neighbor-to-neighbor without a centralized control.

**Multi-robot partitioning** The multi-robot task allocation problem Khamis et al. (2015) involves 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. Therefore, robots are partitioned to form $k$ coalitions based on the utility function. Then, tasks are assigned to coalitions to be executed simultaneously Mazdin and Rinner (2021); Dutta et al. (2019); Zhang et al. (2014). 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.

**Clustering in mobile modular robots** Partitioning the set of modules for configuration generation in modular robots has been studied in Dutta et al. (2015, 2016). In Dutta et al. (2016), an algorithm based on a coalition search graph is proposed to partition a set of modules. The aim is to achieve an efficient shape configuration of scattered mobile modules in a 2D environment 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 that form a coalition are then docked together to form the goal configuration. Another method is proposed for the same purpose in Dutta et al. (2015) where a minimum spanning tree is built to minimize the docking cost. Then, the best coalition or configuration is found by partitioning the built tree taking into account the size, communication, and battery constraints. These methods focus on configuring small sets of separated modules scattered in their environment. Therefore, they are not applicable to solve our problem.

**Clustering robotic-swarms** Clustering has been studied for robotic swarms. The purpose is to divide the swarm into clusters for pattern formation and for better problem-solving efficiency by dividing the problem into subproblems and assigning different tasks to each cluster. Mostly, existing methods are based on robot mobility directed by external stimuli in the environment, so they are not suitable for modular robot-based programmable matter, to cite a few Pinciroli et al. (2009); Hayes et al. (2003); Kazadi et al. (2004); Wahby et al. (2019). Other methods based on token clustering were proposed. In Di Caro et al. (2012), a fully distributed algorithm based on consensus and load balancing is proposed to partition robots with wireless communication into two spatially separated clusters. Then it was extended in Bulla Cruz et al. (2017) 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 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 in wireless sensor networks** Clustering for Wireless Sensor Network (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, limiting inter-cluster communications to CHs only, thus reducing the communication load Afsar and Tayarani-N (2014). Many clustering algorithms have been proposed for WSN Xiangning and Yulin (2007); Mukherjee (2020); Bhola et al. (2020); Pietrabissa and Liberati (2019); Prorok et al. (2010) but they are not suitable for modular robots due to their specific constraints, which make them inapplicable on modular robots: wireless communication, existence of a base station, and pre-election of cluster heads.

**Clustering in lattice-based modular robots** In Bassil et al. (2020) we proposed a fully distributed and adapted version of the DCut algorithm originally proposed by Shao et al. (2018) Shao et al. (2018) in the context of modular robots. It takes into account the geometrical aspect of the ensemble and captures the density between adjacent modules locally using the Jaccard coefficient. The idea is to build a Density-Connected Tree (DCT) that captures the topological similarities between the modules relative to the fixed points at the extremities of the geometric 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 and moving modules from one cluster to another. Additionally, 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.

Table 2.1 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 from ours. The existing work mentioned in this section does not satisfy the requirements to solve the size-constrained $k$-partitioning problem for modular robots. The solution must be distributed, based on the limited local knowledge of each module about its neighborhood, and satisfies the size constraint.

Table 2.1: Comparative table of existing clustering methods.

| Contribution | distributed | local knowledge | size-constraint |
|---|---|---|---|
| Global SearchHager et al. (2013); Huang et al. (2019); Liu and Han (2018); Ansari et al. (2019); Meyerhenke et al. (2014); Kim et al. (2011a) | × | × | × |
| Local SearchOsipov and Sanders (2010); Rolland et al. (1996); Yan et al. (2019); Predari and Esnard (2016) | × | ✓ | × |
| CCP Mulvey and Beck (1984); Zhou et al. (2019); Scheuerer and Wendolsky (2006); Liu et al. (2022); Gnägi and Baumann (2021) | × | × | ✓ |
| Distributed Methods Rahimian et al. (2015); Adoni et al. (2020b) | ✓ | ✓ | × |
| SWARM clustering Bulla Cruz et al. (2017) | ✓ | ✓ | × |
| WSN clustering Xiangning and Yulin (2007); Mukherjee (2020); Bhola et al. (2020); Pietrabissa and Liberati (2019) | ✓ | ✓ | × |
| DCut Bassil et al. (2020) | ✓ | ✓ | × |
| SC_Clust (Our contribution in Chapter 4) | ✓ | ✓ | ✓ |

**Contribution** To solve the size-constrained clustering problem in the context of MSR, we present SC-Clust in Chapter 4, a distributed solution for the size-constrained $k$-partitioning problem for modular robots that uses local knowledge of the modules to cluster the ensemble. The algorithm consists of creating a tree structure and then cutting it into branches that will be adapted to have the desired sizes.

### 2.3.2/ Shape Recognition

The shape recognition problem consists in allowing the modules to discover a representation of their current shape. Shape representation is one of the main challenges to achieve PM. It consists of encoding a shape in a compact and memory-efficient way. Till now shape representation is studied to represent a goal configuration and transmit it to the modules, which provides enough information for self-reconfiguration. The modules can check whether their position belongs to the goal configuration. If not, they can move to fill an empty position in the goal configuration. Nevertheless, it is also useful to allow the ensemble to recognize its current shape. For example, it can be useful for the interactive CAD application to send the shape to the connected computer to update the digital object displayed by the CAD software.

Moreover, in a large modular robot, modules have limited local knowledge about the entire configuration. They can only access their directly connected neighbors. Thus, allowing modules to recognize the whole shape of their configuration can facilitate distributed self-reconfiguration planning, which consists of finding the sequence of movements to re-

configure into a goal shape. Knowing the current configuration modules can calculate the difference between the current and the goal shape to optimize the movements and the re-arrangements to be made to reach the goal shape while ensuring safe and mechanically stable movements.

#### 2.3.2.1/ Existing Work on Shape Recognition

In this section, we review existing work on shape recognition.

**Shape Representation** In Stoy and Nagpal (2007); Fitch and Butler (2008) the authors propose to transform a CAD model into overlapping bricks to make it easier for the modules to identify their position relative to the goal configuration required for the self-reconfiguration process. In Tucci et al. (2017) the authors proposed to use a Constructive Solid Geometry (CSG) tree. The leaves of the tree contain basic geometrical objects, and the intermediate nodes contain geometrical transformations and combination operators to form the final shape on the root. The objective of these methods is to encode a shape using a centralized computer to transmit it to the module to self-reconfigure it.

**Configuration Recognition** The configuration recognition problem, which consists of matching and mapping a configuration to a library of configurations, is studied in Baca et al. (2015); Shiu et al. (2010); Liu and Yim (2020). First, a discovery phase is executed to find a representation of the current configuration as an interconnectivity graph, where nodes represent modules and edges represent the connections between the modules. Then, they match the graph with an existing one and map the physical modules to their logical one.

**Goal Configuration Matching** In Butler et al. (2002), the authors solve the matching problem with a distributed goal recognition algorithm that verifies if a configuration matches a given goal shape without the need to discover the whole configuration. In Baca et al. (2015) a distributed real-time algorithm is presented for configuration discovery. It allows modules to discover other modules using wireless infra-red communication and construct the connectivity graph.

The drawback of connectivity graphs is that they suffer from scalability issues since they depend on the number of modules, which might increase drastically, especially when building high-fidelity programmable matter with millimeter-scale robots. Furthermore, in lattice-based modular robots, we can exploit geometric information to create a compact shape representation.

**Contribution** We propose to solve the shape recognition problem that aims to find a representation of a modular robot current configuration using a fully distributed algorithm described in Chapter 5 that finds a set of overlapping boxes, each of which can be seen as a cluster whose union forms the current shape.

## 2.4/ TIME SYNCHRONIZATION

Many applications with distributed control require the same notion of a global time in each module. For example, user interfaces changing shapes using modular robots Pruszko et al. (2021) where the modules that make up the interface must be synchronized to efficiently handle interactions between humans and the interface.

Synchronization can be achieved using dedicated pins to share a timing signal, but this requires specific hardware design. In our model, each module has its own notion of time provided by its hardware clock. Unfortunately, local hardware clocks are insufficient to achieve a global notion of time, since the accuracy of hardware clocks is affected by environmental conditions such as voltage, temperature, aging, etc. Therefore, they tend to drift apart quasi-linearly over time even in a perfect environment. Therefore, each module needs to have an estimate of a common global time determined by a distributed time-synchronization protocol.

Time synchronization algorithms and protocols have been proposed for computer networks Gusella and Zatti (1989); Cristian (1989); Mills (1991); IEEE (2008). In addition, time synchronization has been widely studied in Wireless Sensor Network (WSN) that form peer-to-peer networks for resource-constrained devices similarly to modular robots. Many time synchronization protocols have been proposed for WSNs such as Maróti et al. (2004); Elson et al. (2003); Lenzen et al. (2014); Yildirim and Gürcan (2014); Shi et al. (2022). For modular robots, to our knowledge, Modular Robot Time Protocol (MRTP) Naz et al. (2016) is the only protocol that has been proposed for time synchronization in modular robots with neighbor-to-neighbor communication. It is the protocol most adapted for large modular robotic systems with low-precision hardware clocks that use low-bitrate neighbor-to-neighbor communication and can form networks with large diameters. It aims to achieve tight network-wide synchronization by keeping a small offset between any local clock and a global reference clock situated at a reference module. The reference module periodically sends a synchronization message that propagates hop-by-hop through a breadth-first communication tree to all modules using a predictive method to compensate for communication delays. Modules compensate for clock skew with least-square linear regression on a window of fixed size containing previously received synchronization points. However, the performance of MRTP decreases as the diameter of the modular robot network increases. The reason is that a small error induced by non-deterministic

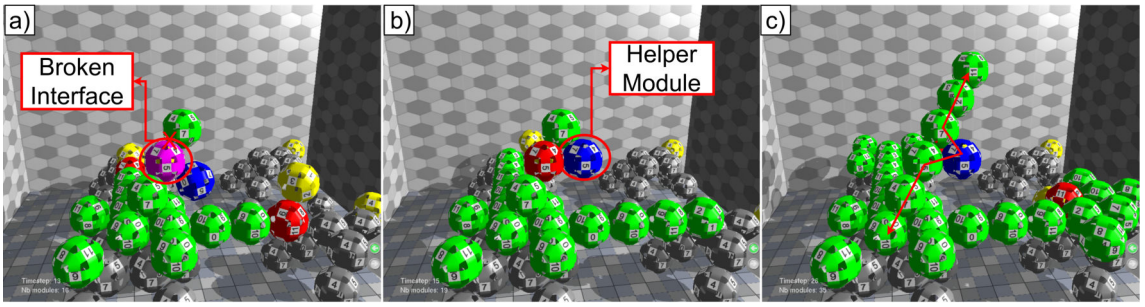



Figure 2.3: A simulation example for handling a broken interface. a) a broken interface is detected. b) the Helper module creates a bridge of communication between disconnected modules . c) communications through the broken interface are passed through the Helper.

delays that can be looked at as the distance from each synchronization point to the best-fit line is accumulated, which can reduce the efficiency of least-square linear regression on large-scale modular robots with a network that spans on large number of hops.

**Contribution** During the thesis work, in Bassil et al. (2021a), we presented and compared two new methods to compensate for clock skew to improve the performance of MRTP in large-diameter modular robot networks. The first, Adaptive Rate Search (ARS) is a lightweight dynamic search method that can provide at any time the adapted rate of the clock of a module by adjusting its provided rate value according to the global time received from the reference module. The second, Enhanced Linear Regression (LR+), uses a Bayesian approach to reduce the uncertainty error induced at each hop along the synchronization path by recursively combining knowledge of a module's parent with a module's local knowledge to calculate an improved estimation of the global time. The results showed that both ARS and LR+ can significantly improve network-wide tight synchronization compared to the linear regression method used in MRTP. While LR+ has the highest precision, ARS has a smaller memory and computation footprint.

## 2.5/ FAULT TOLERANCE

One of the challenges of modular robots is to ensure that they are fault tolerant, that is, they can continue to function even if some of their individual modules fail. Faults such as a broken interface, a loss of power, an incomplete motion, a faulty docking, etc. can probably occur during the self-reconfiguration process. Many self-reconfiguration algorithms exist in the literature, yet, to the best of our knowledge, they do not consider module failures during the self-reconfiguration process. Therefore, to tolerate faults, some mechanisms must be implemented to guarantee the completeness of a self-reconfiguration algorithm to successfully achieve the desired goal shape.

**Contribution** In Bassil et al. (2022), we provided a fault tolerance mechanism to deal with communication failures caused by a broken interface between two connected modules executing the self-reconfiguration algorithm proposed in Thalamy et al. (2021a) where the modules continuously flow upward from a reserve of modules placed below the scene to build a scaffold of an object. During their motion, the modules exchange messages to coordinate their motions in the objective to keep enough space between two following modules to avoid blocking and collisions. A single communication failure can interrupt the flow and results in an incomplete scaffold construction. Our solution consists of using a helper module that serves as a communication bridge between two adjacent modules attached by a broken interface as shown in Figure 2.3.

## 2.6/ CONCLUSION

The chapter presents the challenges addressed in this thesis. They include self-reconfiguration, clustering, time synchronization, and fault-tolerance. Moreover, it presents existing work for each challenge and highlights my contribution to each of these challenges.

# 3

# Experimentation and Simulation Tools

## Contents



## 3.1/ Introduction

This chapter presents the experimentation and simulation tools that were used in this thesis to develop, implement, and analyze the proposed algorithms. Section 3.2 introduces the *BlinkyBlocks* system, a modular robotic system based on cubic lattices that can perform distributed computation and communication. Section 3.3 describes the *3D Catoms*, a lattice-based modular robotic system that can move by rotating on each other using electrostatic forces. Section 3.5 provides a review of existing simulation tools and explains the *VisibleSim* software, a simulation platform that can model the behavior and interactions of various modular robotic systems, including *BlinkyBlocks* and *3D Catoms*.

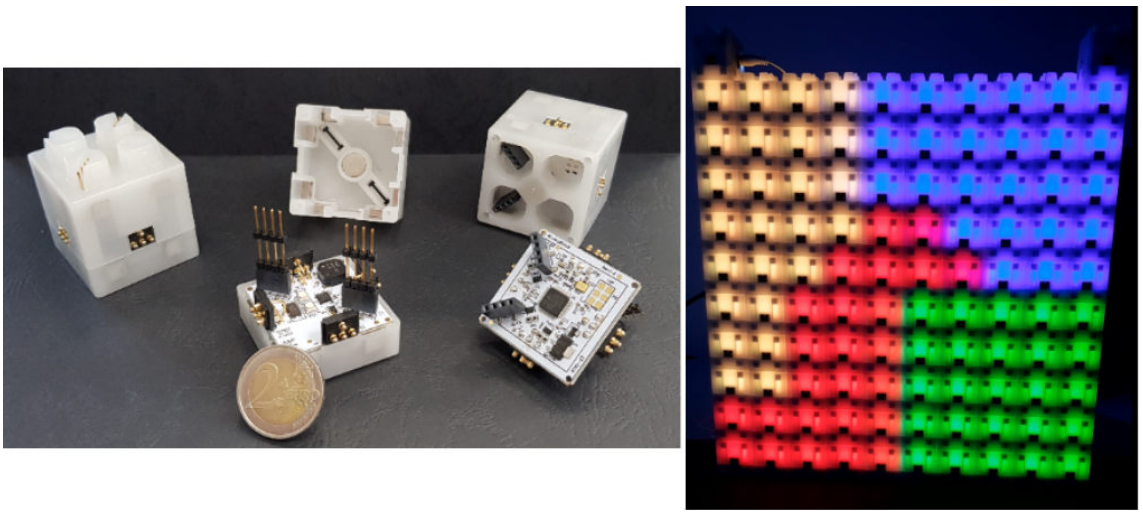


Figure 3.1: On the left a dissection of *BlinkyBlock*. On the right the result of the size-constrained clustering algorithm proposed in this thesis, where each cluster is colored differently.

## 3.2/ BLINKYBLOCKS

*BlinkyBlocks* are centimeter-size cubic robotic blocks. The first version Kirby et al. (2011) is developed as part of the Claytronics project. The second version, used in this thesis work, is developed at FEMTO-ST. In this work, *BlinkyBlocks* are used to implement the proposed size-constrained clustering algorithm and demonstrate its execution on real hardware. In addition, they are used in simulation to validate and analyze the shape recognition algorithm.

Each *BlinkyBlock* module has its own computational power provided by an ARM Cortex M0 32-bit microcontroller. A *BlinkyBlock* module can be attached to up to 6 neighbors using magnets on each of its sides and can communicate with its directly attached neighbors by exchanging messages with a maximum payload size of 227 bytes over serial links controlled by Universal Asynchronous Receiver/Transmitter with 6 Mbps speed. A single block is connected to a power supply, and the power is distributed to all blocks through dedicated pins. *BlinkyBlocks* can detect sound using a microphone sensor and can detect its orientation using a gyroscope. They are also equipped with RGB LEDs and speakers. The ensemble can be reconfigured manually at will by detaching and attaching the blocks by hand. They are aware of the existence of an attached neighbor and can be programmed to react to neighbor-addned or neighbor-removed events.

All *BlinkyBlock* run the same program. The executable program is written in C language, is compiled on an external computer, and is disseminated to the blocks by chunks of bytes through a spanning tree rooted at a block connected to the computer. A custom firmware is pre-programmed on the blocks to handle control commands such as creating

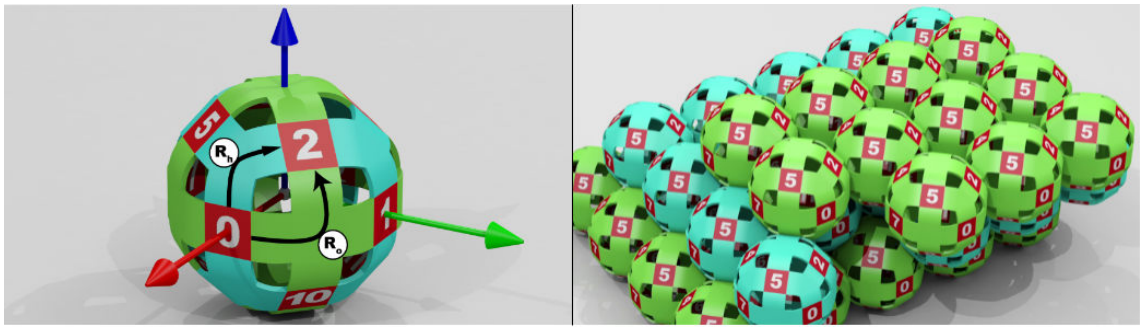


Figure 3.2: On the left, a *3D Catom* with two rotation methods: The first on the octagonal surface area $R_o$ and the second on the hexagonal surface area $R_h$. An arrangement of *3D Catoms* in a FCC lattice where a *3D Catom* can have up to 12 neighbors.

a spanning tree, distributing a unique identifier for each block, and disseminate a program. A logging system can be used to send messages from the blocks to be printed on a connected computer console.

*BlinkyBlocks* are currently used mainly for education and research on distributed systems and to create interactive artistic structures ( ReactiveMatter). Many publications on distributed algorithms in the context of PM were implemented and validated using *BlinkyBlocks*. They include algorithms on centrality leader election Naz et al. (2015), time synchronization Naz et al. (2018b); Bassil et al. (2021b), coordinate distribution Piranda et al. (2023), and mechanical stability Piranda et al. (2021).

## 3.3/ 3D CATOMS

*3D Catoms*, first proposed in Piranda and Bourgeois (2018), are quasi-spherical modules with a diameter on the millimeter scale that can be arranged in a regular 3D grid described as a Face-Centered Cubic (FCC) lattice. Unlike *BlinkyBlocks*, *3D Catoms* have motion capabilities. A *3D Catom* can move around the FCC grid by rotating on the surfaces of fixed *3D Catoms*.

The geometry of a *3D Catom* consists of 12 flat squares used as connectors to latch with other *3D Catoms* using electrostatic forces (shown in red and numbered 0 to 11 in Figure 3.2) which form an hexagonal close packing of modules. Two neighboring connectors are separated by two types of surfaces: a hexagonal surface (colored blue in Figure 3.2)

Table 3.1: Coordinates of cells in the neighborhood of a *3D Catom*

| Plane | $z$ is even | $z$ is odd |
|---|---|---|
| $z-1$ | $(x-1,y-1)(x-1,y)(x,y)(x,y-1)$ | $(x,y)(x,y+1)(x+1,y)(x+1,y+1)$ |
| $z$ | $(x-1,y)(x+1,y)(x,y-1)(x,y+1)$ | $(x-1,y)(x+1,y)(x,y-1)(x,y+1)$ |
| $z+1$ | $(x-1,y-1)(x-1,y)(x,y)(x,y-1)$ | $(x,y)(x,y+1)(x+1,y)(x+1,y+1)$ |

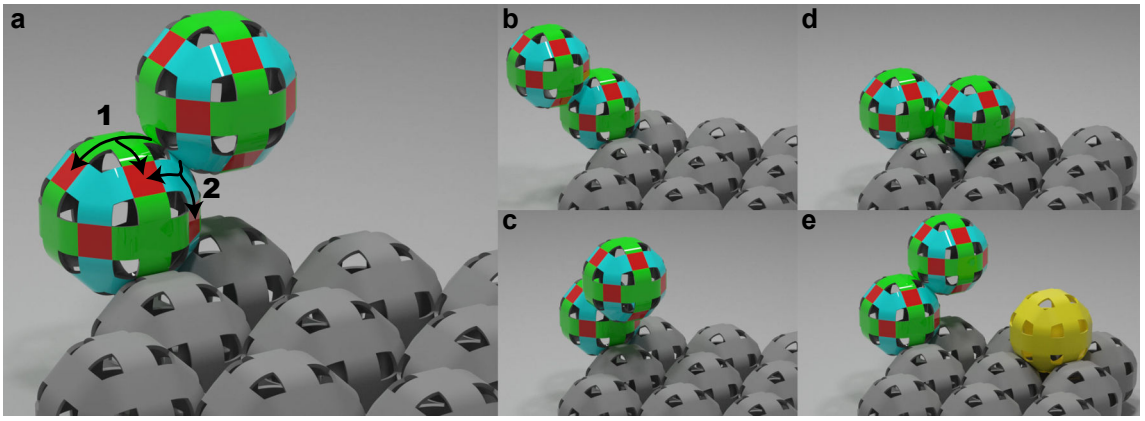



Figure 3.3: Motion capabilities of the *3D Catom*. a) Arrows #1 and #2 shows the two kind of rotations respectively along octagonal surface (in green) and hexagonal surface (in blue). b) Shows the final position of the top *3D Catom* after a rotation along #1 arrow. c) Shows the final position of the top *3D Catom* after a rotation along #2 arrow. In figure d) the final position of the top *3D Catom* is reachable but the same position is not reachable in figure e) due to yellow module.

separates 3 connectors and an octagonal surface (colored green in Figure 3.2) surface separates 4 connectors. A coordinate system is used to define a position in the FCC lattice. A module in a position $(x, y, z)$ can be connected with up to 4 neighbors in each of the planes $z$, $z-1$ and $z+1$. The positions of the neighbors are given in Table 3.1.

Electrostatic connectors are used on the surface of a *3D Catom*, along with latching and motion actuation, to communicate with its latched neighbors by exchanging messages. Therefore, communication is carried out locally, where a *3D Catom* can only communicate with its latched neighbors.

A *3D Catom* can move and change its position in the FCC lattice by rotating on the surface of a fixed neighbor that acts as a pivot. To do so, they use electrostatic actuators placed on the hexagonal and octagonal surfaces to rotate from one connector on their surface to another on the pivot module. A pivot module must not move while actuating a moving module, and a rotation actuation moves one and only one module from its position to an adjacent empty cell (a module cannot carry other modules while rotating). The motions capabilities of a *3D Catom* are shown in Figure 3.3.

The motion of a *3D Catom* is subject to the following constraints that require some coordination and mechanisms to overcome:

- **Collision constraint**: no more than one *3D Catom* should move to the same empty position simultaneously to avoid collisions. In other words, the paths of two motions must not intersect.
- **Bridging constraint**: a *3D Catom* cannot enter a free position that has two occupied positions in opposite directions, as shown in Figure 3.3.

- **Blocking constraint**: a *3D Catom* entering a free position must not be blocked by another *3D Catoms* on its motion path.
- **Connectivity constraint**: a *3D Catom*'s motion cannot disconnect the ensemble.

A self-reconfiguration planner must take into consideration motion constraints to ensure the correctness and completeness of the process.

## 3.4/ Programming Model and System Assumptions

In this section, we enumerate the system assumptions and the programming model considered in our work to program the distributed system formed with connected ensembles of *3D Catoms* and *BlinkyBlocks*:

- All modules run the same distributed program and perform the computations locally on each module.
- All communications are done in a local fashion, where a module can only communicate with its directly connected neighbors when it is docked by exchanging messages.
- All operations are performed asynchronously.
- The interconnections graph must be connected all the time. This adds an additional constraint to be considered by a self-reconfiguration algorithm.
- A module is aware of the presence of a connected neighbor in an adjacent cell.
- All modules share the same coordinate system. Each module stores its coordinates locally and updates them after each movement. A distributed algorithm to distribute the coordinates in the modules is proposed in Piranda et al. (2023).
- A module can react to the reception of a message, the connection and disconnection of a neighbor, and any internal event such as a timer event or a rotation end event (in the case of a *3D Catom*).
- Each module is assigned a unique identifier. An algorithm to distribute unique identifiers in modules is presented in Assaker et al. (2022).

## 3.5/ Simulation Environment

Simulation is extremely important in research and engineering, as it allows the design, testing, and refinement of products or systems in a virtual environment before building

physical prototypes or deploying them in real-world scenarios. In the context of autonomous robots, simulation allows faster and cheaper development cycles of physical hardware and software. It provides an environment for efficient learning and adaptation of robots by providing diverse and rich data and feedback. Furthermore, simulators play an important role in facilitating collaboration and communication between engineers and researchers, as they provide a tool to share and compare models, methods, and results. In the context of PM, simulation is useful for developing software foundations and understanding the dynamics of such large-scale shape-shifting complex structures, especially when testing and evaluating algorithms on a large number of modules or when hardware components are not yet available.

Furthermore, simulators ensure reproducibility of simulated scenarios. Being able to repeatedly run the same simulation with exactly the same order of events at the same time is of utmost importance. They facilitate debugging by allowing researchers and engineers to consistently verify and understand results, isolate issues, and reliably identify errors. In a real-world distributed system, the order of messages and events as well as the environment can be subject to stochasticity, making it challenging to analyze problems and investigate root causes. By providing reproducibility, a simulator enables controlled replay of specific scenarios, allowing precise observation of module interactions and reactions, and more effective troubleshooting.

In this section, we present the existing simulation tools for MSR, then we present *VisibleSim*: a simulator for lattice-based MSR used to implement and validate the work of this thesis.

### 3.5.1/ Existing Simulation Tools

Table 3.2: Modular robot simulators comparison.

| **Simulators** | **Characteristics** | **Open source** | **Last Activity** |
|---|---|---|---|
| Gazebo<br>CoppeliaSim<br>Webots | General purpose physics simulation for mobile robots | yes<br>no<br>yes | Up to date |
| ARGos | Heterogeneous swarm robots | yes | Up to date |
| Rebots | GUI Drag-and-drop, Physics-based, Roombots, Superbot, Smores | yes | Up to date |
| USSR | Physics-based, ATRON, M-TRAN, Odin | yes | 2012 |
| DPRSim | Physics-based, Claytronics | yes | Discontinued |
| VisibleSim | Behavioral, large-scale lattice-based modular robots | yes | Up to date |

There are many existing robot simulators available Collins et al. (2021), each with different features and capabilities. Table 3.2 shows a list of simulators that can be used for modular robots. Some of the most popular are Gazebo simulator Koenig and Howard (2004), CoppeliaSim Rohmer et al. (2013) (formerly called V-REP), and Webots Michel (2004). They are general robotic simulation tools that allow users to create and test mobile robots in realistic 3D environments. They support various physics engines, sensors, actuators, and communication protocols, and are used mainly for monolithic robots, but sometimes they are used to simulate modular robots, such as V-REP in Liu et al. (2017).

Some specialized simulators include ARGos Pinciroli et al. (2012), a physics-based real-time simulator swarm robotics that can support large heterogeneous multi-robot systems.

There are many simulators specific for modular robots, such as Rebots Collins and Shen (2016), a physics-based simulator that allows user interactions via drag and drop GUI. It supports multiple modular robots with multiple architectures, such as Roombots Spröwitz et al. (2010), Superbot Salemi et al. (2006), Smores Liu et al. (2023) and can be extended to support new modules. USSR Christensen et al. (2008) is another general-purpose simulator for modular robots that supports multiple modules such as M-TRAN Kurokawa et al. (2008), Atron Østergaard et al. (2006) and Odin Lyder et al. (2008). DPRSim Ashley-Rollman et al. (2011) is a simulator developed under the Claytronics project. It has a physics engine and supports simulations with millions of 2D and 3D Catoms in the purpose of performance evaluation of distributed algorithms and to visualize the complex behavior of these tiny modules. It also integrates advanced debugging, visualization, and tracing features. However, its development has been discontinued.

*VisibleSim* Thalamy et al. (2021b) is a simulator developed in the Programmable Matter project specific to lattice-based modular robots. It does not include a physics engine, which can be prohibitive when simulating a large number of modules that can scale up to 32 million. It is a behavioral simulator that allows its users to fundamentally study the distributed control algorithms of a large ensemble of modules in a perfect environment and visualize the complex behaviors of the modules that execute parallel actions. It supports all modules developed under the Programmable Matter project (see Section 1.4). *VisibleSim* is used to evaluate all algorithms proposed in this thesis and is detailed in the next section.

