Active Subtraction: A Viable Method of Self-Reconfiguration for Modular Robotic Systems

Matthew D. Hall

Supervisors: Dr. Roderich Groß
Dr. Shuhei Miyashita

Department of Automatic Control and Systems Engineering
The University of Sheffield

A Thesis Submitted for the Degree of
Doctor of Philosophy

September 2022

Abstract

Modular robotic systems comprise individual robots, termed *modules*, that physically connect to one another to create a system that is more capable than the sum of the individuals. The way in which the modules connect together alters the attributes of the formed configuration. The ability to change between configurations enables the systems to adapt to match the demands of a task. For example, a system could form a long, narrow shape to navigate a pipe, before being configured to a large, strong shape capable of lifting heavy objects. The problem of autonomously changing between these configurations provides a unique challenge, as modules must coordinate to allow for movement, remain connected, and position themselves correctly. One reliable and robust method of reconfiguration is through subtraction, where modules are removed from an initially connected structure to leave behind the configuration desired. This thesis is concerned with the development and analysis of novel control algorithms to achieve this subtractive reconfiguration without external intervention, instead using modules that are able to remove themselves. The approach is termed *active subtraction*. To enable deployment on a variety of modular robotic systems, a variety of control strategies are developed, that utilise centralised or distributed control and allow modules to move sequentially or in parallel. It is formally proven that the algorithms enable a system to form arbitrary shapes via active subtraction, and maintain the connectedness of the configuration throughout, preventing collapse. The time-based performance of each algorithm is formally characterised, as well as being validated through the simulation of thousands of randomly generated configurations and configurations specifically designed to compare active subtraction to existing reconfiguration solutions. In addition to these algorithmic contributions, multiple existing modular robotic systems are assessed in detail, identifying their suitability for the use of the proposed active subtraction algorithms as a method of self-reconfiguration.

Acknowledgements

I would like to begin by thanking my supervisor, Dr. Roderich Groß. His support and guidance have helped me to become an academic capable of producing an entire thesis, something that, until writing the final words of it, I wasn't sure would be possible. Furthermore, his expertise and work ethic have been inspirational, and I am truly grateful to him for all that he has passed on to me. I would also like to thank my second supervisors, Dr. Tony Dodd and Dr. Shuhei Miyashita, for their academic and personal assistance.

It has been my pleasure and privilege to be a part of the Natural Robotics Lab, and I would like to thank every member, past and present, that I had the honour to work alongside. I would also like to extend a hearty thanks to the members of F01—my colleagues in work, coffee and procrastination—they've all been amazing friends and I couldn't have done it without them. Extra thanks go to Anıl for being a mentor to me throughout my academic pursuits, as well as at the pub.

I want to acknowledge all the friends and family near and far who have helped me along the way; the list of people who have made this possible is too long to fit on the page, so if you are reading this and think that you should have made the cut, I'll buy you a drink instead.

Finally, to Mum and Dad: your encouragement, support, wisdom and love are what keep me going and I wouldn't be where I am—or who I am—if not for you. Thanks for everything.

"I forgot my mantra."

JEFF GOLDBLUM
Annie Hall (1977)

Table of contents

1 Introduction **1**

1.1 Motivation 1

1.2 Problem Definition 4

1.3 Aims and Objectives 4

1.4 Preview of Contributions 5

1.5 Publications 5

1.6 Thesis Overview 6

2 Background and Related Work **9**

2.1 Introduction 9

2.2 Modular Robotic Systems 9

2.3 Control of Modular Robotic Systems 29

2.4 Reconfiguration Methods 32

3 Introducing Active Subtraction **37**

3.1 Introduction 37

3.2 Problem Formulation 38

3.3 System Considerations 40

3.4 Simulated Environment 41

3.5 Conclusion 47

4 Centralised Active Subtraction 49

4.1 Introduction 49

4.2 Controller Design 50

4.3 Mathematical Analysis 52

4.4 Simulation Studies 57

4.5 Conclusion 67

5 Distributed Active Subtraction 69

5.1 Introduction 69

5.2 Excluded Row Removal 70

5.3 Solution Design 73

5.4 Example Priority Order Index Assignment Process 80

5.5 Mathematical Analysis 83

5.6 Simulation Studies 86

5.7 Conclusion 100

6 Active Subtraction on Real World Systems 103

6.1 Introduction 103

6.2 Comparison of Systems 104

6.3 Conclusion 114

7 Conclusions and Future Work 117

7.1 Conclusions 117

7.2 Future Work 119

References 121

Chapter 1

Introduction

1.1 Motivation

Since their inception, robots have been destined to aid, assist and, in some cases, replace humans in many aspects of our lives. As the development of robotic systems progresses, these systems become better and better at outperforming humans for specific tasks; what started as solutions for simple, laborious and repetitive tasks such as assembly in industry, warehouse transportation and mass production, have evolved to replace humans in many other aspects such as precision surgery, personal assistance and the exploration of complex environments. A shortcoming that most robotic systems have, however, is that they are still designed for specific tasks and their performance suffers when deployed to solve a problem that they were not designed for. Furthermore, they often create a single point of failure in a system, where a failure on the part of the robotic system would cause a failure for the entire operation.

Modular robotic systems is a specific vein of robotic systems that aims to negate these shortcomings. Modular robots are groups of robotic units, termed *modules*, that combine their abilities and attributes to become a system that is greater than the sum of its parts. They do this by physically connecting together, and can do so in different ways to form morphologies with varying characteristics. Through the collaboration of these lesser parts, the whole can complete tasks that would be challenging for the individual, much like living cells [3] or a colony of ants [4]. Compared to traditional, specialised robotic systems, modular robots grant a number of benefits. Firstly, they are adaptable systems, due to their reconfigurability. This means that, rather than requiring multiple robots to complete a task, a single modular robotic system can reconfigure to meet the demands of the task. For example, a system that can manufacture a small sports car, before reconfiguring to produce a large SUV, would be of much greater use to a manufacturer than an assembly line that requires replacement for each product. Similarly, a robot exploring unknown environments that can separate into individual modules to fit through small spaces, before reconnecting to combine their strength to lift heavy loads, would be able to get much further than a robot with fixed size and strength. The systems also grant robustness due to the ease with which modules can be replaced; a single interchangeable module being inexpensive and more easily replaced than a specialised robot. Once autonomy is introduced the benefits are increased even further.

As the hardware of modular robotic systems advances [5], the question of further advancing the technology moves towards considerations of how the systems can be made autonomous. For modular systems to become autonomous, it is required that they are able to change their morphology without intervention from a user. This is termed self-reconfiguration and more specifically concerns a connected group of modules changing their physical properties by adding, subtracting or changing the position of individual modules. With a variety of systems comes a variety of solutions, many of which are specific to the system for which they are designed, with no generic solution apparent.

The majority of reconfiguration solutions consist of adding modules to an existing configuration [6–9], known as *additive* reconfiguration, or rearranging them to form the desired shapes [10, 11], termed *morphing* reconfiguration. Comparatively, Gilpin et al. developed a method of achieving modular reconfiguration through disassembly, whereby a group of robots in a given starting formation can detach unnecessary modules to create a desired shape [12]. This is known as *subtractive* reconfiguration, and has been researched considerably less than additive reconfiguration, despite the promise it holds.

Research into subtractive reconfiguration is motivated by a number of potential advantages:

Problem simplification The problem of optimally reconfiguring from an initial, possibly random, configuration to a desired one is computationally intractable [13, 14]. By addressing situations where the desired configuration is a subset of the initial configuration, as with subtractive reconfiguration, a reliable solution with at least relatively good performance, if not optimal, could be produced.

Ease of communication As discussed by Gauci et al. [15], an initially connected structure enables the modules to communicate from initialization. Because of this, vital information can be passed between all members, and potentially processed, before reconfiguration begins.

Increased reliability of module movement Typically, modules require high precision to form connections between themselves, unless connections are already present in the starting configuration, as is the case in a subtractive approach. Although accurate alignment between connectors could still be required, modules that begin connected to neighbours will have reference points with which to align, rather than converging from potentially arbitrary locations, as with additive reconfiguration.

Physical stability There is also the increased potential for guarantees of physical stability as, if the initial configuration is physically stable and each subsequent removal of a module does not violate the stability rules, the resulting configuration will also be physically stable. In the case of the work of Gilpin et al., this amounts to ensuring that the target structure is one connected entity [12]. In the case of two-dimensional reconfiguration on a flat surface [16], the considerations are removed entirely as physical stability does not require the modules to be connected given that they are resting on a stable surface.

Although promising, the subtractive approach from Gilpin et al. requires external intervention to reconfigure, so cannot be considered an autonomous self-reconfiguring solution. Without any locomotion capabilities, the modules rely entirely on external forces to remove from the structure, such as human intervention or falling due to gravity. This approach has limited applicability for real-world use; as previously discussed, a key benefit of modular robotic systems is their flexibility, notably in environments that are unsafe or inaccessible for humans, however, if the system is incapable of autonomous self-reconfiguration then this benefit is negated. Furthermore, when relying on gravity or some other limiting extraction technique to remove modules, the shapes that can be formed are limited, as modules must be able to fall or be reachable from the exterior of the configuration. Finally, if modules remove by falling they could sustain damage, which is especially unsuitable if expensive or delicate modules were to be used.

By utilising modules capable of locomotion, and extending the solution to enable their autonomous removal, these drawbacks could be overcome, as modules could safely remove themselves without the need for intervention. The configurations formed could be complex shapes, with deep corridors, as the modules would be able to navigate to the exterior themselves. The guarantees of physical stability would be harder to maintain, as the connectedness and support would need to be ensured while modules move across the structure. Nevertheless, the concept of applying subtractive reconfiguration to a system capable of autonomous reconfiguration holds potential, and is thus the focus of the studies presented in this thesis.

1.2 Problem Definition

In the aforementioned works of Gilpin et al. [12, 16], reconfiguration through subtraction is introduced, but limited to modules incapable of locomotion, instead relying on external forces to be removed. This style of reconfiguration is therefore called *passive* subtractive reconfiguration. By contrast, this thesis considers a solution to the self-reconfiguration problem that uses actuated modules that are able to actively remove themselves. This is termed *active* subtraction.

The goal of active subtraction is as follows. Given a regular, densely populated, fully connected starting structure comprising modular robots extending vertically from the ground, and a desired structure that is a subset of the former, create a reconfiguration solution by which the modules that are not part of the desired structure remove themselves from the starting structure with no intervention. To be considered removed the modules must move along the boundary of the structure and reach a *sink* location, external to the structure and situated on the ground. The desired structures can be arbitrary, with the stipulation that they contain no hollow spaces, as this would prevent modules from removing themselves [12]. Throughout the removal process, the reconfiguration solution must guarantee that all modules that are not yet removed must remain connected in some way, so as to avoid collapse.

1.3 Aims and Objectives

Given the aforementioned problem statement, the main focus of this thesis is the creation, analysis and iteration of self-reconfiguration algorithms for modular robotic systems. More specifically, utilising the under-researched concept of shape formation via the disassembly of initially connected modules. By applying this concept to modular robotic systems that are capable of autonomously reconfiguring, it will be shown that actively subtracting modules to yield a desired configuration is a valid, and effective, method of self-reconfiguration.

The specific objectives to fulfil this aim are:

- To conduct a state-of-the-art literature review into modular robotic systems, and the control algorithms that are used for their reconfiguration.
- To develop a novel and viable method of self-reconfiguration for modular robotic systems in a two-dimensional environment.
- To create control algorithms for the realisation of such a method.

- To analyse and characterise the performance of the control algorithms, both mathematically and through simulation.
- To iterate new control algorithms to improve overall performance and robustness.
- To assess the viability of the developed solutions on real-world modular robotic systems.

1.4 Preview of Contributions

The following form the contributions presented in this thesis:

- A centralised control strategy utilising a subtractive approach for modular reconfigurable robotic systems, by which extraneous modules actively remove themselves from a vertical, rectangular starting configuration, to leave behind a given structure. Throughout the reconfiguration process, the structures remain connected and stable.
- An expansion to the control strategy, allowing multiple modules to move simultaneously. In this case the centralised controller conducts more complex computation.
- Design of an alternative control strategy for the same problem, allowing for distributed control, that is, without the need for any centralised entity. Here the modules may also move simultaneously.
- Mathematical analysis of all control strategies, proving the structural stability of the solutions at all times and characterising the completion times. The alternative (distributed) control strategy is shown to be asymptotically optimal.
- Simulation studies to further verify the solutions and analyse the completion time for a variety of scenarios.
- Discussions as to how these control strategies could be deployed on a selection of existing real-world modular robotic systems.

1.5 Publications

This thesis comprises original contributions to scientific knowledge made by the author. The work presented herein has so far led to the peer-reviewed paper:

[1] **M. D. Hall**, A. Özdemir and R. Groß. "Self-reconfiguration in two-dimensions via active subtraction with modular robots," in *Robotics: Science and Systems XVI*, RSS Foundation, 2020.

This paper provides the foundation for Chapter 4 and was presented virtually by the author at the first fully online Robotics: Science and Systems Conference (RSS). This presentation is available online [17].

Additionally, the author contributed to a separate project not presented in this thesis, this led to the publication of the following peer-reviewed paper:

[2] A. Özdemir, M. Gauci, A. Kolling, **M. D. Hall**, and R. Groß, "Spatial Coverage Without Computation," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9674–9680, IEEE, 2019.

This work was also presented by the author in Montréal, Canada, at the 2019 IEEE International Conference on Robotics and Automation (ICRA).

1.6 Thesis Overview

The structure of this thesis is as follows:

- Chapter 2 serves as a background for the work contained in this thesis, presenting and discussing related works. Section 2.2 introduces modular robotic systems, outlining a number of design characteristics that must be considered in their creation. Section 2.3 presents methods of control for modular robotic systems, detailing the differences between centralised and distributed control in existing systems. Finally Section 2.4 presents reconfiguration methods for modular robotic systems, specifically highlighting the benefits and drawbacks of additive, morphing and subtractive reconfiguration.

- Chapter 3 introduces the concept of active subtraction as a method of self-reconfiguration for modular robotic systems. An overview of the concept is presented in Section 3.1. Section 3.2 formulates the problem, specifying the environment in which the problem will be solved, and giving the objective for the solutions to fulfil. Considerations about modular robotic systems that can employ active subtraction are presented in Section 3.3. Section 3.4 details the modelled system used in Chapters 4 and 5, as well as the control algorithm that the modules use to locomote around configurations. Section 3.5 concludes the chapter.

- Chapter 4 presents a centralised solution to the problem definition outlined in the preceding chapter. Two centralised control solutions for the reconfiguration problem are then presented in Section 4.2, of which one allows for sequential movement and the other parallel. In Section 4.3, the solutions are formally proved to be correct, and the run time performance is analysed. Following this, Section 4.4 reports the simulation studies carried out using the solutions, assessing the performance over a range of criteria, and discussing the results in depth. Consequently, Section 4.5 concludes the chapter. This chapter is based on the author's original work, given in [1].

- Chapter 5 presents a novel, distributed solution to the problem. Section 5.1 outlines the shortcomings of the solution from the previous chapter, and details ways in which a distributed solution would overcome these. Details of this distributed solution are given in Section 5.3, presenting the multiple phases of the control algorithm. Section 5.5 contains formal proofs for the correctness of the new solution, as well as analysis of the run time performance. To verify the performance, numerous simulation studies were conducted, the results of which are reported in Section 5.6, including user-defined scenarios and randomly generated ones, alongside discussion of the results. Section 5.7 then concludes the chapter.

- Chapter 6 presents considerations on how the solutions of Chapters 4 and 5 could be applied to existing real world systems. In Section 6.1, the relevant considerations are outlined before an overview table is presented in Section 6.2, illustrating the suitability of various modular robotic systems with regards to self-reconfiguration via active subtraction, alongside detailed analysis of the potential each system holds. Section 6.3 summarises this comparison and concludes the chapter.

- Chapter 7 summarises and concludes the thesis. In Section 7.1 the contributions are discussed, before possible directions for the future of the work are presented in Section 7.2.

Chapter 2

Background and Related Work

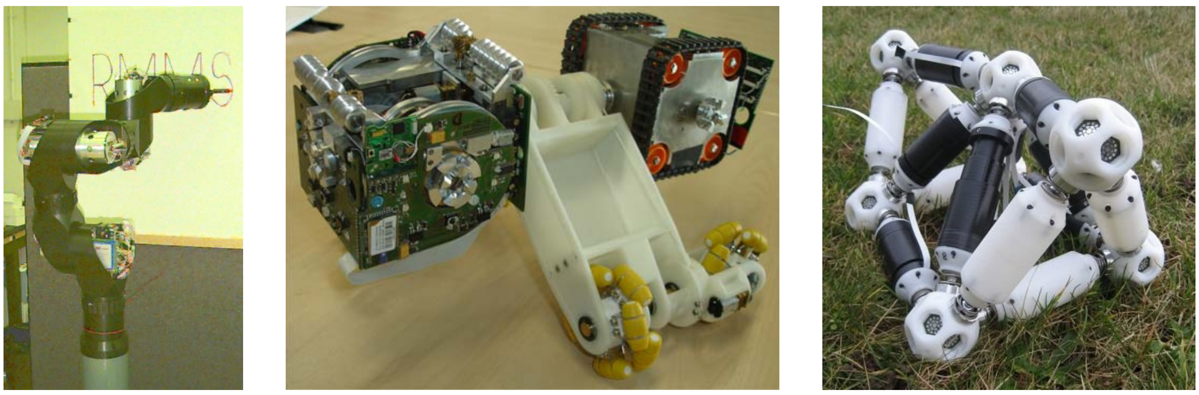
2.1 Introduction

This chapter consists of three sections. The first, Section 2.2, details the history of modular robotic systems by explaining various aspects of design that must be considered during their development. Examples of systems that demonstrate each facet of design are given, and the advantages and disadvantages of each are explored. This gives context for the more specific work involved in this thesis, that is, the development of control algorithms for the reconfiguration of such systems. Section 2.3 outlines the types of control algorithms that exist for modular robotic systems, comparing the advantages and disadvantages of each. Section 2.4 introduces existing reconfiguration methods for modular robotic systems, detailing the overarching concepts behind different approaches, as well as some specifics as to how these approaches are implemented.

2.2 Modular Robotic Systems

Modular robotic systems are collections of robots that combine their abilities and attributes to become a system that is greater than the sum of its parts. Through the collaboration of these lesser parts the whole can complete tasks impossible for the individual, much like a colony of ants, or a swarm of bees [4].

Since the concept of a modular robotic system was first presented by Fukuda et al. as a way to overcome the shortcomings of existing robotic platforms [3], the research and development of such systems now spans decades. In that time, many modular robotic systems have been



(a) (b) (c)

Fig. 2.1 Examples of heterogeneous modular robotic systems; (a) *RMMS* joint and link modules connected to form a manipulator arm © 1996 IEEE, (b) Backbone, Active Wheel and Scout modules from the *Symbrion* and *Replicator* projects connected as a single configuration © 2010 IEEE, (c) *Odin* system composed of joint and link modules © 2008 IEEE. Reprinted from [18–20], respectively.

developed, each with the goal of advancing the field in some way by introducing a novel aspect of design or implementation. Included in these design aspects are the methods of locomotion, the ways in which modules connect to one another, the structures formed when they are connected and many task specific considerations. The following is a brief overview of some existing modular robotic systems and their approaches to these facets of design.

2.2.1 Modular System Type

Modular robotic systems fall into two categories in terms of the type of modular system they can be. The first is *heterogeneous*, where modules are varied and have specific capabilities. In this case, the system will not function at its full potential without a combination of the different modules. The second is *homogeneous*, where all modules are the same, and the system can function with any number of modules.

Heterogeneous

Heterogeneous modular robotic systems are commonplace in manufacturing applications, where the robot retains a certain level of specialism, whilst being adaptable within that limitation. One such example of heterogeneous modular robotic systems in manufacturing is the concept of modular manipulators. These systems are heterogeneous as they generally consist of a base, a number of hinges to grant the manipulator degrees of freedom (DoFs), either modules of their own or comprising joint and link modules, and an end effector to grasp. One such example is the *CMU RMMS* by Schmitz et al. [21], seen in Figure 2.1(a). The system requires joints and links to be connected in order to form a fully operational manipulator. These joints are either pivot joints or rotational joints, and can be combined to add DoFs in many planes. Although not the first modular manipulator, it does introduce the concept of reconfigurability by adding connectors to the end of each module that allow for easy connection and disconnection. Similar systems have since been developed such as *TOMMS* by Matsumaru [22] and *ModMan* by Yun et al. [23].

Much focus has also been paid to researching control methods for the manipulators, as the kinematics of such systems become much more complex as more DoFs are introduced. The fact that the systems are modular and reconfigurable also leads to some interesting challenges as the system should be able to autonomously react to changes in the configuration. The connection methods also introduce some level of misalignment as it is impossible to create a completely seamless connection. Even if small, this error would be propagated through the system and magnified by the length of the manipulator. Because of these factors, much research is involved in developing robust kinematic and control schemes [24–26].

A disadvantage of heterogeneous systems is that specific modules are required for specific roles, which could hamper deployment when the task or environment is not fully known. However, specialised modules do have the benefit of being able to better perform their designated task, as well as allowing for smaller or simpler modules, as was discovered by Kernbach et al. in the development of the *Replicator* and *Symbrion* projects [19, 27, 28]. The initial design for the project was a homogeneous swarm of small reconfigurable robots, but with increased functional requirements the decision was made to develop a heterogeneous system, so that the different modules could be more capable when performing a specific function without the excessive complexity that would be required for each module to perform all functions. The resulting system comprises Backbone, Active Wheel and Scout types of modules, each of which can be seen in Figure 2.1(b). The Backbone module features four connection points, and has limited manoeuvrability due to a unique screw drive mechanism. It also has a strong hinge joint in the middle, enabling it to hold other modules in desired positions, hence the name Backbone. The Active Wheel module is equipped with omnidirectional wheels to allow motion in all directions, two connection points, and a hinge in the centre to lift away from the ground. The Scout module is similar in size to the Backbone, but features tracks around the edge for easier movement across challenging terrain. The module also has sensors to extract information from the environment, and four connectors, one of which can be moved on a hinge which is less strong than the Backbone, meaning that it cannot lift other modules. Although envisioned as a homogeneous system, the transition to heterogeneity has enabled Kernbach et al. to develop the system in a way they deem most effective.

A heterogeneous modular system that was designed to be that way from the start is *EDHMoR* by Faíña et al. [29]. The system comprises a small number of primary active modules, capable of various motions, that can be combined with specialised modules such as manipulators or sensors. Similarly, systems such as *SMART* by Baca et al. [30] and *Mom's Friend* by Ahn et al. [31] are heterogeneous modular robotic systems that make use of task specific modules.

Conversely, a heterogeneous modular robotic system where the core functions require differing modules is *Odin* by Lyder et al. [20], shown in Figure 2.1(c). This system is composed of spherical ball-joint modules with many connection points, and link modules which connect between two ball-joint modules and are capable of extending and retracting. Without both types of modules present in a configuration, the system cannot function. This type of configuration is referred to as *truss* type, and is further discussed in Section 2.2.2. *SABER* by Romanov et al. [32] is not a truss type system, but requires both module types to operate. It is composed of modular track pieces and a platform that can locomote along the track. The track modules are able to join in a continuous wheel to allow omni-wheeled movement in free space, or flatten to create a structured space for the platform to traverse.

These heterogeneous systems all benefit from the advanced capabilities afforded by specialist modules; however, the benefits from Section 1.1 are minimised somewhat. Due to the fact that multiple types of modules are involved, the ease of production is reduced, as is the repairability. The redundancy of the system also suffers, as modules which fail must be replaced by identical modules. These shortcomings can be removed by using identical modules, as in homogeneous systems.

Homogeneous

Homogeneous systems are more commonplace in modular robotics as they more fully realise the benefits outlined in Section 1.1. The identical modules are also popular in research as they afford more generalised control schemes and can be more easily applied to other research areas, such as swarm behaviours [33]. The work contained within this thesis is concerned with homogeneous systems. As such, the remainder of systems discussed in this chapter are examples of homogeneous systems, unless otherwise stated.

2.2.2 Configuration of Modules

Although there are many ways of classifying various modular robotic systems, the most common is by discussing the type of structure the modules create when connected. This structure is known as a *configuration*. The characteristics granted by the type of configuration are very important as it could determine what tasks a system is suited for.

The types of configurations that modular robotic systems can create form a growing list of categories [5, 34–36]. One of the most common is a *chain* configuration, where the modules connect end to end, sometimes capable of branching, but with no fixed positions within a space. Another common architecture is *lattice* based, where the robots are restricted to a regular lattice configuration. More recently however, a configuration type known as *hybrid* has been introduced, whereby the modules are able to connect together as either a chain or a lattice, or both, depending on the configuration requirements. There also exists the *mobile* robot system, where the modules are capable of moving freely in a space before connecting together, as well as the *truss* type, consisting of joints and links, and *free-form* systems that defy characterisation into any of these categories.

Chain

A characteristic of chain type modules is that they have few DoFs individually, but when connected are capable of complex macro-locomotion, such as caterpillar or rolling wheel, as discussed in Section 2.2.5. They are capable of forming linear or branching structures so can also form legged walkers for macro-locomotion. These structures grant the system high levels of versatility, as the number of DoFs can be increased with each module added. While the kinematics are well understood, the control can become complex with an increase in DoFs, and errors in angle control can propagate through the system to greater detriment, as was the case with the previously discussed modular manipulators.

Examples of this type of robot include *Polypod* by Yim [37] which comprises cubic modules with two connection faces and two compressible faces, allowing the modules to expand, contract and change angle, depicted in Figure 2.2(a). Yim demonstrates the capabilities of the system by forming configurations comprised of many *Polypod* modules. These range from simple open-ended chains to connected loops to complex walkers.

Also in this category are systems such as *CEBOT* by Fukuda et al. [3], *CONRO* by Castano et al. [38], the successor to the *Polypod*, the *PolyBot*, also developed by Yim et al. [39] and

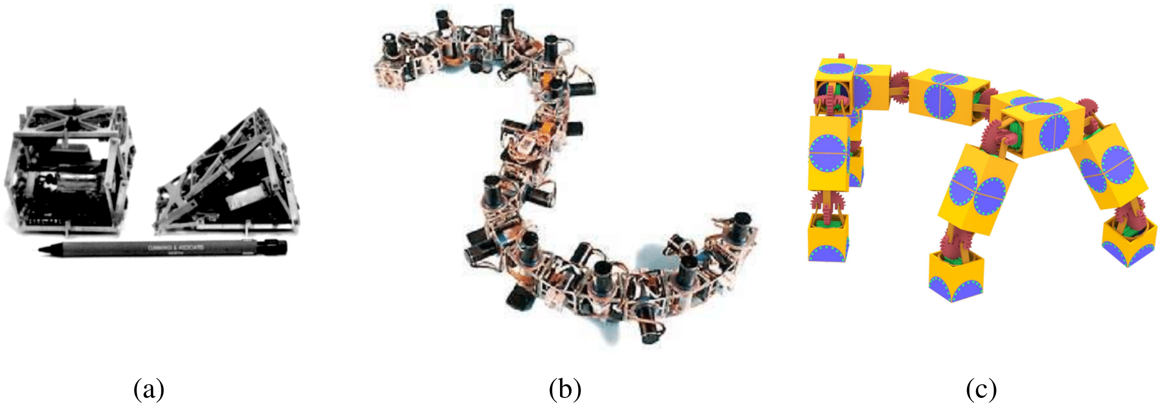


Fig. 2.2 Examples of modular robot systems that form chain configurations; (a) *Polypod* modules, one fully extended and one retracted on one side, (b) *Polybot* modules connected in a chain © 2002 IEEE, (c) simulated *2DxoPod* modules mimicking a vertebrate[1]. Reprinted from [37, 42, 41], respectively.

shown in Figure 2.2(b), *KAIRO 3* by Pfotzer et al. [40] and *2DxoPod* by Sankhar Reddy CH. et al. [41], seen in Figure 2.2(c).

Lattice

Lattice modules are those which form regular patterns when connected together. They can be further separated into sub-categories in a number of ways, such as the number of dimensions in which the lattice extends. There are those that form two-dimensional lattices, such as *Fractum* by Murata et al. [43] which forms a hexagonal lattice, the *Crystalline* system by Rus and Vona [44] which is a compressible module forming a square or rectangular lattice depending on the state and can be seen in Figure 2.3(a), or *Robot Pebbles* by Gilpin et al. [16] which form a square lattice and can be seen in Figure 2.6(a). There also exists *CHOBIE II* by Koseki et al. [45] which forms a square lattice, but vertically, or the *Distributed Flight Array* by Oung et al. [46], which forms a hexagonal lattice, but of modules equipped with propellers for flight, shown in Figure 2.6(f). Similarly, the *MHP* by Doyle et al. [47] forms a square lattice, but on the surface of water, shown in Figure 2.3(b).

There are also modules capable of forming three-dimensional lattices, such as the *3-D Unit* by Murata et al. [49], *Miche* by Gilpin et al. [12] as seen in Figure 2.3(c), *3D M-Blocks* by

[1]Reprinted by permission from Springer Nature: Springer Journal of Intelligent & Robotic Systems, “2DxoPod – a modular robot for mimicking locomotion in vertebrates,” S. Sankhar Reddy CH, Abhimanyu, R. Godiyal, T. Zodage and T. Rane, © 2021.