### 3.5.2/ VISIBLESIM

*VisibleSim* is an open-source behavioral simulator designed for lattice-based modular robots. It can be accessed publicly on GitHub[1]. It is the simulator of choice for developing and evaluating the algorithms proposed in this thesis. Its development started in early

[1] *VisibleSim* git repository: https://github.com/ProgrammableMatterProject/VisibleSim

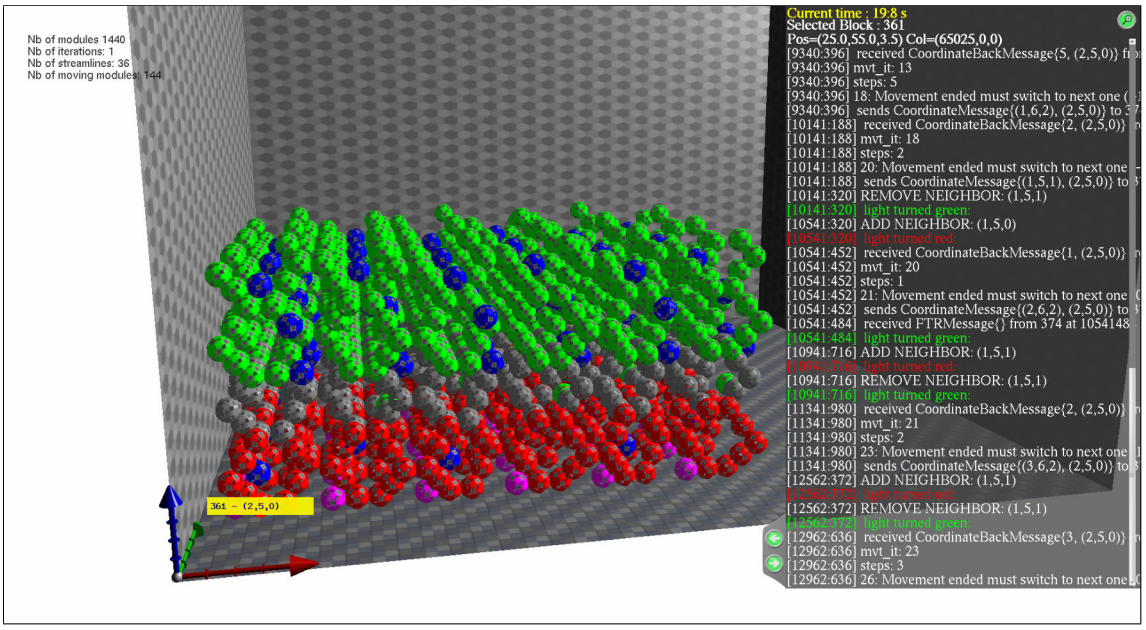



Figure 3.4: A screenshot taken during the execution of a self-reconfiguration algorithm using *3D Catoms*. On the right a console shows the trace of events executed by the selected module at the bottom left. The modules are colored differently to reflect user defined states. On the top left, important real-time statistics are displayed.

2010 by our team at FEMTO-ST, and since then it has been updated, and new features have been continuously added. It supports all modular robot modules with different lattice types developed in the Programmable Matter project, including the *BlinkyBlocks* and *3D Catoms* used in this thesis. New types of modules can also be easily added.

*VisibleSim* is a C++ framework for simulating lattice-based modular robots based on a discrete-event engine where a scheduler handles a sequence of discrete events generated by the modules. Event processing might generate new events to be scheduled. When a module generates an event, the event is pushed to the scheduling queue with a specified delay. Then the event will be executed when the scheduler time corresponds to the event execution time. A class called *BlockCode*, represents a module controller. The user must extend two main components of the *BlockCode* class: a startup function used to initialize the module's state and event handling functions executed in response to the handling of simulation events by the scheduler. For example, a user can program the module to handle events such as a message received, an interruption event, a neighbor added or removed, a motion ended, or any other custom-defined event.

*VisibleSim* is equipped with a Graphical User Interface (GUI) written in OpenGL shown in Figure 3.4. Initially, it reads an XML configuration file that specifies the positions of the modules in the lattice and places them in the simulation world. During execution, the user can navigate and interact with the world by adding, removing, and tapping modules, which are events that can be handled in real time. A console built into the GUI shows

the sequence of events executed by all modules or by a single module when clicking on it. This provides the user with valuable information during execution to debug and trace the chain of events that led to the current state. The GUI is also a powerful visualization tool that allows users to take screenshots and record execution, generating high-quality illustrations and videos that help communicate scientific results.

At the end of a simulation, *VisibleSim* provides the user with important statistics such as simulation time, the number of messages exchanged, and the number of motions at the ensemble and module levels, allowing evaluation and comparison of algorithms.

## 3.6/ CONCLUSION

To conclude, this chapter has presented an overview of the research environment in which the thesis is conducted, focusing on the modular robotic systems employed. Specifically, the *BlinkyBlock* real cubic modules were used in this study to validate algorithms that do not rely on movements on physical hardware. On the other hand, the *3D Catoms* spherical modules, which are capable of moving in a face-centered cubic lattice, were used to apply the proposed self-reconfiguration methods.

Furthermore, this chapter introduced the programming model and system assumptions, which serve as the foundations for the algorithms proposed in subsequent chapters. Additionally, the chapter discusses the simulation environment using *VisibleSim* discrete-event behavioral simulator, which provides a controlled environment to test and validate the proposed distributed algorithms.

# II

# CLUSTERING

# 4

# SIZE-CONSTRAINED CLUSTERING

## Contents



## 4.1/ INTRODUCTION

To enhance the self-reconfiguration process, clusters of modules can be reconfigured in parallel. It has been shown in Moussa et al. (2021) that cluster-based self-reconfiguration can improve the execution time and communication efficiency of the process. In fact, the authors showed that reconfiguring clusters of modules in parallel offers a performance improvement proportional to the number of clusters, compared to the reconfiguration of the entire ensemble of modules.

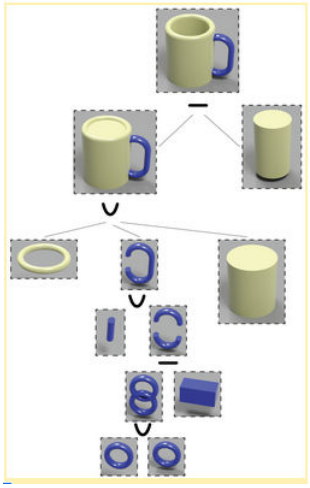


Figure 4.1: A mug represented in CSG tree.

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. The proposed clustering algorithm serves as a preprocessing step to allow the self-reconfiguration process to be enhanced by applying cluster-based approaches. A tree-based density-cut algorithm was proposed in Bassil et al. (2020) 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.

Prior to self-reconfiguring, modules are not aware of their initial configuration, they only know of the existence of their directly connected neighbors. 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 so it knows if it is already in the goal shape or must move to fill an empty position in the goal shape. In Tucci et al. (2017) a solution based on Constructive Solid Geometry (CSG) Requicha et al. (1977) is proposed. It can be used to determine the number and sizes of clusters. It defines the goal shape as a tree made up of basic geometric objects and transformations (union, intersection, difference) that, when combined, form the final scene as shown in Figure 4.1. First, the 3D object to be formed is discretized and encoded in the CSG tree by 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 cluster sizes to a master module, to be then flooded and stored in all modules in an initial phase before starting the clustering process.

In this chapter, we present SC-Clust a size-constrained clustering algorithm that groups modules in any initial configuration into $k$ clusters where $k$, the number of clusters and their

sizes are predefined a priori based on the goal configuration and can be disseminated with the goal shape representation to all the modules. The algorithm proceeds by creating a spanning-tree, cutting then adjusting branches to form the clusters.

Section 4.2 presents the problem and the system assumptions in a formalized way and demonstrates that the size-constrained clustering problem is NP-Complete. In Section 4.3 the SC-Clust algorithm is described; then its time and communication complexity are analyzed in Section 4.4. Section 4.5 presents the simulations and results and the comparison between SC-Clust and DCut: a clustering algorithm for lattice-based modular robots without a size-constraint.

## 4.2/ 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, $w : E \longrightarrow \mathbb{R}^{+*}$.

The modules are homogeneous, placed in a regular lattice, and are attached border-to-border. Since they can only communicate with their directly connected neighbors in their adjacent cells, they form a sparse communication graph with a large network diameter Naz et al. (2018a).

**Definition 1: Size-constrained partition**

A size-constrained partition $G_i(V_i, E_i, W_i)$ is a connected subgraph of G that has a predefined number of nodes $s_i$, that is, $|V_i| = s_i$.

**Definition 2: Size-constrained k-partitioning**

partitions the graph $G$ into $k$ size-constrained partitions (Definition 1) such that:

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 that: $\forall i \neq j, V_i \cap V_j = \emptyset$
3. The size of each size-constrained partition $G_i$ is predefined before partitioning, such that: $\sum_{i=1}^{k} s_i = |V|$

Figure 4.2 (a) shows a correct size-constrained $k$-partitioning. Figure 4.2 (b) shows an incorrect solution since the green cluster is disconnected. The objective of this work is

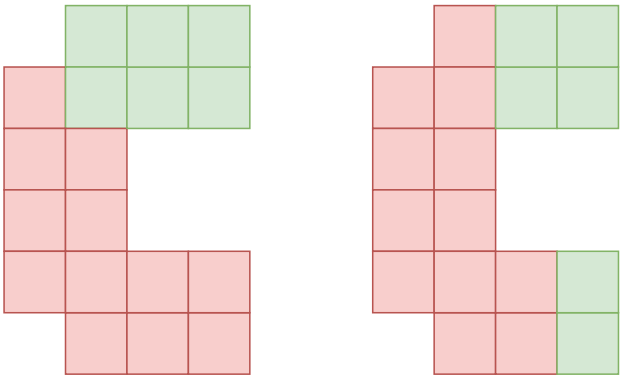



Figure 4.2: An example of two possible size-constrained k-partitioning where $k = 2$, $s_1 = 12$ (Red) and $s_2 = 6$ (Green). (a) shows a correct size-constrained k-partitioning. (b) shows an incorrect size-constrained k-partitioning because the second partition shown in green is disconnected.

.

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 Tucci et al. (2017).
- Each module is identified by a 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.

### 4.2.1/ 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 Berenger et al. (2018) and 3-partition in Feldmann (2013). 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 Garey and Johnson (1979), 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.

## 4.3/ ALGORITHM DESCRIPTION

In this section, we propose the SC-Clust algorithm, a solution to 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 4.3.1). Second, a Minimum Spanning Tree (MST) is built. A fully distributed and asynchronous algorithm Gallager et al. (1983) 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 that have the desired number of modules (Section 4.3.3).

### 4.3.1/ WEIGHT CALCULATION

In this section, an edge weight measure is defined. We start with the following definitions:

**Definition 4: Anchors**

Given a geometric 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 results in having lower weights at the edges that connect the modules near the configuration boundary. This will subsequently result in clusters being positioned closer to borders, making it easier to move modules around for self-reconfiguration.

Anchor 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 and then transmitted to all modules via the built tree. Upon reception, modules can calculate and store the distance to their nearest anchor, and then store their connected edges weights.

### 4.3.2/ TREE CONSTRUCTION

After all modules have stored their adjacent edge weights, a Minimum Spanning Tree (MST) is built. It minimizes $\sum_{(u,v)\in V_{MST}} w(u,v)$. Any distributed algorithm to find an MST can be used. We use a fully distributed asynchronous algorithm called GHS proposed in Gallager et al. (1983). 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 Pandurangan et al. (2018); Blin and Butelle (2001); Haeupler et al. (2018); Mashreghi and King (2021), 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 have a 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 the lower identity number 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, the fragments are extended by merging with other fragments. The 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 merge through their minimum outgoing edge. To find the minimum outgoing edge of a fragment, a message is broadcast 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 upward 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. If the two fragments choose the same minimum outgoing edge, they merge, and the edge chosen by the two fragments is called *core edge*.

During the last phase, two fragments will merge through a *core edge* into one large fragment that forms the MST. We refer the reader to Gallager et al. (1983) for a complete description of the algorithm. Once the MST is formed, we can proceed with its partition. One can choose one of the core nodes connected to the core edge as the root of the tree. However, to have clusters distributed closer to the borders as much as possible, we choose the root to be the node with a 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 in a 2D regular lattice is shown in Figure 4.3.

### 4.3.3/ TREE PARTITIONING

In this phase, given the set of desired cluster sizes $S$, the MST is partitioned to obtain $k = |S|$ size-constrained clusters.

The idea is to find the cut-edge that results in a branch in a way to minimize the difference between the number of modules in the branch and the desired number of modules in the

cluster. The detached branch after removing the cut-edge will form the cluster. 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 originally 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 $V_i$ connected to $c_i$.
Given the set of desired partition sizes $S$, in order to satisfy the size constraint described in Section 4.2, $|V_i|$ should be equal to $s_i$. However, a cut-edge that satisfies this constraint may not exist, since a branch in the MST that has 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, we present a method in Section 4.3.4 that makes additional cuts and

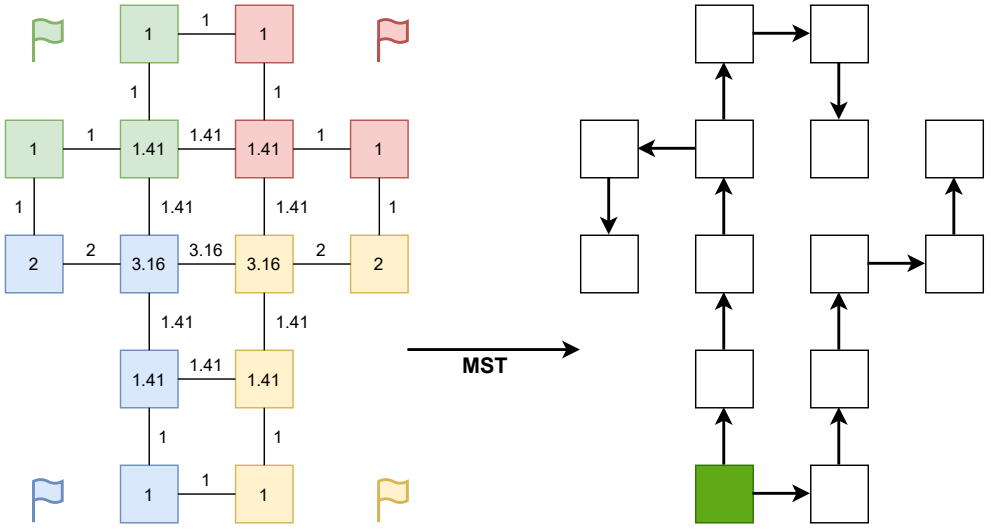



Figure 4.3: An example of MST construction. On the left the weight distributed according to the distance to the nearest anchor (The flag of the same color). On the right, the MST is constructed and the root is colored in green.

associates the resulting branches with the erroneous cluster until it has the desired size.

### 4.3.4/ Additional Cuts

In this section, a new method is presented to deal with size differences after a cut. It requires 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 (a branch that contains at least one module that has a neighbor in $V_i$) with a size equal to $Diff_{c_i}$ 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 4.4. 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 $V_i$.

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

This method guarantees that the resultant 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 can be made, resulting in partitions that contain one module each.

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

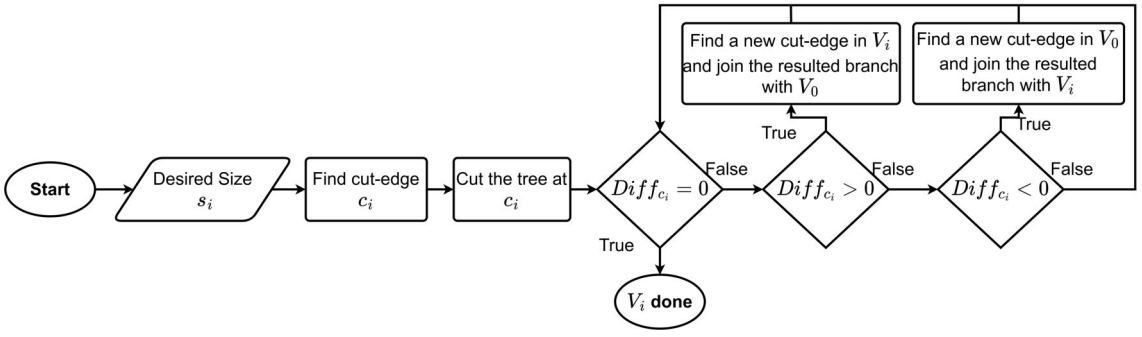



Figure 4.4: Flow chart for the $i^{th}$ partition.

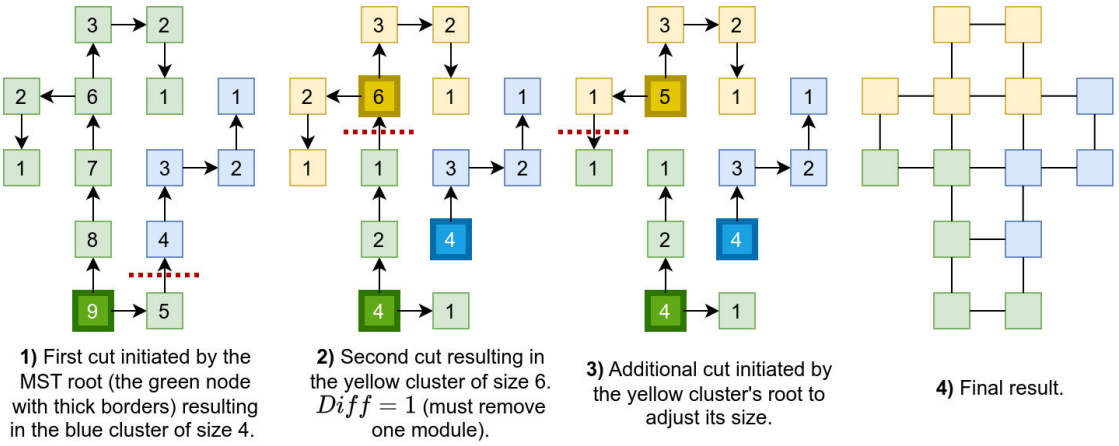



Figure 4.5: A partitioning example given clusters sizes of $\{4,5,5\}$ using additional cut. A cut is shown as the red dotted line. The number on each node corresponds to its sub-tree size. Nodes with larger borders and darker colors are the clusters roots.

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 partition size $i$ ($Diff_{c_i} \neq 0$), $adj$ takes the value of the partition id $i$ to which the resultant 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 initiates 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 and 2. 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 3, lines 32, 35). A neighbor in $V_i$ will always exist because $V_0$ and the erroneous cluster were initially connected. 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.

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 resultant cluster has modules in excess), it calls $cut(true, Diff, 0)$ to find a new $cut\text{-}edge$ within its partition, and the resultant branch is rejoined with the initial partition $V_0$ to minimize the difference (algorithm 3, line 45,51). Otherwise, it sends a

**Algorithm 1:** Partitioning algorithm: initialisation and **FIND_CUT** message handler

```
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
```

REPORT_CUT message with the value of $Diff_{c_i}$ to the root of partition $V_0$ (algorithm 3, 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 resultant branch to the partition $V_i$ (algorithm 4, line 61,63). If after joining a branch with $V_i$ the size of $V_i$ becomes larger than the desired size $s_i$, the root sends the 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 4, 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 4, lines 67, 70). The tree must be updated to join the resultant branches after additional cuts with their corresponding partitions. The $updateTree()$ procedure depends on the algorithm used to build the MST. After considering the nodes in additional branches as disconnected nodes, we use the tree maintenance algorithm described in Diaz and Mendez (2019) where the GHS algorithm for building the MST is re-launched inside the partition to join an additional disconnected branch.

**Algorithm 2: RESP_CUT** message handler

```
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 resultant branch
                      to cluster 0
41                 send CUT(0) to toBestCut
```

Figure 4.5 shows an example of partitioning the MST shown in Figure 4.3 into three clusters with given desired sizes of $S = \{4, 5, 5\}$. The first cut initiated by the MST root results in a branch of size four that forms the first cluster $V_1$ colored blue. The second cut results in a branch of size six, but the desired size is five. So, the second cluster $V_2$ colored yellow has one additional node ($Diff_{c2} = 1$). Therefore, an additional cut in $V_2$ is made to cut a branch with one node adjacent to $V_0$, and the resultant branch is associated

**Algorithm 3: CUT** message handler

```
43 Msg Handler CUT(i):
44   if recut = false then
45     toLastcutAt ← toBestCut
46   if toBestCut = NULL then
47     cutAt ← true
48     Cluster ← i
49     myDiff ← subTreeSize − desiredSize
50     assign sub-tree to cluster i
51     if myDiff > 0 then
         // Case 1: cluster i has an excess of modules. Must find a new
            cut to join the resulting branch to cluster 0
52       execute cut(true, myDiff, 0)
53     else
         // Case 2: Cluster i has a deficit of modules. Report the
            difference to the root
54       send REPORT_CUT(−myDiff) to parent
55   else
56     send CUT(i) to toBestCut
```

with $V_0$. The end result is three clusters with the desired sizes: the blue one with size four, the yellow one with size five, and the green one with size five.

## 4.4/ 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.

### 4.4.1/ 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 connected modules. Therefore, the communication complexity of the first phase is $O(n + m)$.

The second phase consists of building a minimum spanning tree. We use the GHS algorithm described in Gallager et al. (1983) which has a complexity of $O(m + n \log n)$ in addition to $O(n)$ to find the root and redirect the edges towards it.

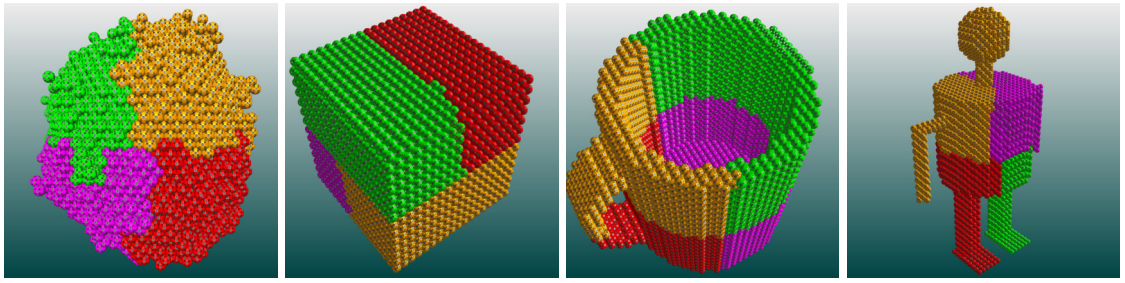
During the third phase, the tree is partitioned to obtain $k$ partitions. The SC-Clust requires

**Algorithm 4: REPORT_CUT** message handler

```
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
           // Case 3: diff = 0. Cluster i is found and it satisfies the size
              constraint.
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
```

$k-1$ 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 resulting from additional cuts, which require $O(k \log n)$ messages.

Overall, by summing the complexities of the three phases, the communication complexity



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

Figure 4.6: DCut results on 4 different shapes with 4 clusters.

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 complexity of communication can be expressed with the number of modules in the system $n$ and is equal to $O(n)+O(n\log n)=O(n\log n)$.

### 4.4.2/ Execution Time

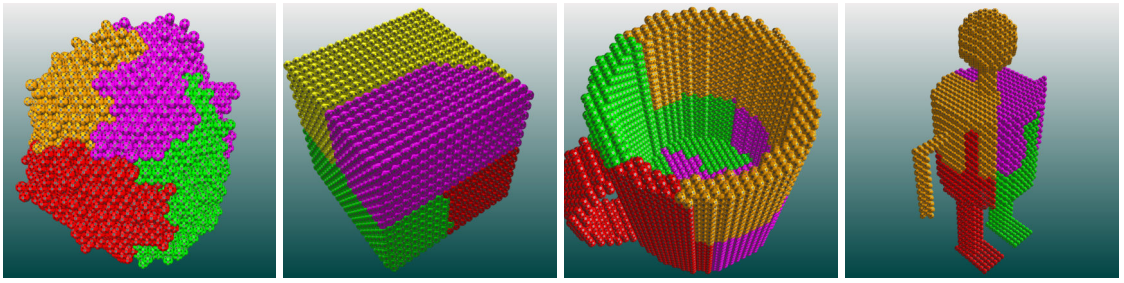
The time required for the first phase in which anchor positions are found and edge weights are calculated depends on the diameter $d$ of the network, since the maximum tree length is bounded by $d$. Three tree traversals are required. Thus, the time complexity of the first phase is $O(d)$.

The time complexity of building the tree in the second phase is $O(n\log n)$ Gallager et al. (1983). 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 to find 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 cluster 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 mainly due to the construction of the MST.



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

Figure 4.7: SC-Clust results on 4 different shapes with 4 equal size clusters.

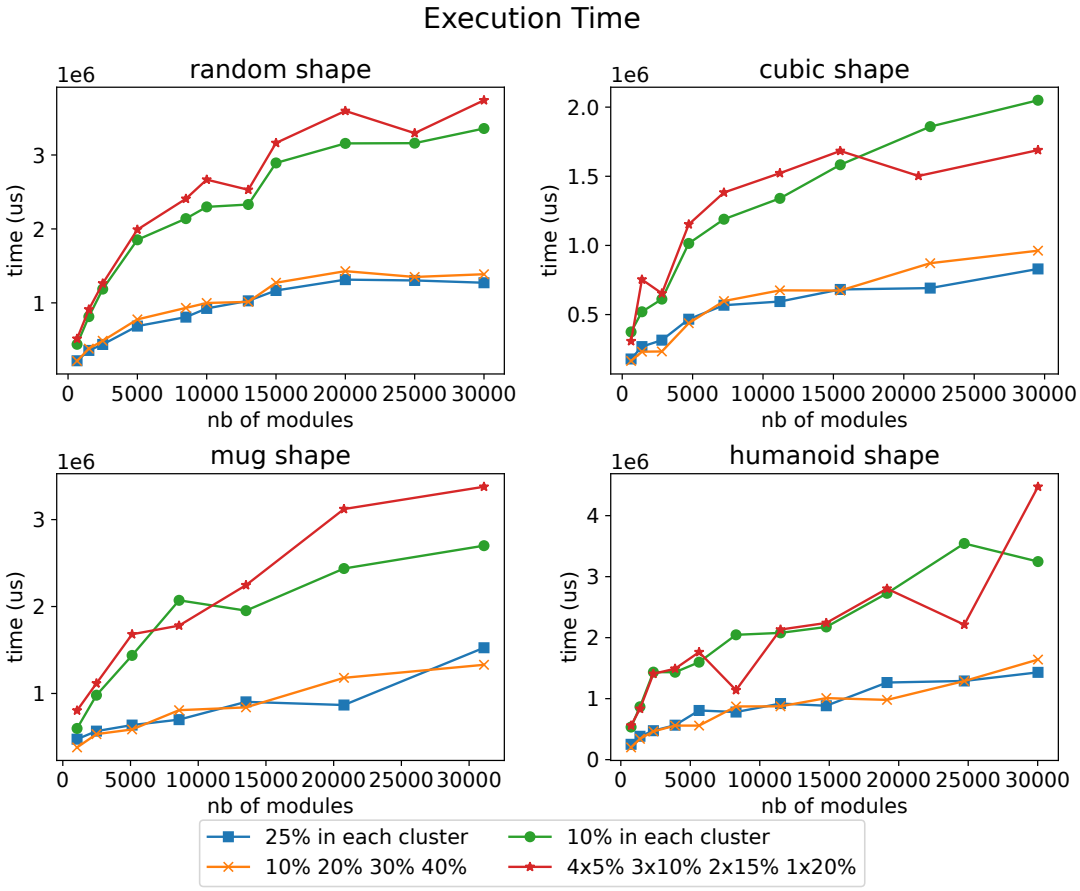



Figure 4.8: SC-Clust execution time evaluation.

## 4.5/ Simulations and Results

Figure 4.6 shows 4 clusters created by DCut Bassil et al. (2020), a clustering algorithm for lattice-based modular robots that results in randomly 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 com-

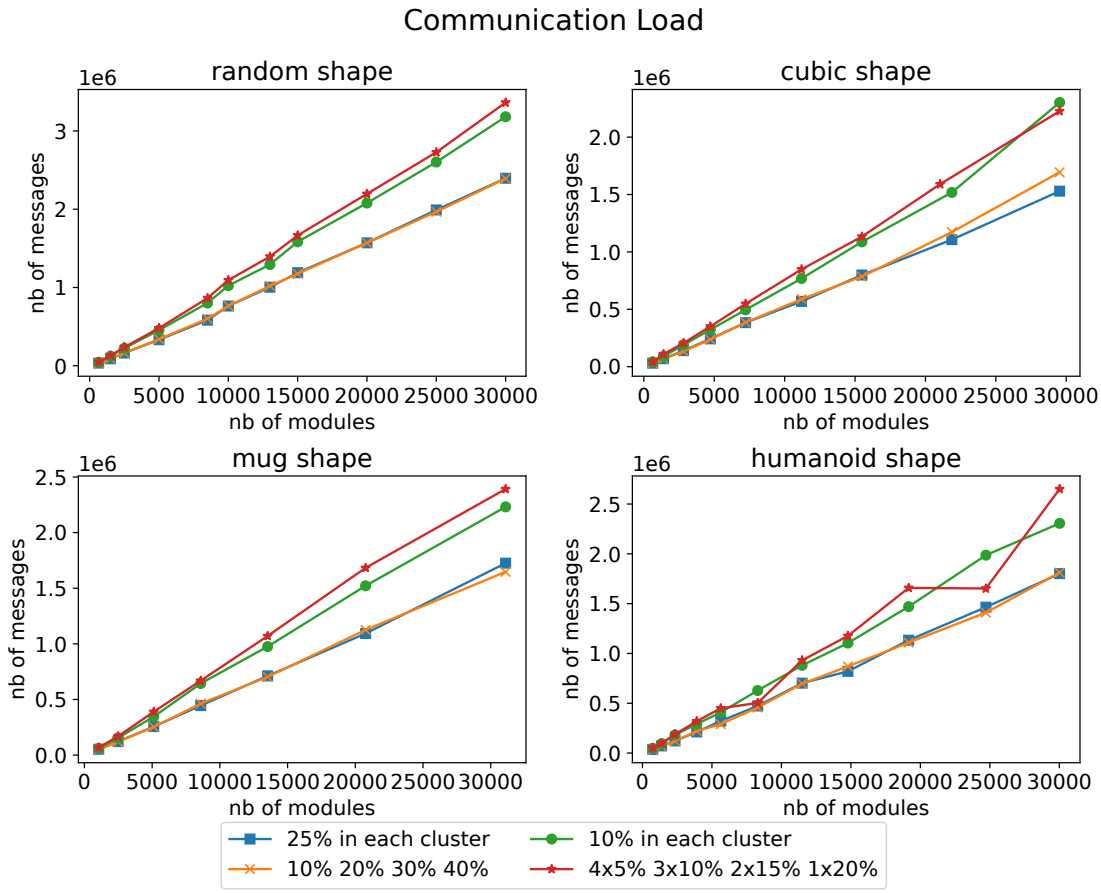



Figure 4.9: SC-Clust communication load evaluation.

ponents of different densities. The 4 clusters are distributed regularly along the borders of the configuration. Figure 4.7 shows 4 clusters created by SC-Clust of equal sizes on the same shapes as in Figure 4.6. 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 4.3.4.

We validated the SC-Clust algorithm on real robotic systems called *BlinkyBlock*. The video[1] shows 6 different experiments on 144 real *BlinkyBlock* 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 *BlinkyBlock* and another time to create heterogeneous clusters with $10\,\%$, $20\,\%$, $30\,\%$ and $40\,\%$ of the set.

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

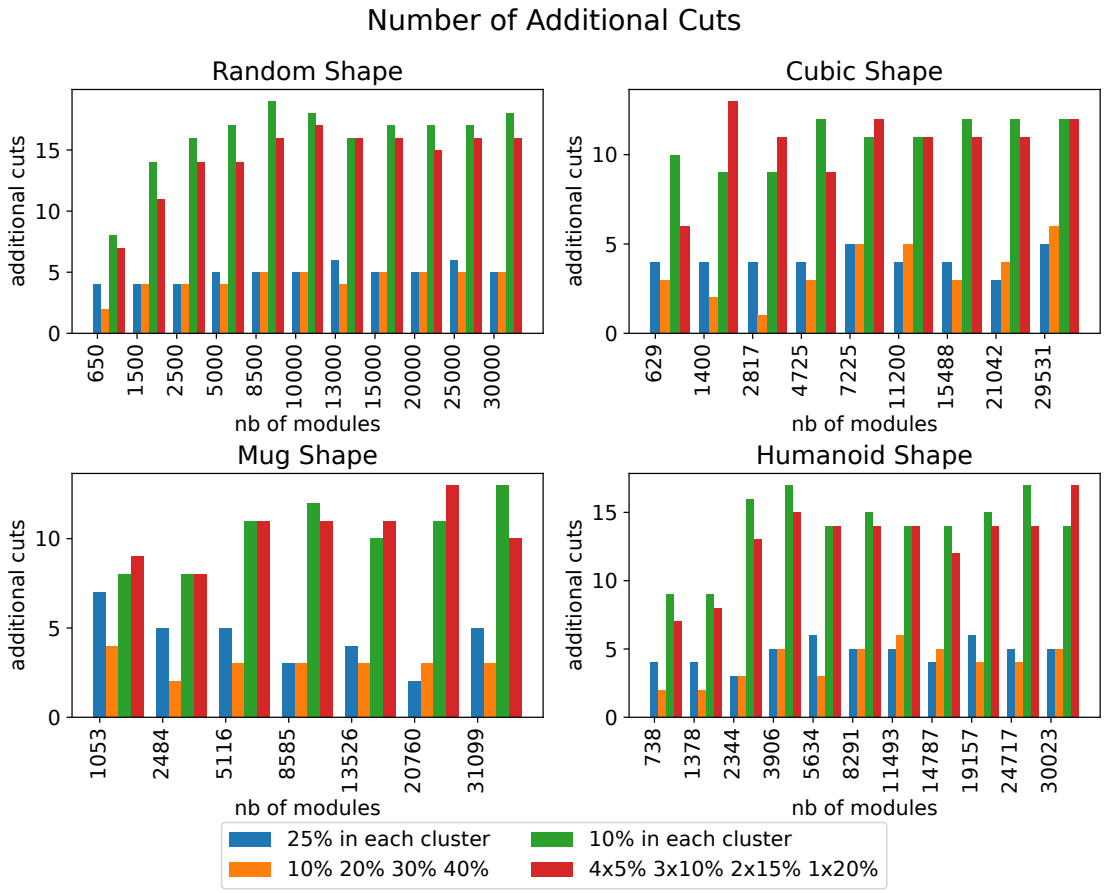



Figure 4.10: Number of additional cuts.

### 4.5.1/ EVALUATING SC-CLUST

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 geometric properties to show that the proposed algorithm finds a solution independently of the geometric 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 $20\,\%$.

#### 4.5.1.1/ Execution Time

Figure 4.8 shows the execution time of the SC-Clust algorithm. We can see that the execution time increases logarithmically as 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 4.4.2 because as the number of clusters increases, the number of cuts to be found increases. Moreover, 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 4.8, the humanoid shape requires more time than the other shapes. Furthermore, when the number of clusters is the same and the cluster size distribution differs, the execution time is affected due to the additional number of cuts (see in Figure 4.10) used to satisfy the size-constraint and the search space to find these cuts which vary according to the clusters sizes.

#### 4.5.1.2/ Communication Load

The communication load is shown in Figure 4.9. The number of exchanged messages for all shapes increases linearly as 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 complexity of the communication load in Section 4.4.1 depends on the number of modules and the connections between the modules. The random shape presents the largest number of connections between its modules; thus, it requires a larger number of exchanged messages. Moreover, when the sizes of the clusters differ while the number of clusters is the same, the number of exchanged messages, which are needed to find additional cuts and join branches to satisfy the size-constraint, is slightly affected.

#### 4.5.1.3/ Additional Cuts

We recall that additional cuts are needed when the resultant cluster size after an initial cut does not satisfy the size constraint. So, additional cuts are made until the cluster size is equal to the desired size. Figure 4.10 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 increases, 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 is affected 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.

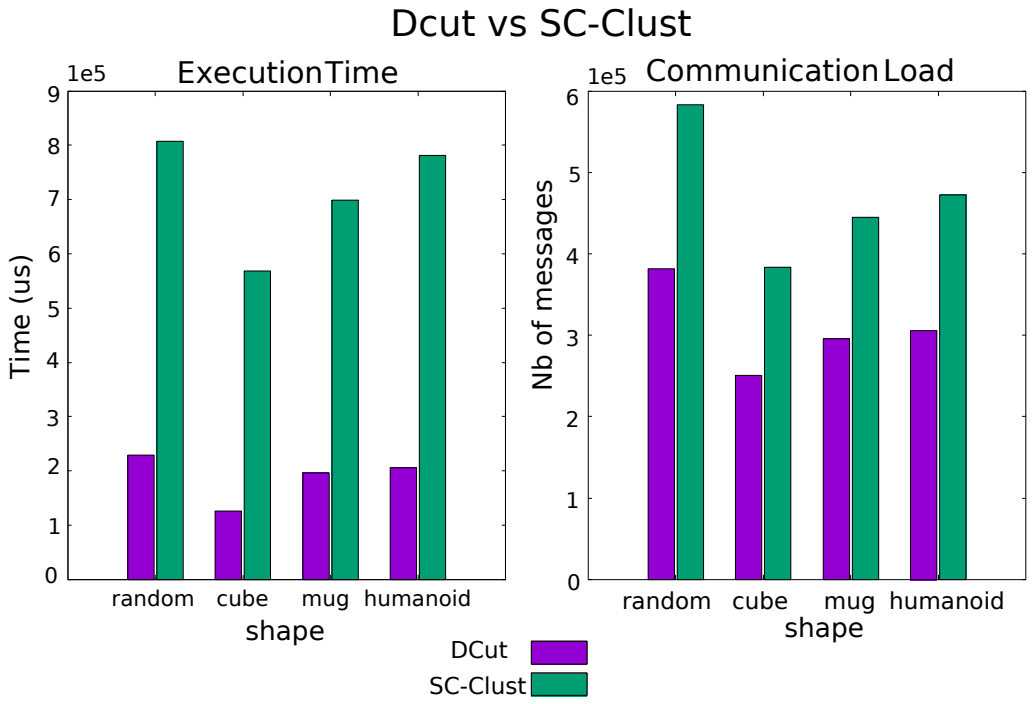



Figure 4.11: Comparing DCut and SC-Clust.

### 4.5.2/ COMPARING DCUT WITH SC-CLUST

Here, we compare the DCut algorithm with SC-Clust. Figure 4.11 compares DCut with SC-Clust in terms of execution time and communication load on the shapes of Figures 4.6 and 4.7 with the same number of modules for each shape and 4 clusters. As seen in Figure 4.11, the SC-Clust requires more messages exchanged in all shapes since for each cut, additional cuts may be necessary 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.

## 4.6/ CONCLUSION

In this chapter, we proposed SC-Clust, a fully distributed size-constrained clustering algorithm based on graph cuts. It groups 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. We evaluated our algorithm on multiple shapes with different geometric properties while varying the number of modules, the number of clusters, and the sizes of the clusters. The results show that our algorithm is scalable and efficient with $O(nlogn)$ time and communication complexity.

# 5

# SHAPE RECOGNITION

## Contents



## 5.1/ INTRODUCTION

The ability of modules of a modular robot to recognize the global shape of the ensemble provides invaluable information that can be used to facilitate and improve various tasks. One key benefit of recognizing the global shape is the facilitation of efficient coordination and collaboration among modules. When modules have knowledge of the robot's global shape, they can better coordinate their actions, movements, and behaviors to achieve efficient self-reconfiguration. Furthermore, by comparing the expected global shape with the actual observed shape, modules can detect the termination of self-reconfiguration and identify inconsistencies, such as module failures, disconnections, or misalignments. Morevorer, shape recognition can be used to report the shape to an external system or operator.

In this chapter, we introduce a shape recognition algorithm for lattice-based modular robots with neighbor-to-neighbor communication. It consists of finding a set of overlapping boxes whose union forms a representation of the current configuration. Using a distributed approach, each module communicates with its direct neighbors to collectively determine the shape of the global ensemble.

Section 5.2 describes the proposed distributed shape recognition algorithm. In Section 5.3 we analyze its computation and communication complexities. Section 5.4 evaluates and analyzes the algorithm where we implemented it in *VisibleSim* using *Blinky-*

*Blocks* modules and compared it with a classic coordinate collection method where each module sends its coordinates through a tree rooted on the connected computer. The results obtained show the efficiency of our algorithm in detecting the current shape of the robot, while also outperforming the coordinate collection algorithm.

## 5.2/ ALGORITHM DESCRIPTION

The main idea of the algorithm is to find a set of full boxes that cover the whole configuration. In this section, we describe the distributed algorithm, shown in Algorithms 5 and 6 for finding the boxes on modular robotic systems where modules communicate using message-passing with their directly attached neighbors.

We assume that all modules share the same 3D coordinate system and that each module stores its coordinates in its memory. This can be done efficiently in lattice-based modular robots, as explained in Piranda et al. (2023); Hołobut et al. (2016). The algorithm is not affected by the orientation of the coordinate axis. However, for simplicity, we use the

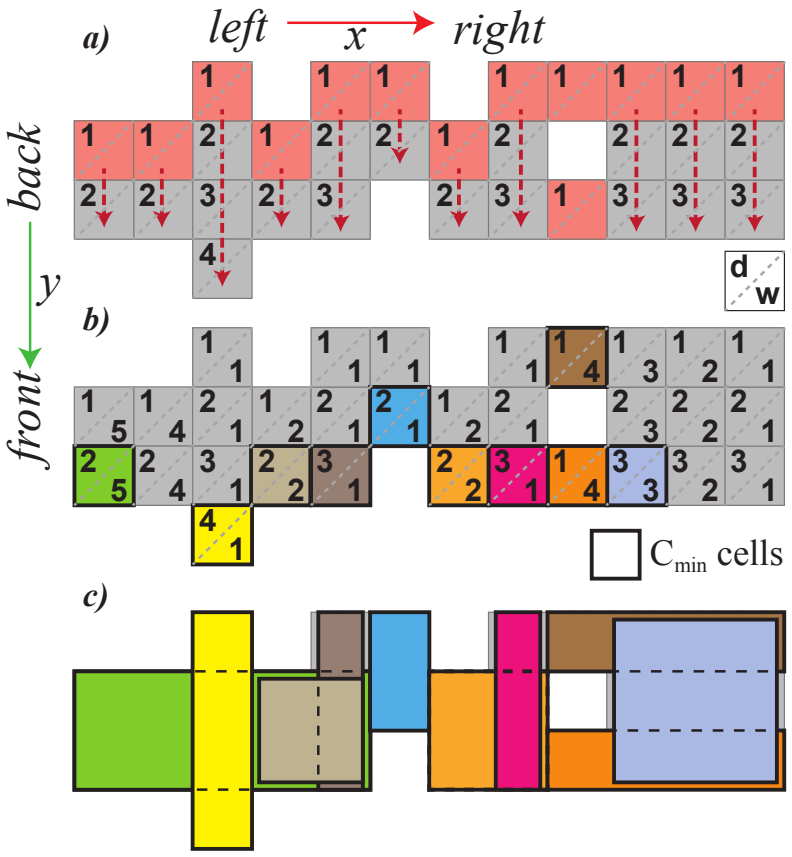



Figure 5.1: Distributed computation of the boxes on 2D. a) Computation of the vertical distance $d$. b) Election of $C_{min}$ cells and computation of $w$. c) Boxes covering the plane using $C_{min}$ cells colors.

orientation of the axis and the direction notation as shown in Figure 5.1. A Box $B$ is defined by two vectors $C_{min}(x_{min}, y_{min}, z_{min})$ and $C_{max}(x_{max}, y_{max}, z_{max})$.

$$X(x,y,z) \in B \Leftrightarrow \begin{cases} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max} \\ z_{min} \leq z \leq z_{max} \end{cases} \tag{5.1}$$

The algorithm starts by creating a vertical decomposition consisting of overlapping rectangles on each 2D layer along the $\overrightarrow{Z}$ axis. Subsequently, these rectangles are extended in the $\overrightarrow{Z}$ axis direction to form boxes. Finally, once a box is found, it can be sent to the connected computer or broadcasted to all the modules.

Let $M$ denote the set of modules. A straight sequence of connected modules in one direction is referred to as a line of modules. The initial step of the algorithm involves determining the values $d_m$ for every module $m \in M$. The value $d_m$ represents the rank of the module in the line (from 1 to n), moving 'backward' along the $\overrightarrow{Y}$ axis starting from module $m$, until an empty position is reached. Let $l_m$ be the vertical line in the $\overrightarrow{Y}$ axis direction that contains $m$, $d_m$ can be expressed by:

$$d_m = 1 + |\{n \in l_m \ s.t \ y_n > y_m\}| \tag{5.2}$$

where $|X|$ is the cardinality of $X$.

**Algorithm 5:** Distributed shape recognition - Part 1

**input:** $(x, y, z)$, neighbors

**Initialization :**
  **if** $empty((x, y+1, z))$ **then**
    **send** SET_D_MSG(1) to $neighbor((x, y-1, z))$

**Msg Handler** SET_D_MSG($d_{sent}$)**:**
  $d \leftarrow d_{sent} + 1$
  **if** $\neg empty((x, y-1, z))$ **then**
    **send** SET_D_MSG($d$) to $neighbor((x, y-1, z))$
  **else**
    **if** isRmin **then**
      **if** $\neg empty((x+1, y, z))$ **then**
        **send** FIND_W_MSG($id$, $d$) to $neighbor((x+1, y, z))$
      **else**
        $w = 1$
        Notify front line of $w$
        **if** $\neg empty((x, y, z+1)$ **then**
          **send** FIND_H_MSG($id$, $d$, $w$) to $neighbor((x, y, z+1))$
        **else**
          myBox = $(\{x, y, z\}, \{x+w-1, y+d-1, z+h-1\})$

**Algorithm 6:** Distributed shape recognition - Part 2

```
1  Msg Handler FIND_W_MSG(id, d_sent):
2    if d < d_sent then
3      send SET_W_MSG(id, d_sent, 0) to neighbor((x, y − 1, z))
4    else
5      if empty((x + 1, y, z)) then
6        w ← 1
7        Notify back line of w
8        send SET_W_MSG(id, d_sent, w) to neighbor((x, y − 1, z))
9      else
10       send FIND_W_MSG(id, d_sent) to neighbor((x + 1, y, z))
11 Msg Handler SET_W_MSG(id, d_sent, w_sent):
12   if d_sent ≥ d then
13     w ← w_sent + 1
14     Notify back line of w
15   if myid = id then
16     if ¬empty((x, y, z + 1)) and isCmin) then
17       send FIND_H_MSG(id, d, w) to neighbor((x, y, z + 1))
18     else
19       myBox = ({x, y, z}, {x + w − 1, y + d − 1, z})
20   else
21     send SET_W_MSG(id, d_sent, w_sent + 1) to neighbor((x, y − 1, z))
22 Msg Handler FIND_H_MSG(id, d_sent, w_sent):
23   if w < w_sent ∨ d < d_sent then
24     send SET_H_MSG(id, 0) to neighbor((x, y, z − 1))
25   else
26     if empty((x, y, z + 1)) then
27       h ← 1
28       send SET_H_MSG(id, h) to neighbor((x, y, z − 1))
29     else
30       send FIND_H_MSG(id, d, w) to neighbor((x, y, z + 1))
31 Msg Handler SET_H_MSG(id, h_sent):
32   h ← h_sent + 1
33   if myid = id then
34     myBox = ({x, y, z}, {x + w − 1, y + d − 1, z + h − 1})
35   else
36     send SET_H_MSG(id, h) to neighbor((x, y, z − 1))
```

To compute $d_m$, the algorithm proceeds as follows (see example Figure 5.1a) :

1. Initially, modules without an attached neighbor in the backward direction must set $d$ to 1.

2. Then, they send a message, denoted as `SET_D_MSG(`$d$`)`, to their front neighbor. Upon receiving the message, the module sets its $d$ value as the received value plus one

and, subsequently, forwards the message to its front neighbor.

3. This process continues until an empty position is encountered in the front direction (cf. Algorithm 5 lines 1-7).

A module is considered as an $R_{min}$ if it occupies the foremost left corner of a rectangle. Once $d_m$ is set, module $m$ can determine locally if it is $R_{min}$. To do so, we denote as $d_{left}$ the $d$ value of the module on the left of the module $m$. Then, a module is $R_{min}$ if it verifies the following condition:

$$empty(front) \wedge (d \neq d_{left} \vee \neg empty(left + front)) \tag{5.3}$$

Next, we define $w_m$ for each module $m$ in the set $M$ as the maximum number of connected vertical lines with the same or higher height toward the right (cf. Figure 5.1b).

We can express $w_m$ by the following rule:

$$w_m = \max_{n \text{ connected}} (y_m = y_n \wedge (d_i \geq d_m \forall i \in [m, n])) \tag{5.4}$$

To determine $w_m$ for all modules $m$ in the set $M$, the following distributed process is employed: each module located in a position $R_{min}$ sends a message called FIND_W_MSG$(id, d_{sent})$ to its right neighbor (cf. Algorithm 5 lines 9-10). As the message is forwarded, if the message reaches a module $n$ with $d_n < d_{sent}$, the module $n$ responds to the sender on its left with a message SET_W_MSG$(id, d_{sent}, w_{sent} = 0)$ (cf. Algorithm 6 lines 2-3). Otherwise, when it reaches a module $n$ that lacks a right neighbor, it assigns $w_n$ the value of 1 and responds to the sender on the left using a message denoted SET_W_MSG$(id, d_{sent}, w_{sent} = 1)$ (cf. Algorithm 6 lines 4-8). Upon receiving this message, module $r$ sets its $w_r$ value as the received value plus one if and only if $d_{sent} \geq d_r$ to ensure that the line at the back of module $r$ can accommodate $d_{sent}$ modules (cf. Algorithm 6 lines 12-14).

From each $R_{min}$ module, we use local values of $d_m$ and $w_m$ to define a rectangle $(R_{min}, R_{max}) = (\{x_m, y_m\}, \{x_m + w_m - 1, y_m + d_m - 1\})$. This construction leads to a vertical decomposition characterized by overlapping filled rectangles within each 2D layer along the $\overrightarrow{Z}$ axis, ensuring full coverage of all modules as can be seen in Figure 5.1c.

Once the value of $w_m$ is determined, we can refer to the values of $d$ and $w$ of the module at the bottom (at $z - 1$) of module $m$ as $d_{bottom}$ and $w_{bottom}$, respectively. We also use $R_{min}(bottom)$ to indicate whether the bottom module is at an $R_{min}$ position. Then a module $m$ is at a $C_{min}$ position if it verifies:

$$R_{min} \wedge \neg(R_{min}(bottom) \wedge d = d_{bottom} \wedge w = w_{bottom}) \tag{5.5}$$

The next step consists of determining the height $h$, which represents the maximum num-

ber of rectangles on top of the one associated to $C_{min}$. Consequently, this results in the formation of overlapping boxes. To accomplish this, the module located at $C_{min}$ initiates a message called FIND_H_MSG($id, d_{sent} = d, w_{sent} = w$). This message serves to count the number of top neighbors along the $\overrightarrow{Z}$ axis that meet conditions $d \geq d_{sent}$ and $w \geq w_{sent}$ (cf. Algorithm 6 lines 16-19).

Upon receiving the FIND_H_MSG message by a module $r$, if the values $d_r > d_{sent}$ or $w_r > w_{sent}$, it replies with a message SET_H_MSG(id, h=0) to the sender at the bottom (cf. Algorithm 6 lines 23-25). Otherwise, if the receiver module $r$ lacks a top neighbor, it responds by sending a message labeled SET_H_MSG(id, h=1) to the sender at the bottom (cf. Algorithm 6 lines 26-28). Upon receiving the SET_H_MSG message, the receiver increments the received $h$ value by one and forwards the message to its bottom neighbor until reaching the initiator (cf. Algorithm 6 lines 32-36).

The algorithm operates asynchronously. Therefore, a module can receive a FIND_W_MSG before its $d$ value is defined or a FIND_H_MSG before the $d$ and $w$ values are defined. To solve this, if a module receives a message and the values required for its handling are not yet defined, the module stores the received message in its memory and handles it once the values are set.

When module $m$ at a $C_{min}$ receives its $h_m$ value, it can set its Box as $(\{x_m, y_m, z_m\}, \{x_m + w_m - 1, y_m + d_m - 1, z_m + h_m - 1\})$. The algorithm terminates when all modules at a $C_{min}$ position have determined their boxes.

The termination of the algorithm can be detected by comparing the volume of the union of boxes with the number of modules. A count can be found using a convergecast operation and subsequently broadcast this count to all the modules within the configuration.

Figure 5.2 shows an example of overlapping boxes. It can be seen that the resultant boxes may differ according to the orientation of the coordinate axis. However, in both cases a) and b) the resultant boxes cover all the configuration. Moreover, having an overlapping box results in boxes that are completely inside a bigger one, such as the blue and purple in Figure 5.2 (a) and the yellow in Figure 5.2 (b). When multiple boxes must be stored, they can be aggregated by neglecting the boxes that are inside another one.

## 5.3/ COMPLEXITY ANALYSIS

The complexity in terms of the number of boxes has a lower bound of $\Omega(1)$ in the case of a cubic configuration. As the shape becomes increasingly irregular and incorporates holes, the complexity approaches an upper bound of $O(n)$ where $n$ is the number of modules.

Next, we assess the time and communication complexities of the shape recognition al-

gorithms presented in Section **??**. The number of messages used to find the boxes is proportional to the number of modules. To find the the values of $d$ and $w$ of each module on a line along the $\overrightarrow{X}$ and $\overrightarrow{Y}$ axes, $O(n)$ messages are exchanged, where $n$ is the number of modules. Then, to find the height of each box, the box's corner $C_{min}$ initiates a message that passes to the line of modules along the $\overrightarrow{Z}$. The number of these messages is also bounded by $O(n)$. Therefore, the communication complexity can be expressed as $O(n)$.

As for the time complexity, the search for boxes is done in parallel. Setting the values of $d$ and $w$ requires $O(D + W)$ time, where $D$ and $W$ are the depth and width of the entire configuration. The time complexity to search for the height $h$ of the boxes is $O(H)$, where $H$ is the height of the configuration. Consequently, the overall time complexity can be expressed as $O(D + W + H)$. Thus, it depends on the geometry of the configuration.

## 5.4/ EXPERIMENTS AND ANALYSIS

We implemented the algorithm using *VisibleSim*, a discrete event-based simulator for distributed modular robotic systems that support *BlinkyBlocks*. Figure 5.2c shows a *VisibleSim* capture of the simulated example of the box cover shown in Figure 5.2. We recall from Chapter 3 that *BlinkyBlocks* system is a modular robotic system made up of centimetre-size blocks that are attached to each other via magnets in a square cubic lattice. Each block is a cube of roughly $40\,$mm, with processing, storage, and communication capabilities. Each *BlinkyBlock* communicates through serial links with its directly connected neighbors by sending packets with a payload size of 227 bytes.

The objective of the experiment is to compare the shape recognition algorithm with exhaustive coordinates collection in order for modules to send their current shape to a computer to be used by an interactive CAD software. The coordinates collection method consists of sending the list of coordinates to the root of a breadth-first spanning tree con-

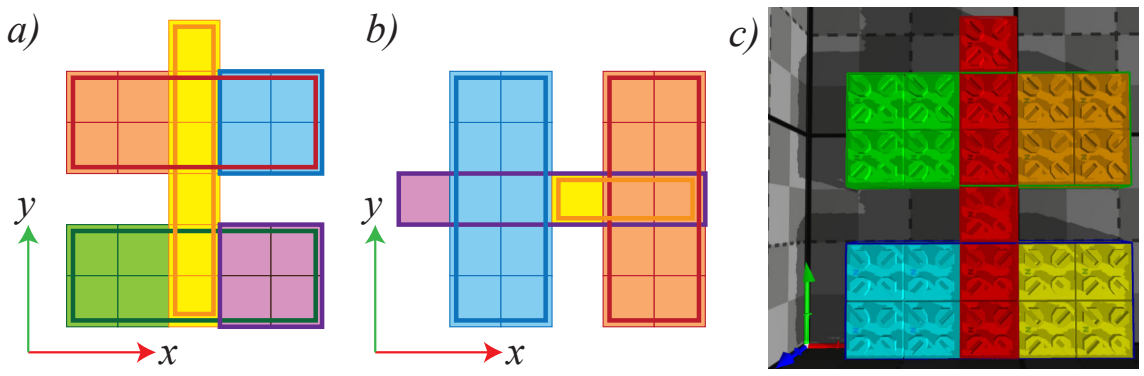



Figure 5.2: Three basic example: a) A basic shape defined by 5 boxes. b) The same shape but rotated producing 4 boxes. c) The same shape computed on *VisibleSim*.

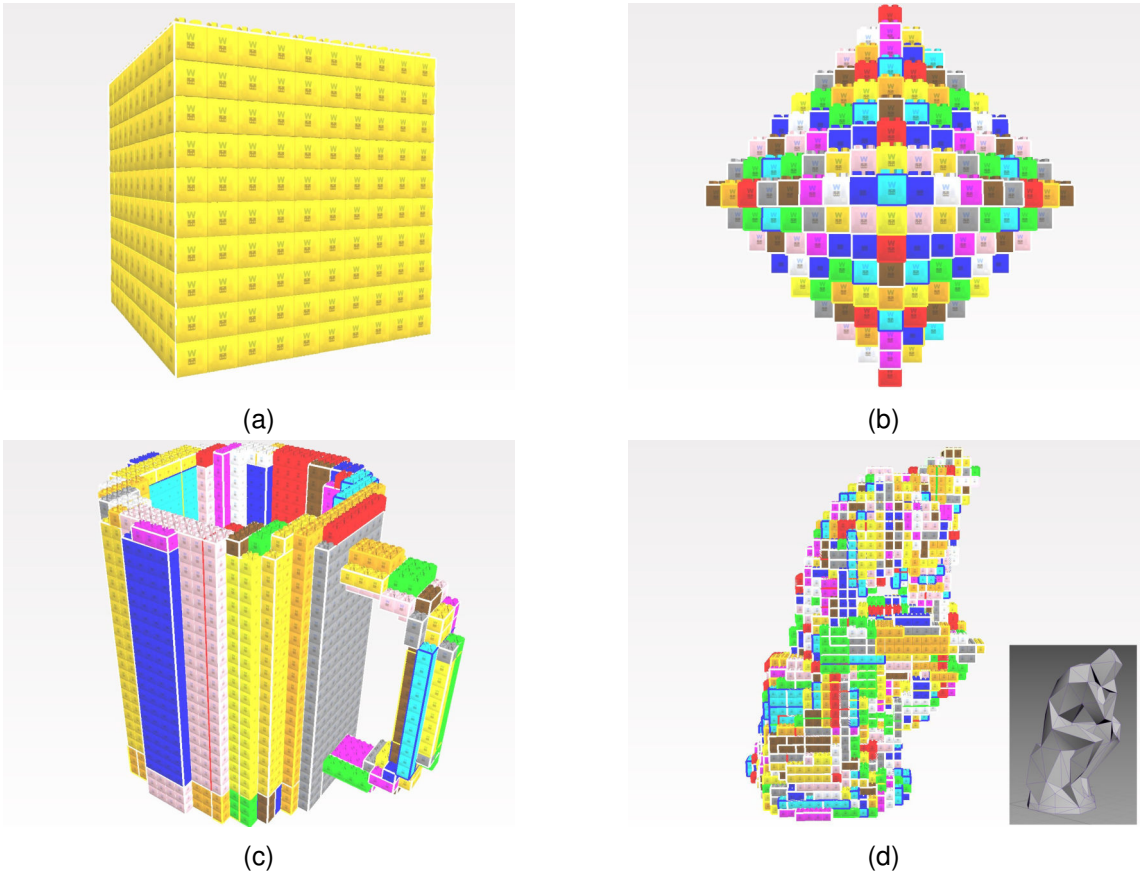


Figure 5.3: Four configurations examples captured from *VisibleSim*. (a) Cube configuration with 1000 *BlinkyBlocks*. (b) Ball configuration with radius 8 (833 *BlinkyBlocks*). (c) Mug configuration with 4019 *BlinkyBlocks*. (d) Thinker configuration with 5814 *BlinkyBlocks*.

nected to the CAD computer. Each module sends three bytes for its coordinates $x$, $y$ and $z$. The leaf modules start by sending their coordinates. The coordinates are merged at intermediate modules before being sent to their parent in the tree when the data are received from all their children modules or the payload is totally used. Using the shape recognition method, once a box is found, the module at $C_{min}$ sends the box information to the root also via a breadth-first tree.