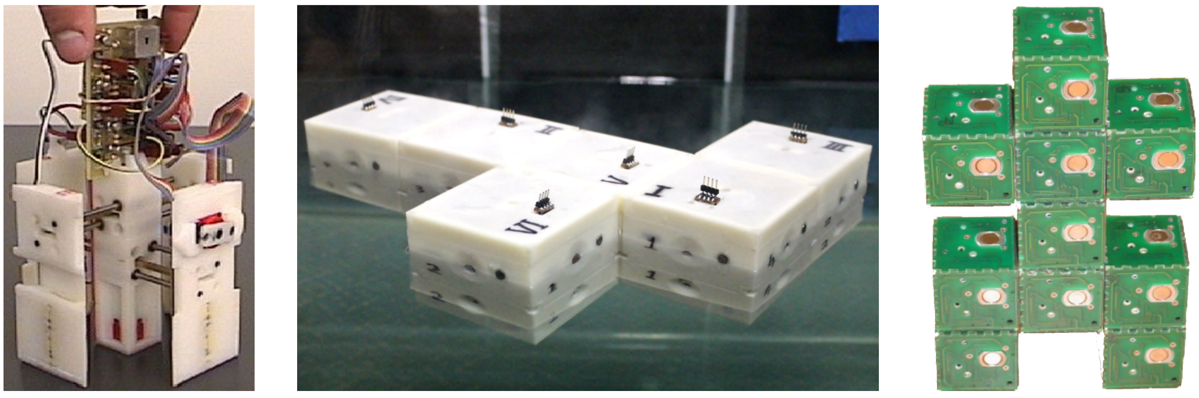

(a) (b) (c)

Fig. 2.3 Examples of modular robot systems that form lattice configurations; (a) a *Crystalline* module © 2000 IEEE, (b) *MHP* modules connected in a square lattice formation on the surface of a body of water © 2016 IEEE, (c) *Miche* modules connected vertically in a square lattice to form a humanoid structure © 2008 Sage Publications. Reprinted from [48, 47, 12], respectively.

Romanishin et al. [50], *Soldercubes* by Neubert and Lipson [51] and *3D Catoms* by Piranda et al. [52].

A shortcoming of lattice configurations is the fact that modules in densely populated lattices are restricted by their neighbours. However, by creating modules that have spherical footprints within cubic lattices (or circular footprints in square lattices for two-dimensions), some modules are capable of rotation or movement within a dense lattice. This is achieved simply by conceiving spherical modules such as *FreeBOT* by Liang et al. [53], or quasi-spherical modules, such as *3D Catoms* by Piranda et al. [52], although the modules are not fully spherical, so some limitations still apply. Another method, conceived by Parrott et al. with the *HyMod* system [54], seen in Figure 2.4(c), is to create connectors that retract when not in use, allowing for in-place rotation, although the *HyMod* system is capable of forming chain structures too, so is considered a hybrid system.

Another method of classifying lattice robots is to characterise the way in which the modules move within the lattice [55]. These categories include translational lattice, where modules slide around neighbours to move, demonstrated by *CHOBIE II* [45], rotational lattice, where modules rotate around a point to alter their position in the lattice, such as *ATRON* [56], seen in Figure 2.4(b), morphable lattice, where the physical characteristics of the modules are altered to allow movement, as with the *Crystalline* [44] system of Figure 2.3(a), and fixed lattice, where modules are incapable of changing position within the lattice, like *Miche* [12] and *Robot Pebbles* [16] seen in Figures 2.3(c) and 2.6(a), respectively.

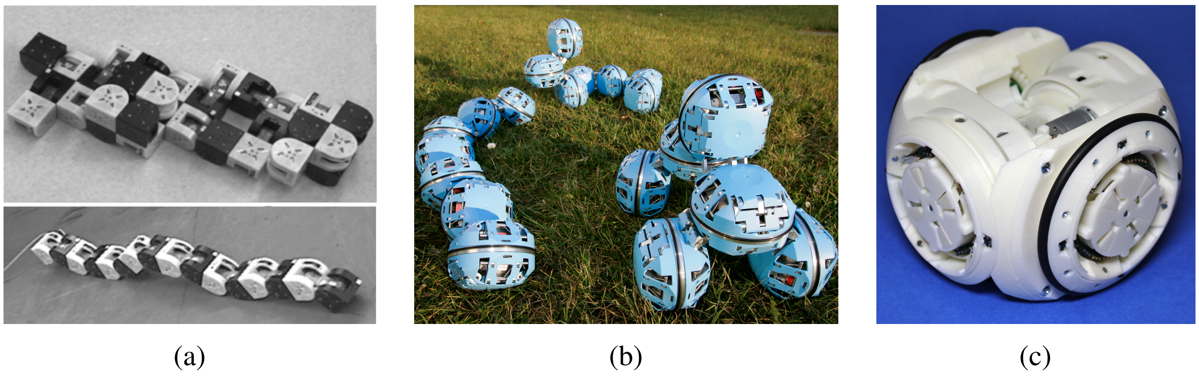


Fig. 2.4 Examples of modular robot systems that form both chain and lattice configurations, known as hybrid systems; (a) *M-Tran III* modules demonstrating lattice and chain configurations © 2008 Sage Publications, (b) *ATRON* modules connected in chain and lattice configurations[2], (c) a *HyMod* module. Reprinted from [10, 56, 54], respectively.

Hybrid

Modules that are capable of existing in both a lattice or a chain architecture are termed hybrid modules and combine the benefits of each architecture [34]. Due to these combined benefits and as technology has progressed, the capabilities of individual modules has improved to the point where hybrid modules are among the most prevalent type of module.

A common method of achieving such behaviour is to create a module that works in effect like a meta-module, where two identical (or similar) halves are connected in the centre by a joint or hinge. An early example of this is the *M-TRAN* module by Murata et al. [57], which features two cubic ends, one with active connectors and one passive, that are connected in the centre by a link, around which each module can rotate 180°. This was iterated upon with *M-TRAN II* by Kurokawa et al. [58]. This version features magnetic connections and a smaller form factor, but is still capable of hybrid configuration formation, as demonstrated by the authors. A further version, *M-TRAN III* was also created by Kurokawa et al. [10], this time reverting to the mechanical connections of *M-TRAN*, while maintaining the smaller form factor of *M-TRAN II*. Examples of *M-TRAN III* modules in lattice and chain formations can be seen in Figure 2.4(a). This approach to hybrid systems inspired similar systems, such as *SuperBot* by Salemi et al. [59], and *Roombots* by Sproewitz et al. [60], which can be seen in Figures 2.6(b) and 2.6(d), respectively.

Other hybrid systems comprise what could be considered one half of the *M-TRAN* modules, and require connections in order to enable micro-locomotion within the lattice, as discussed

[2]Reprinted by permission from Springer Nature: Springer Autonomous Robots, “Design of the ATRON lattice-based self-reconfigurable robot,” E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund, © 2006.

in Section 2.2.4. Examples include *ATRON* by Østergaard et al. [56], seen in Figure 2.4(b), *Molecubes* by Zykov et al. [61], which more closely resemble half of a Roombots module and can be seen in Figure 2.5(e), *SMORES* by Davey et al. [62] and *SMORES-EP* by Jing et al. [63], depicted in Figure 2.5(d) and 2.6(c), *M*^{3} and its successor, *M*^{3} *Express* by Kutzer and Wolfe et al. [64, 65], *UBot* by Bie et al. [66], seen in Figure 2.7(f), as well as the *HyMod* system by Parrott et al. [54], a module of which is shown in Figure 2.4(c).

Truss

Truss modular robots consist of links, which are usually the part of the system that morphs, and joints, making them heterogeneous systems, as discussed in Section 2.2.1. An example of a truss type system with morphable links is the aforementioned *Odin* by Lyder et al. [20], shown in Figure 2.1(c). The links are telescopic bars and the joints are spherical in nature with twelve female connections for the links to be manually inserted in to. The *Tetrobot* by Hamlin and Sanderson is another example of a truss based robot which comprises active and passive links, and concentric multi-link spherical joints to join them [67]. Spinos et al. more recently introduced the Variable Topology Truss [68]. The system advances the technology of truss modular robots by enabling autonomous self-reconfiguration to alter the topology of the structure.

Free-Form

The term free-form is given to modular robots that deny categorisation by any other structure, for example, programmable matter that could theoretically take any form. This is shown with the original *Catoms* modules, as part of the *Claytronics* system. The modules are cylindrical robots of diameter 44 mm, equipped with twenty-four electromagnets around the perimeter to connect to, and to move around, neighbouring modules [69]. Another example is the *Slimebot* by Shimizu et al., which are cylindrical robots equipped with Velcro connectors on all sides that are equipped with telescopic arms. The robots also have anchors so that while one robot extends its arm it can push the other away whilst anchored to the surface, before releasing the anchor and retracting the arm to move towards the moved robot [70]. Liang et al. take the concept even further with *FreeBOT*, a fully spherical module with an internal magnet capable of being positioned at any point to allow freedom of connectivity between modules [53]. The internals also include motorised rubber wheels, which drive the spheres as well as alter the position of the internal magnet.

2.2.3 Connection Methods

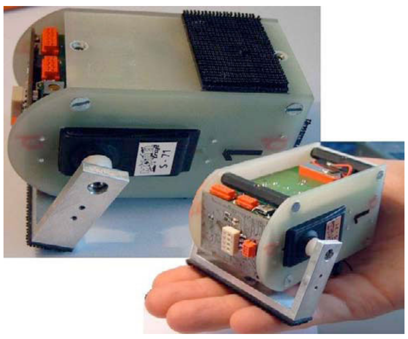
The method by which modules connect to one another is an important design consideration, as it is one of the defining features of reconfigurable modular robots. Multiple methods have been conceived of and implemented, each with advantages and disadvantages associated with them.

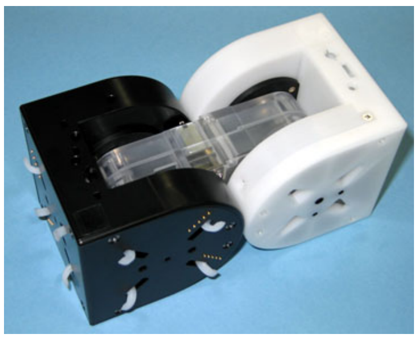
Mechanical

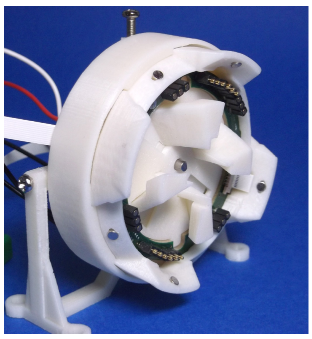
One such method is through mechanical means, the most rudimentary of which is to provide holes or fasteners that require human intervention to actuate the docking and undocking. This is demonstrated on the *YaMoR* system by Moeckel et al. [71] which uses Velcro-like connections, as seen in Figure 2.5(a), but which was later replaced with screw and pin connections that still required manual connection [72]. *GZ-I* by Zhang et al. [73] and *iMobot* by Ryland and Chang [74] also require manual connections, as well as *Odin*, which requires a human intervention to insert the links into the joints and create the desired configuration, an example of which is seen in Figure 2.1(c) [20].

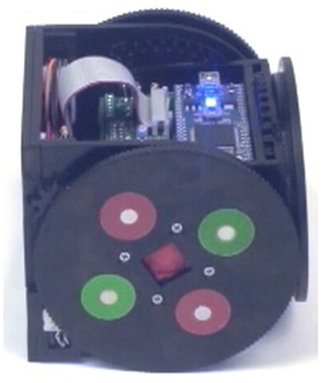
Another mechanical solution is the use of pins, with a latch to grip once inserted into the neighbouring module. However, this requires male and female components, reducing the rotational symmetry of the modules. This method has been demonstrated with the *CONRO* modules [75], seen in Figure 2.7(a). More simply, *M-TRAN III* [10] modules are equipped with extending pins, but which grip onto the connecting face themselves when actuated, meaning the female connector is passive. The extended pins can be seen on the black half of the module in Figure 2.5(b), with the receiving holes of the female half of the connector on the white side. Similar connection methods are employed with *Roombots* [60] shown in Figure 2.6(d). This type of connection, needing a male and female connector for docking, is termed gendered connection.

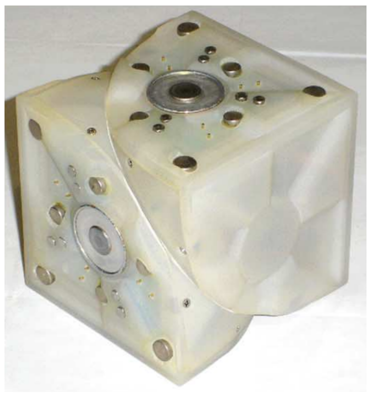
Similarly, *SuperBot*, in Figure 2.6(b), uses a connector developed by Shen at al. called *SINGO* [76], which consists of sliding jaws to establish a connection between modules through mechanical methods. This improves upon the connectors from *CONRO* and *M-TRAN III* by removing the need to have male and female connectors, instead allowing all connectors to both provide and receive connections. As such, this type of connection is known as genderless. *HyMod* modules are equipped with custom mechanical connectors termed *HiGen* connectors, which are also genderless [77]. *HiGen* connectors also allow for single-sided disconnection, that is, only one connector needs to retract for the modules to no longer be connected. This attribute is particularly useful when considering how robust a system is, as it means that a module which experiences a failure can be removed from the system, even if it cannot actuate its connectors.


(a)


(b)


(c)


(d)


(e)

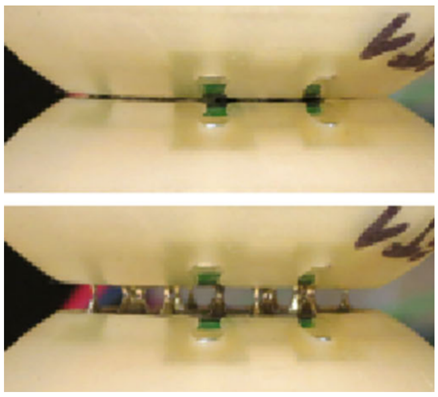

(f)

Fig. 2.5 Examples of connectors utilised by modular robotic systems; (a) *YaMoR* modules, which require connecting manually via the Velcro-like strips © 2005 IEEE, (b) an *M-TRAN III* module with the connection pins extended, (c) an extended *HiGen* connector, four of which are used on a *HyMod* module © 2014 IEEE, (d) a *SMORES* module, equipped with four permanent magnets and a slot in the centre of each face used to misalign the magnets for disconnection © 2012 IEEE (the electromagnetic version, *SMORES-EP*, can be seen in Figure 2.6(c)), (e) an original *Molecube* module with electromagnets to aid connection © 2007 IEEE, (f) two connected *Soldercubes*, shown fully connected and pulled slightly apart as the solder is heated[3]. Reprinted from [78, 10, 77, 62, 79, 51], respectively.

Parrott posits that this is the true definition of a genderless connector, and connectors that are identical, but require both connectors to actuate in order to form or break connections are bi-gendered connectors. An extended *HiGen* connector can be seen in Figure 2.5(c).

One aspect of the pin-connection mechanisms that can be improved upon is how precise the alignment needs to be in order to form a connection. This is addressed with the *s-bots*, which use mechanical jaws to grip to a designated rail around other modules [80], as can be

[3]Reprinted by permission from Springer Nature: Springer Autonomous Robots, "Soldercubes: A self-soldering self-reconfiguring modular robot system," J. Neubert and H. Lipson, © 2015.

seen in Figure 2.7(e). The connection can be initiated at any point around the system by simply attaching the jaw of one module to the rail of another.

Permanent Magnets

Permanent magnets are a reliable means of connecting modules as it avoids the need for any complex electronics or potential failures of the magnet mechanism. However, it does require that a specialised disconnection strategy is employed. The *M-TRAN* and *M-TRAN II* modules make use of a shape memory alloy to push apart from one another and disconnect the magnetic connections [57, 58]. The *Fracta* system uses permanent magnets to initiate a connection, and a switch electro-magnet to separate when desired [43]. *M-Blocks* and *3D M-Blocks*, shown in Figure 2.6(e), also rely on magnetic connection for forming lattices, overcoming the attractive force with the force generated by the inertial motor discussed in Section 2.2.4 [50, 81]. Another solution, employed by *SMORES*, is to use a key to extend from the centre of one module face into the face of another, holding it in place while the module rotates, misaligning the magnets and causing a repelling force [62]. A *SMORES* module can be seen in Figure 2.5(d) Although effective, it has since been replaced with electromagnets [63].

Electromagnets

Electromagnets allow fine control of connections, and do not require any moving parts to operate. Zykov et al. use them as connection mechanisms on the original *Molecubes* [79], shown in Figure 2.5(e). In later versions, however, they move away from focussing on reconfiguration and replace the magnets with passive connectors [61]. The *Miche* and *Robot Pebbles* modules, of Figures 2.3(c) and 2.6(a), respectively, also make use of electromagnets, using the electric current to switch the polarity of the magnets rather than turn them on or off, thus reducing the power necessary to maintain connection [12, 16]. As mentioned, *SMORES* were also iterated upon, changing to using electromagnets in *SMORES-EP* by Jing et al. [63], and can be seen in Figure 2.6(c). This allows the connectors to be simpler, meaning that the modules themselves can be more complex whilst retaining a similar footprint.

Other Connection Methods

A unique method of connection is employed by the *Soldercube* modules, where small areas of solder are heated and can be used to connect to neighbouring modules, seen in Figure 2.5(f).

Not only does this provide enough structural support to lift up to ten modules, but also allows for the passing of signals and power [51].

Another solution is demonstrated with the *3D Catoms* modules, where the modules are sufficiently small that electrostatic forces are able to provide connective forces [82].

2.2.4 Micro-Locomotion of Modules

When designing a modular robotic system, an important aspect to consider is the movement of a single module, termed micro-locomotion [83]. This characteristic becomes especially important when the system is desired to be self-reconfigurable, as it allows for the modules to position themselves in a configuration, as well as add or remove themselves from it, as detailed in Section 2.4.

Externally Propelled

The simplest method of micro-locomotion is to have no internal locomotion at all and rely on external forces to move the modules. The aforementioned *Robot Pebble* [16], is a one centimetre cube, with the ability to connect magnetically to a neighbouring module but has no self-locomotion capabilities. The authors propose that the modules reside on a vibrating table in order to move together and apart stochastically, attaching to or detaching from neighbouring modules if required. Figure 2.6(a) shows a variety of Tetris inspired shapes formed by *Robot Pebble* modules. This lack of propulsion is much like its predecessor, *Miche* [12], seen in Figure 2.3(c), which also has no locomotion capabilities but is capable of connections in three-dimensions and relies on gravity to disassemble. Whilst being a novel and simplistic method of movement, it limits the usefulness of the system, and restricts the reconfiguration of the systems, as modules can only be removed if the external forces allow. Another method of moving cubes without locomotion capabilities could be to use magnets, as shown by Bhattacharjee et al. [84]. They demonstrate simple magnetic cubes, similar to *Robot Pebbles*, being controlled on a magnetic table, locomoting and forming various configurations.

Other systems that rely on external forces for movement include the *Stochastic systems* by White et al. [88], the *Programmable Parts* system by Bishop et al. [89], and the *evo-bots* by Escalera et al. [90] all of which use an air-table to enable movement, and further air currents or

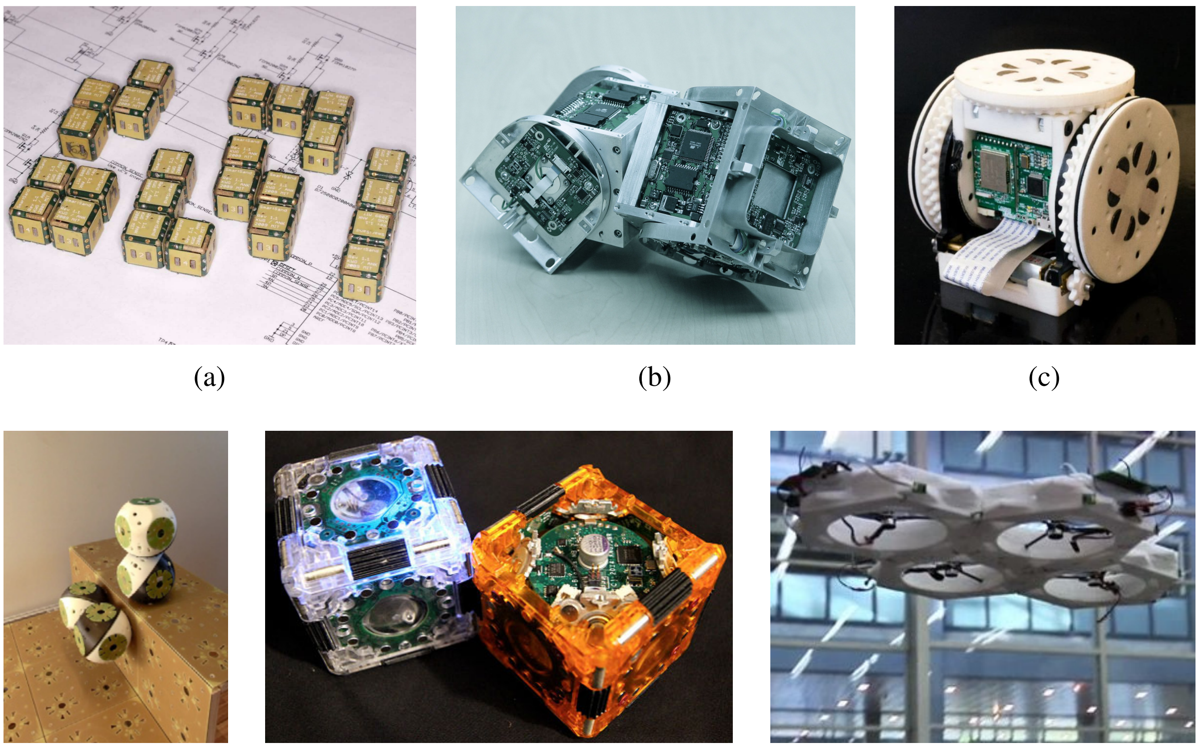


Fig. 2.6 Examples of modular robotic systems that utilise a variety of methods for micro-locomotion; (a) *Robot Pebbles* modules which are incapable of micro-locomotion © 2011 IEEE, (b) a *SuperBot* module, capable of using hinges to slide across a surface in the same manner as an inchworm[4], (c) a *SMORES-EP* module, equipped with differential drive wheels, (d) *Roombots* modules locomoting across a structured space[5], (e) *M-Blocks* modules, which utilise a spinning flywheel to move freely © 2015 IEEE, (f) the *Distributed Flight Array* comprising multiple propeller based modules in order to take flight © 2009 IEEE. Reprinted from [85, 86, 63, 87, 50, 46], respectively.

excitation of the platform to propel the modules. There also exists the *X-bots* system developed by White and Yim [91], which uses an actuated stage to move free modules around a fixed module before attaching to one another. These systems, while demonstrating simplistic modules and novel solutions to locomotion, still suffer from a limited usefulness when translating to real-world use cases, as the lack of predictability makes any planning a difficult and time-consuming task. However, with vast numbers of modules, statistical planing can create a viable

[4]Reprinted by permission from Springer Nature: Springer Towards Autonomous Robotic Systems, "Dynamic power sharing for self-reconfigurable modular robots," C.-A. Chen, A. Kamikura, L. Barrios and W.-M. Shen, © 2014.

[5]Reprinted from Robotics and Autonomous Systems, Vol. 62, No. 7, A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi and A. J. Ijspeert, "Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot," 1016–1033, © 2014, with permission from Elsevier.

control strategy, as shown by Tolley et al. [92] and Neubert et al. [93] and further discussed in Section 2.4.1.

Inchworm

More commonly the movement of modules is enabled through an inchworm method, whereby the modules have at least one DoF and can bend in their centre, pulling the rear towards the front, before straightening again to move forward slightly. Examples of this method include all iterations of the *M-TRAN* robots, by Murata et al. [57], seen in Figure 2.4(a), the *SuperBot* by Salemi et al. [59] which can be seen in Figure 2.6(b), the *four degrees of freedom* system by Chu et al. [94], *Transmote* by Qiao et al. [95], the *Xmobot* by Wang et al. [96], and *ModRED* by Baca et al. [97]. This method of micro-locomotion allows the modules to retain symmetry as no extra parts are needed to enable the movement, unlike wheeled locomotion. However, the range of movement is limited to back and forth, and can be hard to control depending on the environment and surface.

Wheeled Micro-Locomotion

Also commonly used, with increasing popularity, are wheeled modules with differential drive [83]. This is seen in systems such as *Swarm-Bots* by Dorigo et al. [80], the aforementioned *M^3* and, *M^3 Express* [64, 65], *SMORES-EP* [62], depicted in Figure 2.6(c), *KAIRO 3* [40] and in the *HyMod* system [54], shown in Figure 2.4(c). Through the use of differential drive, the modules are easily controlled and can move freely through an unstructured environment. They do, however, forgo a level of symmetry and simplicity as the wheels must extend from the module to allow ground clearance, must be able to freely rotate and contain the necessary motors and drivers for the added capability. These shortcomings can be integrated into the design, as with *HyMod*, where the rotation of the wheel also rotates the connector, adding extra DoFs when forming connections.

The *iMobot*, by Ryland and Cheng [74], makes use of wheeled ends for differential drive, and a joint in the middle to employ the inchworm method. The wheels are rounded squares, so do not protrude from the module unless rotating, meaning that the robot can move in tight spaces in any direction, as well as being able to lift itself.

Structured Space

Another method of micro-locomotion is the use of a structured space, as shown by Sproewitz et al. with the *Roombots* system [60]. Although the modules are capable of movement through rotation, without being affixed at one point, they do not locomote. With a grid of connectors the *Roombots* modules can rotate to move to adjacent spaces, connecting at the new location and disconnecting from the old one, even moving up vertical surfaces, as seen in Figure 2.6(d). This method of locomotion lends itself well to precision tasks as it is always known exactly where a module is, and also enables fast and reliable communication methods. This has been demonstrated with the *HyMod* system, utilising the concept of meta-modules [98] to traverse a grid of fixed connectors in a simulated setting [55].

The concept of utilising a structured space to aid in micro-locomotion can be extended to consider the use of static modules as a structure over which to traverse. By using the movements that are possible with *Roombots* or *M-TRAN*, or meta-modules of robots with central joints such as *SMORES* or *HyMod*, a connected structure of modules could be traversed, similar to a structured space. This was demonstrated with the ATRON system, the individual modules of which are incapable of micro-locomotion, but three modules combined as meta-modules are demonstrated in simulation traversing a configuration [99]. Similarly, but on a module by module basis, Leal-Naranjo et al. present a method of moving across a structure of modules by using connectors on hinges that can extend out from the cubic body of each module to facilitate movement [100]. This micro-locomotion is particularly useful for the reconfiguration of a structure of modules, and further discussed in Section 2.4.

Other Methods of Note

A novel micro-locomotion scheme is employed by Romanishin et al. with the *M-Blocks* [81] and *3D M-Blocks* [50]. By spinning a weight very fast in the centre of the module before applying rapid braking, the *M-Blocks* transfer the momentum to the chassis of the module and flip to move across a surface or over one another, overcoming the magnetic attraction keeping them connected. This solution is unique and allows for movement across both an unstructured environment and other *M-Blocks*. It does, however, require that the modules be simple cubes, as can be seen in Figure 2.6(e), limiting the usefulness in applications other than structural.

A system concerned with locomotion in a different space is the *Distributed Flight Array* [46], shown in Figure 2.6(f). Although the system aims to create a flying modular robot, individual modules are equipped with only a single propeller, and as such are not stable enough for micro-locomotion in the air. For this reason each module has three small omni-wheels on the bottom, to enable micro-locomotion on a surface, before docking with other modules and then flying.

Another form of locomotion in a different space is the previously discussed *MHP* [47], from Figure 2.3(b). The modules exist on the surface of water and route fluid through themselves to locomote across the surface of the water. Doyle et al. go on to propose an iteration of the system, called *MFP*, that would be capable of routing fluids in three-dimensions, suitable for use underwater or in zero gravity environments [101].

There are many more modular systems that lack individual locomotion capabilities, but when connected together can produce methods of movement not possible alone, as detailed in the following section.

2.2.5 Macro-Locomotion of Configurations

The other locomotion consideration to make in the field of modular robotic systems, is that of how a configuration of robots will move once the modules are connected together, termed macro-locomotion. It is possible that the combination of movements produced by the modules creates a locomotion gait, but also possible that locomotion is achieved through the reconfiguration of the modules.

Caterpillar

A common method of macro-locomotion, especially for chain type robots, is that of a caterpillar motion. The modules need only have one DoF and, when properly sequenced, a chain of modules can lift the front to perform a sinusoidal movement to locomote. This solution has been employed in many robotic systems, such as the aforementioned *M-TRAN I, II & III* robots [58] which can be seen in Figure 2.4(a), *SuperBot* [102], *Transmote* [95], *ModRED* [97] and *MSR* by Chu et al. [94]. As well as other systems not capable of micro-locomotion, such as *PolyBot* [39] and *CONRO* [103], shown in Figures 2.2(b) and 2.7(a), respectively.

This is only one method of movement for a chain or hybrid type configuration, allowing for movement over uneven terrain at a slow pace. From a simple straight line, it has been hypothesised that it would be a simple task to autonomously join the ends together to make a rolling-loop for more rapid locomotion [104]; however, this proved more difficult than first expected [35] with much work devoted to finding a reliable solution [105, 106]. Many of the aforementioned systems have demonstrated macro-locomotion as a rolling loop, as evidenced by Yim et al. with *PolyBot* in Figure 2.7(b), but without the self-reconfiguration between the two. This concept is further discussed in Section 2.4.2.