We have done the comparison on four different configurations shown in Fig. 5.3 with different geometries and characteristics:

1. Cube: A simple and regular connected cubic shape. One box is required to cover the whole configuration.

2. Ball: It contains modules whose distance from the center is less than or equal to a given radius Naz et al. (2018a).

3. Mug: A mug shape that exhibits a few irregularities.

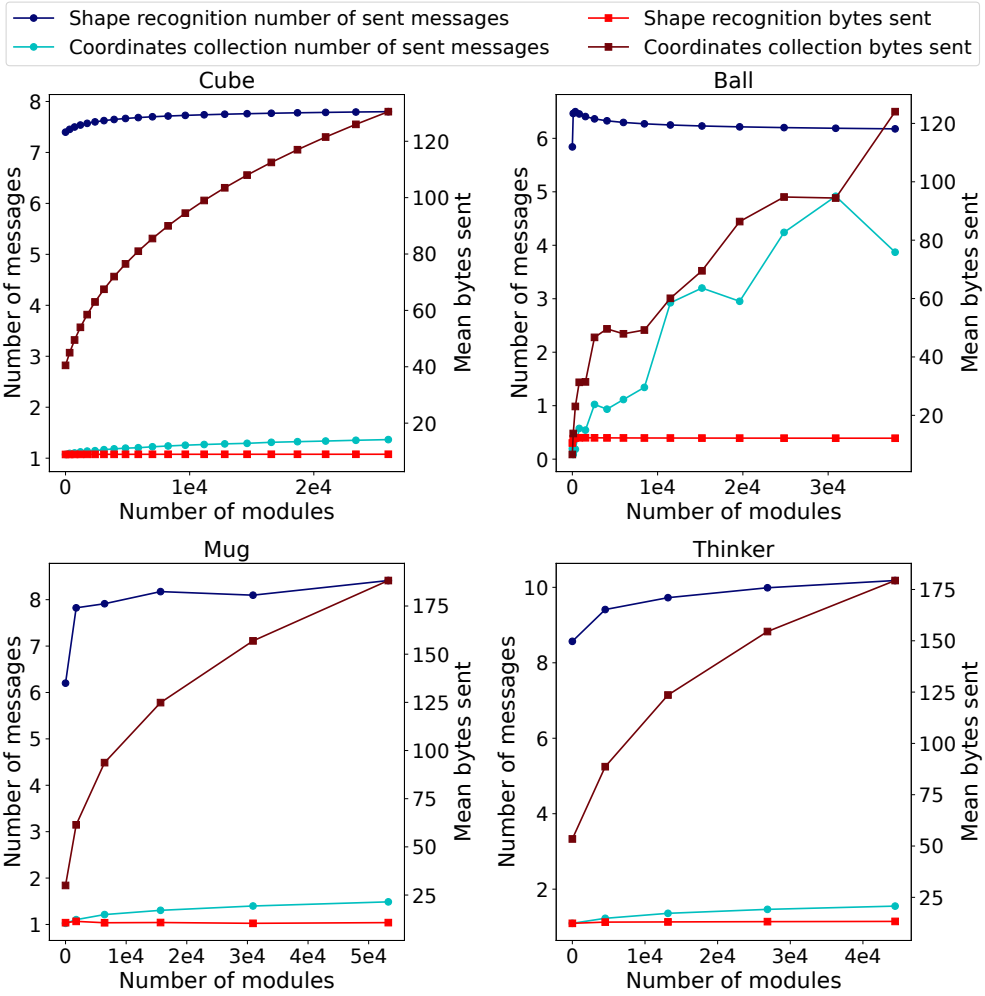



Figure 5.4: Mean number of messages and mean number of bytes sent per module on the four configurations examples.

4. Thinker: The thinker statue defined by a low resolution mesh.

We evaluated the mean number of messages sent per module, the mean number of bytes sent by a module, the time taken to complete the shape recognition, and the ratio between the number of modules and the number of boxes while increasing the sizes of the configurations.

Fig. 5.4 shows the mean number of messages and the mean number of bytes sent by a module on the four configurations. The mean number of messages sent by a module executing the shape recognition method is larger than the coordinates collection on the four configurations. This is due to the communication required to find the dimensions of the boxes. As for the coordinates collection method, each module must send one message to its parent that contains the coordinates of its subtree, but due to the limitation of the packet size, when a packet is filled, it is directly sent to the root, which increases

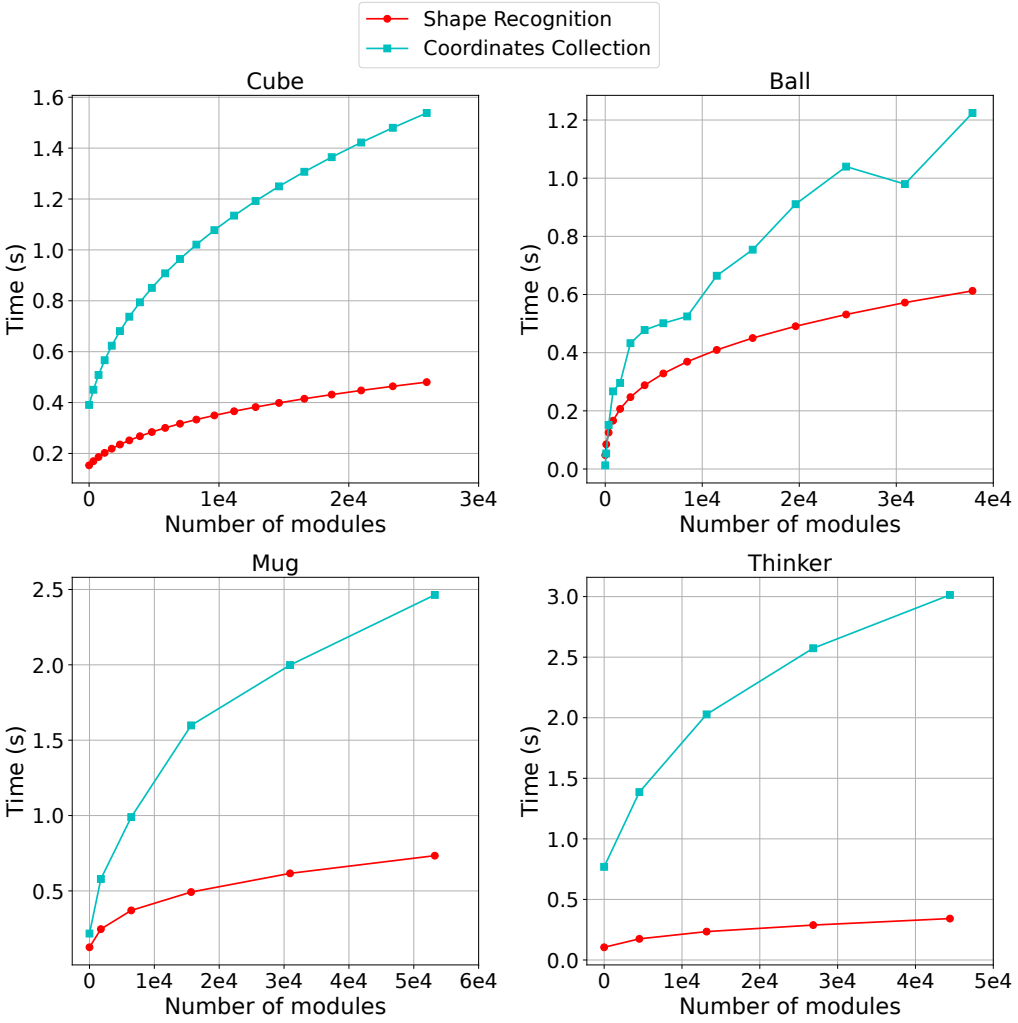



Figure 5.5: Time for receiving the whole current shape by the root.

the number of messages sent by a module. More packets will be full as the size of the configuration increases. Regarding the mean number of bytes sent by a module, the sizes of the messages exchanged by the shape recognition algorithm to find the boxes are limited. The maximum size of a used message is 6 bytes which contain the coordinates of the $C_{min}$ and $C_{max}$ of the box. For the coordinates collection method, the mean message size increases with the size of the configuration. The maximum packet size can go up to its payload capacity.

We conducted an experimental study on *BlinkyBlock* hardware that showed that the time $t$ required per message is affected by the message length $l$ (number of bytes contained in a message) and can be modeled with the linear function: $t = 0.08935 \times l + 1.516$. Therefore, the global executed time is affected by the number of exchanged messages and the length of the messages. Fig. 5.5 shows the time taken by both methods. Although the shape recognition method requires more messages, it can be seen that the shape recognition method is more efficient in time in the four configurations due to the increase in the length

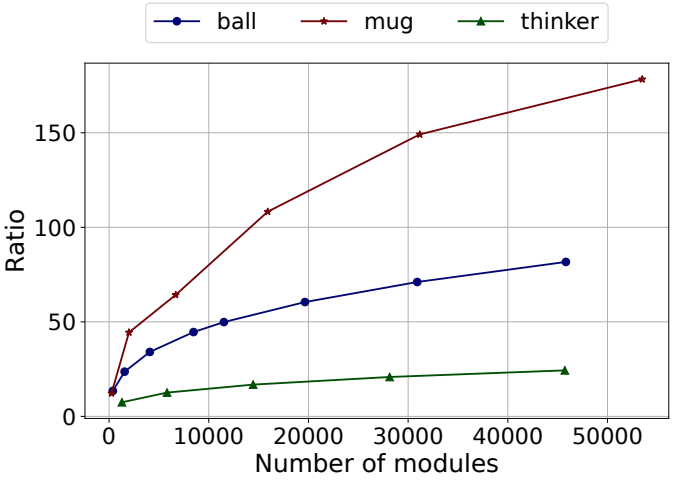



Figure 5.6: Ratio between the number of modules and the number of boxes.

of the messages used in the coordinated collection method as the configuration size increases. The time taken by the shape recognition method depends on the dimensions of the configuration as explained in Section 5.3,

Fig. 5.6 illustrates the ratio between the number of modules and the number of boxes in different configurations. We excluded the graph of the cube shape from the presentation for the sake of visual clarity, since all modules can fit in a single box regardless of the configuration size. As the configuration becomes more irregular, as in the case of the thinker configuration, the ratio decreases. This decrease is a consequence of the need for additional boxes with smaller dimensions to accommodate the irregularities present in the structure. In addition, as the dimensions of the analyzed configurations increase, the ratio decreases due to the rise in irregularities. This is inversely proportional to the regularity of the configuration, which means that more regular configurations will result in a higher ratio. For example, for the cube configuration, we will have a ratio that increases linearly with a slope of one since the ratio will be equal to the number of modules.

## 5.5/ CONCLUSION

In this chapter, we proposed a new shape recognition algorithm that allows modules in a lattice-based modular robot to discover their current shape. The modules search for overlapping boxes to cover the whole configuration. The union of these boxes gives the current shape. We evaluated the algorithm in simulation on *BlinkyBlocks* on different configurations with different geometrical properties and compared it with a coordinates collection method to retrieve the current shape of the ensemble and send it to a central entity. The results show that the shape recognition method outperforms the coordinates collection in time efficiency while using a smaller memory footprint.

# III

# SELF-RECONFIGURATION

# 6

# THE POROUS STRUCTURE

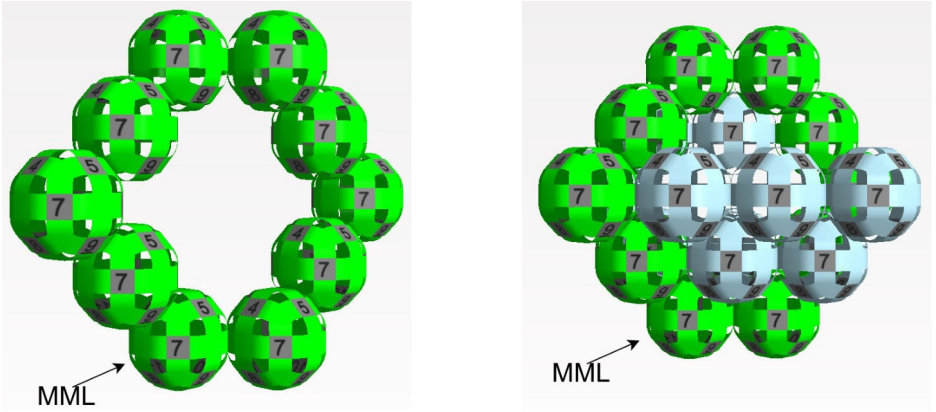
## Contents



## 6.1/ INTRODUCTION

In Chapter 2, the discussion revolved around the difficulty of finding a fast optimal solution for the self-reconfiguration challenge. Two approaches, scaffolding and tunneling, have been used to tackle this issue. Scaffolding entails the construction of the modular robot using hollow substructures or meta-modules, which provide enough empty space for tunneling while simultaneously enhancing holonomy. Tunneling involves the movement of modules through these empty structures to alter the shape of the robot.

In this chapter, we propose a porous scaffolding structure made of *3D Catoms* meta-modules that has enough empty volume to allow tunneling. Usually meta-modules are used to facilitate the movement of modules by pre-planning unitary moves. Thus, with these meta-modules we obtain lattices at the meta-module scale composed of cells that can be empty or full. We propose a three-state model, where each meta-module cell can be absent, present and sparse, or present and filled. Meta-modules can switch from the "FULL" state to the "SPARSE" state by dumping its filling modules into a neighboring cell, and similarly a cell can be emptied by switching from a "SPARSE" state to the state. In the opposite direction, meta-modules can go from the "SPARSE" state to the "FULL" state by receiving modules from a neighboring "SPARSE" or "FULL" cell, and similarly, an empty cell can be filled by switching from an "EMPTY" state to the "SPARSE" state.




(a) Empty 3D meta-module (b) Filled 3D meta-module

Figure 6.1: Meta-module's anatomy. MML refers to a meta-module leader (cf. Section 6.4)

This three-state model provides a new functionality: "FULL" meta-modules can be used as a reserve of material or storage places. Their presence in the grid allows one to reduce the distances covered by the modules during the self-reconfiguration, either by proposing a place to store incoming modules or by proposing outgoing modules close to their delivery place. This new functionality also gives a new property to the structure: the ability to compress (by filling some meta-modules) or expand (by emptying some meta-modules) which drops the self-reconfiguration constraint to have the number of meta-modules in the initial structure to be equal to the one in the goal structure.

The structure anatomy is presented in Section 6.2. Then, the basic operations that move the modules from one meta-module to another in all directions to change their states are presented in Section 6.3. A motion coordination algorithm that will be executed by the modules that execute an operation is presented in Section 6.4. Finally, a conclusion is provided in Section 6.5.

## 6.2/ POROUS STRUCTURE ANATOMY

Our proposed meta-module can be in two states: "FULL" or "SPARSE". A "SPARSE" meta-module is made up of ten *3D Catoms* assembled in a 3D hexagonal shape in an FCC lattice as in Figure 6.1. The positions of the modules in a meta-module are given in Figure 6.3. They can be vertically flipped according to their position in the meta-module-scale lattice. The size of the meta-module has an impact on the granularity of the system, since the description of the shape will be done at the meta-module level. The size of the modules $10$ is chosen so that the meta-module can store the size of another one and maintain enough space between it and its neighbor meta-modules to allow the modules to flow between them without blocking. Furthermore, $10$ is the smallest size that allows

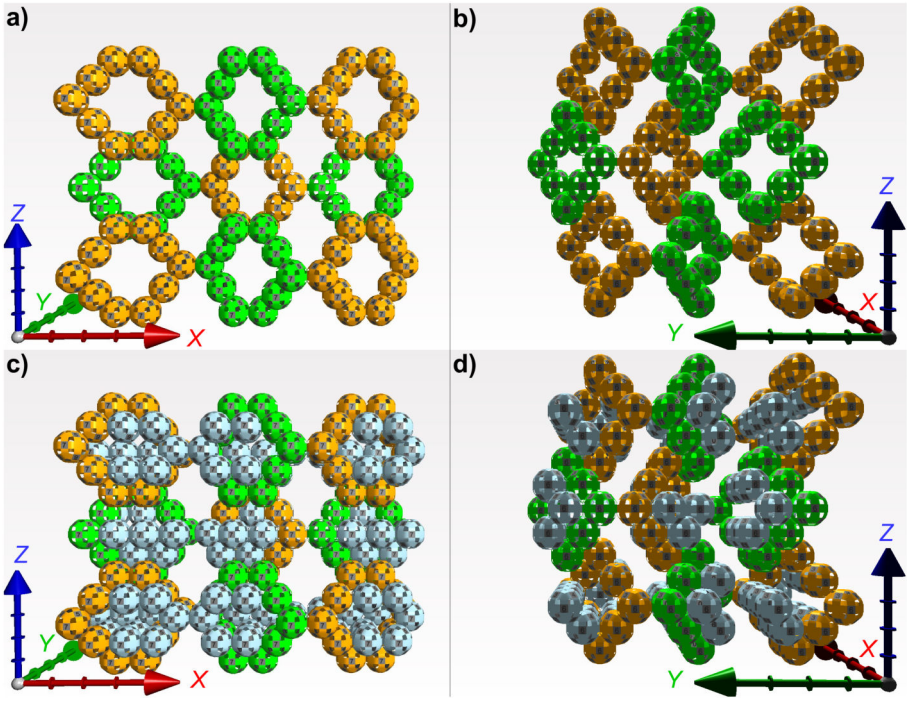



Figure 6.2: Meta-modules structure in a 3D cubic lattice. a) "SPARSE" meta-modules in the $XZ$ plane. b) "SPARSE" meta-modules in the $YZ$ plane. c) "FULL" meta-modules in the $XZ$ plane. d) "FULL" meta-modules in the $YZ$ plane.

modules to flow through the empty internal volume of a "SPARSE" meta-module without being blocked by modules that form the "SPARSE" meta-module through which they traverse. Each "SPARSE" meta-module can fit ten additional modules into its empty internal volume to form a "FULL" meta-module. Therefore, a "FULL" groups 20 *3D Catoms*, twice the size of a "SPARSE" meta-module.

Filling a meta-module allows the structure to compress or expand by a factor of $2$ since each "SPARSE" meta-module can store in its empty volume the size of another meta-module and the "FULL" meta-module can expand by discarding its filling modules so they can be reassembled into a "SPARSE" meta-module at an "EMPTY" position. Let $N$ be the total number of modules in the initial shape and $S_G$ the size of the goal shape in terms of meta-module ("FULL" or "SPARSE"). Due to the expandability and compressibility properties of the proposed structure: $\lceil \frac{N}{20} \rceil \leq S_G \leq \frac{N}{10}$.

The positions of the filling modules are carefully chosen to avoid blockage. They must be free to be discarded in all directions. For this reason, their positions differ according to the position of their meta-module in the meta-module scale lattice. Table 6.1 shows the assortments of positions for each meta-module position.

The meta-modules are arranged in a 3D regular cubic lattice as shown in Figure 6.2. A cell in the grid can be "EMPTY" or present. A present cell contains a meta-module that can be "SPARSE" or "FULL". Each meta-module is attached to an adjacent one with at least

Table 6.1: Positions of filling modules relative to the bottom left one (MML). X, Y and Z are the coordinates of meta-module in the square cubic lattice.

| $(X+Y)\%2=0 \wedge$ $Z\%2=0$ | $(X+Y)\%2=1 \wedge$ $Z\%2=0$ | $(X+Y)\%2=0 \wedge$ $Z\%2=1$ | $(X+Y)\%2=1 \wedge$ $Z\%2=1$ |
|---|---|---|---|
| (0,-1,1) | (-1,-1,1) | (0,1,2) | (1,1,2) |
| (1,-1,1) | (0,-1,1) | (1,1,2) | (0,1,2) |
| (0,-1,2) | (0,-1,2) | (-1,1,2) | (2,0,2) |
| (1,-1,2) | (1,-1,2) | (-1,0,1) | (1,0,1) |
| (2,-1,2) | (-1,0,2) | (0,0,1) | (0,0,1) |
| (0,0,1) | (0,0,1) | (0,-1,1) | (0,-1,1) |
| (-1,0,1) | (1,0,1) | (1,-1,1) | (-1,-1,1) |
| (-1,0,2) | (2,0,2) | (2,0,2) | (-1,0,2) |
| (-1,0,3) | (1,0,3) | (1,-1,3) | (-1,-1,3) |
| (0,0,3) | (0,0,3) | (0,-1,3) | (0,-1,3) |

one module. The left, right, front, and back adjacent meta-modules in the $XY$ plane have the positions of their modules flipped vertically if $(X+Y)\%2=1$ and the top and bottom adjacent meta-modules in the $XZ$ plane are attached to the front or the back of the two top or the two bottom modules according to the $\overrightarrow{Z}$ axis to preserve structure symmetry.

The structure is designed to make module transportation easier. An "EMPTY" cell and a "SPARSE" meta-module with a complete neighborhood are never blocked, which means that they can freely receive or release modules from any direction. In a "SPARSE" meta-module, at least one module is always free to move, allowing the meta-module to be disassembled and its modules to be transported in any direction. Furthermore, an "EMPTY" cell can receive modules from any direction and connect to its neighboring meta-modules without obstruction. This allows the lattice structure to be assembled and disassembled with ease without the need to rearrange other modules or cells. This feature allows the

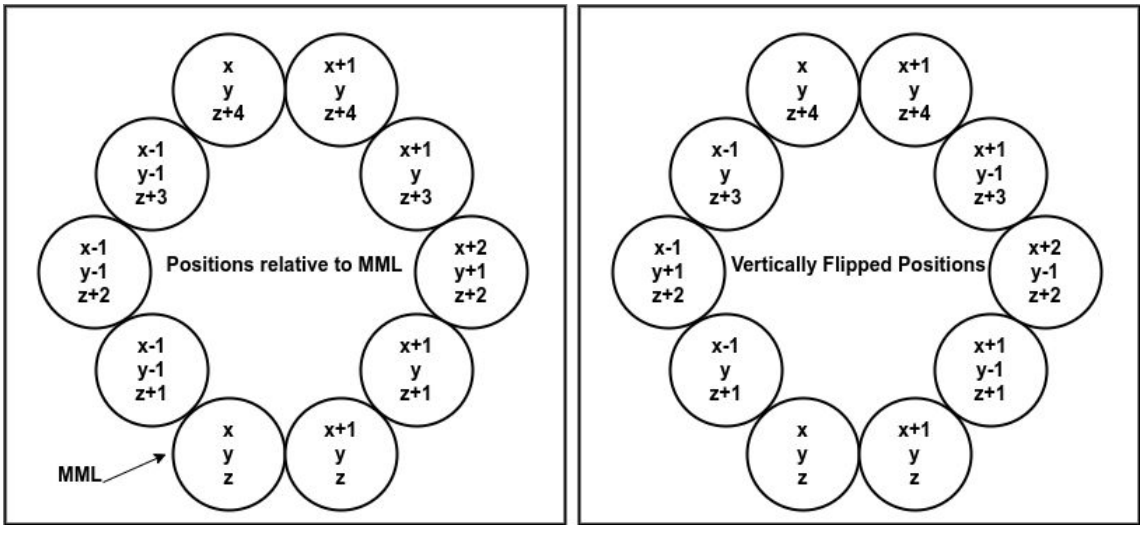



Figure 6.3: Modules positions of a "SPARSE" meta-module in the FCC lattice.

bridging constraint, which prevents the insertion of a module in an empty position between two modules facing opposite directions, to be relaxed at the level of meta-modules. This allows for greater flexibility in the placement and removal of meta-modules within the lattice structure. Therefore, it also facilitates self-reconfiguration planning.

## 6.3/ MOTION OPERATIONS

To change the shape of the entire structure, we define three basic operations that can be executed by a meta-module to change the state of its cell:

1. *Dismantle* operation changes the state of an occupied cell from "SPARSE" to "EMPTY" or "FULL" to "SPARSE". It breaks or empty the meta-module and transports its modules to an adjacent cell.
2. *Transfer* operation does not change the state of a cell. It is only used to transport modules through a "SPARSE" cell.
3. *Assemble* operation changes the state of a cell from "EMPTY" to "SPARSE" or "SPARSE" to "FULL".

Each operation can be executed in the six directions (left, right, up, down, back, and front) in the cubic meta-module scale lattice. These operations can be exploited by a self-reconfiguration planner whose purpose will be to specify which operation to execute on which meta-module. An operation is defined as a sequence of hand-coded movements to navigate the modules of a meta-module from one position to another. Each movement is coded by a triplet in the form of <*current_position, next_position, state*> where the three possible values of *state* are:

1. MOVING: to indicate that the module must continue to move when *next_position* is reached.
2. WAITING: to indicate that the module must stop and wait when *next_position* is reached to serve as a bridge for the next flowing modules or to wait when filling a meta-module before entering its final position to avoid blocking.
3. IN_POSITION: to indicate that the module will reach the final position for the current operation.

All moving operations can be applied by a *3D Catom* to perform a sequence of basic movements. All the sequences must be pre-stored in the robot memory for the six possible directions of motion. However, some operations can be deduced from others. For

Table 6.2: Number of movements per operation

| Operation | Direction | Nb of Movements |
|---|---|---|
| *Assemble* ("SPARSE") | Up/Down | 41 |
| *Assemble* ("SPARSE") | Right/Left | 59 |
| *Assemble* ("SPARSE") | Back/Front | 64 |
| *Assemble* ("FULL") | Up/Down | 72 |
| *Assemble* ("FULL") | Right/Left | 78 |
| *Assemble* ("FULL") | Back/Front | 91 |
| *Transfer* | Up/Down | 50 |
| *Transfer* | Right/Left | 60 |
| *Transfer* | Back/Front | 44 |
| | | Total: 559 |

example, a module needs only to store the movements of the *Transfer* operations in 3 directions, e.g. up, left, and back. The movements for the other directions can be deduced from the stored ones by executing them in reverse order. In addition, the operations *Assemble* and *Dismantle* are homologous, the movements required to dismantle a meta-module are the same as those required to build it, but in reverse order. Hence, reducing the number of operations to be stored in each module.

Table 6.2 shows the number of movements for *Assemble* operations into "SPARSE" or "FULL" meta-module and *Transfer* operations that must be stored in a module. Other operations can be deduced from the stored one. Each movement is stored in the database using 4 bytes to embed the $6 \times 4bits$ used for coordinates plus $2bits$ for the state. As presented in Table 6.2, there are 559 records in total stored in the movements database requiring $2.24$ kB of predefined movements data. It is important to note that despite the fact that this memory is quite large in the context of modular robots, it is a constant size.

The video[1] presents in the first part the various possible operations. First, it shows the *Dismantle* operation on a "SPARSE" meta-module whose modules are transferred and reassembled in an "EMPTY" cell that becomes "SPARSE". Second, the "SPARSE" meta-module is dismantled and transferred to be reassembled in a "SPARSE" cell that becomes "FULL". Third, it shows the *Dismantle* on a "FULL" meta-module whose filling modules are reassembled on an "EMPTY" cell that becomes "SPARSE".

## 6.4/ MODULES MOTION COORDINATION

To execute the operations, a module can take three different roles:

- Meta-Module Leader (MML) is a module chosen in each meta-module whose purpose is to handle computation and communications between meta-modules. Mes-

[1] Youtube video: https://youtu.be/6dtkXBY8t6k

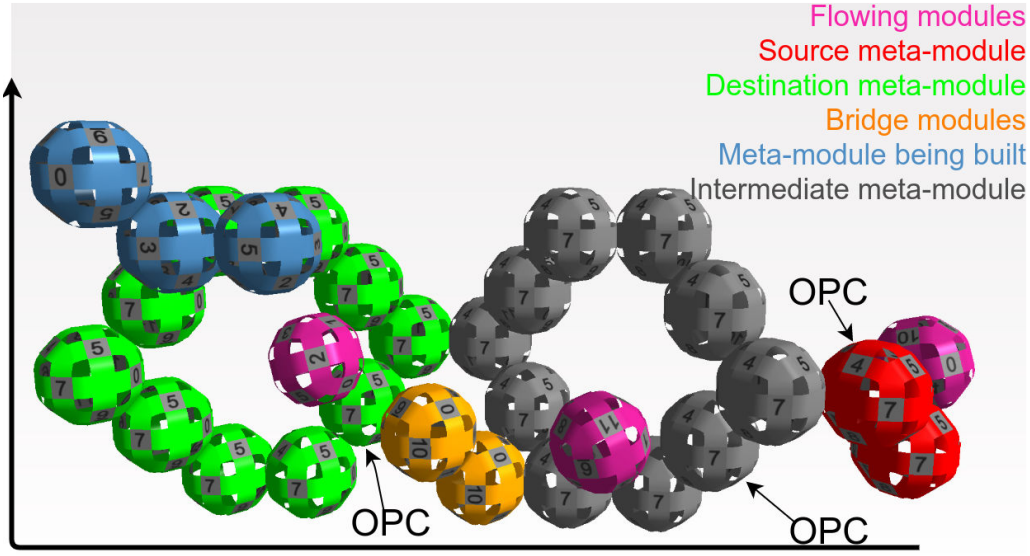



Figure 6.4: Simulation snapshot during modules transportation (best viewed in color). The source at the right is dismantled to be built back at the top of the destination on the left.

sages between meta-modules are sent from one MML to another. The MML can be any of the ten modules that form a meta-module. We chose the one on the bottom left as shown in Figure 6.1.

- Operation Coordinator (OPC) is a module that coordinates operations at the meta-module level by choosing the sequence of movements to execute by a flowing module. The OPC is the first module to which a moving module from a previous operation is connected. Or, in the case of dismantle operations, it is the last module connected to the meta-module in the operation direction.

- Flowing Module (FM) is a module in motion executing an operation's motion sequence.

Figure 6.4 shows modules transportation from a source meta-module on the right to a destination meta-module on the left. The modules of the source (in red) are executing a *Dismantle* operation in the left direction. When they are traversing the intermediary meta-module (in grey) they execute a *Transfer* operation in the left direction. When they reach the destination meta-module they start executing the *Assemble* operation in the up direction. In this figure, the orange modules are in a waiting state that serves as a bridge for the purple moving modules to flow without blocking. Once all the modules pass the bridge, the OPC will inform the waiting bridging modules to continue the execution of their current operation.

To avoid blocking and collisions during the flow of multiple modules, the modules flow in

one line following the same path. A message-passing traffic-light style motion coordination protocol described in Thalamy et al. (2020b) is used to maintain a space gap between every two moving modules. Furthermore, the structure maintains enough space between meta-modules to allow the modules to flow in parallel in multiple adjacent streamlines without collisions.

**Algorithm 7:** Distributed control algorithm for a FM

**Data:** $Operation$: The operation in execution.
**Data:** $mvt\_it = 0$: Iterator on Operation's movements.
1 **Msg Handler** COORDINATE_MSG($Op$, $it$)**:**
2 $Operation \leftarrow Op$ ;
3 $mvt\_it \leftarrow it$ ;
4 rotateTo($Operation[mvt\_it].nextPosition$) ;
5 **Event** ROTATION_END**:**
6 **if** $mvt\_it = Operation.size \wedge Operation.isAssemble$ **then**
7 meta-module reached goal position;
8 **else**
9 **if** $Operation.state = MOVING$ **then**
10 $mvt\_it \leftarrow mvt\_it + 1$ ;
11 rotateTo($Operation[mvt\_it].nextPosition$) ;
12 **else**
13 **if** $Operation.isDismantle \wedge Operation.state = IN\_POSITION$ **then**
14 **send** POSITION_REACHED() to OPC ;
15 **Event** REMOVE_NEIGHBOR**:**
16 **if** Operation.state = WAITING **then**
// Bridge
17 **if** all modules have passed **then**
18 $mvt\_it \leftarrow mvt\_it + 1$ ;
19 rotateTo($Operation[mvt\_it].nextPosition$);

Each moving module performing an operation keeps an iterator on the sequence of movements of the operation being executed. The algorithm executed by a FMs is described in Algorithm 7. When a FM module is attached to OPC, it means that it ended the previous operation and is ready to start executing the sequence of movements of the next operation. So, the OPC sends the `COORDINATE_MSG` containing the operation to be executed by the FM and the value of the iterator so that the FM knows from which movement it must begin. On reception, FM will start to move until it reaches the state IN_POSITION, which means that it ends the movements to be executed for the current operation, or WAITING which means that the module must stop and serve as a bridge for the next modules to pass it (Algorithm lines 6-14). If the operation is a *Dismantle* operation, when an FM becomes IN_POSITION, it notifies its operation coordinator so it can proceed to the next module. Note that the function $rotateTo$ executed by an FM encompasses the motion coordination algorithm that requires additional exchanged messages between the moving

module, its pivot, and its future latching points. Briefly, the light state of a module is red if it serves as a pivot for a FM module motion. Otherwise, it is green. Before each motion, a FM probe the light state of the modules at its next latching points to verify if they are all green before moving. Otherwise, it must wait until they become green. The reader can refer to Thalamy et al. (2020b) for a detailed description of the motion coordination algorithm.

## 6.5/ CONCLUSION

In this chapter, a porous structure made up of hexagonal meta-modules placed in a 3D cubic lattice is proposed and the basic operations to perform to change the state of each cell are presented. The structure allows the storage of excess modules in "SPARSE" meta-modules, allowing it to compress and expand, so we are not constrained during self-reconfiguration to have the initial size of the shape equal to the one of the goal shape. Operations can be exploited by a self-reconfiguration planner whose purpose is to specify which operation to execute on which meta-module in the initial configuration to achieve a given goal configuration. In the next two chapters, we provide two self-reconfiguration planning algorithms. The first in Chapter7 is a fully distributed round-based algorithm. The second in Chapter8 performs the planning in a centralized manner, then, the modules flow asynchronously in a single round to form the goal shape.

# 7

# RePoSt: A Fully Distributed Synchronous Algorithm

## Contents



## 7.1/ Introduction

The previous chapter described a porous structure composed of meta-modules and shows how meta-modules can perform the operations: *Assemble*, *Dismantle* and *Transfer* in all directions to change the state of a cell of the 3D cubic lattice between "Empty", "Sparse" and "Full".

In this chapter, *RePoSt*: A fully distributed algorithm is proposed to plan the reconfiguration of the structure by specifying the operations to be executed by the meta-modules to reach the goal configuration. It operates in rounds where in each round a set of disjoint paths is found using a distributed maximum flow algorithm that connects source meta-modules that must be dismantled to destination meta-modules that must handle the assembly at an cell.

The chapter is organized as follows. In Section 7.2 the algorithm is described. Section 8.3 analyzes the time complexity expressed as the number of motions. Then in Section 7.4

a simulated 3D self-reconfiguration example is presented and analyzed. Finally, a brief conclusion is given in Section 7.5.

## 7.2/ ALGORITHM DESCRIPTION

In this section, *RePoSt*: a synchronous self-reconfiguration algorithm is described to transform a structure composed of the meta-modules described in Chapter 6 from its current shape to a given goal shape.

The *RePoSt* algorithm runs on all modules of the system. This algorithm is mainly divided into three steps repeated sequentially until the convergence to the goal shape:

1. Determining sources and destinations meta-modules.
2. Find the maximum number of possible streamlines connecting sources and destinations.
3. Dismantling sources meta-modules and transporting their composing modules to destinations.

The steps are detailed in the following subsections.

During the reconfiguration process, a Global Coordinator (GC) module coordinates the sub-stepping scheme. It is a fixed module that initially belongs to a meta-module in the goal shape, so it will not change position during the reconfiguration process. At each iteration, the GC initiates each step, detects its termination, and then initiates the next

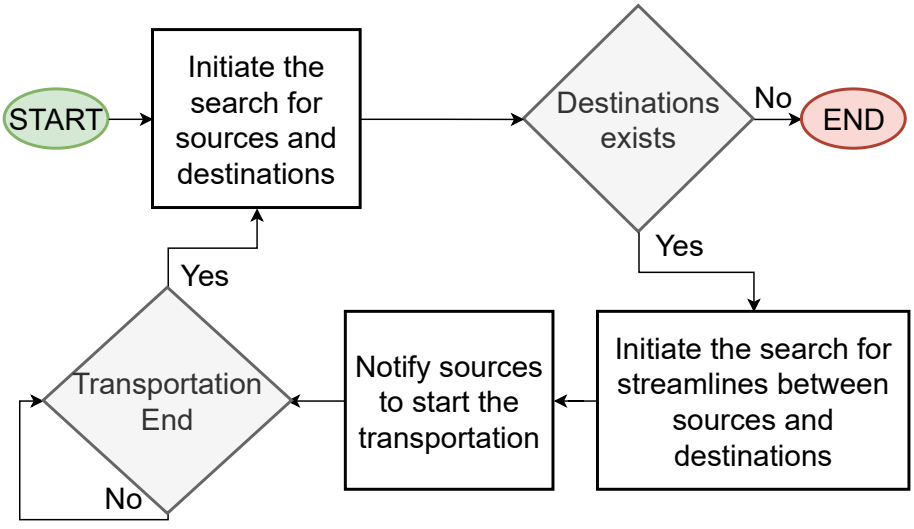



Figure 7.1: GC agent's flow chart

step until the goal shape is achieved, as shown in Figure 7.1. The termination of the reconfiguration process is determined if no destinations are found after the first step.

We assume that each module knows, in addition to its coordinates in the Face-Centered Cubic (FCC) lattice, the coordinates of its meta-module in the square cubic lattice. This can be efficiently disseminated by the GC whose coordinates are the origin. On reception, an MML sets the coordinates of its meta-module according to the direction of the sender. Moreover, all meta-modules know the goal shape and can determine if they are in it or not. This can be done efficiently using the method described in Tucci et al. (2017) applied at the meta-module scale where a module is replaced by a meta-module and the computations are performed by the MMLs. Moreover, we assume that the modules have no prior knowledge of their initial configuration and its size.

### 7.2.1/ Determination of Sources and Destinations

To reconfigure the system from its current shape to a given goal shape, we must first determine the sources and destinations. Sources are any meta-modules that are "Full" or do not belong to the goal shape. A "Sparse" source can be dismantled and a "Full" source can discard its filling modules. Modules from a source are transported to a destination meta-module. The destinations are "Sparse" meta-modules adjacent to an empty cell belonging to the goal shape that needs to be occupied or a "Sparse" meta-module to be filled. When a source module arrives at a destination, the destination handles its transportation to its meta-module goal position. We assume that each meta-module must know the shape of the goal and can locally determine whether it belongs to it or not.

#### 7.2.1.1/ Sources Determination

A potential source is defined as a "Sparse" meta-module in the initial shape that does not belong to the goal shape or a "Full" meta-module at any position. Initially, all "Sparse" meta-modules that do not belong to the goal shape are potential sources. Then, a potential source is confirmed to be a source if it does not disconnect the structure after it has been dismantled. We use the connectivity preservation method described in Lengiewicz and Hołobut (2019) to choose meta-modules which, when dismantled, do not disconnect the structure. Briefly, the connectivity preservation method consists of building a tree rooted at the GC in a way that the leaves meta-modules, when removed, do not disconnect the structure. To build the tree, each meta-module $M_i$ store a value $s_i$ that indicates the minimum number of potential sources that must be traversed to reach the module $M_i$ starting from the GC. Each meta-module in the tree chooses as the parent the neighbor that has the minimum value of $s$. Therefore, the leaves of the tree, if they are potential sources, are confirmed to be sources, since they do not disconnect the structure if

removed.

#### 7.2.1.2/ DESTINATIONS DETERMINATION

Potential destinations are meta-modules adjacent to empty positions in the goal shape or "SPARSE" meta-modules to be filled if the initial shape is larger than the goal shape. A source meta-module will be dismantled and transported to a destination meta-module that coordinates the process of reassembling the source.

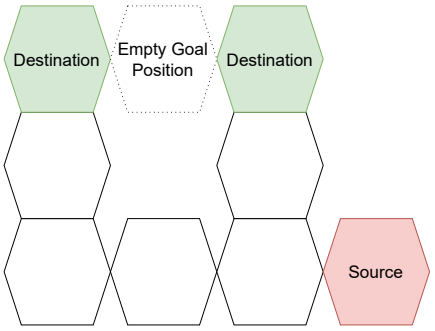



Figure 7.2: Two potential destinations for one empty goal position.

A problem that can occur is having an empty goal position adjacent to multiple potential meta-modules destinations, as seen in Figure 7.2. When a potential destination is determined, to avoid collision, it must be a destination for only one empty goal position. One solution is to report back to the GC which empty goal positions a destination corresponds to. In its turn, the GC chooses an empty goal position for each potential destination and notifies each destination about its associated empty goal position. This issue can also be solved if the modules know their current configuration, so they know about the existence of another meta-module at a cell next to the empty goal position. They can break the symmetry based on simple coordinate rules. The algorithm proposed in Chapter 5 can be used to make the modules recognize their current reconfiguration.

### 7.2.2/ FINDING STREAMLINES

After the sources and destinations are determined, the maximum number of streamlines connecting the sources and destinations is found. A streamline is defined as a path of adjacent meta-modules that starts from a source and ends at a destination. Streamlines must be disjoint to avoid collisions at intersections. This can be achieved by solving the classical problem of maximum-flow in graphs between many sources to many destinations with a unit edge capacity. A distributed asynchronous version of the Edmondkarp max-flow algorithm Edmonds and Karp (1972a) proposed in Lengiewicz and Hołobut (2019) is used and adapted for this purpose. Each source initiates a breadth-first search

for destinations. When a destination is reached, a unique path is backtracked, and the unused branches of the tree are cut off leaving a place for other trees to grow. Then, the source confirms the path by sending a message along it to confirm the streamline. The algorithm terminates when no additional streamlines can be found. The GC needs to detect the termination of this step before proceeding to modules transportation. We use the distributed termination detection method described in Brzezinski et al. (1993).

### 7.2.3/ MODULES TRANSPORTATION

After the establishment of disjoint streamlines connecting sources and destinations, the sources must be dismantled and transported along the streamlines to the destinations. This is done by executing the meta-modules operations described in Section 6.3.

Each MML in a streamline knows the position of the previous and next meta-module in the same streamline. This information is used to determine the direction of the operation to be executed on each meta-module. The *Dismantle* operation is executed on source meta-modules, *Transfer* operation is executed on intermediate meta-modules to transfer modules to the next meta-module in the streamline, and the *Assemble* operation is executed at destinations.

## 7.3/ COMPLEXITY

As presented in Section 7.2, the algorithm repeats $M$ rounds, which consists of transporting modules along the streamlines. The value of $M$ varies enormously in function of the initial, intermediate, and final shapes of the self-reconfiguration. It is mainly affected by the number of streamlines that can be established at each iteration.

We consider that the duration of a basic motion can be mainly determined by time $t_m$ and that the duration of the whole self-reconfiguration is mainly due to the number of motions, which are much longer than the communication times.

The longest stage is the motion of modules along the longest streamline of a round. The length of such a streamline can be majored by the diameter of the configuration divided by the diameter of a meta-module, we call $d$ this dimension. Considering that the number of movements given from the stored *Dismantle* and *Assemble* operations can be majored by $N_0$ and that the number of movements applied for a *Transfer* operation can be majored by $N_1$, we can express the number of basic motions performed to cross a streamline of length $x$ by: $N_{motions} = 2 \times N_0 + (x-1) \times N_1$.

The longest streamline being $d$ long, we can majorize $x$ by $d$, then we have: $N_{motions} < 2 \times N_0 + (d-1) \times N_1$. Therefore, the time complexity can be expressed as $O(M\ N_{motions}\ t_m)$.

## 7.4/ EXPERIMENTS

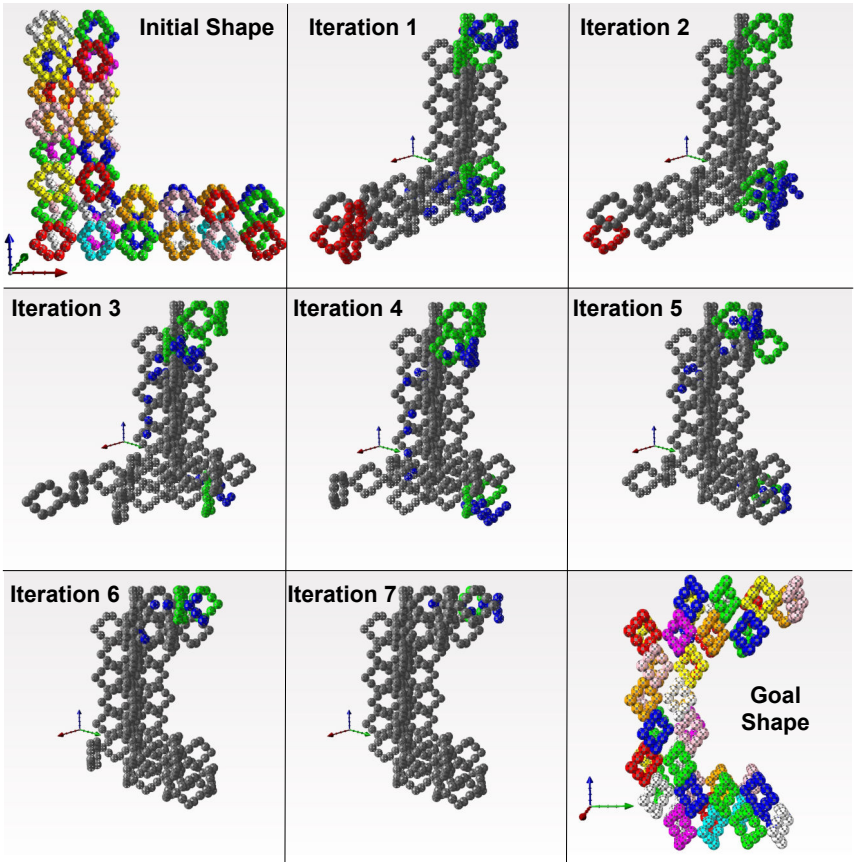



Figure 7.3: Simulation snapshots for the 7 iterations during the reconfiguration of 48 meta-modules in an L shape to a C shape.

In this section, the functioning of our method and the different operations performed in parallel by the meta-modules along the streamlines is shown. Moreover, the simulations show that our algorithm is capable of using the predefined operations to follow the streamlines in order to reconfigure a structure from an initial shape to a goal one in a 3D space.

All simulations are performed in *VisibleSim*. Figure 7.3 shows snapshots of the simulation during the reconfiguration of an L shape made of 48 meta-modules placed in the XZ plane to a C shape in the YZ plane.

The video[1] shows in its second part a simulation of the reconfiguration of 270 "SPARSE" meta-modules placed in a 3 layer square shape into a humanoid shape of size 267 meta-modules. The additional 3 meta-modules in the initial shape are filled inside the structure during the last iterations.

In its last part, the video shows the expansion of a $6 \times 6 \times 3$ configuration with "FULL" meta-modules at the bottom layer into a $6 \times 6 \times 4$ configuration. All "FULL" meta-modules

[1] Youtube video: https://youtu.be/9EIDp7Wv5iw

are emptied and their filling modules are transported in parallel to form an additional layer of ”SPARSE” meta-modules at the top of the configuration in one iteration.

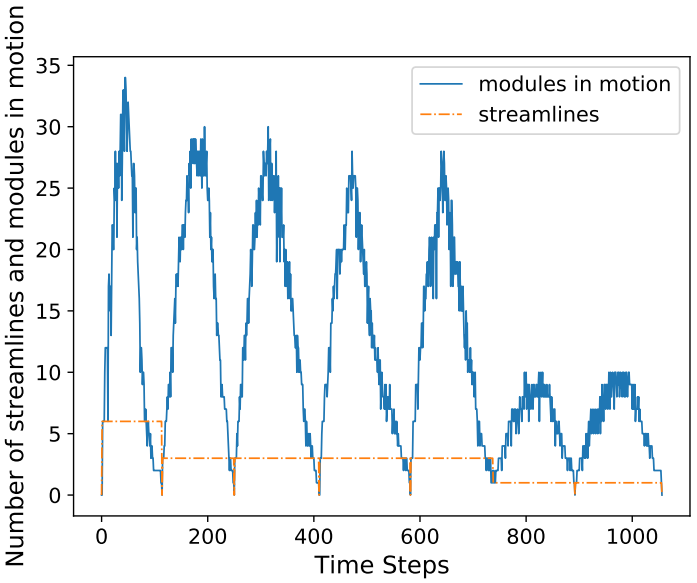
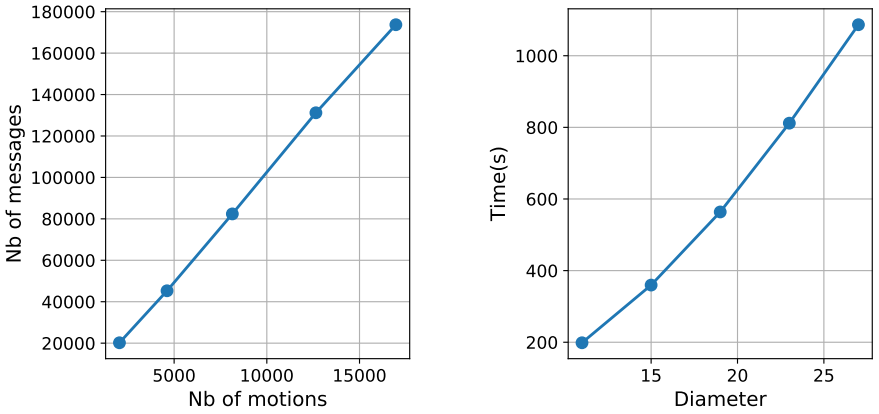



Figure 7.4: Motion parallelism and number of streamlines during the reconfiguration of 48 meta-modules from a L shape to a C shape.

Figure 7.4 shows the number of streamlines and the number of modules that move concurrently along the streamlines against time steps during the reconfiguration example in Figure 7.3. One time step corresponds to the time it takes a module to move from one position in the grid to an adjacent one. Each bell curve in the graph corresponds to an iteration. The number of streamlines and motions becomes null between two peaks, which corresponds to the time required for the first two steps of the algorithm: finding sources and destinations and determining the streamlines. It can be seen that it is negligible compared to the time required to transport the modules. The maximum number of concurrent motions corresponds to the size of the meta-module times the number of streamlines, meaning that all the modules of the dismantled source are moving at the same time. It reaches maximum in iterations 2 to 7. At iteration 1 it is less than the maximum because some modules have reached their goal position, while others have not yet started their movements.

Figure 7.5 evaluates the number of communications and the time for the reconfiguration of an L shape to a C shape while varying the size of the configuration. Figure 7.5a shows that the number of messages exchanged is proportional to the number of movements that increases as the size of the configuration increases. Figure 7.5b shows that the execution time increases linearly with the diameter of the system which is mainly due to the parallel motion in multiple streamlines.




(a) Exchanged Messages (b) Simulation time

Figure 7.5: Number of messages exchanged versus number of motions and simulation time versus the diameter of the system for the reconfiguration of an L shape to a C shape while varying the configuration sizes in $\{20, 28, 36, 44, 52\}$ meta-modules

## 7.5/ CONCLUSION

This chapter presented *RePoSt* a fully distributed algorithm for porous structures to reconfigure an initial configuration into a goal one by dismantling meta-modules, transferring their composing modules along disjoint streamlines and re-building them in empty positions in the goal shape. Examples of reconfiguration in simulation were shown and the motion parallelism was studied. A communication and time analysis was provided that showed that the number of exchanged communications is linear in the number of motions and the time is linear in the diameter of the ensemble.

# 8

# ASAPs: A Hybrid Asynchronous Algorithm

## Contents



## 8.1/ Introduction

In previous chapters, we introduced a porous structure made up of meta-modules that can execute predefined motions operations to change the state of lattice cells. We also introduced *RePoSt*, a fully distributed self-reconfiguration algorithm that operates in rounds to specify the operations for the meta-modules to execute to reach a goal configuration. This chapter presents a novel hybrid centralized/distributed self-reconfiguration planning algorithm called *ASAPs*.

*ASAPs* divides the self-reconfiguration process into two distinct stages. In the first stage, a centralized planner with knowledge of the current and goal configuration calculates all the flowing paths. These paths determine the operations to be executed on each meta-module. By computing the flowing paths a priori, *ASAPs* ensures efficient paths for the modules to follow at the second stage to reach the goal configuration.

In the second stage, the modules follow the precalculated flowing paths under the control of a distributed asynchronous algorithm. The distributed control algorithm enables modules to autonomously execute the operations while cooperating to prevent collisions at paths intersection.

Section 8.2 provides a comprehensive description of both the global planner and the distributed control algorithm used in *ASAPs*. Then, in Section 8.3, an analysis of the complexity of *ASAPs* is presented. The algorithm is evaluated in different self-reconfiguration scenarios, and the results are analyzed and compared with *RePoSt* in Section 8.4. The chapter is concluded in Section 8.5.

## 8.2/ Algorithm Description

In order to transform an initial configuration $I$ into a goal configuration $G$, we consider 3 different groups of meta-modules: Meta-modules that are in the initial configuration but not in the final one ($I \setminus G$), meta-modules present in both configurations ($I \cap G$), and others that are in $G \setminus I$.

Meta-modules in $I \setminus G$ must be dismantled, and their composing modules must flow inside the structure to fill empty positions in $G \setminus I$ by building new meta-modules.

Figure 8.1 shows the general flow of the algorithm. Given the initial and goal configurations, a global planner will perform a centralized computation that finds the flow paths of the modules from $I \setminus G$ to $G \setminus I$ in an initialization phase using a Max-Flow algorithm. The resultant flowing paths will allow to specify which operation to execute on each meta-modules. Then, the operations are transferred to their respective meta-modules on flowing paths. Once the operations are assigned, the modules execute a distributed algorithm based on the traffic light system that controls the flow of the modules on concurrent paths without collisions.

In this section, we first describe a global planning algorithm based on Max-Flow search to determine the operations to execute on each meta-module and in which direction. Then we describe an asynchronous distributed algorithm to control modules' flow on the paths without collisions.

### 8.2.1/ Global Planning

Given an initial configuration $I$ and a goal configuration $G$, a global planner must specify the operations to execute on each meta-module. It is a centralized process that is executed in an initialization phase.

The meta-modules configuration can be represented as a lattice graph in which the

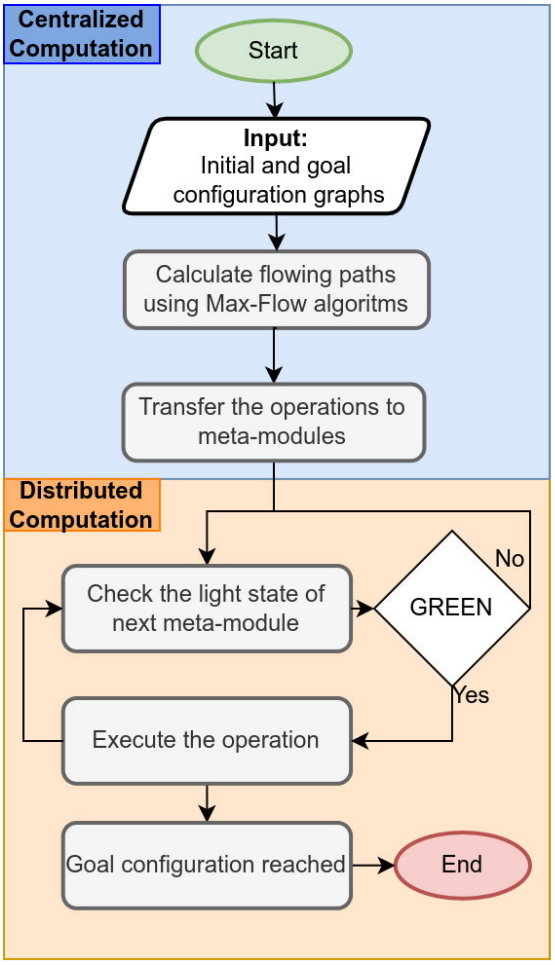



Figure 8.1: The general flow of the algorithm.

nodes represent meta-modules and the edges represent the connections between adjacent meta-modules. The global planner starts by constructing a graph $\mathcal{G}$ representing $I \cup G$ (the whole space reached by the reconfiguration process). A demand region $R_d$ is defined as a connected subgraph that contains nodes in $G \setminus I$. A supply region $R_s$ is a connected subgraph that contains nodes in $I \setminus G$. The nodes in $R_d$ and $R_s$ are connected with edges with infinite capacity to the closest neighbor in terms of hop distance to any node in $I \cap G$. Multiple supply and demand regions can exist in a single graph $\mathcal{G}$ depending on the symmetrical difference between the initial shape $I$ and the goal shape $G$ ($I \triangle G$). A super-supply $S_s$ node is added to the graph and is connected to all nodes in all supply regions with an edge of capacity $1$. All nodes in the demand regions are connected to a super-demand $S_d$ node with an edge of capacity $1$. The nodes in $G \cap I$ are connected with edges of infinite capacity. An example of this construction is shown in Figure 8.2.

Once the graph $\mathcal{G}$ is built, we apply the Edmonds-Karp algorithm Edmonds and Karp (1972b) to find the maximum flow between the super-supply node $S_s$ and the super-demand node $S_d$. The algorithm proceeds by finding the shortest augmenting paths

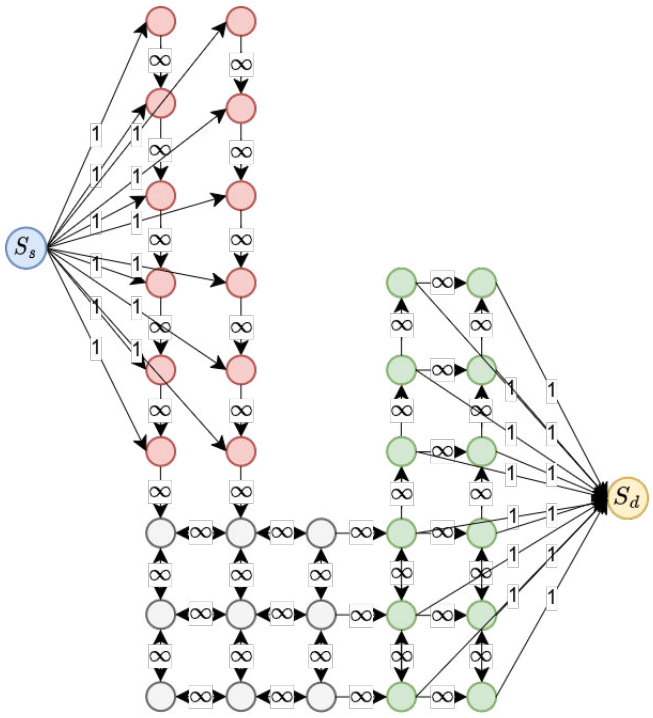



Figure 8.2: $\mathcal{G} = G \cup I$ construction example. Nodes in $R_s$ are colored in red. Nodes in $R_d$ are colored in green.

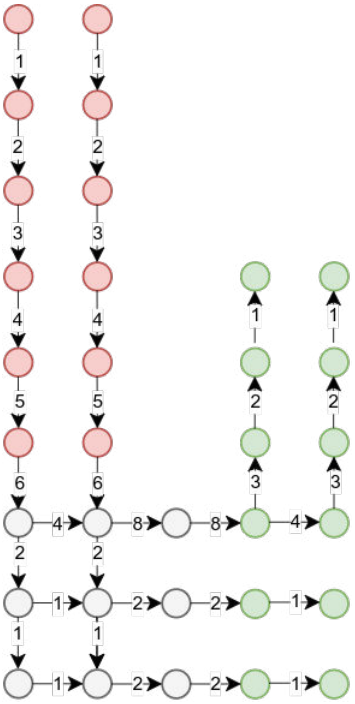


Figure 8.3: The resultant flow after applying the max-flow algorithm.

between $S_s$ and $S_d$ using breadth-first searches on the residual graph. It terminates when no more augmentations can be found.

The flow resulting after applying the Edmonds-karp algorithm on the graph of Figure 8.2 at each edge is shown in Figure 8.3. Nodes $S_s$ and $S_d$ are removed because they are virtual nodes and do not represent any meta-module. The flow value $f_{uv}$ at an edge $u, v$ indicates the number of modules to be routed by meta-module $u$ to meta-module $v$.

Each meta-module must know the flow value towards its neighbor meta-modules. These values must be sent from the central station to all meta-modules. This can be done at an initialization phase through tree-based broadcasts starting from a root module wired to the central station.

#### 8.2.1.1/ FLOW PROPERTIES

Applying the Edmonds-Karp algorithm to the graph construction mentioned above produces a flow with the following properties:

**Property 1.** *The flow covers all demand regions i.e. in the resultant flow, a path exists that connects a supply node to a demand node.*

Edmonds-karp algorithm satisfies the flow conservation constraint at the terminal nodes, which states that the sum of the flow flowing out of the source $S_s$ is equal to the sum of the flow flowing into the sink $S_d$. Since the number of supply nodes is equal to the number of demand nodes and the number of edges going out of $S_s$ is equal to the number of edges going in $S_d$, each augmenting path, excluding $S_s$ and $S_d$, starts with a supply node and ends at a demand node.

**Property 2.** *The total length in terms of hop distance of the paths connecting the supply nodes to the demand nodes is minimized.*

The Edmonds-Karp algorithm finds the shortest possible augmenting path using a breadth-first search. The length of the paths found at each iteration increases monotonically. Therefore, the total length of the path set is minimized. This is an important property, as minimizing the distance traveled reduces the number of commands required for the modules. Therefore, the total energy consumed during self-reconfiguration is also reduced.

**Property 3.** *No two paths connecting supply nodes to demand nodes share a common edge with opposite directions, which may cause a head-on collision.*

Having two paths in the resultant flow with two edges with opposite directions connecting the same two nodes contradicts property 2. This is because switching the destinations on those paths will reduce the total length.

### 8.2.2/ Distributed Flow Control Algorithm

Once the flow values on connections between neighbor meta-modules are received, the modules can start to flow from the supply regions to the demand regions. To do so, the meta-modules executes the operations set according to the flow values and their directions.