Walking

Another frequently demonstrated method of macro-locomotion is through walking. Some systems are able to be configured with varying numbers of legs that afford locomotion on uneven terrain and in various directions for effective exploration. This has been demonstrated with the previously discussed *CONRO* [103], *M-TRAN* [58], shown as a walker in Figure 2.7(c), *2DxoPod* [41] and *Roombots* [107]. As well as other systems such as *Y1*, developed by Gonzalez-Gomez with the walker movement proved by Ranganath et al. [108] and *CoSMO* by Liedke et al. [109]. This has also been shown with the truss based *Tetrobot* [67]. Furthermore, certain heterogeneous systems have developed specific leg modules in order to enable walking locomotion, such as the aforementioned *SMART* system [30], which can be seen in Figure 2.7(d).

Wheeled Macro-Locomotion

For the wheeled modules capable of complex micro-locomotion, it is possible to continue to use the wheels when connected together, provided the configuration is such that there is not an excess of friction perpendicular to the desired direction of travel. This problem is negated by *s-bots* through their ability to align their wheels in the direction of travel [80], demonstrated in Figure 2.7(e). Wheeled macro-locomotion has been shown with *KAIRO 3* [40], and *SMORES* modules [63, 110]. Although not individually wheeled modules, the *ATRON* system has been shown in a configuration that can make use of the rotational DoF in the centre of the module to act as wheels [56], shown in the final configuration of Figure 2.4(b).

A number of heterogeneous systems make use of wheeled modules to enable macro-locomotion. The *CEBOT* [3] consists of binding joint modules, rotation joint modules and wheeled mobile modules, which can combine to form different structures but primarily use the wheeled modules to locomote. This technique is also employed in the system developed for the *Symbrion* and *Replicator* project, which can be seen in Figure 2.1(b), using the previously described Active Wheel module for macro-locomotion when connected, but with each module capable of micro-locomotion too [19].

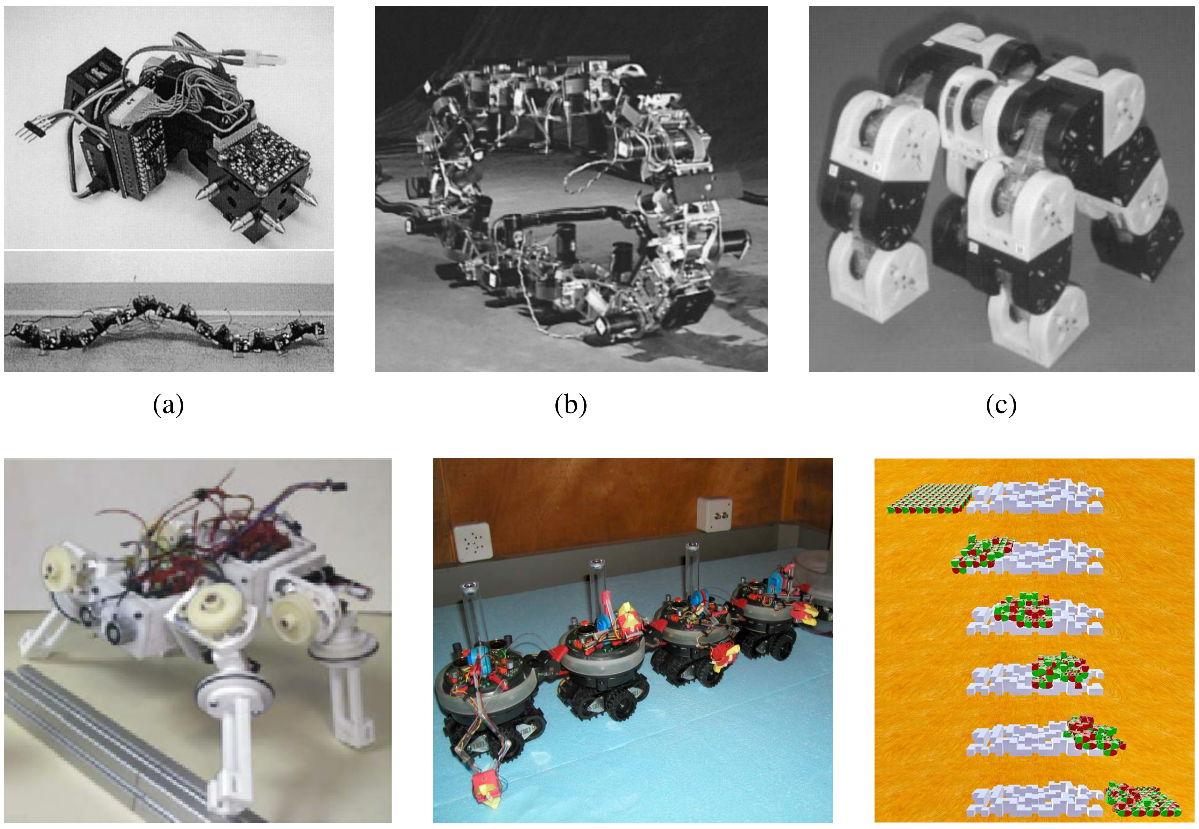


Fig. 2.7 Examples of modular robot systems that utilise a variety of methods for macro-locomotion; (a) a single *CONRO* module and a chain configuration of *CONRO* modules locomoting via a caterpillar motion © 2002 IEEE, (b) *PolyBot* modules connected together to form a rolling loop © 2002 IEEE, (c) *M-TRAN III* modules walking as a quadruped configuration © 2008 Sage Publications, (d) *SMART* system modules configured as a 4-legged walker[6], (e) connected *s-bots* using their ability of aligning their wheels to move as a connected configuration[7], (f) simulated *UBot* modules using macro-locomotion via reconfiguration to navigate rough terrain[8]. Reprinted from [103, 104, 10, 30, 80, 111], respectively.

[6]Reprinted from Robotics and Autonomous Systems, Vol. 60, No. 4, J. Baca, M. Ferre and R. Aracil, "A heterogeneous modular robotic design for fast response to a diversity of tasks," 522–531, © 2012, with permission from Elsevier.

[7]Reprinted by permission from Springer Nature: Springer 1st International Workshop on Swarm Robotics, "The SWARM-BOTS project," M. Dorigo, E. Tuci, R. Groß, V. Triani, T. H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano and L. M. Gambardella, © 2005.

[8]Reprinted by permission from Springer Nature: Springer Journal of Intelligent and Robotic Systems, Vol. 79, "A simplified approach to realize cellular automata for ubot modular self-reconfigurable robots," Y. Zhu, D. Bie, S. Iqbal, X. Wang, Y. Gao and J. Zhao, © 2014.

Macro-Locomotion through Reconfiguration

For those lattice or hybrid based modules not capable of micro-locomotion, one method of locomotion is via reconfiguration. Through moving the modules from a rear location in the configuration to the front, the overall position of the system is changed. This shifts the purpose of reconfiguration from task based to locomotion based, requiring a different mindset to consider the control of the system. There are a number of two-dimensional systems that have shown such macro-locomotion, such as the *Fractum* [43], the self organising robots in the virtual plane by Hosakawa et al. [112], the shape memory alloy based micro-robot by Yoshida at al. [113], and *CHOBIE II* by Koseki et al. [45]. This has also been demonstrated in three-dimensions by Murata with *M-TRAN* [57], Butler et al. with the heterogeneous *Molecule* robot [98], Bonardi et al. with *Roombots* [87] and Bie et al. with *UBots* [66], simulated modules of which are shown locomoting via reconfiguration in Figure 2.7(f). The movement of a module across the surface of a configuration has also been demonstrated in simulation with the *HyMod* system [55], although the full extent of locomotion via reconfiguration is yet to be analysed.

The previously mentioned *Odin* system, consisting of trusses and joints and depicted in Figure 2.1(c), also locomotes through reconfiguration [20]. However, it is through a changing of the physical structure of the modules rather than a reordering of the modules. By extending or retracting the beams, the system changes its shape and can move. This altering of the physical attributes of each module as a method of locomotion has been employed by other authors too. In two-dimensions there are examples such as the *Metamorphic System* by Chirikjian [114], where the hexagonal modules can deform at the vertices, and *Crystalline* [44], where the modules can expand outwards to make themselves larger in a lattice. In three-dimensions this has also been shown by Suh et al. with the *Telecubes* system [115], employing a similar technique to *Crystalline* but on all six faces of the cube.

There are also a number of heterogeneous systems that use reconfiguration to locomote. Most commonly they have a passive type of module and an active type, with the active type performing the reconfiguration of the passive modules, thereby moving the overall system. This is demonstrated by Ünsal et al. with the *I(CES)-Cubes* system [116], which comprises cubic passive modules and active link modules that can move these, as well as by Hjelle and Lipson with their hinge robot that travels along trusses and can reconfigure them [117].

2.3 Control of Modular Robotic Systems

The control of modular robotic systems presents a number of challenges when compared to the control of traditional robotic systems. Firstly, the systems comprise multiple entities, sometimes in the thousands. Secondly, each entity interacts with and can affect upon the other entities in the system. Furthermore, the individual modules could have varying levels of computational capabilities ranging from comprehensive, to no internal processing at all. Because of these factors, control schemes for modular robotic systems often depend heavily upon the specific platform for which they are designed.

There are two approaches when it comes to the design of control schemes for modular robotic systems: *centralised control*, whereby a single control unit sends instructions to each member of the system for them to follow, and *distributed control*, in which each module is responsible for its own actions given its own knowledge.

Each approach offers a number of advantages and disadvantages. As such, there are existing examples of both, a selection of which are detailed as follows.

2.3.1 Centralised Control

As previously outlined, the defining characteristic of centralised control is a single, central control unit. In terms of modular robotic systems, this could be a leader module within the system, or an external computer that can communicate with the modules.

One type of modular robotic system that primarily relies on an external computer for centralised control is the aforementioned modular manipulators. For these systems, it makes no sense to utilise a leader module within the system as the systems tend to be static, so can be easily connected to an external controller. Furthermore, the levels of computation required to calculate the kinematics of the structure can be very complex with many degrees of freedom, so a designated computer that has more computational power than a module can accommodate is required, as discussed by Matsumaru in the design of the previously discussed *TOMMS* system [22].

Another aspect of modular robotic system control that is often centralised regards the macro-locomotion of a configuration. As explored in Section 2.2.5, much collaboration is required between modules to locomote an entire system, something that a central controller is more suited to. A common way to achieve this is by using a gait control table [37]. A gait control table comprises pre-defined movements for each module in a system that, when combined, produce the desired locomotion. The central controller is able to synchronously send instructions from the table to each module to do so. While being a reliable method of locomoting through pre-determined means, the movements are static, and the systems unable to adapt. Yim demonstrates this approach with the *Polypod* modules, moving via caterpillar and rolling methods [37]. The concept is exploited by Jing et al. too, but for the reconfiguration of *SMORES-EP* modules between pre-defined configurations, rather than locomotion [63].

A further concept that is utilised in centralised control is that of hierarchical planning. Due to the aforementioned problem of optimal reconfiguration planning being a computationally intractable one [13], certain heuristics can be used, of which hierarchical planning is one example. Despite being solvable, the solutions are still computationally demanding, so are used in centralised control. The concept is that a high level planner initially creates a movement plan by ignoring physical constraints of the system, before a low level planner uses these plans to generate the detailed actions of each module. Ünsal et al. propose and develop such a solution for the previously detailed *I(CES)-Cubes* system [118, 119], while Jing et al. create something similar for simulated *SMORES-EP* modules [63]. Kotay and Rus go on to adapt the hierarchical planner to account for meta-module level micro-locomotion [120], as discussed in Section 2.2.4. While effective for creating a plan on a centralised controller, it limits the granularity of the movement within the system.

In general, the central controller in a system is limited by computational power, so using a leader module is favoured in heterogeneous systems, or for systems where there is no great need for computational complexity. This is demonstrated by Zykov et al. with the previously introduced iteration of the *Molecubes* system [61]. Here a controller module is developed, which uses local communication to deduce the morphology of the system, before using an evolutionary controller to produce movement for the system.

While effective for controlling systems, including computationally expensive control, centralised control suffers from the fact that the single controller compromises the robustness of the system. The failure of the controller, or disconnection from it, results in the failure of the entire modular robotic system.

2.3.2 Distributed Control

Distributed control requires no central leader, instead having each module in the system run identical algorithms to create emergent behaviour. This approach means that there is no single point of failure in the system; if one module fails, the others continue to operate as they previously had done. The modules also operate based on local information only, not requiring knowledge of the entire system. This means that the system can be dynamically scalable, as the behaviour of each module is not defined by how many modules are present in the configuration. As such, distributed control has been employed in a number of scenarios.

A major inspiration for distributed control comes from nature, where swarms of living organisms are able to collaborate to perform tasks without any centralised control [4]. This can be achieved through the use of hormone signalling. Members of the swarm react to the hormones they detect from neighbours and in turn release their own hormones to signal what they and their neighbours should do. Similarly, modules in a robotic system can pass messages to neighbours to create emergent behaviours. This has been illustrated in simulation many times, including in two-dimensions by Walter et al. [121], and subsequently in three-dimensions by Yim et al. [122]. This has also been demonstrated on hardware by Yoshida et al. with the *Fractum* modules [123], Shen et al. with the previously mentioned *CONRO* modules [124], Kernbach et al. with the *Replicator* and *Symbrion* project [27], Escalera et al. with *evo-bots* [90] and Gilpin et al. with the *Miche* modules [12], among many others. Although an effective way to enable distributed control, each module must be aware of all possible hormones, or messages, it could receive, and the steps that it must perform in response.

Similarly, Støy and Nagpal use hormone inspired control to create a method of additive reconfiguration [6]. Here, rather than the messages being used to directly dictate how the modules react, the strength of a message is instead used to influence where in a configuration a module moves towards. The authors do not consider how the modules are able to move or connect to one another, rather focussing on the high level control of the system via simulation.

Piranda and Bourgeois propose a distributed method of morphing reconfiguration in two-dimensions [125]. In this work, they demonstrate shapes being formed in two-dimensions on a flat surface, by conceptual modules capable of movement by sliding across neighbours. The modules reconfigure from an initial shape to a goal shape, where these shapes are next to one another in the environment, and at least one module overlaps them. The shapes are limited to those that do not contain holes, and do not contain spaces around the edge that are only wide enough for one module to fit into. The algorithm consists of the modules in the outer layer of modules in the initial shape, moving from their starting positions to a position that is inside the goal shape. This is achieved by forming trains of modules that move together along the periphery of the shape. A variation of this is presented by Naz et al. [126]. Here, the rules for the shapes are the same, but the authors investigate shapes the vertical plane, using the cylindrical *Catoms* modules as the basis of the work. Trains of modules are not used, instead modules maintain a gap between moving neighbours, so as to avoid collisions. It is required that the modules know both their location within the configuration, and the goal shape, a requirement that is unrealistic for the *Catoms* modules in practice.

2.4 Reconfiguration Methods

In order for a modular robotic system to form a variety of shapes, a reconfiguration strategy must be created. The majority of strategies are additive, where modules are added to locations around a configuration to form the new configuration. These modules are ones that were not yet involved in the configuration. Another reconfiguration strategy is to not alter the number of modules in a configuration, but to relocate modules from positions in the configuration where they are no longer needed to positions where no module currently resides but is required. This method is termed morphing reconfiguration. The final type of reconfiguration strategy is subtractive reconfiguration, where connections are broken to remove modules from the initial configuration, leaving behind the desired shape. A number of instances of each strategy are outlined below.

2.4.1 Additive Reconfiguration

Autonomous additive reconfiguration is concerned with a shape being built up from an initial seed module, to which connections are made to form the desired shape. This may occur with a configuration of multiple modules that is already constructed and only requires one connection to be made, or from a single seed module with many more modules building around it. Many systems are based on the concept of additive reconfiguration, though few have managed to demonstrate it in practice.

Groß et al. demonstrate this method with the aforementioned *s-bots* [127], as shown in Figure 2.7(e). In this work the *s-bots* are placed in a bounded environment, with a single module acting as a seed. By illuminating red LEDs around its circumference, it attracts other modules to connect. When these modules connect to the seed, they also illuminate red LEDs, attracting more robots to themselves, expanding the configuration. The design of the connection method helps with the success of the strategy, as it is not required that the *s-bots* specifically or accurately align to connect to one another. The wheeled micro-locomotion is also a factor to the success as it means that modules can independently integrate themselves into the final structure. Another demonstration of autonomous additive reconfiguration is shown by Bishop et al. [89] with the aforementioned *Programmable Parts*. In simulation, Sproewitz et al. have shown the *Roombots* modules performing a similar method of additive reconfiguration, utilising meta-modules to enable the movement and testing four different strategies to determine the order of reconfiguration [7].

As previously mentioned, Støy and Nagpal use hormone inspired message passing control to create a gradient of message values in order to direct modules towards a seed module [6]. The work here is accomplished in simulation, but provides a solid example of how simple control could lead to effective additive reconfiguration solutions. However, the authors do not consider the low level movement of the modules, or how they would be able to connect to one another.

A simple solution for this problem is presented by Neubert et al. with an iteration of the *Soldercubes* modules [93]. In this work the modules reside within a fluid filled environment. The authors present a concept whereby a powered seed module initiates a configuration, but rather than modules locomoting to specifically construct a configuration, they move stochastically due to fluid flow, only connecting with the seed if they come into contact with a face that requires connection to a module, and are perfectly aligned. The seed module is the only member of the configuration capable of computation, so is responsible for accepting or declining each potential connection. The concept is demonstrated with simplified models of modules by Tolley et al. [92]. Escalera et al. present a similar concept with the *evo-bots*, using the fluid flow of air currents for stochastic movement on an air table [90]. Here though, modules make connections where possible, and will disconnect from one another if the types of modules that connect together are not the ones desired.

Baca et al. demonstrate a simple additive reconfiguration scheme with *ModRED* [97]. In simulation they show varying number of modules congregating together and connecting to form chains of different lengths of modules.

Additive reconfiguration is also demonstrated by Romanishin et al. with the *M-Blocks* and *3D M-Blocks* [50, 81], modules of which are shown in Figure 2.6(e). As the connection mechanism is simply magnetic, the modules are able to easily connect to, and travel across, configurations of other modules. The authors are not specifically concerned with reconfiguration in the works cited, instead demonstrating the movement of the modules. They show modules moving towards one another autonomously, based on light detection, which is additive reconfiguration, as well as across one another in a pre-defined way, which would be morphing reconfiguration if used for the purposes of reconfiguration.

In an effort to allow systems to form large and complex structures, Thalamy et al. propose the use of a porous scaffold structure, around which the details of the final structure can be added [8]. Here they use a large three-dimensional scaffold tile, comprising *3D Catoms* modules. The authors use simulations to illustrate the formation of initial supporting structures consisting of these scaffold tiles, before further *3D Catoms* modules locomote along the scaffold to form the final configurations. They demonstrate that the approach enables reconfiguration in $O(N^{\frac{2}{3}})$ time, but require excess modules in the formation of these supporting scaffolds.

2.4.2 Morphing Reconfiguration

Morphing reconfiguration can be considered the act of reconfiguration without changing the number of modules in a configuration. For chain based systems this could simply be the configuration altering the position of its modules in continuous space. However, for lattice based systems, modules are required to alter their position within the lattice, either by movement across the structure, or by being moved by another module in the configuration.

Kurokawa et al. demonstrate morphing reconfiguration as a method of locomotion, using *M-TRAN III* modules to create configurations [10]. From these configurations, groups of *M-TRAN III* modules at the rear end of the configuration locomote across the structure to the front, thereby reconfiguring the structure through morphing. The process is shown in progress in Figure 2.4(a). This approach is utilised for reconfiguration in other works too, as discussed in Section 2.2.5.

In an effort to simplify the problem of generalised reconfiguration, Parada et al. introduce a new meta-module concept [128]. Here they present a relatively large meta-module consisting of either twelve modules with hinges connecting two halves, such as *M-TRAN* or *Roombots*, or twenty-four modules with integrated hinges, such as *Molecubes* or *HyMod* modules. By using meta-modules constructed to the specification given, the systems would be able to expand and contract each meta-module in each direction, and could use this motion to move meta-modules around the structure.

An example of morphing reconfiguration concerned with fewer connections is shown by Yim et al. with the *PolyBots* [104]. The work is concerned with reconfiguring a chain of *PolyBots* from a simple straight line into a closed loop, as shown in Figure 2.7(b). Due to the complications associated with aligning two ends of a chain in a three-dimensional space, the experiments reduce the problem to a single plane by lying the configurations flat on a surface. They also reduce one end to a single module and the other to an arm of six modules, both attached to a fixed surface. The authors outline a complex control strategy using a three-phase docking process. This is required as, for a connection between *PolyBots* to be made they must accurately align, highlighting a major stumbling block associated with traditional reconfiguration methods. A similar example is shown by Castano et al. making use of *CONRO* modules to reconfigure from a snake like configuration to a ring [38], and again by Qiao et al. with the *Transmote* system [95].

2.4.3 Subtractive Reconfiguration

Subtractive reconfiguration is a comparatively less researched area of reconfiguration. In this instance, modules are initialised in a connected structure and unnecessary modules removed in order to reveal the final configuration.

There are fewer instances of subtractive reconfiguration being fully demonstrated with a modular robotic system. Using the aforementioned *Miche* system, Gilpin et al. demonstrate subtractive reconfiguration in three-dimensions [12]. The *Miche* modules are small cubes, equipped with magnetic connectors on the six faces. Initialising in a cubic lattice the modules are able to determine whether or not they are required in the final configuration and disconnect from their neighbours if they are not. In an effort to minimise the size of the modules, external actuation is used to move the unnecessary modules. This is achieved in this case using gravity. The authors demonstrate the presented concepts with twenty-eight *Miche* modules, forming a variety of shapes autonomously, such as a dog and a humanoid structure, as seen in Figure 2.3 (a). Gilpin et al. go on to demonstrate subtractive reconfiguration with the *Robot Pebbles* system too [16]. The *Robot Pebbles* are smaller scale robots, inspired by *Miche*, but with connectors on only four of the faces, thereby restricting the lattice to two-dimensions. Again, the modules feature no internal locomotion, instead relying on a vibrating table to locomote the modules, as discussed in Section 2.2.4.

One demonstration with a swarm of robots that are not modular is by Gauci et al. [15]. In this work the author uses a swarm of 725 *Kilobots*, a small-scale robot designed for research in swarm applications [129]. The *Kilobots* are initiated in a tightly packed group and given a user defined goal shape. Those robots that are not required in the final configuration remove themselves through the combination of collision avoidance, phototaxis and anti-phototaxis. Phototaxis is the movement of a robot towards a light source, anti-phototaxis is movement away from the source. The collision avoidance means that the robots will not remove from the structure until there is a clear path ahead, and will not disturb the goal shape in doing so. The phototaxis and anti-phototaxis are used in combination to move the unwanted robots away from the goal shape, and is employed depending on the desired shape and location of the light source. The work shows the strategy successfully yielding certain types of shapes when the correct combination of phototaxis and anti-phototaxis is used. The author discusses the scalability of this approach, hypothesising that the time taken is bounded by the longest path that a robot must take to exit the shape, resulting in $O(\sqrt{n})$ for a square starting shape, or the diameter of the shape in other cases.

The fact that a removal order can be derived by robots simply waiting to be able to move has more general use cases for modular reconfiguration as shown by Tolley et al. [130]. In this work the authors are concerned with additive reconfiguration, utilising simulated modules that are able to float as though in a fluid. However, to obtain an order in which to construct the goal shape, the authors use subtractive reconfiguration. They begin with the goal structure and remove modules that are on the exterior layer and able to move, much like the order in which *Kilobots* move in the aforementioned work by Gauci et al. [15]. From this they determine an order in which modules are able to be removed which, when reversed, yields an order in which the additive reconfiguration can occur. The authors demonstrate physical modules forming a humanoid structure, similar to that presented by Gilpin et al. [12], through additive self-assembly, using an assembly order derived by simulating subtractive reconfiguration. In the worst case, the computation requirements of the assembly planner scales with $O(kn^4)$, where n is the number of modules and k is the sampling rate at each step.

Chapter 3

Introducing Active Subtraction

3.1 Introduction

As presented in Chapter 2, there are numerous ways in which a modular robotic system is able to reconfigure, the majority of which involve adding modules or rearranging them to form shapes, known as additive and morphing self-reconfiguration, respectively. Also described was the concept of subtractive reconfiguration, where a desired shape is formed by initiating a group of modules in a given starting formation, before having unneeded modules detach themselves. This starting formation is a dense, rectilinear structure. Research into subtractive reconfiguration is motivated by a number of factors, as explained in Section 1.1. One such motivation comes from the increased reliability it lends. Typically, modules require high precision to form connections between themselves, unless these connections are already present in the starting configuration, as is the case in a subtractive approach. Furthermore, as discussed by Gauci et al. [15], an initially connected structure can guarantee that the modules are able to communicate from initialization. Because of this, information such as the initial configuration or the control scheme can be easily passed to all members before reconfiguration begins.

Previously, Gilpin et al. presented subtractive reconfiguration with modular robots using the *Miche* and *Robot Pebbles* systems in three- and two-dimensions respectively [12, 16]. However, in these works the modules feature no internal locomotion, instead relying on a vibrating table to locomote once disconnected from the structure.

Subtractive reconfiguration is also possible with non-modular robots, as demonstrated by Gauci et al. [15]. As described in Section 2.4.3, the authors demonstrate a swarm of *Kilobots* initiated in a tightly packed group, forming a user-defined goal shape through the extraneous robots locomoting away from the initial group. In this case the reconfiguration is *active*, as the robots actively remove themselves.

Combining and building on these concepts, and employing lattice-based modular systems with internal locomotion, such as *M-TRAN* [57], *SMORES* [62], *M-Blocks* [81] or *HyMod* [54], it could be possible to have modules remove themselves from an initial starting square or cube, leaving behind the desired structure. This process is termed *active subtraction*.

The problem of reconfiguring via active subtraction is particularly challenging in the presence of gravity, as the redundant modules may have to leave in a particular order to prevent the structure from collapsing at any time. As such, the work contained within this thesis focusses on configurations that extend vertically from the ground. However, the problem of autonomously reconfiguring in three-dimensions is an order of magnitude more difficult than in two-dimensions. One example of where this complication has large ramifications is in the development of a distributed movement algorithm, as explained in Section 3.4.1. Because of this, the work contained within this thesis is concerned with two-dimensional structures, in the vertical plane. Despite this focus, Section 7.2 contains discussion of expanding the work in this thesis to allow for reconfiguration in three-dimensions.

The remainder of this chapter is concerned with the introduction of *active subtraction* and is structured as follows. Section 3.2 first formalises the problem of self-reconfiguration via active subtraction. Context with regards to modular robotic systems and the more general problem of their reconfiguration is then presented in Section 3.3. With this in mind, Section 3.4 details the simulated system used for the work presented in this thesis. Finally, Section 3.5 concludes the chapter.

3.2 Problem Formulation

This section details the formulation of the problem of self-reconfiguration using active subtraction. Solutions to this problem are presented as the main contributions of this thesis, in Chapters 4 and 5.

Consider a group of modules arranged to fill a two-dimensional, rectangular space in a vertical plane. The environment is bounded in one of the four directions by a static surface, which is referred to as the *ground*. In the remaining three directions there exists free space, at least large enough for a module to move into. By default, the modules must remain connected to themselves and to the ground, as otherwise, the structure would collapse due to gravitational forces. The modules, ground, and all connections are considered to be infinitely rigid. A number of modules in this *initial configuration* are required to remove themselves, to leave behind only modules in a predefined configuration, known as the *desired configuration*. As is the case with all existing subtractive reconfiguration, hollow spaces are not permitted in the desired configuration, as modules that are to leave the hollow space behind would not be able to remove themselves [15, 16]. A *sink* location is nominated, which is at ground level and adjacent to the structure. When a module reaches the sink location, it is assumed to be automatically removed from the configuration.

The modules in the desired configuration are henceforth referred to as *included* modules and are stationary throughout the reconfiguration. The excess modules that are to be removed are referred to as *excluded* modules. Whilst in the process of removing itself from the configuration, a module is termed an *active excluded* module. The task is considered complete when all excluded modules have managed to reach the sink location.

3.2.1 Objective

Formally, a robot's configuration is represented as a graph, $G = (V,E)$. Each node $v \in V$ corresponds to a module with coordinates $(v_x, v_y) \in \mathbb{N} \times \mathbb{N}^+$. Edges represent connections between modules; a pair of modules are connected if and only if their Manhattan distance is 1. Let G' denote the *augmented* configuration that includes the original graph, G, as well as an additional node, g, representing the ground, connected to all nodes v where $v_y = 1$. Formally, $G' = (V', E')$ with $V' = V \cup \{g\}$ and $E' = E \cup \{\{g, v\} \,|\, v \in V \text{and} v_y = 1\}$. A configuration G is referred to as *feasible* (i.e., non-collapsing) if the augmented configuration, G', is a connected graph.

It is given that the modules are initially arranged in a rectangular configuration $G_0 = (V_0, E_0)$, of width x_{max} and height y_{max}, where $V_0 = \{(x,y) \,|\, 1 \leq x \leq x_{max}, 1 \leq y \leq y_{max}\}$. Let $G^{\text{inc}} = (V^{\text{inc}}, E^{\text{inc}})$ denote a desired, feasible, non-hollow configuration, where $V^{\text{inc}} \subseteq V_0$ are the included modules. For the robot to self-reconfigure from G_0 to G^{inc}, the excluded modules, $V^{\text{exc}} = V_0 - V^{\text{inc}}$, must be removed. For a module to be removed, it has to reach the sink, which is located at (0, 1).