**Algorithm 8:** Distributed control algorithm on an MML.

**Data:** $F$ : A queue of pairs $< direction, flow >$ representing flow values in each directions.
**Data:** $lightState$
1 **Initialization** MML of a meta-module $u$:
2  mustDismantle($u$)
3 **Function** mustDismantle(u):
4  **if** $u \in R_s$ and no flow is entering $u$ **then**
5  $OPC.Operation \leftarrow$ dismantle($F_0.direction$); **send** REQUEST_START_OP() to MML in direction $F_0.direction$;
6 **Function** setOperation():
7  **if** hasNeighborInDirection($F_0.direction$) **then**
8  **send** SET_OPERATION(transfer($F_0.direction$) to OPC;
9  **else**
10  **send** SET_OPERATION(build($F_0.direction$) to OPC;
11  $F_0.flow \leftarrow F_0.flow - 1$;
12  **if** $F_0.flow = 0$ **then**
13  $F.pop()$;
14 **Msg Handler** REQUEST_START_OP():
15  **if** $lightState = GREEN$ **then**
16  $lightState \leftarrow RED$;
17  setOperation();
18  **send** AUTHORIZE_OP() to $senderMML$;
19  **else**
20  $waiting.push(direction_{sender})$;
21 **Msg Handler** AUTHORIZE_OP():
22  Notify the $OPC$ to start executing the operation;
23 **Event** OPERATION_ENDED on meta-module $u$:
24  **if** $waiting \neq \emptyset$ **then**
25  **send** AUTHORIZE_OP() to $waiting_0$;
26  $waiting.pop()$ ;
27  **else**
28  $lightState \leftarrow GREEN$;
29  mustDismantle($u$);

Algorithms 8, 9 describe the behavior of the OPC and MML modules. The flow starts by executing dismantle operations on meta-modules that are in $R_s$ and do not have an

**Algorithm 9:** Distributed control algorithm for an OPC

**Data:** $Operation$: The operation in execution.
**Data:** $mvt\_it = 0$: Iterator on Operation's movements.

1 **Event** ADD_NEIGHBOR($m$)**:**
2  **if** operation execution is authorized by MML **then**
3  **send** COORDINATE_MSG($Operation$, $mvt\_it$) to $m$ ;
4  $mvt\_it \leftarrow mvt\_it$ + nb of moves to be performed by $m$ ;
5  **else**
6  MML **sends** REQUEST_START_OP to next MML;
7 **Msg Handler** SET_OPERATION($Op$)**:**
8  $Operation \leftarrow Op$
9 **Msg Handler** POSITION_REACHED()**:**
10  **if** mvt_it ¡ Operation.size() **then**
11  m = module at $Operation[mvt\_it].current\_position$ ;
12  **send** COORDINATE_MSG($Operation$, $mvt\_it$) to $m$ ;
13  $mvt\_it \leftarrow mvt\_it$ + nb of moves to be performed by $m$ ;

entry flow. However, before starting the execution of any operation, a meta-module must verify that the next meta-module is not executing any operation to prevent collisions at intersections. To do so, the MML sends a `REQUEST_OP_START` message to the MML of the next meta-module on its path. If the receiver's $lightState$ is green, which means that it is not executing any operation, it sets the next operation to execute on it, then it responds with an `AUTHORIZE_OP` message, and its $lightState$ becomes red. Once the `AUTHORIZE_OP` message is received, the OPC can start the operation. Otherwise, the receiver stores the direction of the sender in a queue and responds once it becomes free. This will cause flowing modules to wait for the next meta-modules they must enter to finish executing an operation in progress (Algorithm 8 line 15-20).

The FMs executes the operations as explained in Section 6.3 except that before starting the operation , the MML requests the authorization to start the execution from the next MML (Algorithm 9 line 1-6).

When an MML detects that the operation's execution ended and there exists a meta-module waiting for its authorization to start the pending operation, it sends an `AUTHORIZE_OP` message to the waiting meta-module to start executing the operation. Otherwise, if it is in $R_s$ and does not have an entry flow, it sets $lightState$ as green and dismantles itself. Therefore, meta-modules in $R_s$ execute the dismantle operation one after the other starting from the end of a path so that they do not disconnect the configuration.

## 8.3/ Complexity Analysis

In this section, we analyze the complexity of the *ASAPs* algorithm. The total time needed for self-reconfiguration includes the time $T_0$ taken by the global planner to find paths, the time for module transport $T_1$, and the time for message transmission $T_2$.

The computational complexity of the global planner is given by the complexity of constructing $\mathcal{G} = G \cup I$ plus the complexity of the max-flow algorithm. The construction of $\mathcal{G}$ depends linearly on the number of nodes $V$, so $O(V)$. The complexity of the Edmonds-Karp algorithm on any graph is given as $O(VE^2)$ Edmonds and Karp (1972b) where $V$ is the number of nodes and $E$ is the number of edges in $\mathcal{G}$. In our case, $\mathcal{G}$ is a graph representing the nodes in a cubic lattice, so the maximum number of edges $E$ is equal to $6.V$. Therefore, the computational complexity of the global planner is $O(V(6V)^2 + V) = O(V^3) = T_0$.

The time to transform an initial shape to the target shape is mainly due to the time of the module flow. The longest distance that a module can travel is the diameter $d_{\mathcal{G}}$ of $\mathcal{G} = G \cup I$ where $G$ is the goal configuration and $I$ is the initial configuration. Modules can flow in parallel following concurrent paths of maximum length $d_{\mathcal{G}}$. Therefore, the time complexity can be expressed as $O(d_{\mathcal{G}}) = T_1$.

It is interesting to compare this complexity with the complexity of the *RePoSt* algorithm, which was $O(dN_{rounds})$, where $N_{rounds}$ was the number of rounds necessary to achieve reconfiguration. In the worst case $N_M - 1$ rounds are required, where $N_M$ is the number of meta-modules. *ASAPs* performs the self-reconfiguration in a single round with an increase in motion parallelization, which takes less time to achieve the goal configuration.

The message complexity of *ASAPs* is due to sending the flow values to their corresponding meta-modules and to the messages used during the flow control algorithm described in Section8.2.2. Sending the flow values can be done through a breadth-first spanning tree rooted at the central station that takes $O(d_{C_I} N_M)$ messages where $C_I$ is the initial configuration. Each movement of each flowing module along the flowing path requires sending a fixed number of messages. Therefore, the flow of modules requires $O(d_{\mathcal{G}} N_m)$ where $N_m$ is the number of modules. The message complexity of both steps can be expressed as $O(d_{C_I} N_M + d_{\mathcal{G}} N_m) = T_2$.

## 8.4/ Simulation and Results

### 8.4.1/ Presentation of the Experiments

In order to evaluate the algorithm, we consider different initial and final configurations with several properties:

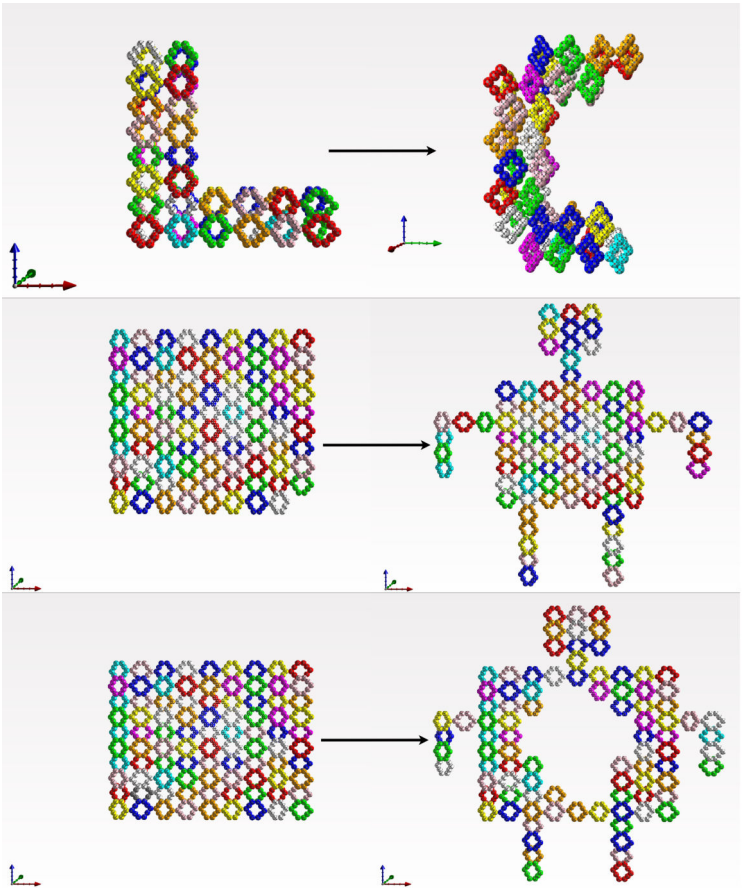


Figure 8.4: Three different self-reconfiguration scenarios, from the top to the bottom: L2C, Human and Hollow. Initial configurations are on the left and goal configuration on the right.

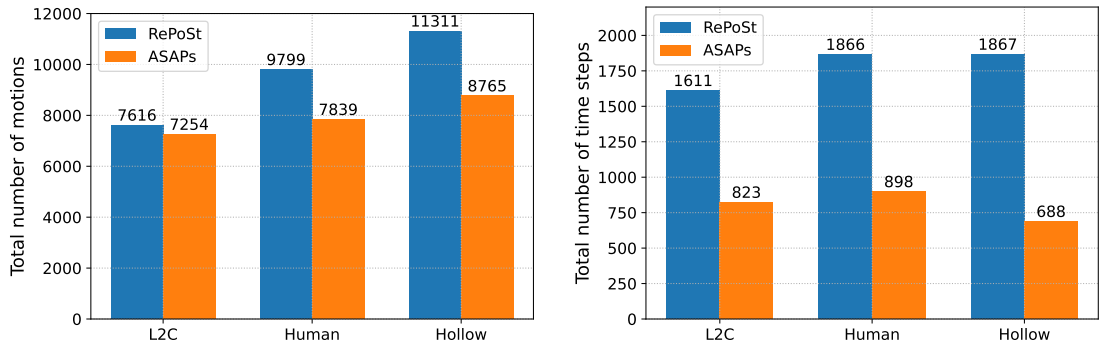
**L2C**: From a L shape made of 48 meta-modules to a C shape. This model shown in Figure 8.4.a, is similar to a narrow line of $2 \times 2$ meta-modules of the section. The narrowness of the line reduces the number of possible simultaneous motions.

**Human**: From a square of 89 meta-modules to a humanoid shape. This model shown in Figure 8.4.b is rich of numerous different paths in the central area (the body of the guy), but there are only some paths to reach the head, arms, and legs area, which will cause bottlenecks. The goal configuration is formed by transporting the meta-modules on the initial configuration borders to the head, arms and legs.

**Hollow**: From a square of 90 meta-modules to a hollow humanoid shape. This model shown in Figure 8.4.c proposes fewer internal meta-modules in its goal configuration than the previous one. The meta-modules in the central area must be dismantled without causing a disconnection.

### 8.4.2/ EXPERIMENTS ANALYSIS AND COMPARISON WITH *RePoSt*

The three self-reconfigurations are presented in a video [1] that shows the dismantling, displacement, and building of meta-modules in parallel to transform the initial shape to obtain the final configuration of meta-modules. It also show a comparison between *RePoSt* and *ASAPs* execution.




(a) Total number of motions.

(b) Total number of time steps.

Figure 8.5: Comparisons of the number of motions (a) and speed (b) of *ASAPs* and *RePoSt* algorithms for the 3 experimental shapes.

Figure 8.5 compares the speed and the total number of motions of the self-reconfiguration process for the *ASAPs* algorithm and *RePoSt* presented in Chapter 7.

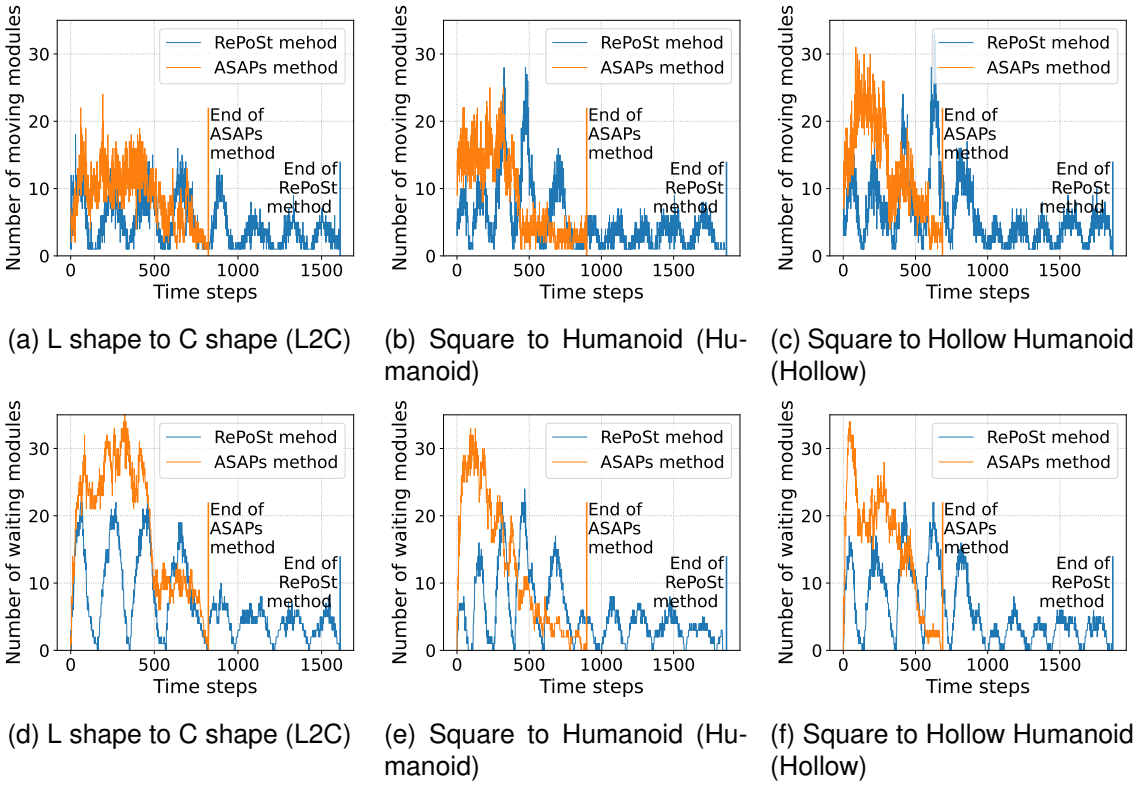
First, we notice that for the three self-reconfigurations, the *ASAPs* algorithm is faster than *RePoSt*, and this is mainly because the parallelization of movements is much more important due to the coordinated flow of modules on the pre-allocated concurrent paths. *ASAPs* algorithm is $1.95$ times faster for L2C, $2.08$ times faster for the Humanoid model and $2.7$ times faster for the Hollow model.

Second, in terms of the total number of motions executed by the modules during self-reconfiguration, *ASAPs* requires fewer motions than *RePoSt* to converge to the goal shape. This is due to the global max-flow planning method that minimizes the total length of the paths found connecting meta-modules in the supply region to the meta-modules in the demand region (cf. property 2). Therefore, *ASAPs* is more energy efficient than *RePoSt*.

Figure 8.6 shows the number of module motions and the number of modules that are waiting per time step when executing *RePoSt* and *ASAPs*. A time step corresponds to the average time required for a *3D Catom* rotation.

The regular oscillations of the curve shown in Figure 8.6 (a), (b) and (c) for the *RePoSt* algorithm are evidence of the successive rounds of this algorithm; they regularly cause

[1] Youtube video: https://youtu.be/Kqick3Am-Q8




(a) L shape to C shape (L2C)

(b) Square to Humanoid (Humanoid)

(c) Square to Hollow Humanoid (Hollow)

(d) L shape to C shape (L2C)

(e) Square to Humanoid (Humanoid)

(f) Square to Hollow Humanoid (Hollow)

Figure 8.6: Comparisons of *ASAPs* and *RePoSt* number of modules in motions per time step (a, b and c) and number of waiting modules (d, e, and f) for the three experimental shapes.

periods with a low number of movements due to the time required for the determination of streamlines at each round. This effect disappears almost completely on the curve given by the self-reconfiguration with the *ASAPs* algorithm. This is because once the meta-modules receive the path information, they all start to flow asynchronously guided by the distributed control algorithm explained in Section 8.2.2. Therefore, the *ASAPs* number of motions curve starts by increasing until it reaches its maximum value, then stabilizes before it starts to decrease when the modules start to reach their goal positions. This shows the increase in motion parallelisms achieved using *ASAPs*.

In Figure 8.6 (d), (b) and (f), the number of waiting modules at a time step varies with the amount of flow. When more modules leave their initial position and start flowing, the number of waiting modules will increase because modules will wait for each other to keep enough space when flowing in a train-like fashion towards their destination. In *ASAPs* an increased number of modules can be found in a waiting state at the beginning time steps for two reasons: first, the number of flowing modules is more important, so more modules are waiting to keep enough space with other flowing modules on the same path, and second, when modules must wait at intersections of paths in case an operation is being executed at the intersection. For the *RePoSt* algorithm, modules follow disjoint paths, so

they are not required to wait at intersection.

#### 8.4.2.1/ BOTTLENECKS EFFECT

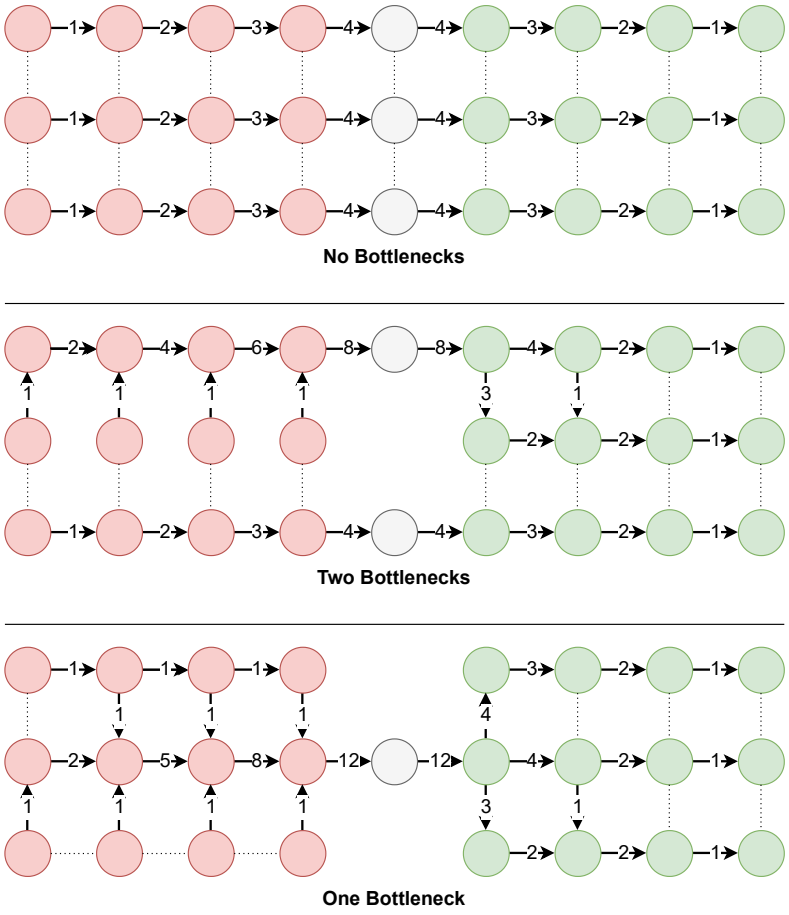



Figure 8.7: The result of paths generated by the max-flow on 3 configurations with different bottleneck sizes. $R_s$ nodes are in red and $R_d$ nodes are in green.

The waiting time of the modules that execute *ASAPs* also depends on the bottlenecks at the intersections of the paths that reach a narrow area. For example, Figure 8.7, shows the paths generated by the max-flow in three configurations where the modules in $R_s$ must cross none, two, or three bottleneck nodes to fill empty positions in $R_d$. Figure 8.8 shows the total self-reconfiguration time of the simulation of the three examples. It can be seen that when multiple paths intersect on one node, causing a bottleneck, the self-reconfiguration time increases. The reason is that only one operation is executed at a time on the bottleneck nodes. Modules that need to go through bottlenecks are waiting for their turn, as explained in Sections 8.2.2.

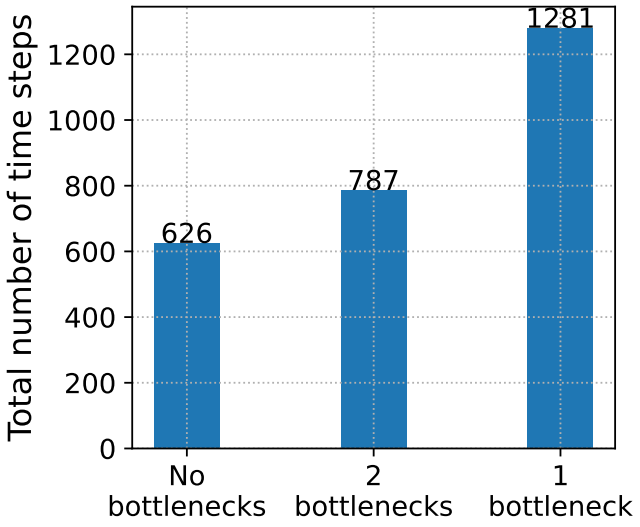



Figure 8.8: Total number of time steps on the self-reconfiguration example of Figure 8.7.

## 8.5/ CONCLUSION

In this chapter *ASAPs*, a hybrid self-reconfiguration algorithm is developed where a central global planner calculates the maximum flow of modules to plan the paths of self-reconfiguration. Unlike *RePoSt*, the paths can intersect. The motions of modules are controlled by a fully distributed and asynchronous flow control algorithm.

We evaluated *ASAPs* in simulation and compared the parallelism, total distance traveled, and self-reconfiguration time with *RePoSt*. The results show an important improvement of efficiency in both the total distance traveled which affect the energy used by the modules and in the self-reconfiguration time.

# 9

# CONCLUSION

## 9.1/ SUMMARY

This thesis is part of a work that aims to achieve a programmable matter by an assembly of a large number of tiny robotic modules. Modules are interconnected and can communicate to coordinate the self-reconfiguration of the ensemble into a goal configuration. The objective of the proposed algorithms is to increase the efficiency of self-reconfiguration. Therefore, we proposed and implemented algorithms for size-constrained clustering, shape recognition, and self-reconfiguration.

A fully distributed clustering algorithm (SC-Clust) which uses message passing to allow modules to cluster themselves into predefined size clusters is presented in Chapter 4. The main motivation of this work is to reduce the search space when planning for self-reconfiguration and to allow clusters to reconfigure in parallel. The clustering algorithm is based on tree cuts that result in branches that form the final clusters. The results showed that we can efficiently and effectively group ensembles of tens of thousands of modules with $O(nlogn)$ communication and time complexities with $n$ the number of modules.

To allow an ensemble of connected modules in a regular lattice to identify a description of their shape, Chapter 5 presented a fully distributed algorithm that finds a set of filled boxes whose union describes the shape. It has a communication complexity $O(n)$ and a time complexity of $O(D + W + H)$ where $n$ the number of modules and $D$, $W$ and $H$ the depth, width, and height of the bounding box of the ensemble, respectively. Detecting the shape of a modular robot ensemble can be beneficial for an efficient self-reconfiguration planner. We also showed in simulations that this approach is more efficient in reporting the shape to an external connected computer than exhaustive coordinate collection.

In order to increase the number of possible motions of modules to enhance self-reconfiguration, Chapter 6 presented a new porous structure that has enough hollow internal volume to allow modules to move through it concurrently. It is built by connected meta-modules placed in a regular cubic lattice. The proposed meta-modules can be in

two states either sparse or full, giving the structure the ability to compress and expand. We also defined a set of operations that meta-modules can perform to change their state or direct modules that must traverse through them.

Then, Chapter 7 and Chapter 8 presented two self-reconfiguration planning algorithms based on a max-flow search that specify which operation to execute on which meta-module to transform the structure into a given goal shape. The first *RePoSt* is a fully distributed round-based algorithm where in each round the modules flow following disjoint paths to reach their goal positions. With *RePoSt* we can reach a goal configuration in $O(Md)$ where $M$ is the number of rounds and $d$ is the diameter of the configuration. The second *ASAPs* follows a hybrid planning approach, where a centralized planner precalculates the flow to be sent in each direction for all meta-modules. This information is then distributed to the meta-modules. The modules then flow asynchronously by performing the operations specified according to the flow values, reducing the time complexity to $O(d)$. We evaluated and compared both algorithms in different configurations in simulation.

## 9.2/ Discussions and Future Work

In this section, I would like to give a general perspective, then take a step back and discuss the advantages and disadvantages of the proposed solutions, and also show some future work for each of the proposed solutions.

### 9.2.1/ General Perspective

Regarding self-reconfiguration, I hope that the development of *3D Catoms* will soon produce operational hardware so that I can implement my algorithms and test their practicality in a real-world setting. I would like to investigate the potential faults that can occur on real hardware, such as incomplete rotation, communication failures, and other hardware dysfunctions. Subsequently, I aim to propose software solutions that effectively employ robustness in the face of these dysfunctions, minimizing the need for human intervention. Some noteworthy solutions have been proposed in related research: Makhoul and Bassil (2023) addresses communication errors, Bassil et al. (2022) focuses on handling communication failures caused by broken interfaces, and Hourany et al. (2022) presents a disconnection detection method. My goal is to propose additional solutions for motion failures, power failures, and other dysfunctions and apply them in practice to ensure the completeness of self-reconfiguration on real hardware.

Furthermore, it would be useful to adapt the algorithms to other models and systems. This includes systems with wireless communication capabilities, as well as systems operating

in different lattice structures and employing different motion mechanisms.

### 9.2.2/ On the Size-Constraint Clustering

Chapter 4 presented a new algorithm called SC-Clust that clusters modules into predefined size groups. The main goal is to improve the self-reconfiguration process by allowing each cluster to reconfigure at the same time. This algorithm represents a significant advancement as it is the first fully distributed approach for size-constrained clustering that relies solely on local module knowledge to decompose the ensemble into clusters. It is an important tool that enables the use of distributed cluster-based solutions for self-reconfiguration. However, there are some limitations that need to be addressed in order to achieve this.

First, to facilitate the transition to the goal configuration, the shape and positions of the resultant clusters must be controlled to prevent blocking positions caused by irregular cluster borders. Furthermore, the shape of the cluster can also be customized to meet specific goals. For example, if the goal is to reconfigure the system into a specific shape, it would be advantageous to design the clusters in a shape or pattern that is present in both the initial and the goal configurations. Such a cluster will not require many transformations, which can simplify the reconfiguration process by decreasing the amount of motion required.

Second, the positions of each cluster must be carefully considered to ensure successful reconfiguration. For example, placing clusters near their target positions may be a useful strategy to reduce the number of steps needed to achieve the desired configuration. This can help minimize the time and resources required to run the reconfiguration process, especially for systems that require rapid or frequent reconfiguration.

Furthermore, in cluster-based self-reconfiguration planning, finding the correct order of assembly is crucial because it affects the overall efficiency and success of the reconfiguration process. The order of constructions is determined by analyzing the dependencies between the connections of the modules or clusters. If a connection between two modules depends on the completion of another connection, those connections must be assembled in a specific order to ensure a successful construction.

Therefore, I would like to investigate these ideas to propose more efficient cluster-based self-reconfiguration algorithms that are compatible with this decomposition of the modules set. In particular, managing the inter-cluster boundaries and the order of assembly to avoid blocking constraints.

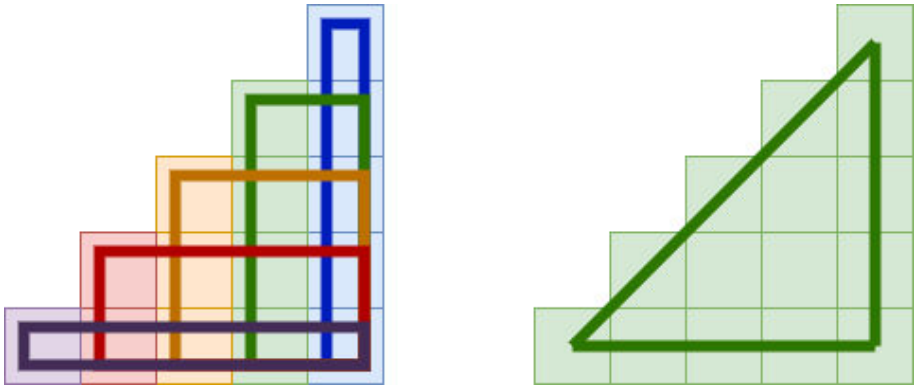


Figure 9.1: On the left, the representation of the shape is using 5 elements. On the right, it shows how using a triangle can reduce the number of elements to 1.

### 9.2.3/ On Shape Recognition

The shape recognition algorithm presented in Chapter 5 finds a representation of the current modular robot shape. The overlapping boxes can be seen as the leaf nodes of a Constructive Solid Geometry (CSG) tree, where the union of these boxes represents the overall shape of the robot.

The presented algorithm holds significant importance in the context of interactive Computer-aided design (CAD) applications. It facilitates accurate and prompt transmission of shape descriptions from the Programmable Matter (PM) to the connected computer. Additionally, the ability of modules to determine an efficient representation of their current shape carries potential benefits for self-reconfiguration planning. By comparing the differences between the current shape and the desired goal shape, modules can make informed decisions during the planning phase.

Extending the algorithm to use a variety of shapes, in addition to a box, as building blocks can potentially lead to more efficient representations of the robot's shape. One possibility is to adapt the algorithm to actively search for polyhedrons rather than solely focusing on boxes. Although representing polyhedrons requires additional information for storage, it can substantially reduce the number of elements required to cover the entire configuration. Figure 9.1 shows an example in which finding a diagonal can reduce the representation from five elements to one.

In the current version of the proposed algorithm, the boxes are aggregated when received by a connected computer. Although this method is effective in representing the overall shape of the robot, it may not be the most efficient way to do so if we want the modules themselves to store a description of the whole configuration shape. To address this limitation, a distributed aggregation method must be developed to minimize the number of boxes used to represent the robot's shape. This method would involve ignoring boxes that are fully included in larger ones, which can reduce redundancy and improve

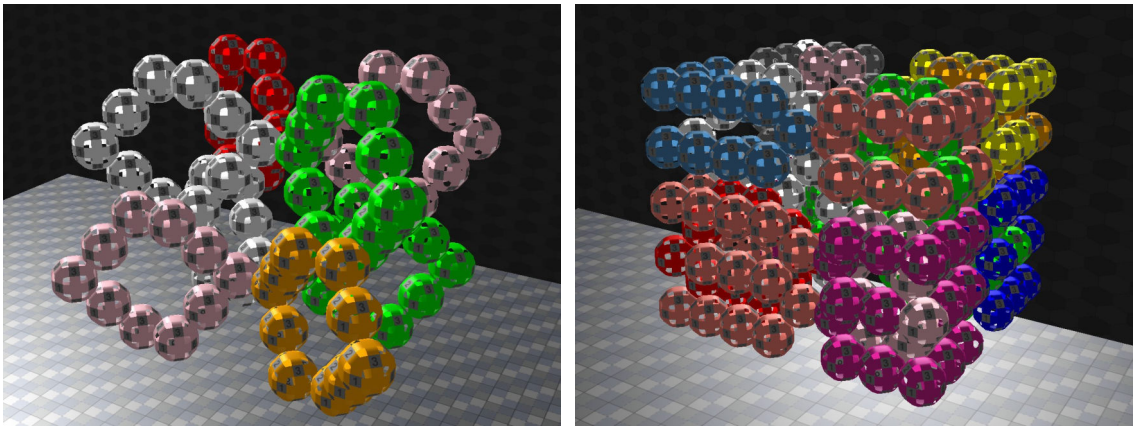
the efficiency of the algorithm.