The reconfiguration problem consists of identifying a finite sequence of configurations $G_1, G_2, \ldots, G_T$ such that

- For all $k = 1, 2, \ldots, T$: G_k is feasible;

- For all $k = 1, 2, \ldots, T$: G_k can be reached from G_{k-1} via one module executing a valid movement, which is potentially followed by removing the module if it has reached the sink;
- $G_T = G^{\text{inc}}$.

3.3 System Considerations

Following the problem definition and conceptualisation of the novel self-reconfiguration method of *active subtraction*, it is important to address the requirements for the system that will be used to develop active subtraction as a method of self-reconfiguration. As detailed in Chapter 2, there are many different types of modular robotic systems, with different attributes and capabilities. This section presents a discussion of characteristics relevant to active subtraction, from both a hardware and software perspective, before Section 3.4 details the idealised system that is utilised in the work presented within the following chapters. Chapter 6 presents a more detailed analysis of how the solutions to the self-reconfiguration problem may be transferred to real world systems.

The first consideration to be made is that of the type of modular system, as introduced in Section 2.2.1. Systems can be heterogeneous, with distinct module types, or homogeneous, where all modules are identical. In the case of modular robotic systems, homogeneous systems are preferable as they allow any module to fill any required cell in a configuration. However, for the centralised solutions of Chapter 4, a leader module is required, which must be capable of computation for successful reconfiguration. In the case of very large configurations, this computational requirement will only grow. Despite only one module needing this as a leader, a homogeneous system would require all modules to be identical, which will become impractical. However, it is possible for the leader to be external to the system, such as a connected computer, or the surface extension of the *HyMod* system [54]. As such, a homogeneous system is appropriate for active subtraction, with an external leader being utilised if needed.

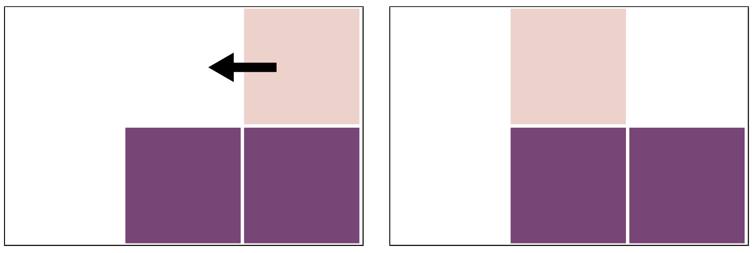
A further consideration is the configuration of the modules, as explained in Section 2.4. In the case of active subtraction, a lattice or hybrid system would be most suitable, as they can form regular initial configurations from which the modules are able to remove themselves. As previously stated, the work in this thesis is concerned with two-dimensional configurations in the vertical plane. As such, a square or cubic lattice makes the most sense, as the modules will be able to directly support those above and the number of movement directions are limited when compared to more complex shaped lattices. The modules should also have a sufficient number of connectors to allow for the creation of such lattices, and have strong enough connections to support neighbours during the reconfiguration process. Examples of appropriate modular systems for this requirement are *M-TRAN* [57], *SMORES* [62], *M-Blocks* [81] or *HyMod* [54].

The system must also be capable of reliable and quickly passing messages between modules. As demonstrated by Gilpin et al. [12], it is important for messages to be passed in order to successfully reconfigure via subtraction. This requirement will be even more important when modules are able to self-reconfigure, as their position within the lattice will change. The precise method of message passing can be achieved in a number of ways, such as each module having a unique identifier and addressing messages to the relevant module. Another method could be by using connectors that are capable of passing messages through them, and sending messages in the direction of the module that the message is intended for. For this to be possible the modules would require a method of determining its orientation, i.e. which way is up. This is utilised for the distributed algorithms in the thesis, where modules are concerned only with the state of their immediate neighbours. For the centralised solutions, the modules can use an initial message passing process to determine their location in the lattice, by simply passing messages with incrementing coordinate values in the relevant direction. Following this, the leader module can address messages to modules in specific coordinates.

3.4 Simulated Environment

This section details the idealised, simulated system that is used for the development of active subtraction. Each solution in the following chapters is first conceptualised and proven to be correct in an abstract way, before applied to the simulator to evaluate the performance under a wide range of conditions. The program developed for this is called Modular Active Subtraction Simulator for 2-Dimensions (MASS2D) and is available as an open source project [131].

The work presented in this thesis uses a model of a robot that, while having similar attributes to the aforementioned lattice-based modular systems, implements them in a simplified manner, akin to the sliding square method of movement [112]. The possible movements can be separated into two types, adjacent and diagonal movement, shown in Figure 3.1(a) and (b), respectively. The movements are expressed in terms of a module's Moore neighbourhood, illustrated in Figure 3.2. Adjacent movement is where a module traverses by a distance of one unit along only one axis, that is North (N), East (E), South (S), or West (W). This takes one time step to complete, as a module will move a total distance of one unit. For diagonal movement, a module traverses a distance of one unit along both axes, that is, North-West (NW), North-East (NE), South-East (SE), or South-West (SW), taking two time steps to complete in total. For


(a)

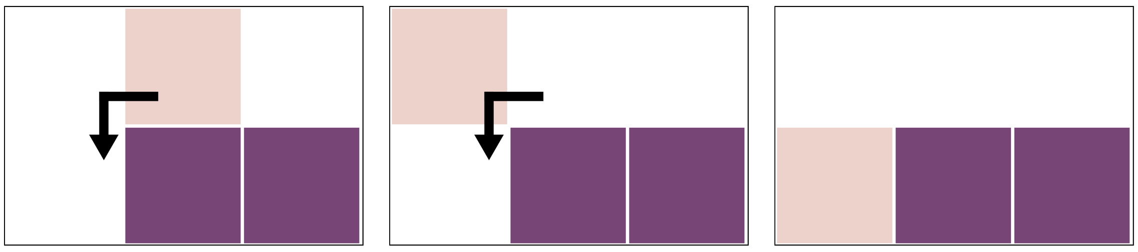

(b)

Fig. 3.1 Example of an active module employing (a) adjacent movement to travel West, taking one time step, followed by (b) diagonal movement to travel South-West, taking two time steps. Dark purple modules are included modules while cream coloured modules are active excluded modules.

visualisation purposes, modules move each part of the diagonal movement in one time step, so can appear suspended in the illustrations, as in Figure 3.1(b). In the simulation friction between modules is ignored.

Each module is aware of the state of its connectors, that is, whether each is connected to a neighbour or not. It is assumed that connected modules are able to communicate with one another and can pass messages through the system, including through the ground, as is possible with the *HyMod* surface extension [54]. Through this communication, the modules are able to establish their relative position and orientation within a global coordinate system, if required by the solution. The time required to pass messages is negligible compared to the time required to complete a movement, so a time step is considered to have passed each time a module moves, i.e. the messages are transmitted instantaneously. As the specifics of message passing is not the focus of this work, the process is simplified within the simulated system, and messages are sequentially resolved, with all messages being resolved within a single time step. As such, the messages can be considered immediately passed to a neighbour in the simulated system.

The concept of a time step is used throughout the simulator as a way to represent the synchronisation of the modules. Each module is able to pass and receive multiple messages in a time step, as these are considered instantaneous, but may only move a distance of one

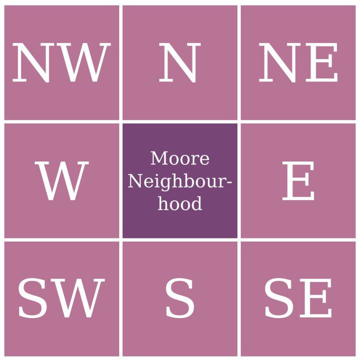


Fig. 3.2 Illustration of the Moore neighbourhood of a module.

unit. The problem of synchronising a system is not a simple one [132], especially when considering a system in which modules will be moving. However, this is not a focus of the work contained within, so is simplified within the simulations. The work is also not concerned with the optimisation of how long a time step is, just the minimisation of the number of time steps. As such, a time step could be arbitrarily long in order to ensure the synchronisation of the system. This synchronisation is also necessary for the movement, as modules will test in real time that the space they wish to move into is free, before moving simultaneously.

The modules also have a limited memory to store simple values and states. As previously discussed, the modules pass messages for a number of reasons, and so must be able to store values passed within these messages, such as a coordinate for localisation. The distributed solution presented in Chapter 5 uses a state based solution, so each module must have the memory to store the actions that each state dictates. As explained in Section 3.3, a homogeneous system is preferable for active subtraction, but with an external leader for scenarios that demand high amounts of computational resources. In the case of the simulator, the external leader is provided by the simulator itself, so a homogeneous system is used.

As explained in the problem formulation, the environment contains a single sink location, external to the initial configuration. A single sink is used in this work to simplify the development of the initial active subtraction algorithms contained within this thesis. Note that adding a second sink, one on each side of the structure, has the potential to allow the formation of the desired structure to be twice as fast. However, this would require modules to have knowledge of their locations in the system, as well as their path to the sink, something that is not required for the locomotion algorithm (detailed in the next section). It would also introduce the possibility of two moving modules colliding with one another when locomoting towards sinks, if they choose opposite sink locations but have to cross paths to reach them. Finally, a future goal for this work, as discussed in Section 7.2, is to extend the configurations in to three-dimensions, at

Algorithm 1: Active Excluded Module Movement Algorithm

```
1 $S_D \leftarrow$ {S, SW, W, NW, N, NE, E, SE}
2 while $position \neq sink$ do
3     $D \leftarrow$ null
4     forall $d \in S_D$ do
5         if can move in direction $d$ and $D =$ null then
6             $D \leftarrow d$
7     move in direction $D$
8     $S_D \leftarrow \{D-2, D-1, D, D+1, D+2, D+3, D+4\}$
```

which point the number of potential sink locations increases dramatically, further increasing the complexity. As such, a single sink is used and the process simplified.

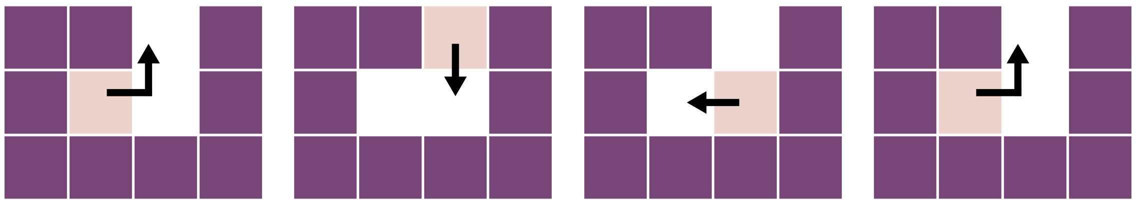
Also outlined in the problem formulation is the fact that connections are considered infinitely rigid. As such, any module with an adjacent module is considered supported by that neighbour. If a module does not have an adjacent neighbour then it is not supported and as such will fall due to gravity. A consideration for the active subtraction process is to ensure that all modules are supported by a connection to the ground. This could be via direct connection to the ground or connected through any number of modules in a tree to the ground. Proofs of this support is included in the presentation of the solutions. For the purposes of simulation gravity is not modelled, but any configuration that has an unsupported module is considered invalid.

The modules must be capable of locomotion in order to reach the sink. If achieved in a distributed way, the same movement algorithm can be used for both centralised and distributed solutions to the self-reconfiguration problem. The following is the distributed algorithm used in simulation by the excluded modules in order to locomote to the sink, which can be translated onto a real-world system in the future.

3.4.1 Active Excluded Module Algorithm

In order to remove themselves from the structure, the excluded modules require a movement algorithm to locomote towards the sink. Algorithm 1 shows a solution that enables a module to reach the sink with only knowledge of its Moore neighbourhood.

The first step is to determine an order, S_D, of directions in which to probe the state of neighbouring cells in its Moore neighbourhood (given a direction $d \in S_D$, let $cell(d)$ be *empty* if the corresponding neighbour cell is not occupied or is occupied by an active excluded


(a)

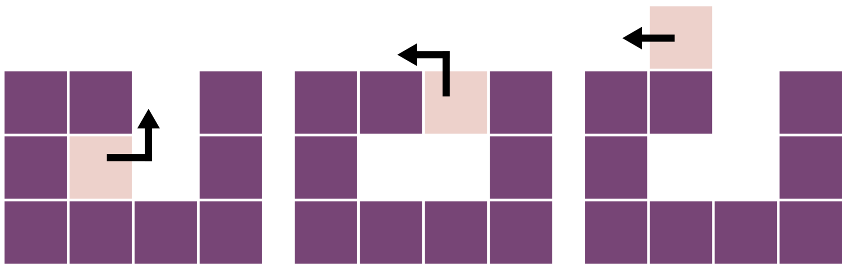

(b)

Fig. 3.3 Example of a stalemate when using (a) static prioritisation, and (b) dynamic prioritisation avoiding the stalemate.

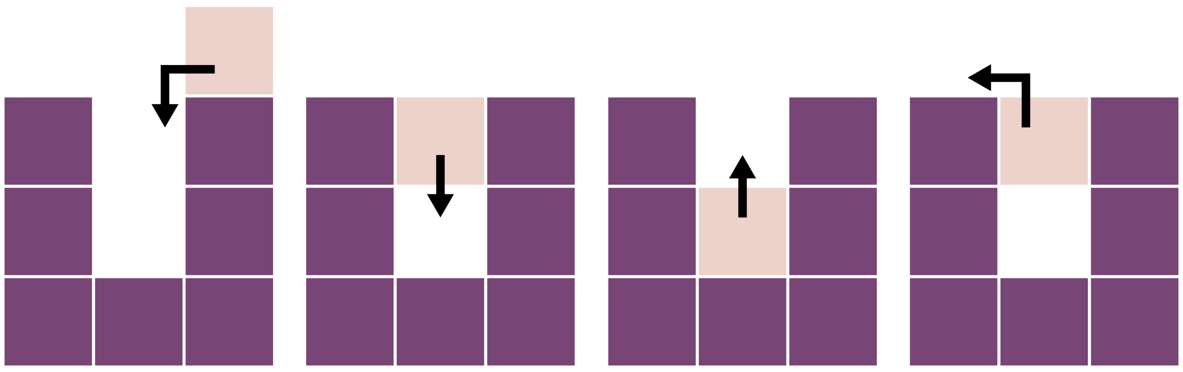


Fig. 3.4 Example of a module escaping a dead-end by checking all possible directions of movement, including opposite to the direction it already travelled.

module[1]). By defining the sink at position (0, 1), the module has a location to aim for, which is always either South, West or South-West. The module therefore prioritises a South-bound direction, continuing in a clockwise order through the directions of the Moore neighbourhood (line 1).

For non-rectilinear structures, such as the one shown in Figure 3.3, the aforementioned fixed priority sequence may result in a deadlock. The active module would first move to the North-East but is then able to move South, which is the prioritised movement, before moving West as the next highest prioritised movement. It would then repeat these three movements indefinitely, not making progress towards the sink, as seen in Figure 3.3(a).

To avoid such situations, the movement of the active module is informed by the direction it last travelled, D. This information is used to alter the priority (i.e., set S_D) by which the next direction is decided (line 8). The numeric alteration consists of steps around the Moore neighbourhood in a clockwise direction. The five Moore neighbourhood positions that do not constitute a 'backward' movement, that is, $D-2, D-1, D, D+1, D+2$, are given priority over the other three possible directions, $D+3, D+4$, so as to avoid retreading the same path, where possible. It is not necessary to check $D+5$, as $D+4$ will always be a possibility, due to it being the direction that the module just came from. Figure 3.3(b) shows this dynamic prioritisation resolving the previous stale-mate.

In some situations, such as in Figure 3.4, a module could find itself at a dead-end. Movement directions $D+3$ or $D+4$ become relevant and are probed in this sequence. This act of dynamically prioritising directions to move means that the active module can always reach the sink, although not necessarily by taking the shortest possible path.

Each movement that a direction, d, would imply falls into the category of adjacent movement or diagonal movement. For an adjacent movement it is simply necessary that $cell(d)$ be empty, and if this is the case then the direction to move, D, can be set to direction d. In the case of a diagonal movement, it is required that an adjacent cell be passed through to reach the potential location. The next neighbouring cell in the Moore neighbourhood is checked in this scenario (if the previous cell was empty, it would have already been selected). This process is implicitly represented on line 5.

Once D has been set, the module moves to the neighbouring cell located in direction D, re-prioritises the order of directions in S_D and repeats the process. This continues until the module reaches the sink, at which point it can be considered removed, and informs the leader accordingly.

[1]Active excluded modules are considered empty cells as they will no longer occupy the location they are currently in once a time step is complete. This is relevant when considering parallel movement as in Section 4.2.2.

3.5 Conclusion

In this chapter, the concept of active subtraction for modular robotic systems was introduced. The problem of self-reconfiguration using active subtraction was formalised, and discussions of system requirements presented. An idealised simulated system was detailed and the first algorithm, a distributed movement algorithm, was presented to be used in the work in the coming chapters.

Chapter 4

Centralised Active Subtraction

This chapter is based on, and expands upon, the author's original contributions to the publication [1].

4.1 Introduction

In the previous chapter, the concept of active subtraction was introduced. Also presented was a simulated modular robotics system, which is to be used to develop a solution to the problem of autonomous self-reconfiguration for modular robotic systems, the formulation of which was also detailed. This problem is addressed in this chapter, where a solution for reconfiguration in two-dimensions using *active subtraction* is presented. Two such solutions are presented: one where modules operate purely sequentially, while the other allows for parallel movement. Both solutions require some level of centralisation, where a leader module is required. Through the development of this solution, it can be seen that active subtraction is a viable method of self-reconfiguration.

This chapter is structured as follows. The solutions to the problem formulated in Section 3.2 are presented in Section 4.2. Section 4.3 presents formal analysis of the validity and run-time performance of the solutions. Subsequently, simulations of the solutions in a range of scenarios are shown in Section 4.4, along with the results. Finally, Section 4.5 concludes the chapter.

4.2 Controller Design

The following section presents two solutions for the problem described in Section 3.2. The first solves the problem while having modules actively remove themselves one at a time, termed Sequential Active Subtraction (`SAS`). The second builds on the first but grants the modules parallel movement, greatly reducing the overall reconfiguration time, which is Parallel Active Subtraction (`PAS`). Both solutions require at least one module to have full knowledge of the system and make centralised decisions for the reconfiguration to be completed successfully.

4.2.1 Sequential Active Subtraction (`SAS`)

This section details the creation of a control algorithm that allows a leader module to select modules to be removed from the structure. When selected, an excluded module becomes an active excluded module and employs Algorithm 1 (see Section 3.4.1) to locomote to the sink location, removing itself from the structure. This will leave behind a set of connected modules that form a given desired configuration.

All modules execute an identical distributed algorithm based solely on local knowledge, except for the leader module, the identity of which can be chosen at initialisation, and which requires knowledge of the initial and desired configurations. The most Westerly included module on the ground is a suitable candidate for the leader module. This is a unique module, as there exists only a single block of included modules on the ground (because no hollow spaces can exist between modules and the ground), and only one of these has a Westerly neighbour that is not an included module. As all modules are initially connected, the location of each module can be defined relative to a single point within the system, by passing a message and incrementing the relevant value. The tuple of coordinate values can be used to identify the modules. The leader module is responsible for choosing and informing sequential excluded modules to remove themselves, one at a time. It does so by executing Algorithm 2.

Algorithm 2 requires the leader module to have knowledge of which modules are excluded (V^{exc}) and which are included (V^{inc}). Assuming that the set of excluded modules is not empty, the leader chooses one module to be removed. The leader can infer from V^{exc} and V^{inc} how many inactive connectors each excluded module has. Let F_v denote the number of inactive connectors for module $v \in V^{\text{exc}}$. For a module v to be considered for removal, the case must be that $F_v > 0$. A module where $F_v > 0$ is termed a free module, as it is free to move in the directions where it has an inactive connector. The leader module first determines the highest horizontal layer of the structure that contains at least one excluded module with $F_v > 0$. It

Algorithm 2: Leader module (SAS)

```
1 Require $V^{\text{exc}}$ and $V^{\text{inc}}$
2 while $V^{exc} \neq \emptyset$ do
3     $V^{\text{top}} \leftarrow \{u \in V^{\text{exc}} \mid u_y = \max_{v \in V^{\text{exc}}}\{v_y \mid F_v > 0\}\}$
4     $m \leftarrow \text{argmin}_{v \in V^{\text{top}}} v_x$
5     notify $m$                        ▷ send signal to activate $m$
6     $V^{\text{exc}} \leftarrow V^{\text{exc}} \setminus \{m\}$      ▷ update the set of excluded modules
7     wait $m$                          ▷ wait for $m$ to reach sink
```

denotes by V^{top} the set of excluded modules with $F_v > 0$ in this layer (line 3). It then chooses from V^{top} the module furthest West, m, which is also nearest to the sink (line 4). It informs the chosen module to remove itself from the configuration (line 5) and updates the set of excluded modules accordingly (line 6). The leader then waits for the active excluded module to reach the sink, which it can be informed of through the structure, or through the ground (line 7). The process is repeated until V^{exc} is empty. As the process is sequential i.e., only one module can be an active excluded module at any time, modules cannot possibly collide.

4.2.2 Parallel Active Subtraction (PAS)

To reduce the time cost of reconfiguration, a version of active subtraction that allows modules to move in parallel has also been designed, termed PAS. It requires at least the leader module to be capable of conducting simulations. Active excluded modules continue to use Algorithm 1 (see Section 3.4.1). PAS differs from SAS by the timings that the excluded modules are activated. However, it preserves the order in which the modules arrive at the sink.

Algorithm 3, which is executed by the leader module, requires the module to simulate Algorithm 1 to obtain the order in which the excluded modules reach the sink, $\varphi(\cdot)$, along with the number of time steps that the ith module reaching the sink was active, Δ_i, where $i = 1, 2, \ldots, |V^{exc}|$. For example, if excluded module m was the second module to arrive at the sink and was active for 20 time steps, $\varphi(m) = 2$ and $\Delta_2 = 20$. The start time of the first module, s_1, is set to 0. Its arrival time, a_1, is set to Δ_1, with all other modules assigned an arrival time of infinity. Subsequent modules must arrive at the sink after their predecessor. Moreover, collisions among active modules must be prevented. A collision is defined as an active module residing within the Manhattan neighbourhood of another active module, meaning the modules could intersect within one time step. Only the Manhattan neighbours are considered, that is neighbours to the North, East, South and West, as it takes two time steps to perform diagonal movement. Without collisions, interaction between active modules is avoided, which could

Algorithm 3: Leader module (PAS)

```
1  Require $\varphi(\cdot)$, $\Delta$                ▷ Obtained by simulating Algorithm 2
2  $s_1 \leftarrow 0$
3  $a_1 \leftarrow \Delta_1$, $a_{j\neq 1} \leftarrow \infty$
4  forall $i = 2$ to $|V^{exc}|$ do
5      $s_i \leftarrow \max(0, a_{i-1} + 2 - \Delta_i)$
6      removal ← false
7      while removal = false do
8          $V^{sim} \leftarrow V^{inc} \cup \{u \in V^{exc} \,|\, a_{\varphi(u)} \geq s_i\}$
9          collision ← simulate $V^{sim}$
10         if collision then
11             $s_i \leftarrow s_i + 1$
12         else
13             removal ← true
14     $a_i \leftarrow s_i + \Delta_i$
15 notify $j$ at $s_j$                 ▷ send signals to modules at start times
```

otherwise affect the direction in which they move (see Algorithm 1). As a consequence, the arrival times of subsequent modules must differ by at least two. The earliest possible start time for the module in question is chosen accordingly (line 5). The leader module then simulates the active subtraction process, to determine whether the module in question is successfully removed without any collision (line 9). To lessen the computational load, it is only necessary to simulate the movement of modules that have not reached the sink by the time the newly considered module is deployed (line 8). Excluded modules that are yet to be assessed are included in the configuration, but as they have no designated start time, their movement is not simulated. If a collision occurs, the start time of the module in question is delayed by one time step and the simulation is repeated. Once a valid start time has been found, the arrival time is calculated accordingly (line 14). Once all excluded modules have been assigned a start time, the leader can begin the reconfiguration of the system. It initializes a clock and activates the excluded modules at their corresponding starting times (line 15). Note that multiple modules may be activated on the same time step.

4.3 Mathematical Analysis

This section formally analyses the correctness and run-time performance of both SAS and PAS.

Lemma 1. *If $V^{exc} \neq \emptyset$, then lines 3 and 4 of Algorithm 2 identify a module to be removed.*

Proof. Consider the set of excluded modules that have at least one inactive connector, $B = \{v \in V^{exc} \,|\, F_v > 0\}$. If $B = \emptyset$, then every excluded module is surrounded by other modules. The number of excluded modules is finite, therefore, the excluded modules would have to be fully encapsulated by the included modules. However, this is not possible, as the desired structure (made of the included modules) is non-hollow. Therefore, $B \neq \emptyset$. As B is finite, it follows that there exist modules in B that have maximal height, that is, $V^{top} \neq \emptyset$. From these modules, the algorithm chooses the most-Westerly one. The choice is unique, as no two modules have identical coordinates. □

Lemma 2. *The module chosen by Algorithm 2 can be removed without the configuration becoming unfeasible.*

Proof. The included modules form, by assumption, a feasible configuration. Hence, module m is not required to support any included module. Let us assume that removing m would make the configuration unfeasible. Consider a vertical block of u excluded modules, that is surrounded by two empty cells at coordinates (x,y) to $(x,y+u+1)$. One can show that u has to be 0:

1. If $(x,y+u+1)$ was removed before (x,y), then $(x,y+u)$ would have been removed before (x,y) as well.
2. If (x,y) was removed before $(x,y+u+1)$, then $(x,y+1)$ would have been removed before $(x,y+u+1)$.

Hence, u must be 0. As a consequence, any excluded module is supported (directly or indirectly) from below by either an included module or the ground, or from above by some included module. □

Lemma 3. *If Algorithm 2 chooses a module, m, to be removed, Algorithm 1 constructs a valid, finite path from m to the sink.*

Proof. Let G_s be the graph, where the nodes are the free faces of any module in the configuration, excluding the focal module m, and where the edges link any pair of adjacent faces. As m is the module to be removed, it must have at least one free face, implying it is adjacent to one or more faces in G_s. It is known that G_s is connected at all times. Each free module face can only be adjacent to one other face per side, that is, each node has two edges, meaning G_s is a path graph of finite length. The sink, s, is located at the ground and adjacent to the configuration. Thus, s is always at one end of the path graph. As m is connected to a module whose free face(s) belongs to G_s and the proposed movement framework allows m to move along a new direction, that is, an edge (see Algorithm 1), m always reaches an end of G_s. If the end that is reached is not s, implying the module was traversing the path in a clockwise direction around the configuration, it will re-traverse G_s in the opposite (counter-clockwise) direction. This causes the module to inevitably reach s. □

Theorem 4. *By employing Algorithms 1 and 2, `SAS` is guaranteed to remove all excluded modules.*

Proof. The theorem is proved by induction. If there are no excluded modules, nothing is to be shown. Assume that the theorem is true if there are $k \geq 0$ excluded modules. Let $|V^{\text{exc}}| = k+1$. According to Lemma 1, the leader module identifies a module, m, to be removed, and activates this module. It follows from Lemma 2, that removal of module m does not cause the configuration to become unfeasible. As stated by Lemma 3, module m traverses a finite path from its initial position to the sink. As only active modules move, and as m is the only active module, module m reaches the sink in finite time. Once the sink is reached, the module is removed, and deactivated, resulting in a configuration of only k excluded modules. For this configuration, all excluded modules are guaranteed to be removed, meaning the proof is complete. □

Theorem 5. *Using `SAS`, the number of time steps required to reach the desired configuration is bounded by $O(|V_0|^2)$.*

Proof. While executing Algorithm 1, the active excluded module moves along the perimeter of the remaining structure. Initially, this may happen in a clockwise direction. Once the module moves in a counter-clockwise direction around the structure, however, it keeps doing so until reaching the sink. The path length along the perimeter is bounded by $|V_0|+1$, where V_0 is the initial configuration (defined in Section 3.2.1). Therefore, the module must reach the sink in at most $2|V_0|+2$ time steps. As at most $|V_0|$ modules are to be removed, the number of time steps to reach the desired configuration is bounded by

$$2|V_0|^2 + 2|V_0| = O\big(|V_0|^2\big). \tag{4.1}$$

□

Theorem 6. *`PAS` guarantees that all excluded modules are removed, each following the same path as in `SAS`. The order in which modules arrive remains the same as in `SAS`.*

Proof. The theorem is proved by induction. If there are no excluded modules, nothing is to be shown. Assume that the theorem is true if there are $k \geq 0$ excluded modules. Let

$|V^{\text{exc}}| = k+1$ and the modules be labelled in the order by which they would arrive at the sink when employing SAS: $1,\dots,k+1$. If module $k+1$ was changed to be an included module, all statements would be true. That is, the k excluded modules would reach the sink at times $a_1 < a_2 < \dots < a_k$, each one following the same path as in SAS. Algorithm 3 determines the arrival times $a_1 < a_2 < \dots < a_k$ independent of whether module $k+1$ is excluded or not. According to line 5 of Algorithm 3,

$$a_{k+1} = s_{k+1} + \Delta_{k+1} \geq \max\left(0, a_k + 2 - \Delta_{k+1}\right) + \Delta_{k+1}, \tag{4.2}$$

$$a_{k+1} = \max\left(\Delta_{k+1}, a_k + 2\right). \tag{4.3}$$

Hence, module $k+1$ starts no earlier than at time 0, and the order of arrival is preserved. If $a_{k+1} = a_k + 1$, one module would reside within the Manhattan neighbourhood of the other, resulting in a collision. Hence, $a_{k+1} = \max(\Delta_{k+1}, a_k + 2)$ would be the earliest possible arrival time. The leader module emulates the movements of modules 1 to $k+1$ to check for possible collisions. Any possible collisions among modules 1 to k have already been resolved. If a collision involving module $k+1$ occurs, s_{k+1} (and hence a_{k+1}) is incremented, and the process repeated (see line 11 of Algorithm 3). As $a_{k+1} = a_k + 2 + \Delta_{k+1}$ could not result in any collision, the number of iterations is bounded. In each iteration, at most $\Delta_{k+1} + \max_{j \leq k} \Delta_j$ steps have to be simulated: If module $k+1$ reaches the sink without any collision, arrival times $a_1 < a_2 < \dots < a_k < a_{k+1}$ have been determined. Moreover, as no collision remains, none of the other excluded modules (1 to k) will ever reside within the Manhattan neighbourhood of module $k+1$. In other words, module $k+1$ follows exactly the same individual path as for SAS. The movements of module $k+1$ can hence not render a configuration unfeasible, meaning the proof is complete. □

Theorem 7. *Using SAS or PAS, in the worst-case, the number of time steps required to reach the desired configuration is* $\Theta(|V_0|^2)$.

Proof. According to Theorem 5, the number of time steps required to reach the desired configuration is $O(|V_0|^2)$, where V_0 is the initial configuration (as defined in Section 3.2.1). In other words, the time grows at most quadratically with the number of modules in the initial configuration. This result equally applies to PAS. What remains to be shown is that quadratic growth is indeed possible, that is, the number of time steps required to reach the desired configuration may be $\Omega(|V_0|^2)$. Consider the initial configuration shown in Figure 4.1(a). It comprises two horizontal blocks of included modules. Although each block contains 7 modules in Figure 4.1, in a more generalized configuration, each block contains $\frac{V_0^{exc}}{2}$ modules, where V_0^{exc} can be arbitrarily large. To obtain a lower bound for the number of time steps required

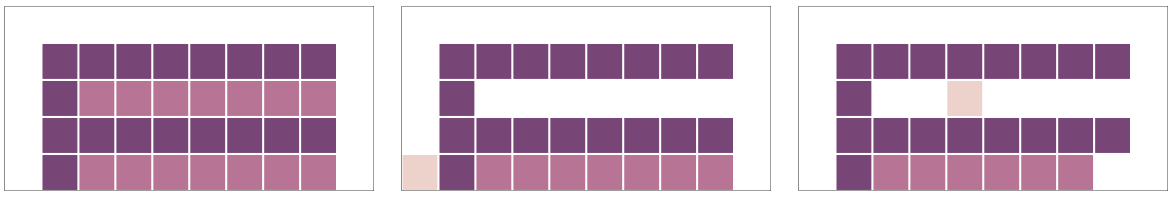


Fig. 4.1 Example of a complex shape with a long corridor that excluded modules must traverse. As before, dark purple modules are included modules and cream coloured modules are active excluded modules. Additionally, light purple modules are the inactive excluded modules. Shown at (a) initialisation, (b) after the top row of excluded modules is removed, and (c) part way through the removal of the lower row of excluded modules. Animated versions of the reconfiguration process available online [133].

to reach the desired configuration, it is assumed that the excluded modules in the upper block have already removed themselves from the structure, as shown in Figure 4.1(b). Any excluded module from the lower block will fully explore the upper corridor, as it cannot determine using local knowledge alone that this is a dead-end, seen in Figure 4.1(c). To prevent collisions, subsequent modules need to be sufficiently separated in time. Formally, the times at which any pair of subsequent modules arrive at the sink have to differ by $|V_0^{exc}|$. The $\frac{|V_0^{exc}|}{2}$ excluded modules in the lower block hence need time steps at least equal to

$$\left(\frac{|V_0^{exc}|}{2} - 1\right)|V_0^{exc}| = \Omega\big(|V_0^{exc}|^2\big) \tag{4.4}$$

in order to be removed. In this example,

$$V_0 = 4 + 2|V_0^{exc}|, \tag{4.5}$$

as such the time steps required corresponds to

$$\Omega\big(|V_0|^2\big). \tag{4.6}$$

$\square$

4.4 Simulation Studies

In this section, a number of simulation studies are presented, demonstrating the capabilities of `SAS` and `PAS`. The algorithms are implemented in Python. The implementation allows the user to input a desired configuration (see Section 4.4.1) or generate random configurations (see Section 4.4.2), as well as graphically illustrating the paths of the excluded modules, if required. The program is called Modular Active Subtraction Simulator for 2-Dimensions (MASS2D) and is available as an open source project [131].

The performance of a simulation trial is quantified by the number of movements required for all excluded modules to be removed, referred to as *time steps*. To account for structures of different sizes, normalised values are presented, that is, the total time steps taken are divided by the number of excluded modules in the initial configuration. Formally,

$$T_{norm} = \frac{T_{total}}{|V_0^{exc}|}. \tag{4.7}$$

A simulation is considered successful if all excluded modules are removed, without the configuration becoming unfeasible at any moment in time.

The performance is also compared to a calculated best-case time for both sequential movement, sequential best-case time ($\mathrm{BCT_S}$), and parallel movement, parallel best-case time ($\mathrm{BCT_P}$). The best-case time is a theoretical lower bound for a given configuration. The A* algorithm [134] is employed to determine the shortest possible route that each excluded module could take to reach the sink, via empty cells or those occupied by excluded modules, while still circumnavigating the included modules. This yields the best-case time for each excluded module separately. $\mathrm{BCT_S}$ is calculated by summing these times together. $\mathrm{BCT_P}$ is calculated by ordering the best-case times for each module, starting with the shortest time. These times are then assessed in much the same way as line 5 from Algorithm 3. That is, each time is taken in the ordered set of individual best-case times, and the gap between the arrival times of adjacent arriving modules is compared. If the gap would be less than two, then two is added to the total to avoid collisions. If the gap would be more than two, that larger value is added. This process is repeated until the best-case time for every module has been assessed. Formally, $\mathrm{BCT_P}$ is,

$$\mathrm{BCT}_{i+1} = \mathrm{BCT}_i + \max(2, \mathrm{BCT}_{i+1} - \mathrm{BCT}_i) \tag{4.8}$$

$$\mathrm{BCT_P} = \mathrm{BCT}_n \tag{4.9}$$

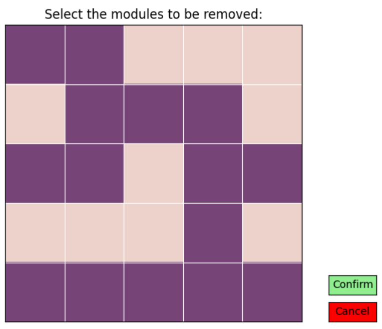


Fig. 4.2 The graphical user interface designed to enable a user to input configurations for simulation.

where $\mathrm{BCT}_1, \mathrm{BCT}_2, \ldots, \mathrm{BCT}_n$ are the ordered best-case times for each excluded module. BCT_S and BCT_P may not be possible in practicality, as a module's shortest route could rely on another excluded module not having been removed and these dependencies could be cyclical.

4.4.1 User-Defined Configurations

In this section, a range of user-defined desired configurations are investigated.

To aid with the production of the configurations, a graphical user interface (GUI) was created. By selecting modules to be removed, the user is able to input any configuration they desire, although only feasible configurations are simulated. Figure 4.2 shows the GUI being used to input a desired configuration from a 5×5 initial configuration.

Figure 4.3(a) shows how SAS subtracts excluded modules to reconfigure from a 4×4 starting square to a simple U-shape, similar to the one considered by Gauci et al. [15]. Such a shape could be useful for collecting tasks or grasping tasks, once it has been formed. When employing SAS, the reconfiguration takes 51 time steps. The same simulation in PAS takes only 17 time steps, a reduction of 67%, and is shown in Figure 4.3(b). The BCT_S and BCT_P are 51 and 17 time steps, respectively. As can be seen, the time-based performance of SAS and PAS match the best-case times, meaning each module follows the shortest possible path to the sink.

In the works on *Miche* and *Pebbles* by Gilpin et al. [12, 16], a humanoid structure is formed. As shown in Figure 4.4(a), this is not a feasible configuration in the problem formulation of Section 3.2 (which involves vertical 2-D structures and a ground), because the indicated module is enclosed by included modules and the ground. One possible solution is to rotate the configuration to a feasible one, shown in Figure 4.4(b). This reconfiguration from a 5×5

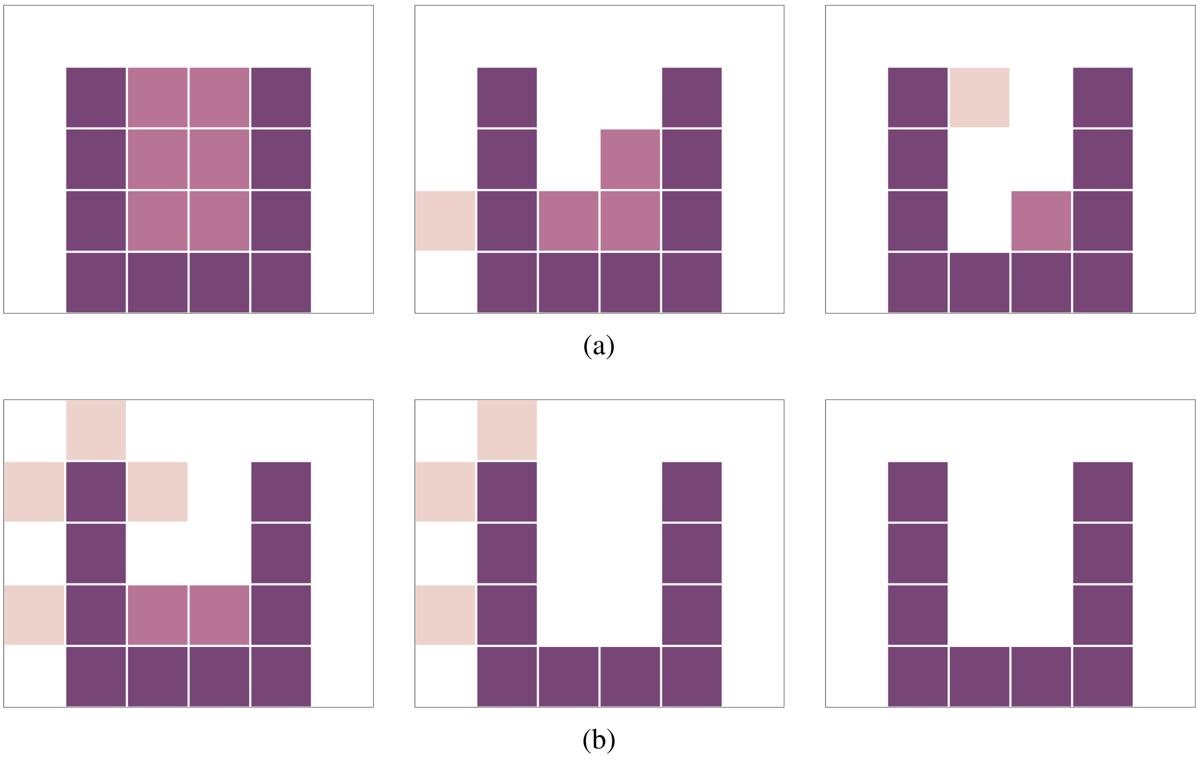


Fig. 4.3 Example of a user-defined U-shape being formed in simulation. (a) SAS shown at time steps 0, 22, and 34. (b) PAS shown at time steps 6, 12 and 18. Animated versions of the reconfiguration process available online [133].

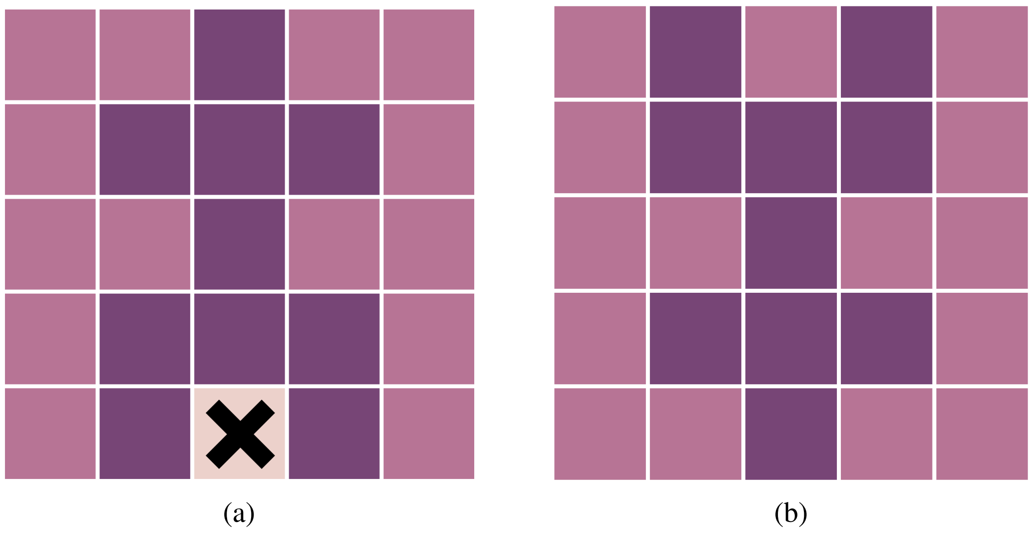
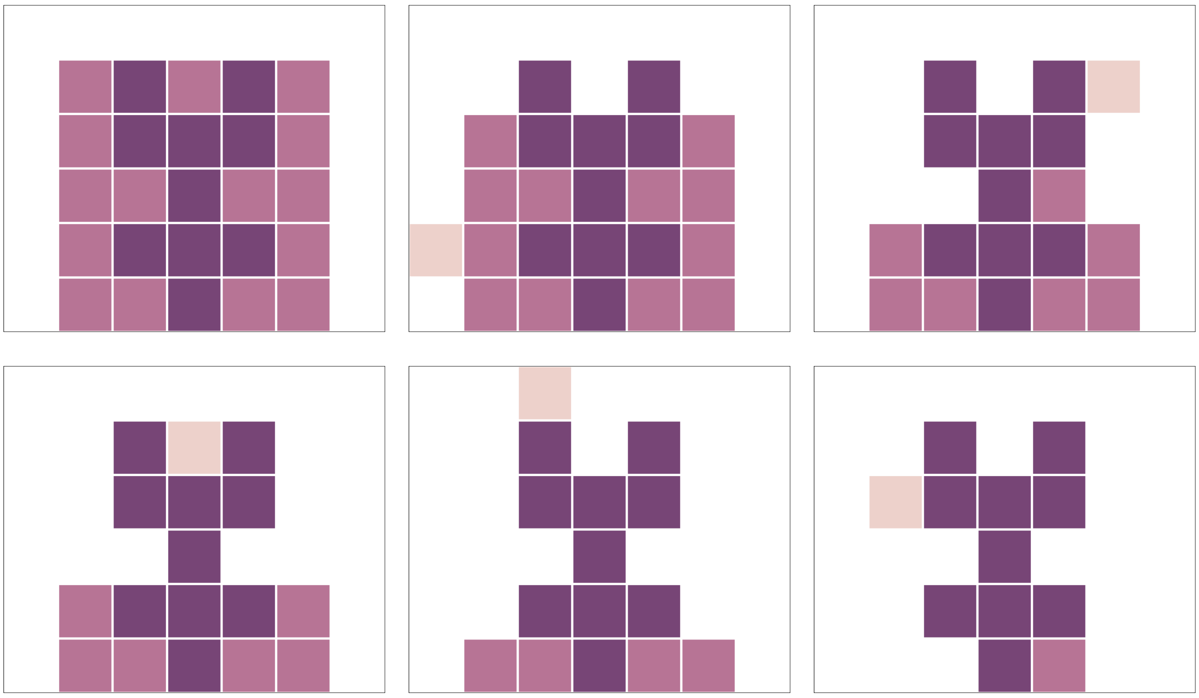


Fig. 4.4 A humanoid shape configuration that is shown as (a) impossible to form due to the enclosed module indicated, and (b) a feasible rotated version of the same shape.


(a)

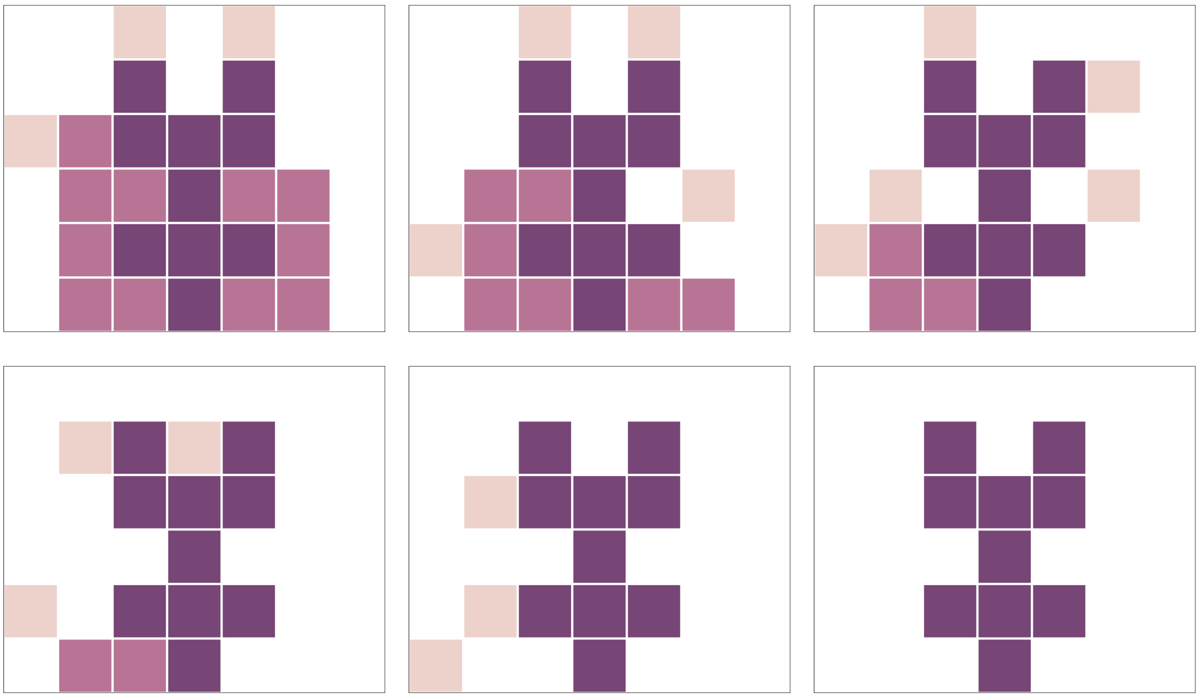
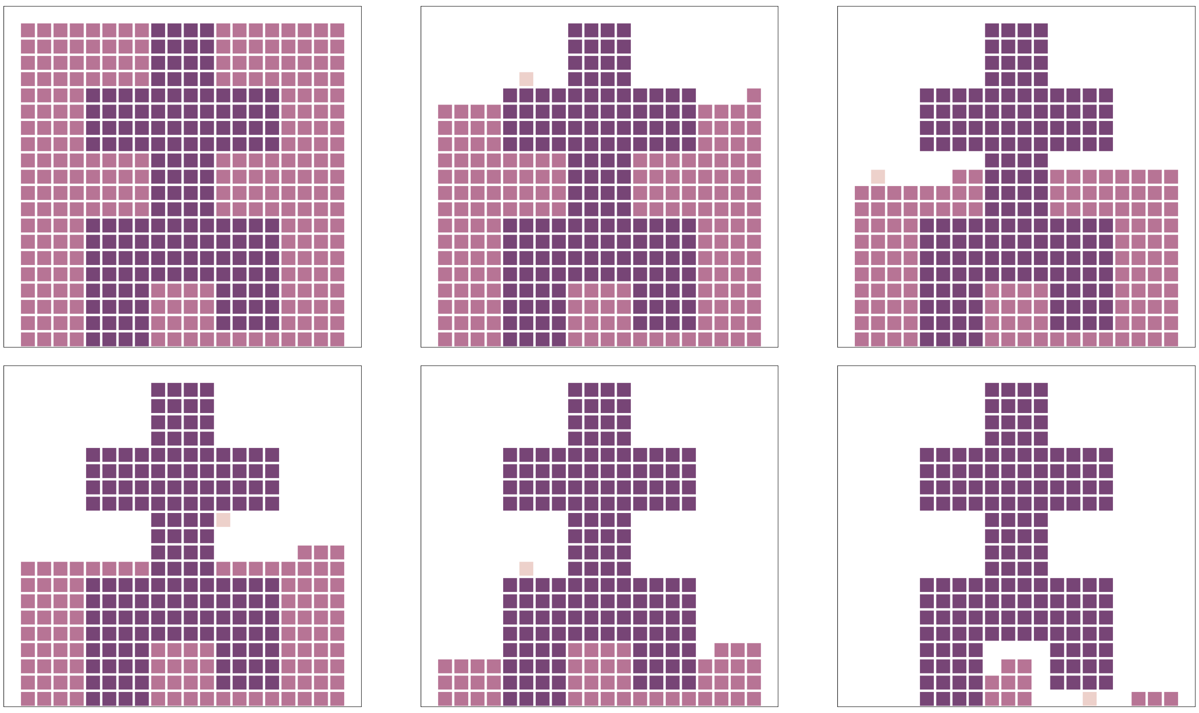

(b)

Fig. 4.5 The steps taken to form an inverted humanoid from a 5×5 grid in simulation. (a) SAS shown at time steps 0, 26, 54, 76, 100 and 127, and (b) PAS shown at time steps 6, 16, 24, 32, 37 and 45. Animated versions of the reconfiguration process available online [133].


(a)

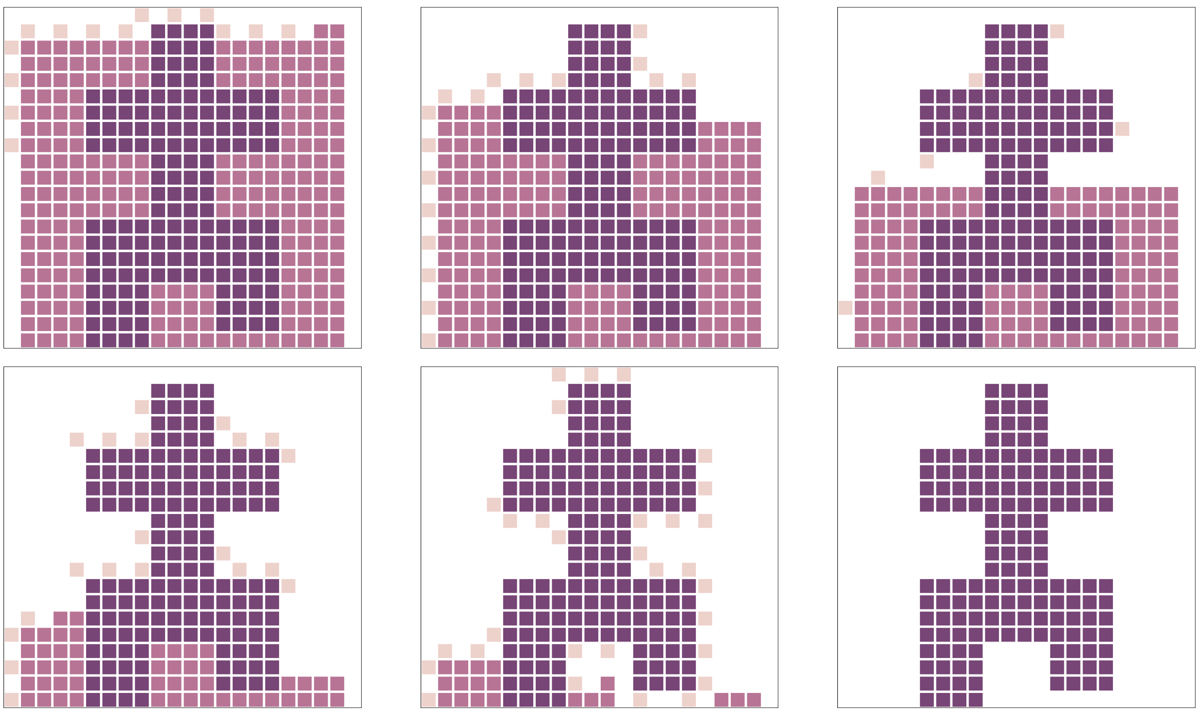

(b)

Fig. 4.6 Example of (a) SAS, and (b) PAS, being used to form a large humanoid structure from a 20×20 grid. Shown at time steps (a) 0, 2129, 3593, 4500, 6682 and 8490, and (b) 8, 138, 350, 496, 538 and 661. Animated versions of the reconfiguration process available online [133].

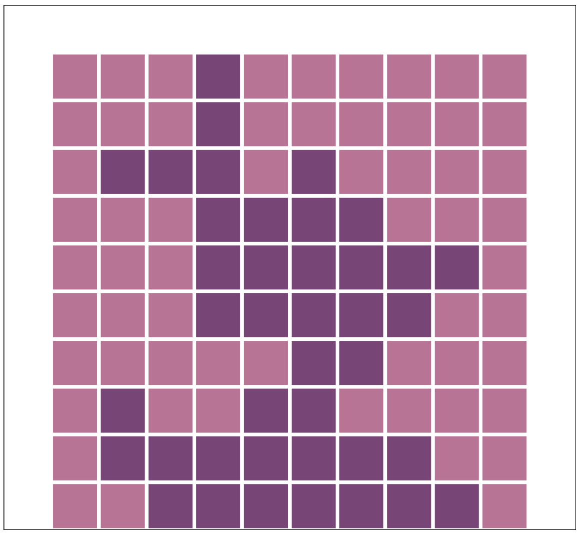


Fig. 4.7 Example of a randomly generated configuration, of size 10×10 and inclusion density (ρ) of 40%.

starting square takes 159 time steps using SAS, and can be seen in Figure 4.5(a). PAS, shown in Figure 4.5(b), takes only 45 time steps, a reduction of 72%. The $\mathrm{BCT_S}$ and $\mathrm{BCT_P}$ for this scenario are 125 and 29 time steps, respectively.

Another solution, shown in Figure 4.6, could be to increase the size of the initial configuration and have one side of the desired configuration slightly removed from the ground. Here the initial configuration is set to 20×20 modules. It takes 9554 time steps to reach this configuration when employing SAS, shown in Figure 4.6(a). The same 20×20 humanoid shape was also simulated using PAS. From Figure 4.6(b), one can see the benefits of multiple modules moving at once. PAS takes only 661 time steps, a reduction of 93%. The $\mathrm{BCT_S}$ and $\mathrm{BCT_P}$ for this scenario are 8190 and 487 time steps, respectively.

4.4.2 Randomly Generated Configurations

This section presents studies where the desired configurations are randomly generated, making it possible to systematically characterise the performance under a wide range of conditions. Initial configurations of different sizes are considered, alongside desired configurations of different inclusion densities. Inclusion density, ρ, is defined here as the percentage of modules within the initial configuration that are retained in the desired configuration, that is, the percentage of included modules. Formally,

$$\rho = \frac{|V^{\text{inc}}|}{|V^{\text{exc}} \cup V^{\text{inc}}|} \times 100\%. \tag{4.10}$$

To generate a desired configuration from a given initial configuration size, all modules in the configuration are initially assigned to be included modules. A set of candidate modules is created, originally consisting of modules that are in the exterior layer of the initial configuration. From the candidate set, a module is randomly chosen to be an excluded module and removed from the set. Each Manhattan neighbour of this new excluded module is assessed in turn, checking whether it is already in the candidate set, and whether changing this module to an excluded module would yield an unfeasible configuration. If neither of these conditions are true, then the neighbour is added to the candidate set. Once each neighbour has been assessed, the random selection of an excluded module from the candidate set is repeated. This process continues until the number of included modules satisfies the given inclusion density. Figure 4.7 shows an example of a randomly generated configuration of 10×10 modules with 40% inclusion density.

Influence of Density of Included Modules

The performance of SAS and PAS are studied for different inclusion densities, from 10% to 90%, in steps of 10%. Three sizes of initial configurations are considered for each inclusion density: 10×10, 20×20 and 30×30. For each combination of density and initial configuration size, the same 100 randomly generated desired configurations are used for simulating both SAS and PAS.

The effect that the inclusion density has on the number of time steps it takes the modules to remove themselves from the structure is depicted for both SAS and PAS in Figure 4.8. Separately, box plots of the results when using SAS or PAS are shown separately in Figure 4.9(a) and (b), alongside line graphs of the median BCT_S and BCT_P for each.

Influence of Configuration Size

The way in which performance of SAS and PAS scale with the configuration size is also studied, considering sizes of 5×5, 10×10, 15×15, 20×20, 25×25, and 30×30. For each case, the inclusion density is 60%. The same 100 randomly generated desired configurations are used for both approaches. The results are shown in Figure 4.10.