Furthermore, I would like to work on developing a dynamic version of the algorithm that allows modules to keep track of their shape in real time, which could provide valuable information for distributed self-reconfiguration planning. By continuously updating the representation of the robot's shape, modules can make more informed decisions about their position relative to the goal configuration and potentially reconfigure themselves more efficiently.

### 9.2.4/ On the Porous Structure

**The meta-module design** The porous structure presented in Chapter 6 is made up of meta-modules. The decision to use hexagonal-shaped meta-modules was made in order to closely mimic a voxel in the square cubic lattice. Its size 10 was experimentally determined in *VisibleSim* to accommodate additional 10 *3D Catoms* within its empty internal volume to form a "FULL" meta-module while also leaving sufficient space for modules to flow within the empty volume of a "SPARSE" meta-module, as well as between two adjacent meta-modules in all directions. Any other meta-module design with the same properties can be used. For example, using other geometries can provide better spatial efficiency by using a smaller number of modules and get better fidelity to the goal shape. Larger meta-modules can be used to build an internal scaffold and smaller meta-modules on the borders. However, the challenge of this approach is the complexity it can cause to self-reconfiguration and assembly planning and in the coordination of the movements of modules between meta-modules with different geometries.

**Advantages** The porous structure allows the modules to flow freely within it. In addition, all meta-modules, "SPARSE" or "FULL" have at least one module that is not blocked free to move, therefore it can be dismantled and transported in any direction regardless of its position. Furthermore, the compressibility and expandability properties of the structure do not restrict the goal shape to have the same number of meta-modules as the initial shape. The excess number of modules in the initial shape can be stored inside "SPARSE" meta-modules and the filling modules can be later used for other purposes, such as coating the structure or replacing faulty modules. Furthermore, the meta-module design allows it to be assembled at any empty position even if the six adjacent positions are filled. This relaxes the bridging constraint that exists at the module level, which states that no modules can be placed between two filled positions in opposite directions. Therefore, it facilitates self-reconfiguration planning.



(a) A 2x2 cube without coating.

(b) A 2x2 cube coated with special meta-modules.

Figure 9.2: Coating example using special meta-modules on the boundaries.

**Coating** Coating the internal scaffolding structure with a thin layer of modules is necessary to better represent the shape. In Thalamy et al. (2020a), the authors presented the challenges of the coating problem and proposed a basic method to build the coating of a scaffold layer by layer in a separate step once the scaffold was established. I would like to investigate another coating solutions that performs the coating simultaneously with the self-reconfiguration of the internal scaffold. One solution would consist of altering the shape of the meta-modules at the boundaries to better approximate the goal shape at these locations. I believe that this can be achieved by using other meta-module shapes to cover the surface, as shown in Figure 9.2. To do so, we can use a new *Coat* operation that can assemble modules on the surface. This requires storing additional movement data and planning.

**Mechanical Stability** A disadvantage of the proposed structure is that two meta-modules in a single layer are horizontally connected with a single latching point that can overload the connection and cause breakage. An advantage of the proposed structure is the ability to distribute the weight of the robot across the entire structure by carefully choosing the positions of "FULL" meta-modules, which can improve stability. In Piranda et al. (2021), a solution is proposed that can be applied to detect the rupture point and stability in a distributed manner, and the result can be taken into account when planning reconfiguration. However, it is computationally prohibitive. The work is ongoing within our team to efficiently and distributedly detect the stability of a structure in real-time. I believe that using structures with regular building blocks such as the meta-modules proposed in this thesis can reduce the calculation by performing it on the meta-module level. For example, to calculate the center of gravity of a structure using the coordinates of its components, we can do it at the meta-module level instead of the module level, reducing

the number of coordinates used for calculation by $n$ times compared to the module level where $n$ is the size in the a meta-module ($n = 10$ in our proposed meta-module).

**Operations movements** To reconfigure the structure, the modules can perform operations consisting of a sequence of predefined motions stored in their memory. The operations will be set by a self-reconfiguration planner. For now, the operations are hand-coded, which is a tedious and time-consuming task. Therefore, I believe it would be interesting to automate this task, which can not only facilitate the process, but also find an optimized sequence of movements. I would like to investigate graph-based path-planning methods, reinforcement learning and other search based motion planning for this purpose.

### 9.2.5/ On the Self-Reconfiguration Algorithms

In chapters 7 and 8, two self-reconfiguration algorithms are presented. They specify which operations to execute and in which direction on each meta-module to reach a given goal configuration. The first *RePoSt* is fully distributed. It uses the goal shape representation that must be known by all meta-modules to determine a set of sources and destinations. Then in each of its rounds, modules flow from sources such that their removal does not disconnect the structure to destinations until the convergence to the goal shape. The second, *ASAPs*, follows a hybrid approach in which a centralized planner, given the initial and goal configurations, calculates the maximum flow between regions that do not belong to the goal configuration and regions that are in the goal but not in the initial configuration. The result of the max-flow is then transmitted to the meta-modules and they start flowing in a simultaneously while applying a traffic-light-like mechanism to schedule operations on flow intersections.

The simulation results showed that *ASAPs* significantly decreases the time of self-reconfiguration due to an increase in flow. However, it requires a central planner and a prior knowledge of the initial and goal configurations, which limits its applicability to certain scenarios: when the ensemble is not connected to a central planner and/or the initial shape is unknown. The shape recognition algorithm of chapter 5 can be used to determine the initial configuration.

Although we talked about the filling procedure and how it can be used to not constrain the size of the goal configuration to be strictly equal to the initial one. I believe that this feature can be used by the self-reconfiguration planner to enhance the process. I would like to test different strategies for resource allocation. A resource allocator can be used to strategically position "SPARSE" meta-modules in areas where deletion is expected and "FULL" meta-modules in regions where creation is anticipated.

For now, the proposed planners use only the space covered by the union of the initial and

goal configurations. I would like to investigate how intermediate configurations that span a larger space can be used to avoid bottlenecks during the self-reconfiguration process of modular robots. Bottlenecks occur when there is a limited space or pathway in the structure that restricts the movement of the modules. This can lead to congestion and delays in the self-reconfiguration process. To avoid bottlenecks, the self-reconfiguration planner can use intermediate configurations to create alternative paths in the structure. For example, if there is a narrow pathway that restricts the movement of the modules, the planner can create an intermediate configuration that rearranges the modules and creates a wider pathway. Future research in this area can focus on developing efficient and effective strategies for using intermediate configurations to avoid bottlenecks and optimize the self-reconfiguration process.

# PERSONAL PUBLICATIONS

## JOURNALS

1. Jad Bassil, Abdallah Makhoul, Benoît Piranda, Julien Bourgeois. **Distributed Size-Constrained Clustering Algorithm for Modular Robot-Based Programmable Matter.** *In ACM Transactions on Autonomous and Adaptive Systems,* Published in March 2023.

2. Jad Bassil, Benoît Piranda, Abdallah Makhoul, Julien Bourgeois. **ASAPs: Asynchronous Hybrid Self-Reconfiguration Algorithm for Porous Modular Robotic Structures.** *Preprint,* Under Review in Autonomous Robots.

## INTERNATIONAL CONFERENCES

1. Jad Bassil, Jean-Paul Abs Yaacoub, Benoît Piranda, Abdallah Makhoul and Julien Bourgeois. **Distributed Shape Recognition Algorithm for Lattice-Based Modular Robots.** *In 2023 International Symposium on Multi-Robot & Multi-Agent Systems (MRS),* Boston, USA, 2023, pp. 1-7.

2. Abdallah Makhoul, Jad Bassil. **Fault Tolerance Technique Using Bidirectional Hetero-Associative Memory for Self-Reconfigurable Programmable Matter.** *In 202 International Wireless Communications and Mobile Computing (IWCMC),* Marrakesh, Morocco, 2022, pp. –, Core Rank: B.

3. Jad Bassil, Benoît Piranda, Abdallah Makhoul, Julien Bourgeois. **RePoSt: Distributed Self-Reconfiguration Algorithm for Modular Robots Based on Porous Structure.** *In 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Kyoto, Japan, 2022, pp. 12651-12658, Core Rank: A.

4. Jad Bassil, Perla Tannoury, Benoît Piranda, Abdallah Makhoul, Julien Bourgeois. **Fault-Tolerance Mechanism for Self-Reconfiguration of Modular Robots.** *In 2022 International Wireless Communications and Mobile Computing (IWCMC),* Dubrovnik, Croatia, 2022, pp. 360-365, Core Rank: B.

5. Jad Bassil, Benoît Piranda, Abdallah Makhoul, Julien Bourgeois. **A new porous structure for modular robots.** *In Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS),* Virtual, 2022, pp. 1539–1541, Core Rank: A*.

6. Jad Bassil, Benoît Piranda, Abdallah Makhoul, Julien Bourgeois. **Enhanced Precision Time Synchronization for Modular Robots.** *In 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA),* Boston, MA, USA, 2021, pp. 1-5, Core Rank: B.

# BIBLIOGRAPHY


**[ReactiveMatter ]** *Reactive Matter*. http://www.scenocosme.com/reactive_matter.htm. – Accessed: 2023-03-28

**[Adoni et al. 2020a]** ADONI, Hamilton Wilfried Y. ; NAHHAL, Tarik ; KRICHEN, Moez ; AGHEZZAF, Brahim ; ELBYED, Abdeltif: *A survey of current challenges in partitioning and processing of graph-structured data in parallel and distributed systems*. jun 2020

**[Adoni et al. 2020b]** ADONI, Wilfried Yves H. ; NAHHAL, Tarik ; KRICHEN, Moez ; EL BYED, Abdeltif ; ASSAYAD, Ismail: **"DHPV: a distributed algorithm for large-scale graph partitioning"**. In *Journal of Big Data* 7 (2020), dec, number 1, pages 1–25. – URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00357-y. – ISSN 21961115. DOI: 10.1186/s40537-020-00357-y

**[Afsar and Tayarani-N 2014]** AFSAR, M. M. ; TAYARANI-N, Mohammad-H.: **"Clustering in sensor networks: A literature survey"**. In *Journal of Network and Computer Applications* 46 (2014), pages 198–226. – URL http://www.sciencedirect.com/science/article/pii/S1084804514002124. – ISSN 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2014.09.005

**[Ahmadzadeh and Masehian 2015]** AHMADZADEH, Hossein ; MASEHIAN, Ellips: **"Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization"**. In *Artificial Intelligence* 223 (2015), jun, pages 27–64. – URL https://linkinghub.elsevier.com/retrieve/pii/S0004370215000260. – ISSN 00043702. DOI: 10.1016/j.artint.2015.02.004

**[Ahmed et al. 2021]** AHMED, Aamir ; ARYA, Sandeep ; GUPTA, Vinay ; FURUKAWA, Hidemitsu ; KHOSLA, Ajit: **"4D printing: Fundamentals, materials, applications and challenges"**. In *Polymer* 228 (2021), pages 123926. – URL https://www.sciencedirect.com/science/article/pii/S0032386121005498. – ISSN 0032-3861. DOI: https://doi.org/10.1016/j.polymer.2021.123926

**[Ansari et al. 2019]** ANSARI, Shahab U. ; HUSSAIN, Masroor ; MAZHAR, Suleman ; MANZOOR, Tareq ; SIDDIQUI, Khalid J. ; ABID, Muhammad ; JAMAL, Habibullah: **"Mesh partitioning and efficient equation solving techniques by distributed finite element methods: A survey"**. In *Archives of Computational Methods in Engineering* 26 (2019), number 1, pages 1–16

**[Ashley-Rollman et al. 2011]** ASHLEY-ROLLMAN, Michael P. ; PILLAI, Padmanabhan ; GOODSTEIN, Michelle L.: **"Simulating multi-million-robot ensembles"**. In *2011 IEEE international conference on robotics and automation* IEEE (event), 2011, pages 1006–

1013

**[Assaker et al. 2022]** ASSAKER, Joseph ; MAKHOUL, Abdallah ; BOURGEOIS, Julien ; PIRANDA, Benoît ; DEMERJIAN, Jacques: **"A Dynamic ID Assignment Approach for Modular Robots"**. In *Advanced Information Networking and Applications: Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), Volume 1* Springer (event), 2022, pages 91–104

**[Baca et al. 2015]** BACA, Jose ; WOOSLEY, Bradley ; DASGUPTA, Prithviraj ; NELSON, Carl: **"Real-time distributed configuration discovery of modular self-reconfigurable robots"**. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2015, pages 1919–1924. – URL http://ieeexplore.ieee.org/document/7139449/. – ISBN 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139449

**[Bassil et al. 2020]** BASSIL, Jad ; MOUSSA, Mouhamad ; MAKHOUL, Abdallah ; PIRANDA, Benoit ; BOURGEOIS, Julien: **"Linear Distributed Clustering Algorithm for Modular Robots Based Programmable Matter"**. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020

**[Bassil et al. 2021a]** BASSIL, Jad ; PIRANDA, Benoît ; MAKHOUL, Abdallah ; BOURGEOIS, Julien: **"Enhanced Precision Time Synchronization for Modular Robots"**. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)* IEEE (event), 2021, pages 1–5

**[Bassil et al. 2021b]** BASSIL, Jad ; PIRANDA, Benoît ; MAKHOUL, Abdallah ; BOURGEOIS, Julien: **"Enhanced Precision Time Synchronization for Modular Robots"**. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, 2021, pages 1–5. DOI: 10.1109/NCA53618.2021.9685103

**[Bassil et al. 2022]** BASSIL, Jad ; TANNOURY, Perla ; PIRANDA, Benoît ; MAKHOUL, Abdallah ; BOURGEOIS, Julien: **"Fault-Tolerance Mechanism for Self-Reconfiguration of Modular Robots"**. In *2022 International Wireless Communications and Mobile Computing (IWCMC)* IEEE (event), 2022, pages 360–365

**[Berenger et al. 2018]** BERENGER, Cedric ; NIEBERT, Peter ; PERROT, Kevin: **"Balanced connected partitioning of unweighted grid graphs"**. In *Leibniz International Proceedings in Informatics, LIPIcs* Volume 117, 2018. – ISBN 9783959770866. DOI: 10.4230/LIPIcs.MFCS.2018.39

**[Bhola et al. 2020]** BHOLA, Jyoti ; SONI, Surender ; CHEEMA, Gagandeep K.: **"Genetic algorithm based optimized leach protocol for energy efficient wireless sensor networks"**. In *Journal of Ambient Intelligence and Humanized Computing* 11 (2020), number 3, pages 1281–1288

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

**[Bourgeois et al. 2016a]** BOURGEOIS, Julien ; PIRANDA, Benoit ; NAZ, Andre ; BOIL-

LOT, Nicolas ; MABED, Hakim ; DHOUTAUT, Dominique ; TUCCI, Thadeu ; LAKHLEF, Hicham: **"Programmable matter as a cyber-physical conjugation"**. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* IEEE (event), IEEE, oct 2016, pages 002942–002947. – URL https://hal.archives-ouvertes.fr/hal-02140347http://ieeexplore.ieee.org/document/7844687/. – ISBN 978-1-5090-1897-0. DOI: 10.1109/SMC.2016.7844687

**[Bourgeois et al. 2016b]** BOURGEOIS, Julien ; PIRANDA, Benoit ; NAZ, Andre ; BOILLOT, Nicolas ; MABED, Hakim ; DHOUTAUT, Dominique ; TUCCI, Thadeu ; LAKHLEF, Hicham: **"Programmable matter as a cyber-physical conjugation"**. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 10 2016, pages 002942–002947. – URL https://hal.archives-ouvertes.fr/hal-02140347http://ieeexplore.ieee.org/document/7844687/. – ISBN 978-1-5090-1897-0. DOI: 10.1109/SMC.2016.7844687

**[Brzezinski et al. 1993]** BRZEZINSKI, J. ; HELARY, J.M. ; RAYNAL, M.: **"Termination detection in a very general distributed computing model"**. In *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, 1993, pages 374–381. DOI: 10.1109/ICDCS.1993.287688

**[Bulla Cruz et al. 2017]** BULLA CRUZ, Nicolas ; NEDJAH, Nadia ; MOURELLE, Luiza: **"Robust distributed spatial clustering for swarm robotic based systems"**. In *Applied Soft Computing Journal* 57 (2017), pages 727–737. – URL http://dx.doi.org/10.1016/j.asoc.2016.06.002. – ISSN 15684946. DOI: 10.1016/j.asoc.2016.06.002

**[Buluç et al. 2016]** BULUÇ, Aydın ; MEYERHENKE, Henning ; SAFRO, Ilya ; SANDERS, Peter ; SCHULZ, Christian: *Recent Advances in Graph Partitioning*. pages 117–158. In *Algorithm Engineering: Selected Results and Surveys*. Cham : Springer International Publishing, 2016. – URL https://doi.org/10.1007/978-3-319-49487-6_4. – ISBN 978-3-319-49487-6. DOI: 10.1007/978-3-319-49487-6_4

**[Butler et al. 2002]** BUTLER, Zack ; FITCH, Robert ; RUS, Daniela ; YUHANG WANG: **"Distributed goal recognition algorithms for modular robots"**. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)* Volume 1, IEEE, 2002, pages 110–116. – URL http://ieeexplore.ieee.org/document/1013347/. – ISBN 0-7803-7272-7. DOI: 10.1109/ROBOT.2002.1013347

**[Castano et al. 2000]** CASTANO, Andres ; SHEN, Wei-Min ; WILL, Peter: **"CONRO: Towards deployable robots with inter-robots metamorphic capabilities"**. In *Autonomous Robots* 8 (2000), number 3, pages 309–324

**[Chennareddy et al. 2017]** CHENNAREDDY, S ; AGRAWAL, Anita ; KARUPPIAH, Anupama: **"Modular Self-Reconfigurable Robotic Systems: A Survey on Hardware Architectures"**. In *Journal of Robotics* 2017 (2017), pages 1–19. – URL https://www.hindawi.com/journals/jr/2017/5013532/. – ISSN 1687-9600. DOI: 10.1155/2017/5013532

**[Christensen et al. 2008]** CHRISTENSEN, David ; BRANDT, David ; STOY, Kasper ; SCHULTZ, Ulrik P.: **"A unified simulator for self-reconfigurable robots"**. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* IEEE (event), 2008, pages 870–876

**[Collins et al. 2021]** COLLINS, Jack ; CHAND, Shelvin ; VANDERKOP, Anthony ; HOWARD, David: **"A review of physics simulators for robotic applications"**. In *IEEE Access* 9 (2021), pages 51416–51431

**[Collins and Shen 2016]** COLLINS, Thomas ; SHEN, Wei-Min: **"Rebots: A drag-and-drop highperformance simulator for modular and self-reconfigurable robots"**. 2016. – Research Report

**[Cristian 1989]** CRISTIAN, Flaviu: **"Probabilistic Clock Synchronization"**. In *Distributed Computing* 3 (1989), September, number 3, pages 146–158. – ISSN 0178-2770

**[Dewey et al. 2008]** DEWEY, Daniel J. ; ASHLEY-ROLLMAN, Michael P. ; DE ROSA, Michael ; GOLDSTEIN, Seth C. ; MOWRY, Todd C. ; SRINIVASA, Siddhartha S. ; PILLAI, Padmanabhan ; CAMPBELL, Jason: **"Generalizing metamodules to simplify planning in modular robotic systems"**. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (2008), September, pages 1338–1345. ISBN 9781424420582. DOI: 10.1109/IROS.2008.4651094

**[Di Caro et al. 2012]** DI CARO, Gianni ; DUCATELLE, Frederick ; GAMBARDELLA, Luca M.: **"A fully distributed communication-based approach for spatial clustering in robotic swarms"**. In *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), pages 153–171

**[Diaz and Mendez 2019]** DIAZ, Sergio ; MENDEZ, Diego: **"Dynamic minimum spanning tree construction and maintenance for Wireless Sensor Networks"**. In *Revista Facultad de IngenierÃa Universidad de Antioquia* (2019), 12, pages 57–69. – URL http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0120-62302019000400057&nrm=iso. – ISSN 0120-6230

**[Diekmann et al. 2000]** DIEKMANN, Ralf ; PREIS, Robert ; SCHLIMBACH, Frank ; WALSHAW, Chris: **"Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM"**. In *Parallel Computing* 26 (2000), number 12, pages 1555–1581

**[Dokuyucu and Özmen 2022]** DOKUYUCU, Halil .. ; ÖZMEN, Nurhan G.: **"Achievements and future directions in self-reconfigurable modular robotic systems"**. In *J. Field Robotics* (2022), December. – URL https://onlinelibrary.wiley.com/doi/10.1002/rob.22139. – ISSN 1556-4959. DOI: 10.1002/rob.22139

**[Dutta et al. 2016]** DUTTA, Ayan ; DASGUPTA, Prithviraj ; NELSON, Carl: **"A bottom-up search algorithm for size-constrained partitioning of modules to generate configurations in modular robots"**. In *Web Intelligence* 14 (2016), number 1, pages 67–82.

– ISSN 24056464. DOI: 10.3233/WEB-160332

**[Dutta et al. 2015]** DUTTA, Ayan ; DASGUPTA, Raj ; BACA, José ; NELSON, Carl A.: **"Spanning Tree Partitioning Approach for Configuration Generation in Modular Robots"**. In RUSSELL, Ingrid (editors) ; EBERLE, William (editors): *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida, USA, May 18-20, 2015*, AAAI Press, 2015, pages 360–365. – URL http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10447

**[Dutta et al. 2019]** DUTTA, Ayan ; UFIMTSEV, Vladimir ; ASAITHAMBI, Asai ; CZARNECKI, Emily: **"Coalition Formation for Multi-Robot Task Allocation via Correlation Clustering"**. In *Cybernetics and Systems* 50 (2019), number 8, pages 711–728. – URL https://doi.org/10.1080/01969722.2019.1677334. DOI: 10.1080/01969722.2019.1677334

**[Edmonds and Karp 1972a]** EDMONDS, Jack ; KARP, Richard M.: **"Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems"**. In *J. ACM* 19 (1972), April, number 2, pages 248–264. – URL https://doi.org/10.1145/321694.321699. – ISSN 0004-5411. DOI: 10.1145/321694.321699

**[Edmonds and Karp 1972b]** EDMONDS, Jack ; KARP, Richard M.: **"Theoretical improvements in algorithmic efficiency for network flow problems"**. In *Journal of the ACM (JACM)* 19 (1972), April, number 2, pages 248–264. – URL https://doi.org/10.1145/321694.321699. – ISSN 0004-5411. DOI: 10.1145/321694.321699

**[Elson et al. 2003]** ELSON, Jeremy ; GIROD, Lewis ; ESTRIN, Deborah: **"Fine-Grained Network Time Synchronization Using Reference Broadcasts"**. In *SIGOPS Oper. Syst. Rev.* 36 (2003), December, number SI, pages 147–163. – ISSN 0163-5980

**[Fan and Pardalos 2010]** FAN, Neng ; PARDALOS, Panos M.: **"Linear and quadratic programming approaches for the general graph partitioning problem"**. In *Journal of Global Optimization* 48 (2010), number 1, pages 57–71

**[Feldmann 2013]** FELDMANN, Andreas E.: **"Fast balanced partitioning is hard even on grids and trees"**. In *Theoretical Computer Science* 485 (2013), may, pages 61–68. – URL http://arxiv.org/abs/1111.6745. – ISSN 03043975. DOI: 10.1016/j.tcs.2013.03.014

**[Fitch and Butler 2008]** FITCH, Robert ; BUTLER, Zack: **"Million Module March: Scalable Locomotion for Large Self-Reconfiguring Robots"**. In *International Journal of Robotics Research* 27 (2008), number 3-4, pages 331–343. – ISSN 02783649. DOI: 10.1177/0278364907085097

**[Fitch et al. 2003]** FITCH, Robert ; BUTLER, Zack ; RUS, Daniela: **"Reconfiguration planning for heterogeneous self-reconfiguring robots"**. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* Volume 3 IEEE (event), 2003, pages 2460–2467

**[Gallager et al. 1983]** GALLAGER, R. G. ; HUMBLET, P. A. ; SPIRA, P. M.: **"A Distributed Algorithm for Minimum-Weight Spanning Trees"**. In *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5 (1983), number 1, pages 66–77. – ISSN 15584593. DOI: 10.1145/357195.357200

**[Garey and Johnson 1979]** GAREY, M. R. ; JOHNSON, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. First Edition. W. H. Freeman, 1979. – URL http://www.amazon.com/Computers-Intractability-NP-Completeness-Mathematical-Sciences/dp/0716710455. – ISBN 0716710455

**[Gilbert et al. 1998]** GILBERT, John R. ; MILLER, Gary L. ; TENG, Shang-Hua: **"Geometric mesh partitioning: Implementation and experiments"**. In *SIAM Journal on Scientific Computing* 19 (1998), number 6, pages 2091–2110

**[Gilpin et al. 2008]** GILPIN, Kyle ; KOTAY, Keith ; RUS, Daniela ; VASILESCU, Iuliu: **"Miche: Modular shape formation by self-disassembly"**. In *The International Journal of Robotics Research* 27 (2008), number 3-4, pages 345–372

**[Gnägi and Baumann 2021]** GNÄGI, Mario ; BAUMANN, Philipp: **"A matheuristic for large-scale capacitated clustering"**. In *Computers & Operations Research* 132 (2021), pages 105304

**[Goldstein and Mowry 2004]** GOLDSTEIN, Seth C. ; MOWRY, Todd C.: **"Claytronics: A scalable basis for future robots"**. (2004), November. – URL http://www.cs.cmu.edu/~claytronics/papers/goldstein-robosphere04.pdf

**[Gusella and Zatti 1989]** GUSELLA, Riccardo ; ZATTI, Stefano: **"The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD"**. In *IEEE transactions on Software Engineering* 15 (1989), number 7, pages 847–853

**[Haeupler et al. 2018]** HAEUPLER, Bernhard ; HERSHKOWITZ, D E. ; WAJC, David: **"Round-and message-optimal distributed graph algorithms"**. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, 2018, pages 119–128

**[Hager et al. 2013]** HAGER, William W. ; PHAN, Dzung T. ; ZHANG, Hongchao: **"An exact algorithm for graph partitioning"**. In *Mathematical Programming* 137 (2013), number 1, pages 531–556

**[Haghighat et al. 2015]** HAGHIGHAT, Bahar ; DROZ, Emmanuel ; MARTINOLI, Alcherio: **"Lily: A miniature floating robotic platform for programmable stochastic self-assembly"**. In *2015 IEEE international conference on robotics and automation (ICRA)* IEEE (event), 2015, pages 1941–1948

**[Hawkes et al. 2010]** HAWKES, Elliot ; AN, B ; BENBERNOU, Nadia M. ; TANAKA, H ; KIM, Sangbae ; DEMAINE, Erik D. ; RUS, D ; WOOD, Robert J.: **"Programmable matter by folding"**. In *Proceedings of the National Academy of Sciences* 107 (2010), number 28, pages 12441–12445

**[Hayes et al. 2003]** HAYES, Adam ; MARTINOLI, A. ; GOODMAN, Rodney: **"Swarm Robotic Odor Localization: Off-Line Optimization and Validation with Real Robots"**. In *Robotica* 21 (2003), 07. DOI: 10.1017/S0263574703004946

**[Hołobut et al. 2016]** HOŁOBUT, Paweł ; CHODKIEWICZ, Paweł ; MACIOS, Anna ; LENGIEWICZ, Jakub: **"Internal localization algorithm based on relative positions for cubic-lattice modular-robotic ensembles"**. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE (event), 2016, pages 3056–3062

**[Hou and Shen 2014]** HOU, Feili ; SHEN, Wei-Min: **"Graph-based optimal reconfiguration planning for self-reconfigurable robots"**. In *Robotics and Autonomous Systems* 62 (2014), number 7, pages 1047–1059

**[Hourany et al. 2022]** HOURANY, Edy ; PIRANDA, Benoit ; MAKHOUL, Abdallah ; BOURGEOIS, Julien ; HABIB, Bachir: **"Detector: Hierarchical Distributed Fault Detection Algorithm for Lattice Based Modular Robots"**. In *International Conference on Advanced Information Networking and Applications* Springer (event), 2022, pages 118–129

**[Hourany et al. 2021]** HOURANY, Edy ; STEPHAN, Christian ; MAKHOUL, Abdallah ; PIRANDA, Benoit ; HABIB, Bachir ; BOURGEOIS, Julien: **"Self-reconfiguration of modular robots using virtual forces"**. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE (event), 2021, pages 6948–6953

**[Huang et al. 2019]** HUANG, Dong ; WANG, Chang-Dong ; WU, Jian-Sheng ; LAI, Jian-Huang ; KWOH, Chee-Keong: **"Ultra-scalable spectral clustering and ensemble clustering"**. In *IEEE Transactions on Knowledge and Data Engineering* 32 (2019), number 6, pages 1212–1226

**[IEEE 2008]** IEEE: **"IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems"**. In *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* (2008), pages 1–300. DOI: 10.1109/IEEESTD.2008.4579760

**[Kawano 2017]** KAWANO, Hiroshi: **"Tunneling-based self-reconfiguration of heterogeneous sliding cube-shaped modular robots in environments with obstacles"**. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pages 825–832. DOI: 10.1109/ICRA.2017.7989100

**[Kawano 2019]** KAWANO, Hiroshi: **"Distributed Tunneling Reconfiguration of Sliding Cubic Modular Robots in Severe Space Requirements"**. In CORRELL, Nikolaus (editors) ; SCHWAGER, Mac (editors) ; OTTE, Michael (editors): *Distributed Autonomous Robotic Systems*. Cham : Springer International Publishing, 2019, pages 1–15. – ISBN 978-3-030-05816-6

**[Kawano 2020]** KAWANO, Hiroshi: **"Distributed tunneling reconfiguration of cubic modular robots without meta-module's disassembling in severe space require-**

**ment"**. In *Robotics and Autonomous Systems* 124 (2020), pages 103369. – URL https://www.sciencedirect.com/science/article/pii/S0921889019301447. – ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2019.103369

**[Kazadi et al. 2004]** KAZADI, Sanza ; CHUNG, M. ; LEE, B. ; CHO, R.: **"On the dynamics of clustering systems"**. In *Robotics and Autonomous Systems* 46 (2004), 01, pages 1–27. DOI: 10.1016/j.robot.2003.10.001

**[Khamis et al. 2015]** KHAMIS, Alaa ; HUSSEIN, Ahmed ; ELMOGY, Ahmed: **"Multi-robot Task Allocation: A Review of the State-of-the-Art"**. In KOUBÂA, Anis (editors) ; DIOS, J.Ramiro Martínez-de (editors): *Cooperative Robots and Sensor Networks 2015* Volume 607. Cham : Springer International Publishing, 2015, pages 31–51. – URL https://doi.org/10.1007/978-3-319-18299-5_2. – ISBN 978-3-319-18299-5. DOI: 10.1007/978-3-319-18299-5_2