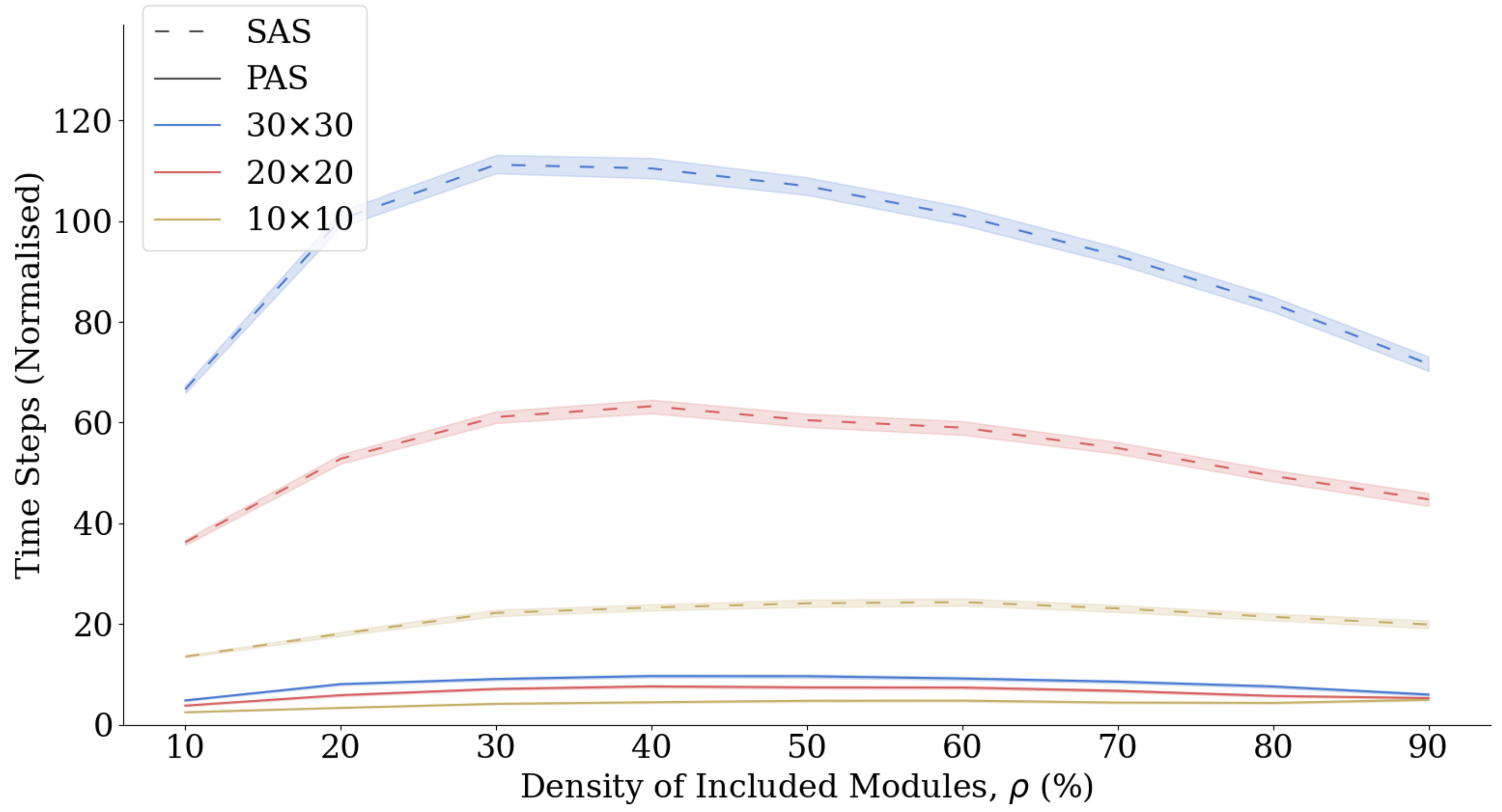
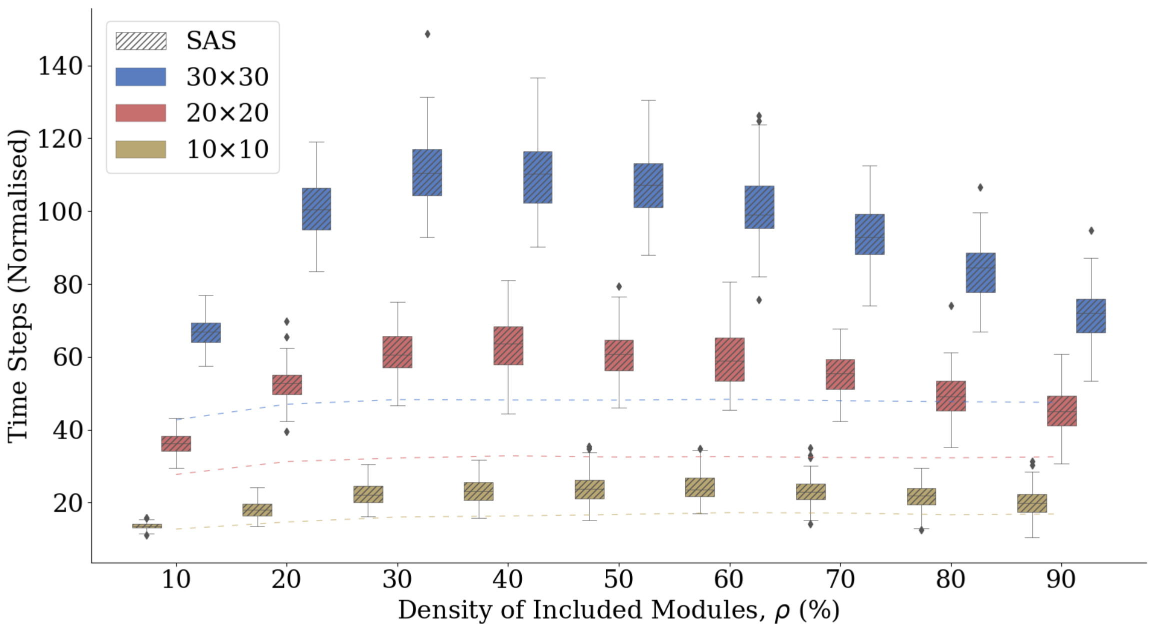
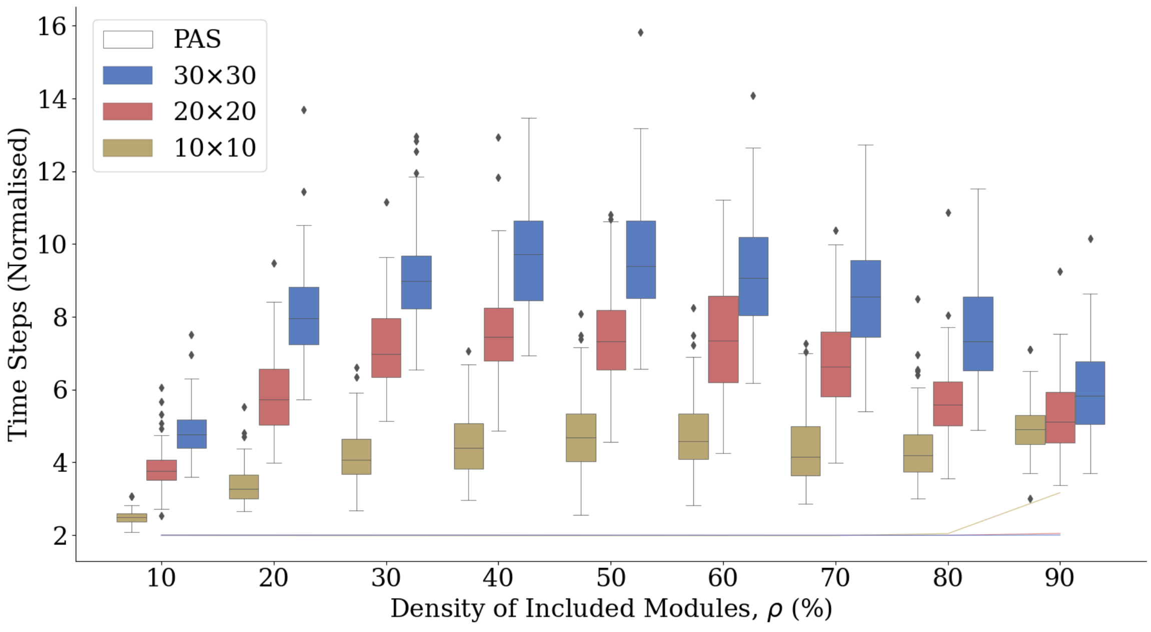


Fig. 4.8 Line plots of the normalised time taken by excluded modules to remove themselves, leaving behind randomly generated desired configurations of various inclusion densities (100 samples per 10% inclusion density step). Initial configurations of sizes 10×10, 20×20 and 30×30 are simulated. Time steps are normalised by the number of excluded modules in the initial configuration.



(a)



(b)

Fig. 4.9 Separate box plots of the simulated performance of (a) SAS and (b) PAS for the randomly generated desired configurations of inclusion densities between 10% and 90% for initial configurations of sizes 10×10, 20×20 and 30×30. Line plots of the median $\mathrm{BCT_S}$ and $\mathrm{BCT_P}$ for each scenario are overlaid on the respective box plots.

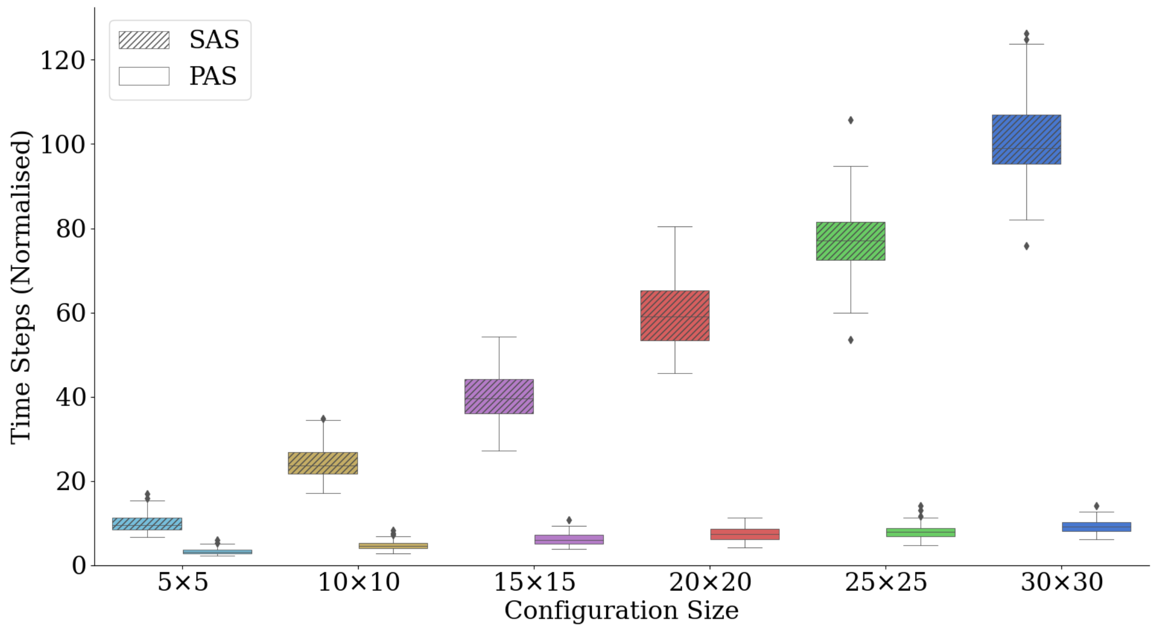


Fig. 4.10 Box plots of the normalised time taken by excluded modules to remove themselves from configurations of various sizes. The desired configurations were generated randomly with inclusion density 60% (100 samples per box). Time steps normalised by the number of excluded modules.

4.4.3 Discussion of Results

In every case simulated, both user-defined and randomly generated, all excluded modules were able to reach the sink and left behind only the desired configuration, thereby demonstrating the algorithms performing as intended.

PAS far outperforms SAS in terms of time steps taken for the same configurations, requiring the data to be shown on separate graphs in order to be fully assessed.

In Figure 4.9(a), the performance of SAS over varying inclusion density exhibits an interesting behaviour, where it can be observed to rise to a point and fall again. It is believed that this is due to the configurations that can be generated rather than SAS itself. Desired configurations of low density may have less chance of containing overhangs and dead-ends, geometry that is time-consuming to circumnavigate. As the density then increases, so does the likelihood of creating more complex geometry. However, once it becomes very high, areas that featured complex geometry are more likely to be filled again. This theory is reinforced by inspecting the performance of PAS in Figure 4.9(b), where a similar behaviour can be seen.

The disparity between best-case performance and real performance can be seen to be proportionally larger between $\mathrm{BCT}_{\mathrm{P}}$ and PAS. This is due to the fact that the module order is changed to find the true lower bound; in practice this would create a longer route for subsequent modules, an effect that is ignored when the A* algorithm obtains the individual best-case times. The $\mathrm{BCT}_{\mathrm{P}}$ tends towards a performance of 2 time steps, as can be seen in Figure 4.9(b). This is because the gap in individual best-case time performances for adjacent modules is 1 time step. It can be seen in Equation 4.9 that the minimum effect that each subsequent module can have on the overall best-case time performance is 2 time steps.

In Figure 4.10 the change in performance attributed to configuration size can be clearly observed. When using SAS, the normalised time that excluded modules take to reach the sink increases with configuration size, while the theory predicts a linear growth (or quadratic, if not normalised). However, when using PAS, the normalised time remained reasonably consistent. Although each module still individually travelled a further distance in a larger configuration, this also allowed for more modules to move in parallel, almost negating the effects of the increased travel distance. This is corroborated by the increased performance of PAS when simulating the configurations in Figures 4.4(b) and 4.6. These findings are in line with the proof in Theorem 7, where it was shown that the predicted growth was linear (i.e., quadratic growth, if not normalised) in the worst-case scenario.

The trade-off between computational cost and performance means that both SAS and PAS have potential use cases. SAS is able to run on simple hardware at the expense of reconfiguration speed, yet is still able to reconfigure large structures. Moreover, the relatively small increase in performance that comes with using PAS over SAS in smaller systems may not justify the need for more capable modules. Whereas, for larger scale systems the improvement may be sizeable enough to negate the cost of complex modules. Where PAS proves computationally too demanding for a leader module, the computations could also be off-loaded to an external computer.

4.5 Conclusion

In this chapter, a novel subtractive reconfiguration approach by which extraneous modules actively remove themselves from a starting configuration, to leave behind a given structure was presented. This approach is referred to as *active subtraction*. Two solutions were presented, one with purely sequential movement, the other with parallel movement. The correctness was formally proved, and the worst-case performance characterised for both solutions. Simulations that validated the solutions in a wider range of conditions were also presented, exploring the effect that varying the sizes and compositions of initial and desired configurations had on the time required for the modules to remove themselves.

Chapter 5

Distributed Active Subtraction

5.1 Introduction

In the previous chapter, centralised control schemes were presented to solve the problem of self-reconfiguration in two-dimensions with modular robots. These solutions, while successful in reconfiguring from an initial structure to a desired structure, display a number of shortcomings.

Firstly, the run-time performance is suboptimal, with modules needing to travel great distances due to the removal order in some cases. The performance worsened dramatically for structures with *cavities* for both Sequential Active Subtraction (SAS) and Parallel Active Subtraction (PAS). A cavity is defined as a space within the bounding box of the configuration that comprises excluded modules at initialisation, and will therefore be empty by the end of the reconfiguration process. As shown in Theorem 7 in Section 4.3, the performance in the worst-case increases quadratically with configuration size, due to cavities in the desired configuration. Optimisation is possible by altering the order in which modules remove themselves from the structure. This chapter details a method of reconfiguration that will improve the time-based performance when applied to configurations with cavities. This will be achieved by optimising the order in which excluded modules are removed, giving priority to modules in cavities further from the sink, thereby negating the need for excluded modules to climb in to and out of cavities as they self-reconfigure.

Secondly, the solutions also require a leader module to operate, which has to perform some relatively involved computation in the case of PAS. By creating a distributed solution to the problem, that would require no central control, the effort on the part of a leader module would be removed. This would also lead to a more robust system, as there would be no single point of failure, a desirable attribute in the case of modular robotic systems. The solutions presented in this chapter will accomplish self-reconfiguration entirely through message passing, meaning that the process is distributed. As such, the following solutions are termed *Distributed Ordering Sequential Active Subtraction (`DO-SAS`)* and *Distributed Ordering Parallel Active Subtraction (`DO-PAS`)*. As `PAS` outperformed `SAS` by such a large margin, this chapter is concerned with further optimising `PAS` in terms of time-based performance, as well as minimising computational load by removing the centralised computation. Therefore, the following will be focussed on `DO-PAS`, with the performance of `DO-SAS` being shown in the simulation studies for completeness.

This chapter presents a solution to the problem of self-reconfiguration in two-dimensions that overcomes these shortcomings. This is achieved via message passing to remain distributed, and deciding the order in which modules are removed based on these messages, inspired by the hormone based control discussed in Section 2.3.2. The problem formulation remains the same as the one presented in Section 3.2, but the approach to solving the problem that is presented in this chapter is a novel one.

The modules used in this chapter are identical to those in Chapter 4, the details of which are presented in Section 3.4. The movement algorithm employed (Algorithm 1, see Section 3.4.1) is distributed, and requires only local knowledge, so remains unchanged for the work in this chapter.

This chapter is structured as follows. Section 5.2 contains details of a new metric to help quantify any improvement in performance, and of a pre-reconfiguration phase for the new solution. Following this, Section 5.3 presents the core machinations of the distributed solution to the self-reconfiguration problem by describing the distributed state based behaviour of the modules. Section 5.4 then shows a step by step demonstration of the reconfiguration process. Formal analysis is presented in Section 5.5, comprising proofs for the correctness and run-time performance of the solution. Subsequently, simulations of the solution in a range of scenarios are shown in Section 5.6, along with the results. Finally, Section 5.7 concludes the chapter.

5.2 Excluded Row Removal

To help quantify any improvement in performance, a metric is defined here called the *maximum travel distance (MTD)*. The MTD is defined as the number of units the free module farthest from the sink must travel in order to reach the sink when employing the movement given by Algorithm 1. Recall from Section 4.2 that a free module is an excluded module with at least

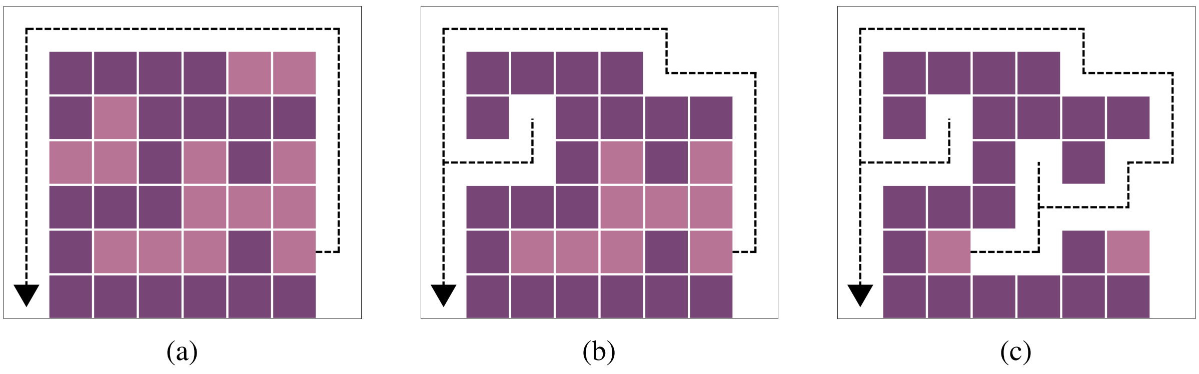


Fig. 5.1 Depictions of the MTD for a configuration at various stages of reconfiguration via SAS. Shown with an MTD of (a) 19, (b) 25 and (c) 31. As in Chapter 4, dark purple modules are included modules, light purple modules are excluded modules and cream coloured modules, not present in this figure, are active excluded modules.

one inactive connector, meaning there exists a free space on that side, and as such can move in that direction. Figure 5.1 demonstrates MTDs for a configuration with various numbers of excluded modules removed through SAS. As can be seen, the modules that are removed via the order given by SAS cause an increase in the MTD. Through the improvements made by DO-PAS this will be mitigated.

DO-PAS consists of an initial phase and a main phase that, in combination, are able to achieve self-reconfiguration. The first phase comprises the removal of any rows at the top of the configuration that are composed entirely of excluded modules. This is done to minimise the MTD before the main subtraction phase begins, as it means that the vertical distance modules on the side farthest from the sink must climb is reduced. This process is achieved via distributed message passing.

Upon initialisation, the module at the most North-Easterly point in the configuration, $v_{\mathtt{NE}}$, checks whether it is excluded or included.[1] If it is an included module then the top layer cannot be entirely composed of excluded modules, so the module sends a *priority* message with value 1 to the most South-Easterly module, $v_{\mathtt{SE}}$, to begin the main phase of assigning P values, the process of which, and details of the message types are introduced in Section 5.3. This case is shown in Figure 5.2(a). However, if $v_{\mathtt{NE}}$ is an excluded module, it sends a message to its Western neighbour, to begin checking whether the entire row is excluded. Each excluded module that receives this message continues to pass it to the West, as in Figures 5.2(b)-(d). If an included module receives the message, a different, *priority* message is returned towards $v_{\mathtt{SE}}$

[1]Modules at the corners of the configuration can deduce their position from the state of their connectors i.e., module $v_{\mathtt{NE}}$ is the only module in the configuration with a free North and East facing connector, module $v_{\mathtt{SE}}$ is the only module whose south facing connector is connected to the ground and a free connector on its East face etc.

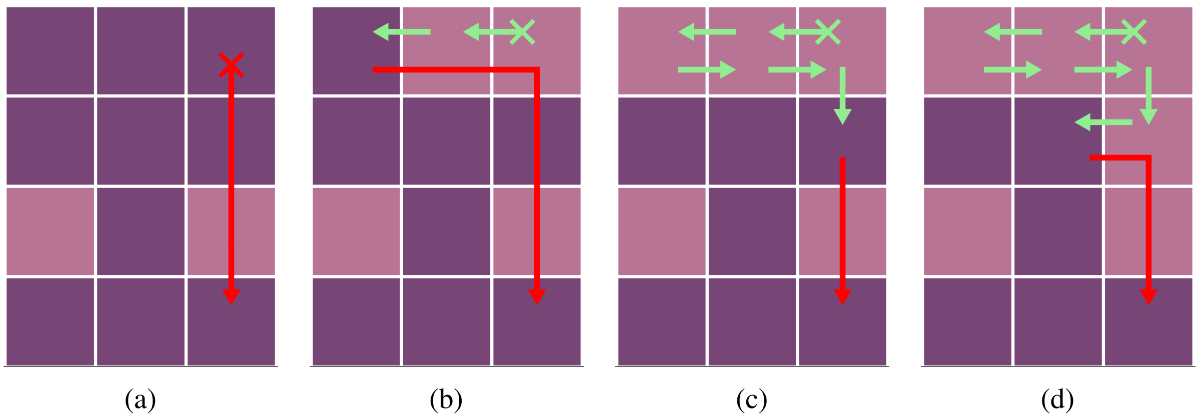


Fig. 5.2 Examples of how messages are passed between modules to assess whether a row is composed entirely of excluded modules. Red arrows indicate a negative message, while green indicates positive. The arrow attached to the $\times$ is the first message sent.

via $v_{\mathtt{NE}}$, indicating that the row is not entirely excluded, and as such the outer layer assessment should begin. This case can be observed in Figure 5.2(b), where the red arrow indicates the negative message. If the message reaches the most Westerly module in that row, and it is also excluded, then the entire row must be excluded. That module then sends a positive message to the East, and begins to remove itself from the structure. Subsequent modules in the row that receive the positive message as it is passed towards $v_{\mathtt{NE}}$ also begin moving, one time step after they receive the positive message, ensuring a gap of two time steps between each module. The modules can be seen passing the positive message to the East in Figures 5.2(c) and (d). This process is repeated on descending rows until a row that is not entirely excluded is found. In the case shown in Figure 5.2(c), the first module in the row is included, so the negative message is immediately sent to $v_{\mathtt{SE}}$. However, in Figure 5.2(d) the most Easterly module in the second row is excluded, so passes the positive message West until it reaches an included module. In all cases a message is sent to $v_{\mathtt{SE}}$ as soon as an included module is reached, and $v_{\mathtt{SE}}$ begins the main phase.

The process of the modules autonomously removing themselves is illustrated in Figure 5.3. By inspecting Figure 5.3, the benefits of excluded row removal can be seen, as the MTD reduces from 23 to 19. The reduction of four can be attributed to the travel distance being lessened by two units of movement North and two South.

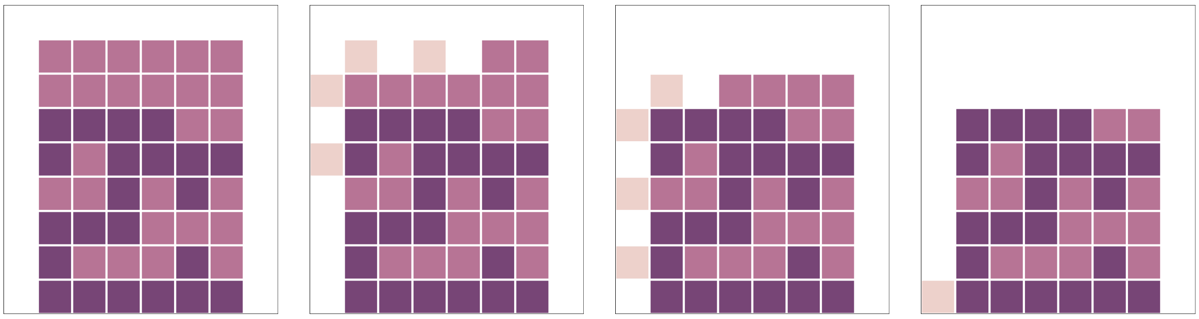


Fig. 5.3 Example of two rows composed entirely of excluded modules being removed from a configuration. Animated versions of the reconfiguration process available online [133].

5.3 Solution Design

The main phase consists of modules using message passing to assign all remaining excluded modules a *priority order index*, P before removing themselves from the structure. This index dictates the order in which modules remove themselves, and is decided before any movement takes place.

Following the removal of entire rows of excluded modules, the challenge of ensuring the remaining excluded modules do not travel further than absolutely necessary becomes more complex, especially under the constraint that the movement and ordering process remain distributed. The movement algorithm that the modules use is deterministic, so their preferred route can be deduced. Using this knowledge, messages can be passed through the excluded modules following a set of rules, assigning each a value indicating its position in the removal order. This value is known as the priority order index, P, and will have a unique value between 1 and $|V^{\text{exc}}|$, where V^{exc} is the set of excluded modules in the configuration. All P values are assigned before any excluded modules begin moving.

Throughout the self-reconfiguration process, the excluded modules will exist in a number of states, each of which has different behaviours and responses to messages. As such we can consider a module to execute a different algorithm depending on the state it is in. Figure 5.4 shows an overview of the four states and how a module transitions between them.

An excluded module will always begin in the **idle** state. It will then transition to either the **prioritised** state if it receives a *priority* message or the **waiting** state if it receives a *wait* message. A module in the **waiting** state will transition to the **prioritised** state upon receiving a *priority* message. Finally, modules that are in the **priority** state will enter the **moving** state once they receive a *move* message.

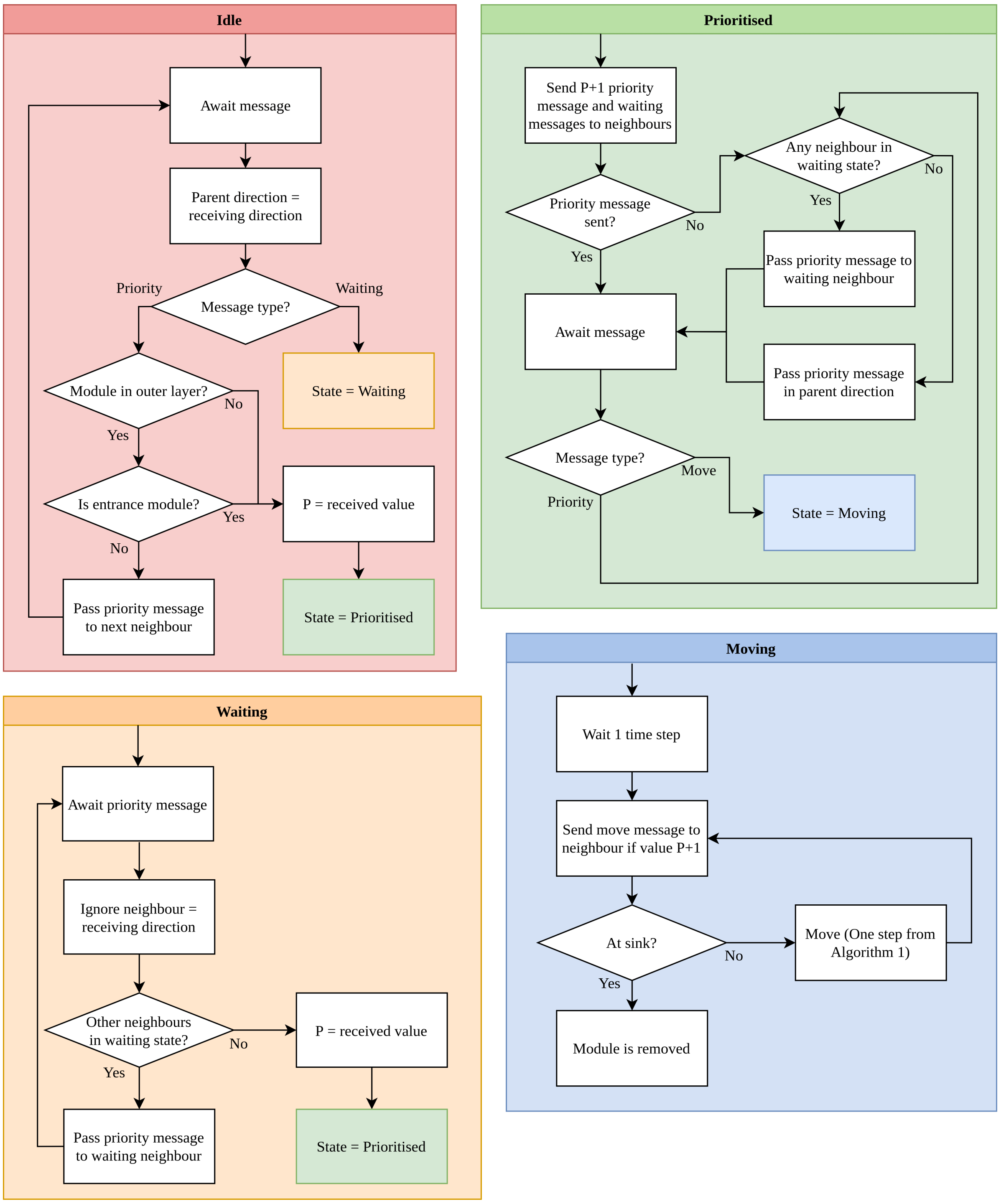

Idle
Await message
Parent direction = receiving direction
Message type?
Priority
Waiting
Module in outer layer?
No
Yes
State = Waiting
Is entrance module?
Yes
No
P = received value
Pass priority message to next neighbour
State = Prioritised
Prioritised
Send P+1 priority message and waiting messages to neighbours
Any neighbour in waiting state?
No
Yes
Priority message sent?
No
Yes
Pass priority message to waiting neighbour
Await message
Pass priority message in parent direction
Message type?
Move
Priority
State = Moving
Moving
Wait 1 time step
Send move message to neighbour if value P+1
At sink?
No
Yes
Move (One step from Algorithm 1)
Module is removed
Waiting
Await priority message
Ignore neighbour = receiving direction
Other neighbours in waiting state?
No
Yes
P = received value
Pass priority message to waiting neighbour
State = Prioritised

Fig. 5.4 Overview of the behaviour of each module state and the way in which a module transitions between them. Boxes indicate actions for the module to take, while diamonds indicate decisions, with the relevant options attached to the arrows.

To enable the distributed behaviour there are a number of messages which can be passed by each module in order to assign *P* values and switch between states. They are summarised here:

Priority The message type which contains the *P* value to assign to the receiving module, or be passed to the module which should be assigned the next priority order index.

Wait The message that is sent to a module to transition it to the **waiting** state.

Move The message sent to an excluded module that will transition it to the **moving** state, upon which it becomes an active excluded module.

Using these messages, modules will transition between the states shown in Figure 5.4, as well as assign *P* values to determine the removal order. A more detailed explanation of each state is given here, before an example of the process is presented in Section 5.4.

5.3.1 Idle State

An excluded module begins in the **idle** state, where it has not yet received a message from any neighbours and as such has no *P* value assigned to it. Upon receiving a message, the module sets its *parent direction* as the direction from which this message was received. If the module is the entrance module (explained below) then the parent direction is set as its free face, to ensure that the correct neighbours are given messages when in the **prioritised** state (See Section 5.3.2). The behaviour is then determined based on the message type received.

If the message is a *wait* message, the module in question transitions to the **waiting** state, detailed in Section 5.3.3.

If the message is a *priority* message then the module action depends on whether it is in the outer layer or not. If it is not in the outer layer then the module assigns its own *P* value as the value contained within the *priority* message, it then enters the **prioritised** state, explained in Section 5.3.2. However, if the module is in the outer layer then it must decide whether it is the *entrance* module or not. The entrance module is defined as the excluded module within a cavity that has the shortest path to the sink. Figure 5.5 shows examples of this, where the marked modules are the entrance modules for their respective cavities. To deduce whether it is the entrance module or not, the module in question will check whether the next neighbour in the outer layer is also included. If so, it is the entrance module, so enters the **prioritised** state. Otherwise, the next neighbour is also excluded, so it passes the *priority* message to its next neighbour and resume awaiting a message in the **idle** state.

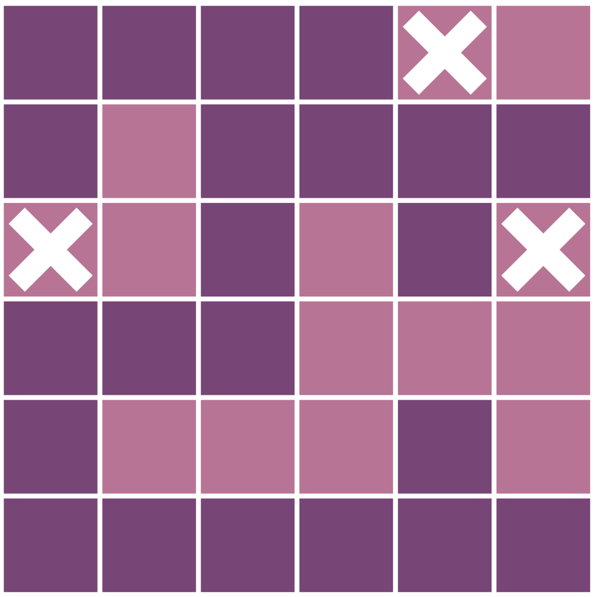


Fig. 5.5 Illustration of which excluded module acts as the entrance module for each cavity, some of which have multiple excluded modules in the outer layer.

If the next neighbour in the outer layer is an included module, then the module in question must be the entrance module for the cavity. It assigns its own P value to be the value associated with the message it received and enters the **prioritised** state, explained in Section 5.3.2.

5.3.2 Prioritised State

Once a module sets its P value, it is in the **prioritised** state, where it can pass messages to help assign other excluded modules their priority values and waits to begin moving.

When the module first enters the **prioritised** state, it sends messages to its neighbours not in the parent direction. Messages are only sent to excluded neighbours[2] and the type is determined by where the neighbour is, relative to the parent direction, as shown in Figure 5.7.

The neighbours are assessed in a clockwise manner, starting with the first after the parent direction. The first excluded neighbour that the module assesses is sent a *priority* message with the value $p+1$, provided it is not the final neighbour to be assessed. This can be seen in Figures 5.7(a)-(d). After a priority order index has been passed, all remaining excluded neighbours are sent a *wait* message, as in Figures 5.7(a), (b) and (d). The final neighbour is always told to wait, seen in Figures 5.7(a), (b) and most obviously in (e). This avoids a

[2]Except for the entrance module, as included neighbours are required to help pass the *priority* message to the next cavity once a cavity is fully assessed, shown in Section 5.4 and further explained below.

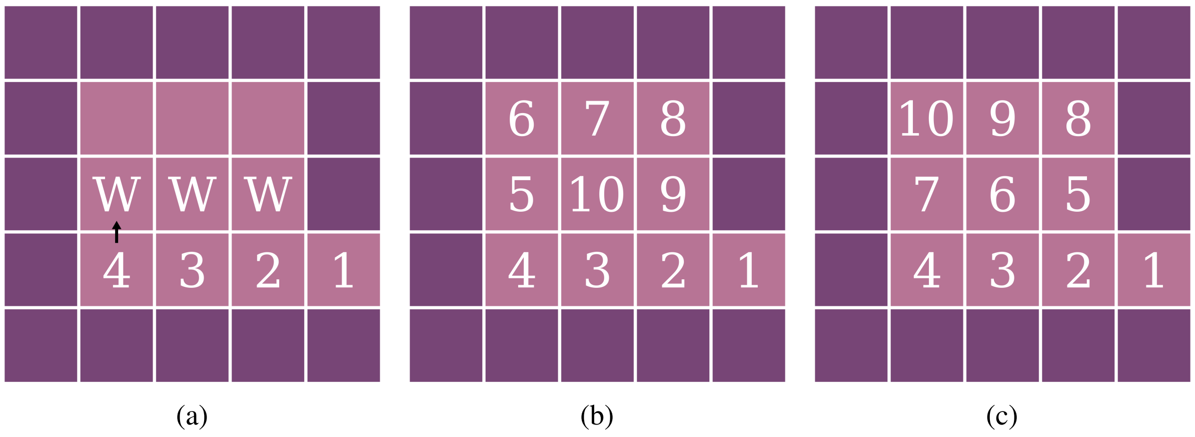
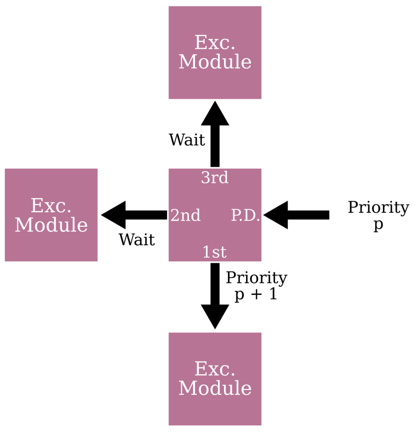


Fig. 5.6 An example of why an excluded module must send a *wait* message to an excluded neighbour in the final direction, and why modules that are in the **waiting** state must check for neighbouring waiting modules before assigning P values. (a) shows the moment at which the proposed behaviour is important. (b) shows an unfeasible priority order when this is not done. (c) shows the correct priority order when following this rule.

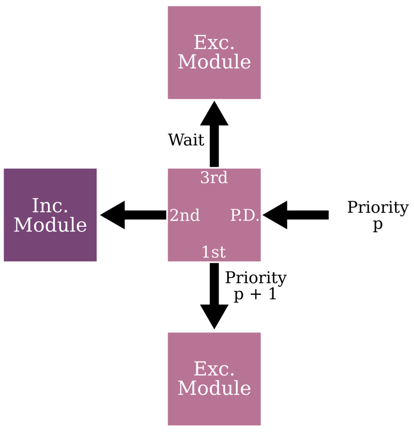
situation where modules are assigned a priority value in such a way that excluded modules fully surrounding another excluded module are not prioritised over the central one. If the final neighbour of the module in Figure 5.6(a) with $P = 4$ (the neighbour to the North) is immediately given a *priority* message, rather than first being given the *wait* message, then the final priority order will be as shown in Figure 5.6(b). As before, this is an unfeasible priority order, as the module with $P = 10$ will not be supported when the other modules have removed themselves. Figure 5.6(c) shows the final priority order following the module with $P = 4$ sending a *wait* message to its Northern neighbour instead, as in Figure 5.6(a). As can be seen, this yields a feasible removal sequence.

If the module has no neighbours that can be sent a *priority* message, as in Figures 5.7(e) and (f), then the sending of the *priority* message is unsuccessful. In this case the module will check if it has a neighbour in the **waiting** state that can be sent the priority message, as in Figure 5.6(a). If it has no neighbours in the **waiting** state then the *priority* message is instead passed in the parent direction. By doing this, the *priority* message will propagate through the system towards the nearest module in the **waiting** state, and the assignment of P values will continue. This is demonstrated in Section 5.4.

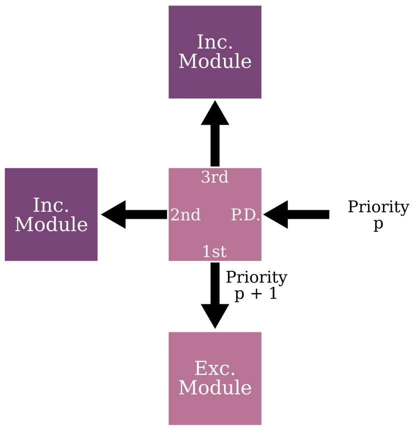
Once the module has done this initial passing of messages, it awaits a message from a neighbour. In the case of a *priority* message, the same behaviour as above is exhibited, passing the *priority* message to either a neighbour in the **waiting** state or the module in the parent



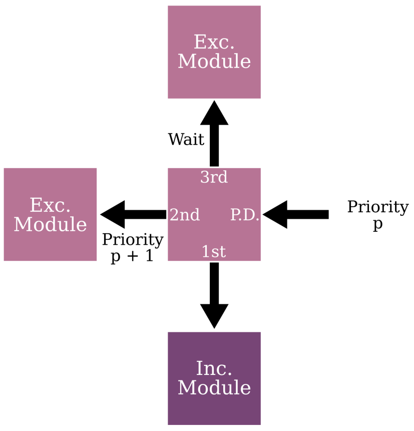
(a)



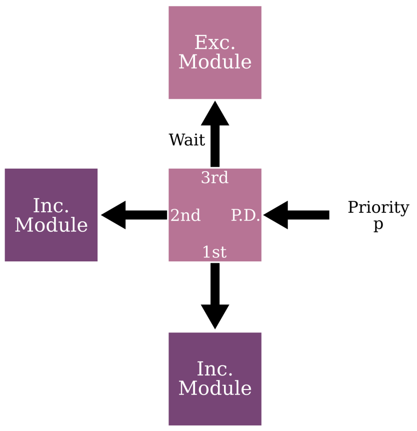
(b)



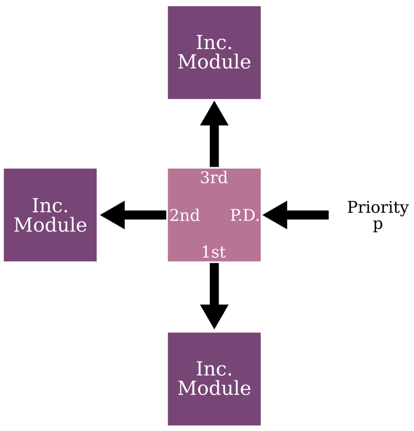
(c)



(d)



(e)



(f)

Fig. 5.7 Examples of situations where an excluded module that is not in the **waiting** state would send *priority* and *wait* messages to neighbours in response to receiving a *priority* message and setting its own priority order index to the received value. (P.D. stands for parent direction.)

direction. In the case of a *move* message, the module transitions to the **moving** state, explained in Section 5.3.4.

Not shown in the diagram but important to the operation of the solution is the behaviour of the entrance module upon receiving a *priority* message when already in the **prioritised** state. In this case, the module passes the *priority* message around the outside layer, via included modules, to enable the next cavity to begin the priority order index assignment process. Once module $v_{\mathtt{SW}}$, which is the module closest to the sink, receives a *priority* message and is either in the **prioritised** state or is an included module, then all modules must have been assigned a P value. It then sends a *move* message around the outer layer to the module with a priority order index value of $P = 1$ and the excluded modules begin the process of removing themselves.

5.3.3 Waiting State

The **waiting** state is entered when a module receives a *wait* message. This indicates that the module has been considered for priority index assignment but the module that sent the message has neighbours that should have a higher priority.

When in the **waiting** state the module is awaiting a *priority* message from a neighbour. Upon receiving this message the module first checks whether it has any waiting neighbours of its own, discounting the direction that the *priority* message was received from. If so, the *priority* message that it received is passed to that waiting neighbour and the module returns to waiting for a *priority* message.

If, however, the module has no waiting neighbours in the remaining directions, then it assigns its own P value as the value contained within the *priority* message, it then enters the **prioritised** state, explained in Section 5.3.2.

5.3.4 Moving State

Following the assignment of all P values, excluded modules can begin moving. When they do so they are in the **moving** state, where they will locomote to the sink and send a *move* message to transition the next highest priority module to the **moving** state. Each module must begin moving after the module that is prioritised one value higher than itself does to preserve the priority order. For example, a module with a priority order index of $P = 2$ leaves after the module with $P = 1$.

When a module receives a *move* message and enters the **moving** state, it first waits one time step. It is required to wait one time step so that the number of time steps between two active excluded modules arriving at the same position is maintained at a value of two.

The module then locomotes to the sink using the distributed active excluded algorithm, Algorithm 1 (see Section 3.4.1). Upon reaching the sink a module is considered removed. Once all excluded modules have reached the sink then the self-reconfiguration process is complete.

5.3.5 Distributed Ordering Sequential Active Subtraction (`DO-SAS`)

As previously outlined, `DO-SAS` is not focussed on here, due to `DO-PAS` negating the downsides of `PAS` whilst improving on the performance. However, for completeness, `DO-SAS` would operate in the same manner to `DO-PAS` for the majority, differing when modules enter the **moving** state. Here the *move* message would be sent only once a module reaches the sink location, rather than as soon as possible, and this message would be propagated through the system by flooding, to reach the module with the next priority order index, irrespective of where that module is located.

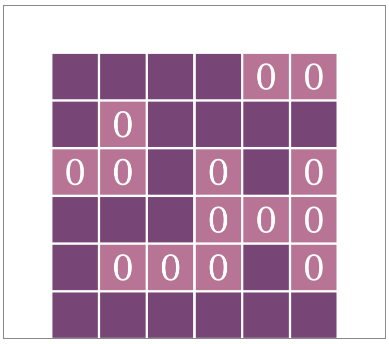
By combining the states shown here and the transitions between them, a complete solution to the self-reconfiguration problem is produced. This is demonstrated in the next section and subsequently proven in Sections 5.5 and 5.6.

5.4 Example Priority Order Index Assignment Process

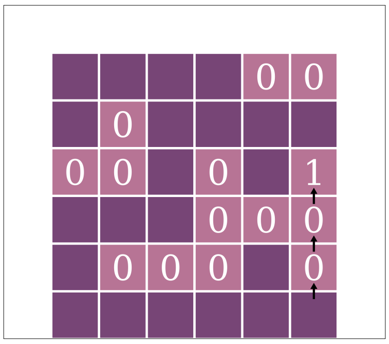
To demonstrate how the state behaviours and transitions enable self-reconfiguration, an example of a configuration using the proposed design to assign priority order indexes and begin reconfiguration is shown in Figure 5.8 and explained here.

All modules are initially in the **idle** state and have not been assigned a P value ($P = 0$). In Figure 5.8(b), a *priority* message with a value of 1 is sent by $v_{\texttt{SW}}$. As the first two excluded modules have excluded neighbours in the outer layer, they pass this message until reaching the entrance module of the cavity. Upon receiving this message, the entrance module transitions to the **prioritised** state and sets its priority order index, $P = 1$. It then sends a message to its only excluded neighbour, to the South. That message is a *priority* message and contains the value $P + 1$, which in this case is 2.

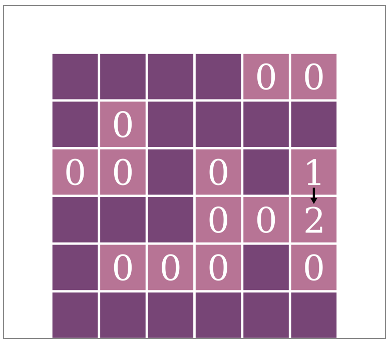
In Figure 5.8(c), the module receives this message, assigning a priority order index of $P = 2$ to itself. Figure 5.8(d) shows that the module with $P = 2$ has two excluded neighbours.



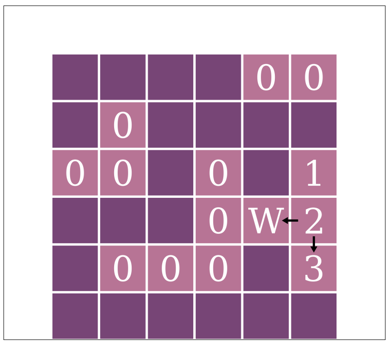
(a)



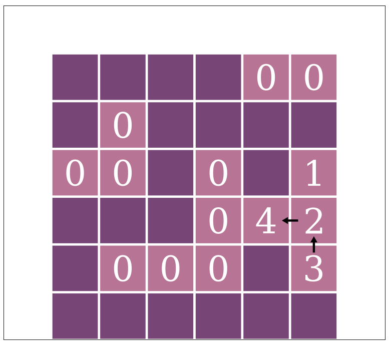
(b)



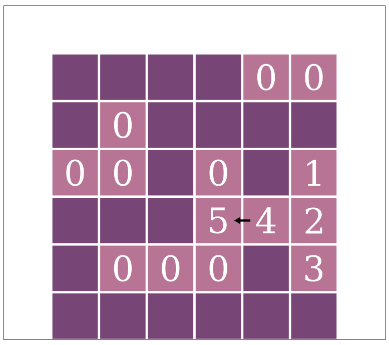
(c)



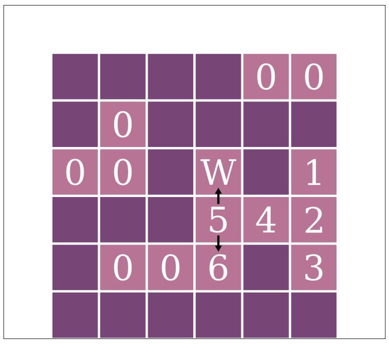
(d)



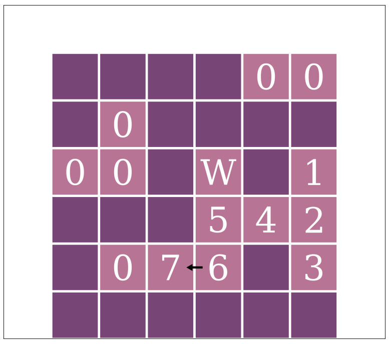
(e)



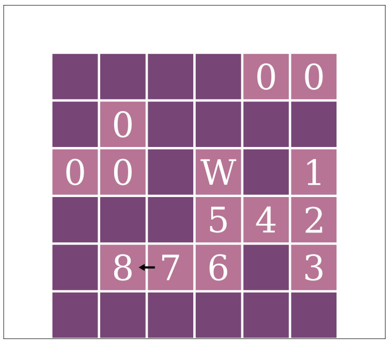
(f)



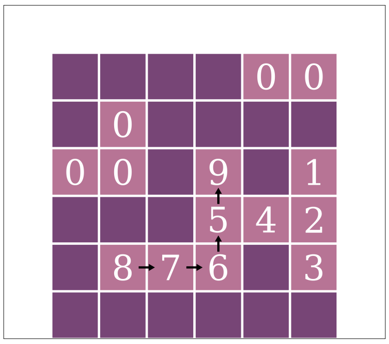
(g)



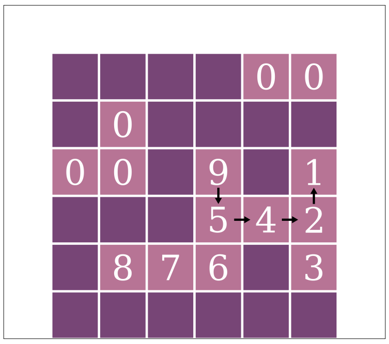
(h)



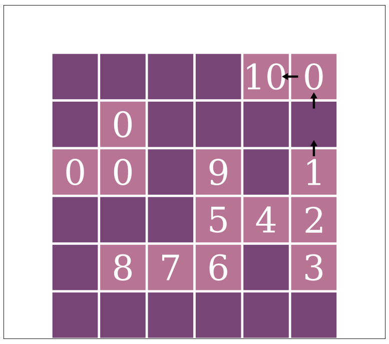
(i)



(j)



(k)



(l)

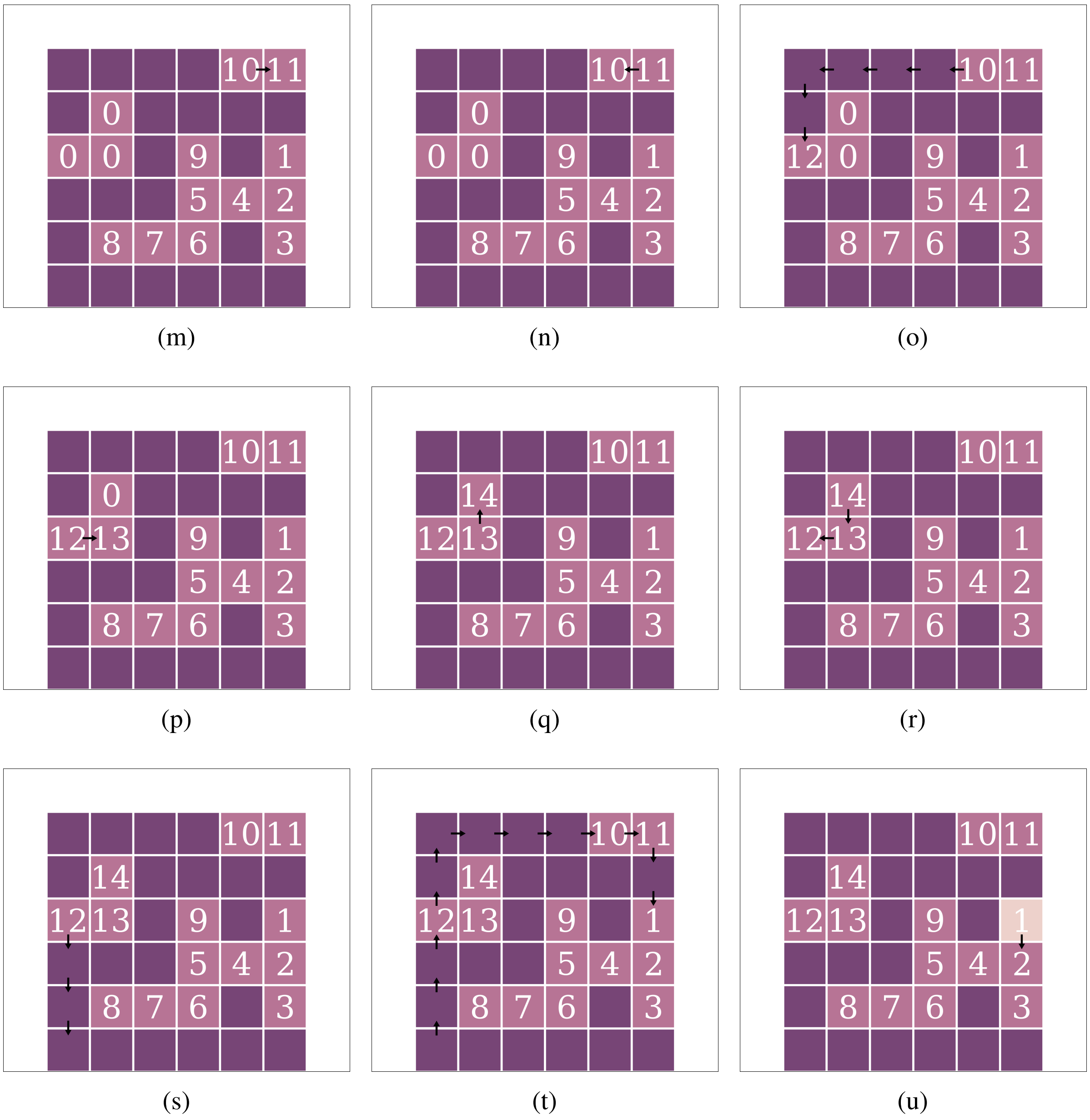


Fig. 5.8 A visual representation of excluded modules initialised with values of $P = 0$ and in the **idle** state, passing messages and transitioning between states to set their priority order indexes and successfully self-reconfigure.

It received a message from, and subsequently set its parent direction as, North. As such it will prioritise its neighbours in the order East, South then West (See Section 5.3.2). It can be seen that the module sends a *priority* message South, leading the module below to enter the **prioritised** state and assign itself $P = 3$, and a *wait* message to the West, causing that module to enter the **waiting** state.

As can be seen in Figure 5.8(d), the module with $P = 3$ has no unassigned excluded neighbours and no neighbours in the **waiting** state, so will send a *priority* message with a value of 4 in its parent direction, in this case North. The neighbour to the North, that is the module with $P = 2$, has a neighbour in the **waiting** state, so passes the *priority* message to that neighbour, which sets its priority order index $P = 4$.

This continues in the same way up until the stage shown in Figure 5.8(k), at which point all modules within the first cavity have been assessed. At this point, each module within the cavity is in the **prioritised** state, so will pass a *priority* message with a value of 10 through parent directions until reaching the entrance module. As previously described, the entrance module must then pass the *priority* message through included modules to find the next cavity, and subsequently the next entrance module, as seen in Figure 5.8(l).

The process continues until all excluded modules have been assigned a priority order index (are in the **prioritised** state) and module $v_{\texttt{SW}}$ receives the *priority* message, as in Figure 5.8(s). At this point module $v_{\texttt{SW}}$ sends a *move* message around the outer layer, intended for the excluded module to be removed first, the one with a priority order index of $P = 1$, shown in Figure 5.8(t). This module then enters the **moving** state, sending a *move* message to the module with the next priority order index before locomoting to the sink, as can be seen in Figure 5.8(u). Snapshots of the modules removing themselves can be seen in Figure 5.10(a).

As can be seen from this demonstration, the states and messages introduced in Section 5.3 create a distributed solution to the problem introduced in Section 1.2. This is further proven in the following sections.

5.5 Mathematical Analysis

This section formally analyses the correctness of `DO-PAS`, and characterises its run time performance.

Lemma 8. *Any excluded module in the initial configuration must either have a free face, or be connected to a module with a free face, via a path of excluded modules.*

Proof. The statement is proven by contradiction. It is assumed that there exists an excluded module, v, for which the statement is not true. S_v denotes the set of excluded modules that are connected to v via a path of excluded modules. As $v \in S_v$, $S_v \neq \emptyset$. For the contradiction to hold, none of the modules in S_v can have a free face. This means that the modules in S_v must be fully enclosed by included modules, and potentially the ground. As the modules in S_v are fully enclosed by included modules and the ground, once all excluded modules are removed, the resulting configuration, which by definition is the desired configuration, would contain a hollow space. However, by definition the desired configuration cannot contain hollow spaces. Therefore, this desired configuration cannot exist, meaning that S_v cannot exist, and the Lemma must be true. □

Lemma 9. *Assuming the entrance module of a given cavity receives a* priority *message of value n, the excluded modules of this cavity are assigned unique priority order index values of* $n, n+1, n+2, \dots, n+|V_{cav}^{exc}|$*, where* V_{cav}^{exc} *is the set of excluded modules within the cavity.*

Proof. If the initial configuration does not contain a cavity, there is nothing to be shown. A particular cavity is considered. Let $\varepsilon \in V_{\text{cav}}^{\text{exc}}$ denote the module with the shortest path to the sink, that is, the entrance module. Module ε receives a *priority* message of value n. This module is considered to be the root of a tree. The root cannot be in the waiting state, and hence assigns its priority order index to n. If the cavity is of size one, there is nothing more to be shown. Otherwise, the root will consider all its neighbouring excluded modules to be nodes, in a manner similar to a depth-first search[3]. According to Lemma 8, all excluded modules are connected via a path of excluded modules to a module with a free face, and hence they are also connected via a path to the entrance module[4]. As such, all excluded modules will be discovered by the tree search (a). Every time a priority order index is assigned to a node of the tree, the *priority* message value is incremented. As the depth-first search explores the nodes sequentially, no two nodes in the cavity can have the same priority order index (b).