**[Kim et al. 2011a]** KIM, Jin ; HWANG, Inwook ; KIM, Yong-Hyuk ; MOON, Byung-Ro: **"Genetic approaches for graph partitioning: a survey"**. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pages 473–480

**[Kim et al. 2011b]** KIM, Jin-Woo ; KIM, Jeong-Hwan ; DEATON, Russell: **"DNA-Linked Nanoparticle Building Blocks for Programmable Matter"**. In *Angewandte Chemie International Edition* 50 (2011), number 39, pages 9185–9190

**[Kirby et al. 2007]** KIRBY, Brian T. ; AKSAK, Burak ; CAMPBELL, Jason D. ; HOBURG, James F. ; MOWRY, Todd C. ; PILLAI, Padmanabhan ; GOLDSTEIN, Seth C.: **"A modular robotic system using magnetic force effectors"**. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* IEEE (event), 2007, pages 2787–2793

**[Kirby et al. 2011]** KIRBY, Brian T. ; ASHLEY-ROLLMAN, Michael ; GOLDSTEIN, Seth C.: **"Blinky Blocks: A Physical Ensemble Programming Platform"**. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA : Association for Computing Machinery, 2011 (CHI EA '11), pages 1111–1116. – URL https://doi.org/10.1145/1979742.1979712. – ISBN 9781450302685. DOI: 10.1145/1979742.1979712

**[Koenig and Howard 2004]** KOENIG, Nathan ; HOWARD, Andrew: **"Design and use paradigms for gazebo, an open-source multi-robot simulator"**. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)* Volume 3 IEEE (event), 2004, pages 2149–2154

**[Kotay and Rus 2000]** KOTAY, K.D. ; RUS, D.L.: **"Algorithms for self-reconfiguring molecule motion planning"**. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)* Volume 3, 2000, pages 2184–2193 vol.3. DOI: 10.1109/IROS.2000.895294

**[Kurokawa et al. 2008]** KUROKAWA, Haruhisa ; TOMITA, Kohji ; KAMIMURA, Akiya ; KOKAJI, Shigeru ; HASUO, Takashi ; MURATA, Satoshi: **"Distributed self-reconfiguration of M-TRAN III modular robotic system"**. In *The International Jour-*

*nal of Robotics Research* 27 (2008), number 3-4, pages 373–386

**[Lengiewicz and Hołobut 2019]** LENGIEWICZ, Jakub ; HOŁOBUT, Paweł: **"Efficient collective shape shifting and locomotion of massively-modular robotic structures"**. In *Autonomous Robots* 43 (2019), February, number 1, pages 97–122. – URL https://doi.org/10.1007/s10514-018-9709-6. – ISSN 15737527. DOI: 10.1007/s10514-018-9709-6

**[Lenzen et al. 2014]** LENZEN, Christoph ; SOMMER, Philipp ; WATTENHOFER, Roger: **"PulseSync: An efficient and scalable clock synchronization protocol"**. In *IEEE/ACM Transactions on Networking* 23 (2014), number 3, pages 717–727

**[Liu et al. 2017]** LIU, Ceyue ; LIU, Jiangong ; MORENO, Rodrigo ; VEENSTRA, Frank ; FAINA, Andres: **"The impact of module morphologies on modular robots"**. In *2017 18th International Conference on Advanced Robotics (ICAR)* IEEE (event), 2017, pages 237–243

**[Liu et al. 2023]** LIU, Chao ; LIN, Qian ; KIM, Hyun ; YIM, Mark: **"SMORES-EP, a modular robot with parallel self-assembly"**. In *Autonomous Robots* 47 (2023), number 2, pages 211–228

**[Liu and Yim 2020]** LIU, Chao ; YIM, Mark: **"Configuration recognition with distributed information for modular robots"**. In *Robotics Research* Volume 10. Springer, 2020, pages 967–983. – URL http://link.springer.com/10.1007/978-3-030-28619-4-65. – ISSN 25111264. DOI: 10.1007/978-3-030-28619-4-65

**[Liu and Han 2018]** LIU, Jialu ; HAN, Jiawei: **"Spectral clustering"**. In *Data Clustering*. Chapman and Hall/CRC, 2018, pages 177–200

**[Liu et al. 2022]** LIU, Yaoyao ; GUO, Ping ; ZENG, Yi: **"MEACCP: A membrane evolutionary algorithm for capacitated clustering problem"**. In *Information Sciences* 591 (2022), pages 319–343. – URL https://www.sciencedirect.com/science/article/pii/S002002552200055X. – ISSN 0020-0255. DOI: https://doi.org/10.1016/j.ins.2022.01.032

**[Lyder et al. 2008]** LYDER, Andreas ; GARCIA, Ricardo Franco M. ; STOY, Kasper: **"Mechanical design of odin, an extendable heterogeneous deformable modular robot"**. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* Ieee (event), 2008, pages 883–888

**[Makhoul and Bassil 2023]** MAKHOUL, Abdallah ; BASSIL, Jad: **"Fault Tolerance Technique Using Bidirectional Hetero-Associative Memory for Self-Reconfigurable Programmable Matter"**. In *2023 International Wireless Communications and Mobile Computing (IWCMC)* IEEE (event), 2023

**[Maróti et al. 2004]** MARÓTI, Miklós ; KUSY, Branislav ; SIMON, Gyula ; LÉDECZI, Akos: **"The flooding time synchronization protocol"**. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA : Association for Computing Machinery, 2004, pages 39–49

**[Martin 2012]** MARTIN, CH: *Spectral clustering: a quick overview*, PhD thesis, PhD Thesis, 2012

**[Mashreghi and King 2021]** MASHREGHI, Ali ; KING, Valerie: **"Broadcast and minimum spanning tree with o (m) messages in the asynchronous CONGEST model"**. In *Distributed Computing* 34 (2021), number 4, pages 283–299

**[Mazdin and Rinner 2021]** MAZDIN, Petra ; RINNER, Bernhard: **"Distributed and Communication-Aware Coalition Formation and Task Assignment in Multi-Robot Systems"**. In *IEEE Access* 9 (2021), pages 35088–35100

**[Meyerhenke et al. 2014]** MEYERHENKE, Henning ; SANDERS, Peter ; SCHULZ, Christian: **"Partitioning complex networks via size-constrained clustering"**. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8504 LNCS (2014), pages 351–363. – ISBN 9783319079585. DOI: 10.1007/978-3-319-07959-2_30

**[Meyerhenke et al. 2017]** MEYERHENKE, Henning ; SANDERS, Peter ; SCHULZ, Christian: **"Parallel graph partitioning for complex networks"**. In *IEEE Transactions on Parallel and Distributed Systems* 28 (2017), number 9, pages 2625–2638

**[Michel 2004]** MICHEL, Olivier: **"Cyberbotics ltd. webots™: professional mobile robot simulation"**. In *International Journal of Advanced Robotic Systems* 1 (2004), number 1, pages 5

**[Mills 1991]** MILLS, D.L.: **"Internet time synchronization: the network time protocol"**. In *IEEE Trans. on Communications* 39 (1991), number 10, pages 1482–1493. DOI: 10.1109/26.103043

**[Moussa et al. 2021]** MOUSSA, Mohamad ; PIRANDA, Benoit ; MAKHOUL, Abdallah ; BOURGEOIS, Julien: **"Cluster-Based Distributed Self-reconfiguration Algorithm for Modular Robots"**. In *Lecture Notes in Networks and Systems* 225 LNNS (2021), pages 332–344. – ISBN 9783030750992. DOI: 10.1007/978-3-030-75100-5_29

**[Mukherjee 2020]** MUKHERJEE, Proshikshya: **"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, 2020, pages 77–85

**[Mulvey and Beck 1984]** MULVEY, John M. ; BECK, Michael P.: **"Solving capacitated clustering problems"**. In *European Journal of Operational Research* 18 (1984), number 3, pages 339–348

**[Nakagaki et al. 2016]** NAKAGAKI, Ken ; DEMENTYEV, Artem ; FOLLMER, Sean ; PARADISO, Joseph A. ; ISHII, Hiroshi: **"ChainFORM: A Linear Integrated Modular Hardware System for Shape Changing Interfaces"**. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. New York, NY, USA : Association for Computing Machinery, 2016 (UIST '16), pages 87–96. – URL https://doi.org/10.1145/2984511.2984587. – ISBN 9781450341899. DOI:

10.1145/2984511.2984587

**[Naz et al. 2015]** NAZ, Andre ; PIRANDA, Benoit ; GOLDSTEIN, Seth C. ; BOURGEOIS, Julien: **"ABC-Center: Approximate-center election in modular robots"**, IEEE, 9 2015, pages 2951–2957. – URL http://ieeexplore.ieee.org/document/7353784/. – ISBN 978-1-4799-9994-1. DOI: 10.1109/IROS.2015.7353784

**[Naz et al. 2016]** NAZ, Andre ; PIRANDA, Benoit ; GOLDSTEIN, Seth C. ; BOURGEOIS, Julien ; NAZ, André ; BENOˆIT PIRANDA, Benoˆıt ; COPEN GOLDSTEIN, Seth: **"A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots"**. URL https://hal.archives-ouvertes.fr/hal-02300456, 2016. – Research Report

**[Naz et al. 2018a]** NAZ, André ; PIRANDA, Benoît ; TUCCI, Thadeu ; COPEN GOLDSTEIN, Seth ; BOURGEOIS, Julien: **"Network characterization of lattice-based modular robots with neighbor-to-neighbor communications"**. In *Distributed Autonomous Robotic Systems: The 13th International Symposium* Springer (event), 2018, pages 415–429

**[Naz et al. 2018b]** NAZ, André ; PIRANDA, Benoît ; BOURGEOIS, Julien ; GOLDSTEIN, Seth C.: **"A time synchronization protocol for large-scale distributed embedded systems with low-precision clocks and neighbor-to-neighbor communications"**. In *Journal of Network and Computer Applications* 105 (2018), 3, pages 123–142. – URL https://linkinghub.elsevier.com/retrieve/pii/S1084804517304253. – ISSN 10848045. DOI: 10.1016/j.jnca.2017.12.018

**[Osipov and Sanders 2010]** OSIPOV, Vitaly ; SANDERS, Peter: **"n-Level Graph Partitioning"**. In *European Symposium on Algorithms* Springer (event), 2010, pages 278–289

**[Østergaard et al. 2006]** ØSTERGAARD, Esben H. ; KASSOW, Kristian ; BECK, Richard ; LUND, Henrik H.: **"Design of the ATRON lattice-based self-reconfigurable robot"**. In *Autonomous Robots* 21 (2006), number 2, pages 165–183

**[Pandurangan et al. 2018]** PANDURANGAN, Gopal ; ROBINSON, Peter ; SCQUIZZATO, Michele ; OTHERS: **"The distributed minimum spanning tree problem"**. In *Bulletin of EATCS* 2 (2018), number 125

**[Parada et al. 2021]** PARADA, Irene ; SACRISTÁN, Vera ; SILVEIRA, Rodrigo I.: **"A new meta-module design for efficient reconfiguration of modular robots"**. In *Autonomous Robots* (2021), mar, pages 1–16. – URL https://doi.org/10.1007/s10514-021-09977-6. – ISSN 15737527. DOI: 10.1007/s10514-021-09977-6

**[Park et al. 2008]** PARK, Michael ; CHITTA, Sachin ; TEICHMAN, Alex ; YIM, Mark: **"Automatic Configuration Recognition Methods in Modular Robots"**. In *The International Journal of Robotics Research* 27 (2008), mar, number 3-4, pages 403–421. – URL http://journals.sagepub.com/doi/10.1177/0278364907089350. – ISSN 0278-3649. DOI: 10.1177/0278364907089350

**[Parrott et al. 2018]** PARROTT, Christopher ; DODD, Tony J. ; GROSS, Roderich: *HyMod:*

*A 3-DOF Hybrid Mobile and Self-Reconfigurable Modular Robot and its Extensions*. pages 401–414. In GROSS, Roderich (editors) ; KOLLING, Andreas (editors) ; BERMAN, Spring (editors) ; FRAZZOLI, Emilio (editors) ; MARTINOLI, Alcherio (editors) ; MATSUNO, Fumitoshi (editors) ; GAUCI, Melvin (editors): *Distributed Autonomous Robotic Systems: The 13th International Symposium*. Cham : Springer International Publishing, 2018. – URL https://doi.org/10.1007/978-3-319-73008-0_28. – ISBN 978-3-319-73008-0. DOI: 10.1007/978-3-319-73008-0_28

**[Pescher et al. 2020]** PESCHER, Florian ; NAPP, Nils ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"GAPCoD: A Generic Assembly Planner by Constrained Disassembly"**. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. Richland, SC : International Foundation for Autonomous Agents and Multiagent Systems, 2020 (AAMAS '20), pages 1028–1036. – ISBN 9781450375184

**[Pietrabissa and Liberati 2019]** PIETRABISSA, A. ; LIBERATI, F.: **"Dynamic distributed clustering in wireless sensor networks via Voronoi tessellation control"**. In *International Journal of Control* 92 (2019), number 5, pages 1001–1014. – URL https://doi.org/10.1080/00207179.2017.1378441. DOI: 10.1080/00207179.2017.1378441

**[Pinciroli et al. 2009]** PINCIROLI, C. ; O'GRADY, R. ; CHRISTENSEN, A. L. ; DORIGO, M.: **"Self-organised recruitment in a heteregeneous swarm"**. In *2009 International Conference on Advanced Robotics*, 2009, pages 1–8

**[Pinciroli et al. 2012]** PINCIROLI, Carlo ; TRIANNI, Vito ; O'GRADY, Rehan ; PINI, Giovanni ; BRUTSCHY, Arne ; BRAMBILLA, Manuele ; MATHEWS, Nithin ; FERRANTE, Eliseo ; DI CARO, Gianni ; DUCATELLE, Frederick ; OTHERS: **"ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems"**. In *Swarm intelligence* 6 (2012), pages 271–295

**[Piranda and Bourgeois 2018]** PIRANDA, Benoit ; BOURGEOIS, Julien: **"Designing a quasi-spherical module for a huge modular robot to create programmable matter"**. In *Autonomous Robots* 42 (2018), dec, number 8, pages 1619–1633. – URL http://link.springer.com/10.1007/s10514-018-9710-0. – ISSN 0929-5593. DOI: 10.1007/s10514-018-9710-0

**[Piranda and Bourgeois 2022]** PIRANDA, Benoît ; BOURGEOIS, Julien: **"Datom: A Deformable Modular Robot for Building Self-reconfigurable Programmable Matter"**. In MATSUNO, Fumitoshi (editors) ; AZUMA, Shun-ichi (editors) ; YAMAMOTO, Masahito (editors): *Distributed Autonomous Robotic Systems*. Cham : Springer International Publishing, 2022, pages 70–81. – ISBN 978-3-030-92790-5

**[Piranda et al. 2021]** PIRANDA, Benoit ; CHODKIEWICZ, Pawel ; HOLOBUT, Pawel ; BORDAS, Stephane ; BOURGEOIS, Julien ; LENGIEWICZ, Jakub: **"Distributed Prediction of Unsafe Reconfiguration Scenarios of Modular Robotic Programmable Matter"**. In *IEEE Transactions on Robotics* 37 (2021), 12, number 6, pages 2226–2233. – URL https://ieeexplore.ieee.org/document/9434946/. – ISSN 1552-3098. DOI:

10.1109/TRO.2021.3074085

**[Piranda et al. 2023]** PIRANDA, Benoît ; LASSABE, Frederic ; BOURGEOIS, Julien: **"DisCo: A Multiagent 3D Coordinate System for Lattice Based Modular Self-Reconfigurable Robots"**. (2023)

**[Predari and Esnard 2016]** PREDARI, Maria ; ESNARD, Aurélien: **"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 (event), 2016, pages 280–287

**[Prorok et al. 2010]** PROROK, Amanda ; CIANCI, Christopher M. ; MARTINOLI, Alcherio: **"Towards optimally efficient field estimation with threshold-based pruning in real robotic sensor networks"**. In *2010 IEEE International Conference on Robotics and Automation* IEEE (event), 2010, pages 5453–5459

**[Pruszko et al. 2021]** PRUSZKO, Laura ; COUTRIX, Céline ; LAURILLAU, Yann ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"Molecular hci: Structuring the cross-disciplinary space of modular shape-changing user interfaces"**. In *Proceedings of the ACM on Human-Computer Interaction* 5 (2021), number EICS, pages 1–33. DOI: 10.1145/3461733

**[Rahimian et al. 2015]** RAHIMIAN, Fatemeh ; PAYBERAH, Amir H. ; GIRDZIJAUSKAS, Sarunas ; JELASITY, Mark ; HARIDI, Seif: **"A distributed algorithm for large-scale graph partitioning"**. In *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10 (2015), number 2, pages 1–24

**[Requicha et al. 1977]** REQUICHA, A.A.G. ; VOELCKER, H.B. ; ROCHESTER. PRODUCTION AUTOMATION PROJECT, University of: *Constructive Solid Geometry*. Production Automation Project, University of Rochester, 1977 (TM (Rochester, PAP)). – URL https://books.google.fr/books?id=hG2lngEACAAJ

**[Rohmer et al. 2013]** ROHMER, Eric ; SINGH, Surya P. ; FREESE, Marc: **"V-REP: A versatile and scalable robot simulation framework"**. In *2013 IEEE/RSJ international conference on intelligent robots and systems* IEEE (event), 2013, pages 1321–1326

**[Rolland et al. 1996]** ROLLAND, Erik ; PIRKUL, Hasan ; GLOVER, Fred: **"Tabu search for graph partitioning"**. In *Annals of operations research* 63 (1996), number 2, pages 209–232

**[Romanishin et al. 2015]** ROMANISHIN, John W. ; GILPIN, Kyle ; CLAICI, Sebastian ; RUS, Daniela: **"3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions"**. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pages 1925–1932. DOI: 10.1109/ICRA.2015.7139450

**[Rus and Vona 2001]** RUS, Daniela ; VONA, Marsette: **"Crystalline robots: Self-reconfiguration with compressible unit modules"**. In *Autonomous Robots* 10 (2001), number 1, pages 107–124

**[Saab et al. 2019]** SAAB, Wael ; RACIOPPO, Peter ; BEN-TZVI, Pinhas: **"A review of**

**coupling mechanism designs for modular reconfigurable robots"**. In *Robotica* 37 (2019), number 2, pages 378–403

**[Salemi et al. 2006]** SALEMI, Behnam ; MOLL, Mark ; SHEN, Wei-min: **"SUPERBOT: A Deployable, Multi-Functional, and Modular Self-Reconfigurable Robotic System"**. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pages 3636–3641. DOI: 10.1109/IROS.2006.281719

**[Schaeffer 2007]** SCHAEFFER, Satu E.: **"Graph clustering"**. In *Computer Science Review* 1 (2007), number 1, pages 27–64. – URL http://www.sciencedirect.com/science/article/pii/S1574013707000020. – ISSN 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2007.05.001

**[Scheuerer and Wendolsky 2006]** SCHEUERER, Stephan ; WENDOLSKY, Rolf: **"A scatter search heuristic for the capacitated clustering problem"**. In *European Journal of Operational Research* 169 (2006), number 2, pages 533–547

**[Shao et al. 2018]** SHAO, Junming ; YANG, Qinli ; ZHANG, Zhong ; LIU, Jinhu ; KRAMER, Stefan: **"Graph Clustering with Local Density-Cut"**. In PEI, Jian (editors) ; MANOLOPOULOS, Yannis (editors) ; SADIQ, Shazia (editors) ; LI, Jianxin (editors): *Database Systems for Advanced Applications*. Cham : Springer International Publishing, 2018, pages 187–202. – ISBN 978-3-319-91452-7

**[Shi et al. 2022]** SHI, Fanrong ; YANG, Simon X. ; TUO, Xianguo ; RAN, Lili ; HUANG, Yuqing: **"A Novel Rapid-Flooding Approach With Real-Time Delay Compensation for Wireless-Sensor Network Time Synchronization"**. In *IEEE Transactions on Cybernetics* 52 (2022), number 3, pages 1415–1428. DOI: 10.1109/TCYB.2020.2987758

**[Shiu et al. 2010]** SHIU, Ming-Chiuan ; FU, Li-Chen ; CHIA, Yen-Jui: **"Graph isomorphism testing method in a self-recognition velcro strap modular robot"**. In *2010 5th IEEE Conference on Industrial Electronics and Applications* IEEE (event), 2010, pages 222–227

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

**[Spröwitz et al. 2010]** SPRÖWITZ, A. ; LAPRADE, P. ; BONARDI, S. ; MAYER, M. ; MOECKEL, R. ; MUDRY, P. A. ; IJSPEERT, A. J.: **"Roombots—Towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules"**. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, October 2010, pages 1126–1132. DOI: 10.1109/IROS.2010.5649504

**[Stoy 2006]** STOY, K.: **"Using cellular automata and gradients to control self-reconfiguration"**. In *Robotics and Autonomous Systems* 54 (2006), 2, pages 135–141. – URL https://linkinghub.elsevier.com/retrieve/pii/S0921889005001521. – ISSN 09218890. DOI: 10.1016/j.robot.2005.09.017

**[Stoy and Nagpal 2007]** STOY, K. ; NAGPAL, R.: **"Self-Reconfiguration Using Di-**

**rected Growth"**. In ALAMI, Rachid (editors) ; CHATILA, Raja (editors) ; ASAMA, Hajime (editors): *Distributed Autonomous Robotic Systems 6*. Tokyo : Springer Japan, 2007, pages 3–12. – URL http://link.springer.com/10.1007/978-4-431-35873-2-1. – ISBN 978-4-431-35873-2. DOI: 10.1007/978-4-431-35873-2-1

**[Thalamy et al. 2019]** THALAMY, Pierre ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"A survey of autonomous self-reconfiguration methods for robot-based programmable matter"**. In *Robotics and Autonomous Systems* 120 (2019), pages 103242. – URL https://doi.org/10.1016/j.robot.2019.07.012. – ISSN 09218890. DOI: 10.1016/j.robot.2019.07.012

**[Thalamy et al. 2020a]** THALAMY, Pierre ; PIRANDA, Benoit ; BOURGEOIS, Julien: **"3D coating self-assembly for modular robotic scaffolds"**. In *IEEE International Conference on Intelligent Robots and Systems* (2020), pages 11688–11695. – ISBN 9781728162126. DOI: 10.1109/IROS45743.2020.9341324

**[Thalamy et al. 2020b]** THALAMY, Pierre ; PIRANDA, Benoît ; LASSABE, Frédéric ; BOURGEOIS, Julien: **"Deterministic scaffold assembly by self-reconfiguring micro-robotic swarms"**. In *Swarm and Evolutionary Computation* 58 (2020), pages 100722. – ISSN 22106502. DOI: 10.1016/j.swevo.2020.100722

**[Thalamy et al. 2021a]** THALAMY, Pierre ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"Engineering efficient and massively parallel 3D self-reconfiguration using sandboxing, scaffolding and coating"**. In *Robotics and Autonomous Systems* 146 (2021), pages 103875. – URL https://www.sciencedirect.com/science/article/pii/S0921889021001603. – ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2021.103875

**[Thalamy et al. 2021b]** THALAMY, Pierre ; PIRANDA, Benoît ; NAZ, André ; BOURGEOIS, Julien: **"VisibleSim: A behavioral simulation framework for lattice modular robots"**. In *Robotics and Autonomous Systems* 147 (2021), pages 103913. – URL https://www.sciencedirect.com/science/article/pii/S0921889021001986. – ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2021.103913

**[Tucci et al. 2017]** TUCCI, Thadeu ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"Efficient scene encoding for programmable matter self-reconfiguration algorithms"**. In *Proceedings of the ACM Symposium on Applied Computing* Part F1280 (2017), pages 256–261. ISBN 9781450344869. DOI: 10.1145/3019612.3019706

**[Tucci et al. 2018]** TUCCI, Thadeu ; PIRANDA, Benoît ; BOURGEOIS, Julien: **"A distributed self-assembly planning algorithm for modular robots: Robotics track"**. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 1 (2018), pages 550–558. – ISBN 9781510868083

**[Vassilvitskii et al. 2002]** VASSILVITSKII, S. ; YIM, M. ; SUH, J.: **"A complete, local and parallel reconfiguration algorithm for cube style modular robots"**. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on* Volume 1,

2002, pages 117–122 vol.1. DOI: 10.1109/ROBOT.2002.1013348

**[Wahby et al. 2019]** WAHBY, Mostafa ; PETZOLD, Julian ; ESCHKE, Catriona ; SCHMICKL, Thomas ; HAMANN, Heiko: **"Collective Change Detection: Adaptivity to Dynamic Swarm Densities and Light Conditions in Robot Swarms"**. In *Artificial Life Conference Proceedings*, 08 2019, DOI: 10.1162/isal_a_00233

**[White et al. 2005]** WHITE, Paul ; ZYKOV, Viktor ; BONGARD, Josh C. ; LIPSON, Hod ; OTHERS: **"Three dimensional stochastic reconfiguration of modular robots."**. In *Robotics: Science and Systems* Citeseer (event), 2005, pages 161–168

**[Xiangning and Yulin 2007]** XIANGNING, Fan ; YULIN, Song: **"Improvement on LEACH protocol of wireless sensor network"**. In *2007 international conference on sensor technologies and applications (SENSORCOMM 2007)* IEEE (event), 2007, pages 260–264

**[Yan et al. 2019]** YAN, Yaowei ; BIAN, Yuchen ; LUO, Dongsheng ; LEE, Dongwon ; ZHANG, Xiang: **"Constrained local graph clustering by colored random walk"**. In *The world wide web conference*, 2019, pages 2137–2146

**[Yildirim and Gürcan 2014]** YILDIRIM, Kasim S. ; GÜRCAN, Önder: **"Efficient time synchronization in a wireless sensor network by adaptive value tracking"**. In *IEEE Transactions on Wireless Communications* 13 (2014), number 7, pages 3650–3664. – ISSN 15361276. DOI: 10.1109/TWC.2014.2316168

**[Yim et al. 2000]** YIM, Mark ; DUFF, David G. ; ROUFAS, Kimon D.: **"PolyBot: a modular reconfigurable robot"**. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)* Volume 1 IEEE (event), 2000, pages 514–520

**[Yim et al. 2007]** YIM, Mark ; SHEN, Wei-Min ; SALEMI, Behnam ; RUS, Daniela ; MOLL, Mark ; LIPSON, Hod ; KLAVINS, Eric ; CHIRIKJIAN, Gregory S.: **"Modular self-reconfigurable robot systems [grand challenges of robotics]"**. In *IEEE Robotics & Automation Magazine* 14 (2007), number 1, pages 43–52

**[Zhang et al. 2014]** ZHANG, Yu ; PARKER, Lynne E. ; KAMBHAMPATI, Subbarao: **"Coalition coordination for tightly coupled multirobot tasks with sensor constraints"**. In *2014 IEEE international conference on robotics and automation (ICRA)* IEEE (event), 2014, pages 1090–1097

**[Zhou et al. 2019]** ZHOU, Qing ; BENLIC, Una ; WU, Qinghua ; HAO, Jin-Kao: **"Heuristic search to the capacitated clustering problem"**. In *European Journal of Operational Research* 273 (2019), number 2, pages 464–487. – URL https://www.sciencedirect.com/science/article/pii/S0377221718307380. – ISSN 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2018.08.043

# List of Figures

# List of Tables

# List of Definitions

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — https://github.com/gallandarakhneorg/tex-templates
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

école doctorale **sciences pour l'ingénieur et microtechniques**

**Title:** Distributed Clustering and Self-Reconfiguration for Modular Robots Based Programmable Matter

**Keywords:** Modular Robots, Programmable Matter, Distributed Algorithms, Clustering, Self-Reconfiguration

**Abstract:**

Programmable matter can be used to create objects that can be programmed to change shape and physical properties on demand. One of the ways to implement programmable matter is to build it as a self-reconfigurable modular robot made up of a large set of simple micro-robots attached to each other. These micro-robots are able to communicate with their directly connected neighboring modules and modify the interconnections to change the overall shape of the robot. This thesis addresses the challenge of self-reconfiguration in modular robotic systems, which involves autonomously rearranging modules to achieve a desired goal shape. The self-reconfiguration problem is difficult due to the very high number of possible configurations, which increases exponentially with the number of modules. In this thesis, we argue that fast self-reconfiguration algorithms for large-scale modular robots can be achieved by clustering the modules. A distributed size-constrained algorithm is presented to form clusters of predefined sizes, along with a novel porous structure composed of two-state meta-modules placed in a regular lattice. This porous structure allows concurrent module flow within it. In addition, two self-reconfiguration algorithms that can be applied on the proposed structure are introduced. The thesis also presents a distributed shape recognition algorithm to detect the current shape of the ensemble. To validate the proposed algorithms, simulations are performed to demonstrate their efficiency in achieving their objectives.

**Titre :** Clustérisation distribuée et auto-reconfiguration pour la matière programmable basée sur des robots modulaires

**Mots-clés :** Robots Modulaires, Matière Programmable, Algorithmes Distribués, Clustering, Auto-Reconfiguration

**Résumé :**

La matière programmable peut être utilisée pour créer des objets qui peuvent être programmés pour changer de forme et de propriétés physiques à la demande. L'une des façons d'implémenter la matière programmable est de la construire comme un robot modulaire auto-reconfigurable composé d'un grand ensemble de micro-robots simples attachés les uns aux autres. Ces micro-robots sont capables de communiquer avec leurs modules voisins directement connectés et de modifier les interconnexions pour changer la forme globale du robot. Cette thèse aborde le défi de l'auto-reconfiguration dans les systèmes robotiques modulaires, qui implique de réorganiser de manière autonome les modules pour atteindre une forme cible. Le problème d'auto-reconfiguration est difficile du fait du nombre très élevé de configurations possibles, qui augmente de façon exponentielle avec le nombre de modules. Dans cette thèse, nous soutenons que des algorithmes d'auto-reconfiguration rapide pour les robots modulaires à grande échelle peuvent être obtenus en regroupant les modules. Un algorithme distribué à contraintes de taille est présenté pour former des clusters de tailles prédéfinies, ainsi qu'une nouvelle structure poreuse composée de méta-modules à deux états placés dans une maille régulière. Cette structure poreuse permet un déplacement simultané des modules à l'intérieur de celle-ci. De plus, deux algorithmes d'auto-reconfiguration applicables sur la structure proposée sont introduits. La thèse présente également un algorithme de reconnaissance de forme distribuée pour détecter la forme actuelle de l'ensemble. Afin de valider les algorithmes proposés, des simulations sont réalisées pour démontrer leur efficacité à atteindre leurs objectifs.

Université Bourgogne-Franche-Comté
32, avenue de l'Observatoire
25000 Besançon, France