Therefore, *all* excluded modules of the cavity are assigned priority order index values (a), each of which is unique and incremented by one after the node before it (b), giving the modules values of $n, n+1, n+2, \dots, n+|V_{\text{cav}}^{\text{exc}}|$. □

Lemma 10. *At no point during the removal of excluded modules from a cavity does a module become unsupported, rendering the overall structure unfeasible.*

[3]The variation comes from assigning a waiting state to any neighbours that are not immediately explored, or that are in the final direction to be checked. Further explanation can be found in Section 5.3.

[4]All modules in a cavity that have a free face are directly connected to one another, one of which is the entrance module. It is not possible to have two entrance modules as this would render the desired structure unfeasible.

Proof. The statement is proven by induction. Consider the tree that corresponds to the priority order index assignments, with the root being the entrance module of the cavity (as defined in Lemma 9). Let S_n be the set of modules in this tree for which the shortest path to any leaf is of length n. $n = 0$ is considered as a base case. The modules in S_0 have a path of length 0 to a leaf, so are the leaves themselves. At the time the tree was constructed, the modules that end up as leaves are exactly those that must have initiated *return* messages (had they sent *priority* messages then this would have created a child node). The cases in which a module can send a *return* message are as follows:

- The remaining neighbours that do not have a priority order index assigned are included modules. In this case, the module is implicitly supported by the included modules.
- The module has at least one excluded module as a neighbour, and all neighbouring excluded modules are in the *waiting* state. In this case, the module already has a priority order index, but the waiting neighbours do not yet. According to Lemma 9, each module has a unique priority order index that increments as they are assigned, therefore these neighbours must have higher priority order indexes and will support the module in question.
- The module rests upon the ground and has no neighbours to pass a *priority* message to. In this case, the module is supported by the ground.

From this it can be seen that all modules in S_0 are supported until they begin moving. Assume that the statement is true for $n = k$, that is all modules in S_k are supported. For $n = k+1$, a module in S_{k+1} is considered, and labelled m. As $n = k+1 \geq 1$, this m cannot be a leaf. The shortest path from module m to any leaf is of length $k+1$. As a result, module m must have a child node for which the shortest path to any leaf is k. As such, module m will be supported by a neighbour in S_k until it begins moving. Therefore, the induction is complete. □

Theorem 11. *At no point during the reconfiguration of an entire structure does a module become unsupported.*

Proof. If a configuration has entire rows of excluded modules at the top, then the process of excluded row removal begins reconfiguration. This is achieved using a top-down approach, as was employed in `SAS` and `PAS`. It was shown in Lemma 2 that modules can be removed in this way without the structure becoming unfeasible. Following this, cavities are identified, and the modules removed from each in turn. Lemma 10 shows that during the removal of modules from a cavity, no module becomes unsupported. As can be seen in Lemma 3, the movement algorithm (Algorithm 1) gives a feasible path for each module to reach the sink. Hence, modules are able to move themselves from their initial positions to the sink while remaining supported throughout. □

Theorem 12. *Using* `DO-PAS`*, the time-based performance grows linearly with the size of the configuration.*

Proof. Consider a configuration, V_0, of width W and height H, with $|V_0^{\text{exc}}|$ excluded modules. The first module to move when using `DO-PAS` is the entrance module furthest from the sink. In the worst case, this module is in the South-East corner of the configuration. The distance that this module would have to travel is $2H+W+2$, taking one time step per movement. As shown in Lemma 9 the modules within each cavity form a connected tree. As such, each module within a cavity can follow its predecessor, maintaining a gap of two time steps. It can be seen that, as the cavities are assessed starting with the furthest from the sink then moving closer, one by one, that the modules from cavities further from the sink will pass over the entrance module of any cavities closer to the sink. When this happens, an entrance module is able to follow the final module from the preceding cavity[5]. Moreover, all active excluded modules move at the same speed, one step at a time. Therefore, all modules following the first one will arrive at the sink two time steps after the preceding module. As such the time it will take to remove all remaining excluded modules after the first is $2(|V_0^{\text{exc}}|-1)$. The total time will then be

$$2H+W+2+2(|V_0^{\text{exc}}|-1) = O\big(|V_0|\big) \tag{5.1}$$

The time complexity of any algorithm cannot be better than $\Theta(|V_0|)$ as it takes at least $2n$ time steps for n excluded modules to arrive at the sink. As such, while the algorithm is not optimal, it is asymptotically optimal. □

5.6 Simulation Studies

This section presents a number of studies, carried out in simulation, proving the effectiveness of `DO-PAS`, as well as `DO-SAS` for comparison. The results demonstrate the improved performance when compared to `PAS` and `SAS`. The solutions are implemented in Python, and allow a user to input a desired configuration (as in Figure 4.2), or generate a random configuration of a given size and ratio of included to excluded modules. The program is called Modular

[5]As the value of the priority order index is retained between cavity assignments (see Section 5.3), the final module from one cavity will have a priority order index one greater than the entrance module of the next cavity, so each entrance module will know when to begin moving.

Active Subtraction Simulator for 2-Dimensions (MASS2D) and is available as an open source project [131].

The performance of each solution is measured by the number of movements required to remove all excluded modules, referred to as time steps. As previously outlined, the time and energy required to pass messages is negligible when compared to the requirements for a module to move, so time steps increment only when a module moves. In general, a simulation is considered a success if all excluded modules are removed without the configuration becoming unfeasible at any point.

The parallel best-case time (BCT_P) and sequential best-case time (BCT_S) are also compared against, as in Section 4.4. The BCT_P and BCT_S are calculated by first using the A* algorithm to obtain the shortest path each excluded module could take to the sink, while circumnavigating included modules. To obtain the BCT_S these values are summed together. However, to calculate the BCT_P, these values are combined, with gaps of two time steps between module arrival times being ensured. The formal calculation is more thoroughly detailed in Section 4.4. The BCT_P and BCT_S may not be feasibly achievable, as the path determined by A* may rely on excluded modules that could have already removed themselves in practice.

5.6.1 User-Defined Configurations

In this section, a number of user defined configurations are simulated to assess the effectiveness of `DO-PAS`, and to compare with other examples of subtractive reconfiguration from Chapter 4 and existing works.

Figure 5.9 shows `DO-PAS` being employed by the excluded modules to reconfigure from an initial configuration of a 4×4 starting square to a simple U-shape, similar to the one considered by Gauci et al. [15], and also in Section 4.4.1. The performance of the simulation using `PAS` took a minimum of 17 time steps. When employing `DO-PAS`, the reconfiguration also takes 17 time steps. This is due to the fact that the order in which the modules are removed in Section 4.4.1 is the same as the order that given by `DO-PAS`, as can be seen when comparing Figures 4.3(b) and 5.9. This can be attributed to the simplicity of the desired configuration, comprising only a single, rectangular cavity. The BCT_P also gives a value of 17 time steps, so in this case the optimal performance had previously been achieved. The same can be said when comparing `SAS`, `DO-SAS` and the BCT_S, which all take 51 time steps to fully reconfigure. Again, this suggests that the order of all is optimal, as well as the route taken by the modules.

A more complex shape, with multiple cavities and less simple geometry is the one shown in Figures 5.1, 5.3 and 5.8. The reconfiguration to this shape from a 5×5 starting square using

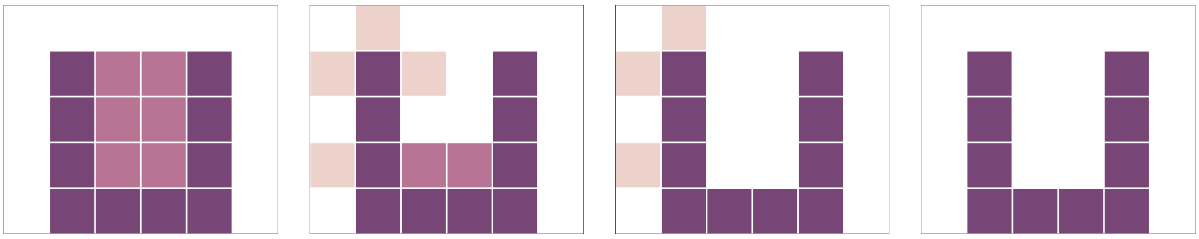
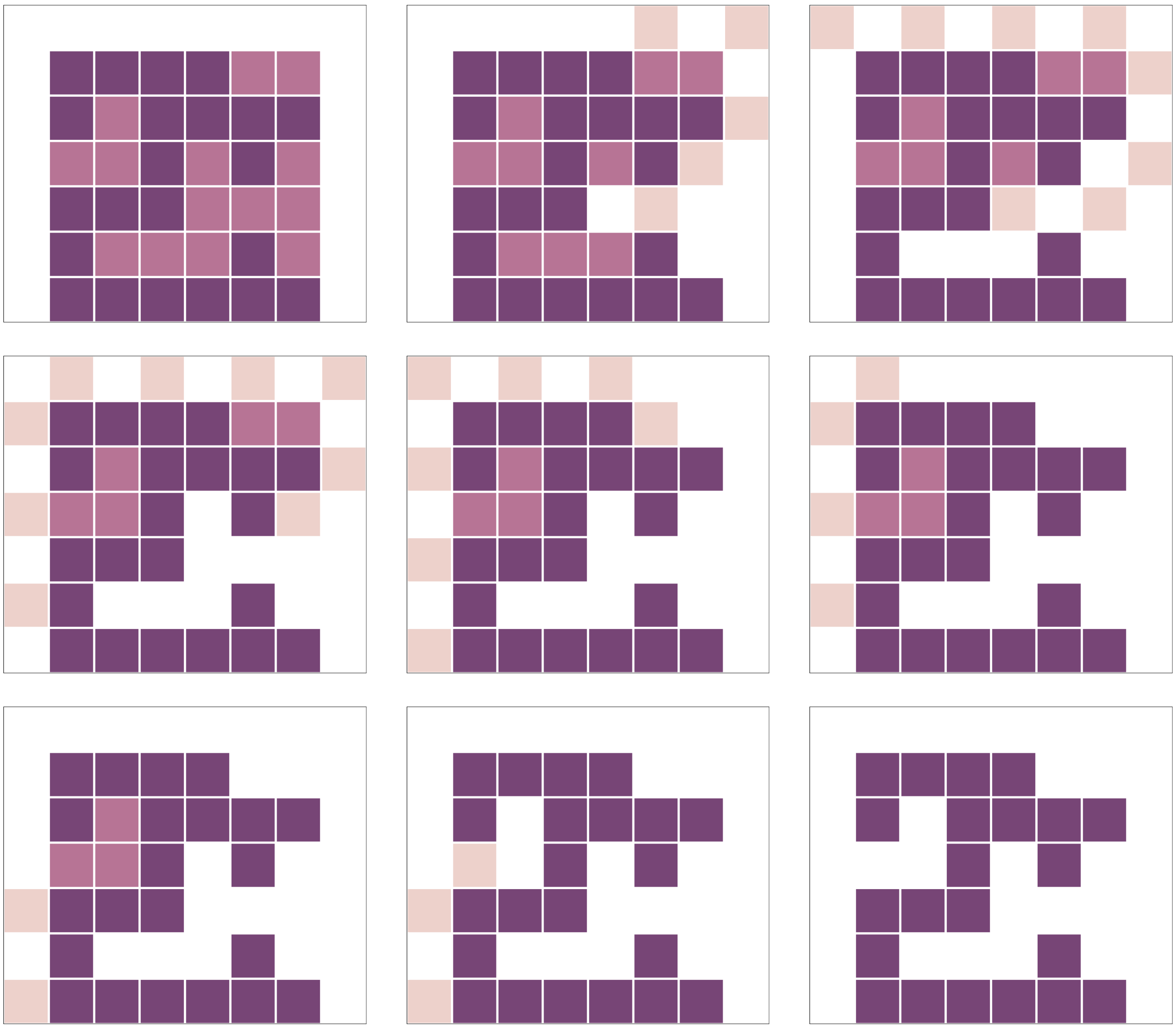


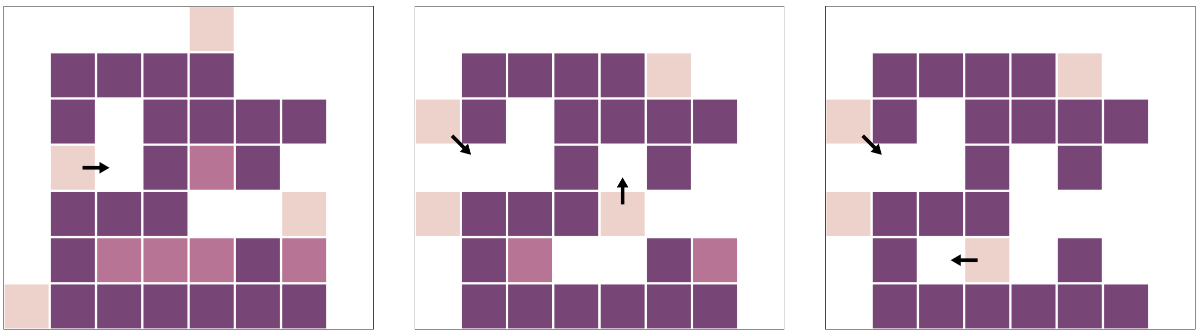
Fig. 5.9 Example of excluded modules employing DO-PAS to self-reconfigure from an initial configuration of a 4×4 square to a U-shape. Shown at time steps 0, 6, 12 and 17. Animated versions of the reconfiguration process available online [133].

DO-PAS can be seen in Figure 5.10(a). Using DO-PAS, the reconfiguration process takes a total of 43 time steps. However, as can be seen in Figure 5.10(b), there are situations where the performance when using PAS will be much worsened. This is confirmed by the reconfiguration time of PAS, requiring 98 time steps, meaning DO-PAS grants a reduction of 56%. The $\mathrm{BCT_P}$ is 33 time steps. The shorter time can be attributed to the fact that the first module to be removed with DO-PAS requires 17 time steps to reach the sink (See Figure 5.8(u)), with subsequent modules only leaving afterwards. Conversely, the calculation for best-case parallel time allows the module closest to the sink to move first, meaning the first module reaches the sink in 4 time steps for the given desired configuration. In this situation, the excluded modules that move later would not feasibly be able to follow their shortest paths in reality, as they would have to enter the cavity left by the modules closest to the sink, as in Figure 5.10(b). Applying SAS to the desired configuration given in Figure 5.10 yields a performance of 292 time steps. Comparatively, using DO-SAS reduces this performance to 220 time steps, a reduction of 25%. The $\mathrm{BCT_S}$ also gives a performance of 220 time steps, suggesting that the order determined by DO-SAS is optimal for the formulated problem.

Another configuration with large cavities that affect the time-based performance is the large humanoid from Section 4.4.1, inspired by the work of Gilpin et al. with the *Miche* and *Robot Pebbles* systems [12, 16]. PAS yields a reconfiguration time of 661 time steps. However, when using DO-PAS, the excluded modules are able to remove themselves from the structure in only 520 time steps, a reduction of 33%. Snapshots of the reconfiguration process using DO-PAS can be seen in Figure 5.11 The $\mathrm{BCT_P}$ is 487 time steps, but again would not be possible as this performance relies on the modules closest to the sink being removed first, which would not allow the later modules to follow the shortest path in practice. Note again that the difference between the performance of DO-PAS and the $\mathrm{BCT_P}$ is the difference between the time it takes the first entrance module to reach the sink, which is the first module to begin moving with DO-PAS, and the time it takes the module closest to the sink to reach the sink, as the calculation of the $\mathrm{BCT_P}$ takes as the first module. This suggests that, by using DO-PAS, the modules follow



(a)



(b)

Fig. 5.10 Example of active subtraction, reconfiguring from a 5×5 starting square to a given shape. (a) DO-PAS shown at time steps 0, 6, 11, 16, 25, 30, 35, 39 and 43. (b) PAS shown at time steps 28, 50 and 66, demonstrating situations where PAS causes modules to take longer routes than necessary. Animated versions of the reconfiguration process available online [133].

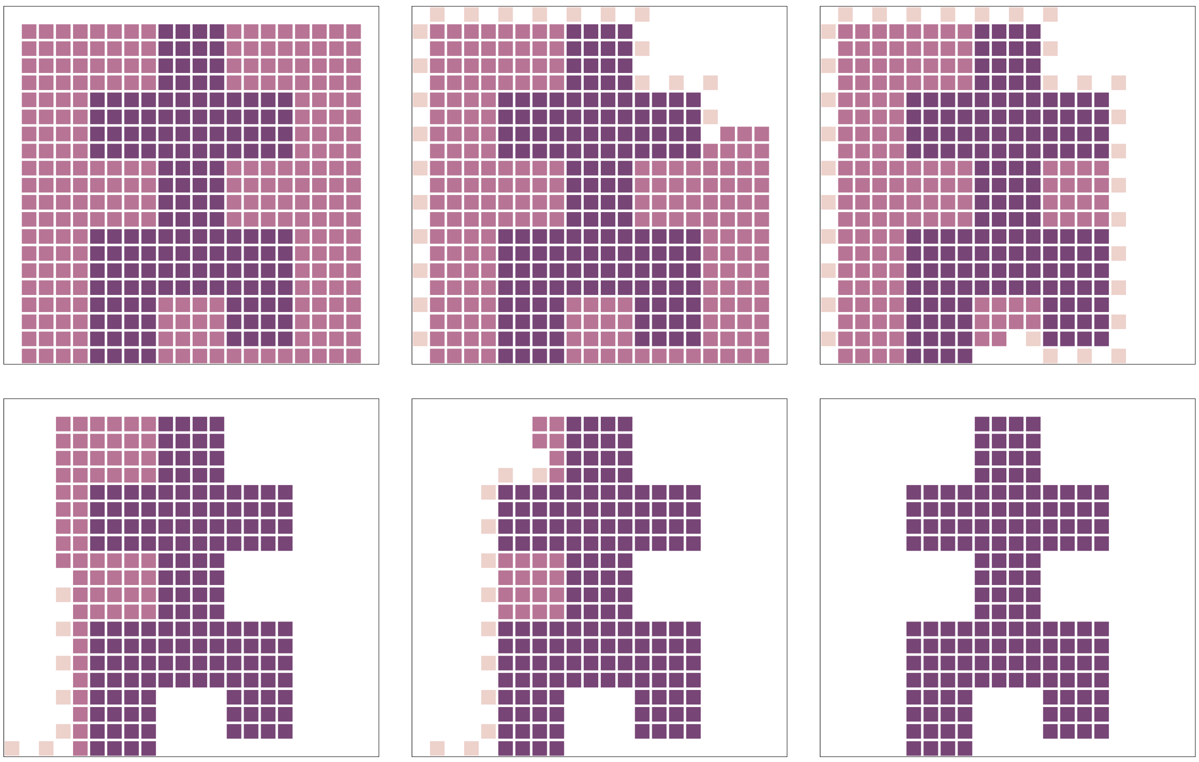


Fig. 5.11 Snapshots of the reconfiguration process when employing DO-PAS, beginning from a 20×20 initial configuration and ending with a large humanoid structure. Shown at time steps 0, 71, 181, 386, 453 and 520. Animated versions of the reconfiguration process available online [133].

the optimal routes, but that it is not possible to follow these routes in the order given by the $\mathrm{BCT_P}$ calculation. When using SAS, the performance was considerably worse for the large humanoid structure, taking 9554 time steps. Using DO-SAS, the performance is reduced to 8190 time steps, a reduction of 14%. This is again the same number of time steps as given by the best-case sequential time.

In Section 4.3, the worst-case performance of PAS is determined, using a specific desired configuration to demonstrate it. This can be seen again in Figure 5.12(a). Figure 5.12(b) shows why this configuration is so time-consuming for PAS, where modules must fully traverse the length of the upper cavity, and each subsequent module must delay its movement while the preceding module navigates the corridor. Figure 5.12(c) shows the order that the modules remove themselves in when DO-PAS is used. In this case, no module is required to traverse the cavity, greatly improving performance. The original reconfiguration time of 140 time steps is reduced to 44, a reduction of 69%. The $\mathrm{BCT_P}$ is 42 time steps. The reason for the difference of 2 time steps can be easily deduced by inspecting Figure 5.12(a), where it can be seen that the entrance module from the top layer of modules would require 2 time steps fewer to reach the

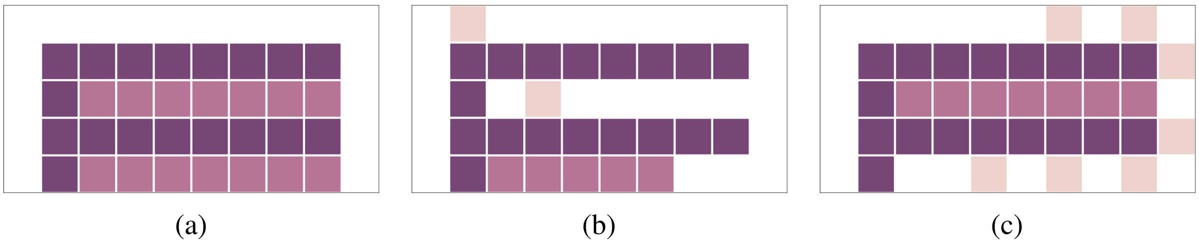


Fig. 5.12 Example of a desired shape with two long cavities, (a) shown at time step 0. Demonstrating how DO-PAS mitigates the time-based performance issues of PAS when traversing long cavities, (b) the reconfiguration using PAS is shown at time step 39, and (c) reconfiguration using DO-PAS at time step 8. Animated versions of the reconfiguration process available online [133].

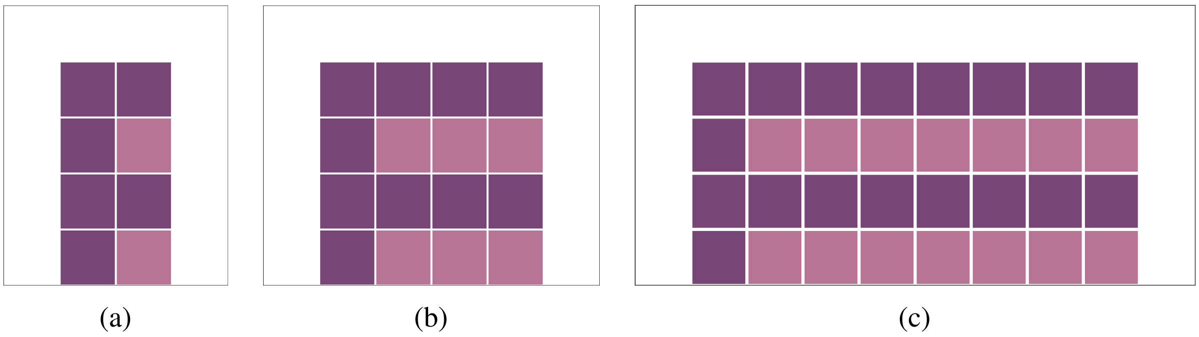


Fig. 5.13 Desired shapes with two long cavities of varying widths, simulated to assess the performance comparison between PAS in the worst-case and DO-PAS. The widths shown are (a) 2, (b) 4, and (c) 8.

sink than the bottom. However, as evidenced by comparing PAS and DO-PAS, if the top layer of modules moved first in practice then the lower layer would have to traverse the resulting cavity, greatly worsening the time-based performance.

To confirm that DO-PAS reduces the worst-case time-based performance of PAS from $\Theta(|V_0|^2)$, similar desired configurations are also simulated. For these configurations, the number of excluded modules in each of the two cavities is equal to one module less than the width. The height remains at four modules, while the widths vary from two modules to forty modules. A selection of these desired configurations is demonstrated in Figure 5.13. The results of this simulation can be seen in Figure 5.14. To account for the variable sizes of configurations and the number of excluded modules, values are normalised by dividing the total time steps required for reconfiguration by the number of excluded modules in the initial configuration. As can be seen, the normalised performance for DO-PAS does not worsen with increased widths, unlike PAS. For all widths, the BCT$_\mathrm{P}$ is two time steps shorter than the performance of DO-PAS. This affirms the aforementioned reasoning, as the time taken by the entrance module of each

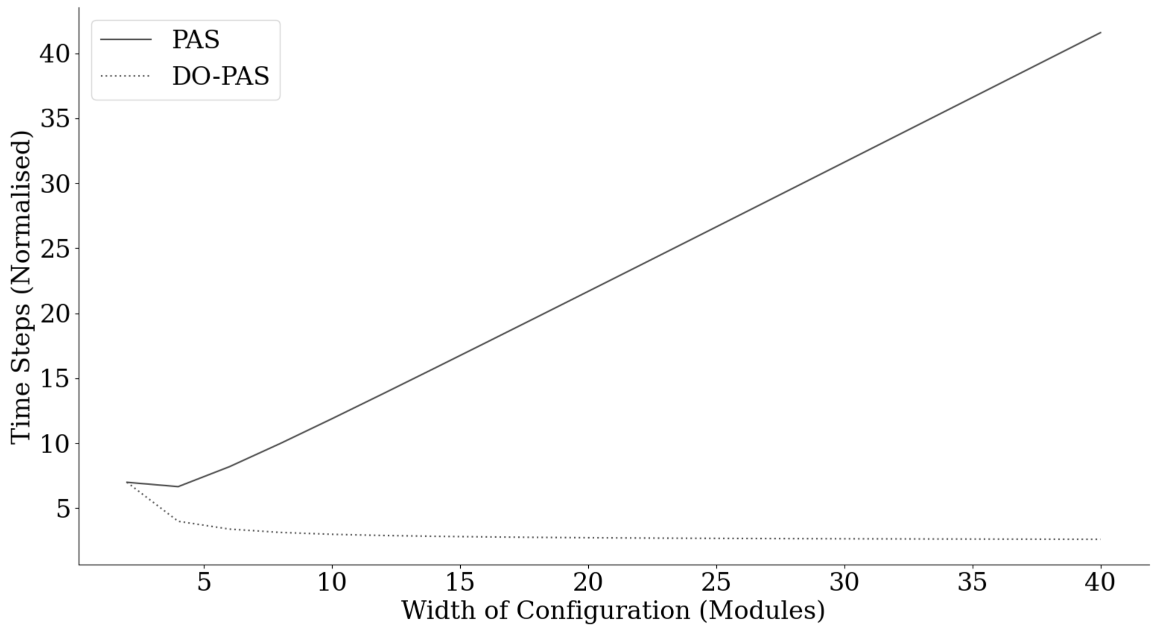


Fig. 5.14 Graph comparing the performance of DO-PAS to that of PAS for a selection of desired configurations with two long cavities of varying widths. Shown is the normalised time taken by excluded modules to remove themselves from initial configurations, leaving behind the desired configurations. Time steps are normalised by the number of excluded modules in the initial configuration.

cavity to reach the sink will differ by two time steps, irrespective of width. It can also be seen that, with wider configurations, the performance of DO-PAS tends towards two time steps. This is because all modules arrive two time steps after the preceding module, other than the first module which must traverse the configuration to reach the sink before any modules have been removed. For larger configurations, the number of modules that are removing themselves two time steps after the one before becomes larger and larger, so the affect that the length of time it takes the first module to reach the sink has on the overall performance is minimised.

The equations for the performance of each can be determined from the results. When the width of the configuration is equal to two, as seen in Figure 5.13(a), the number of excluded modules in each cavity is only one, so with PAS no modules have to wait for their predecessor to traverse the upper cavity. Because of this, the performance of PAS is simply the distance that the module furthest from the sink must travel, as it is able to begin moving at the first time step. If we define T as the total time taken to reconfigure and H and W are the height and width of the initial configuration, respectively, then the number of time steps required to traverse the exterior of the configuration will be:

$$T = 2H + W + 2, \tag{5.2}$$

where the extra 2 time steps are required for the movement around the corners of the structure. The time required to navigate the upper cavity must also be considered, which will be two further time steps:

$$T = 2H + W + 2 + 2. \tag{5.3}$$

Here the height and width are known to be four modules and two modules, respectively, so the performance can be calculated to be fourteen time steps.

However, once the number of modules in a cavity is greater than one, the performance of `PAS` degrades quadratically. This is due to the fact that modules in the lower cavity must traverse the upper cavity once the modules have been removed. Subsequently, each module in the lower cavity must wait for its predecessor to fully negotiate the upper cavity, further increasing the time. Taking this into account, the performance of `PAS` for the corridor configurations of various widths can be expressed as

$$T = \frac{|V^{\text{exc}}|^2}{2} + 5W + 2. \tag{5.4}$$

This confirms the fact that the worst-case time-based performance of `PAS` is relative to $\Theta(|V_0|^2)$. Once normalised by the number of excluded modules in the initial configuration, the relation between the width of the configuration and the time steps required becomes linear, as seen in Figure 5.14.

Conversely, for `DO-PAS` the equation characterising the performance is much more simple. As with the equation for the instance of `PAS` where the width was two, the amount of time steps it takes the module furthest from the sink to traverse the configuration is important. In `DO-PAS` though, there is no need to include time taken to navigate a cavity, as the module furthest from the sink is removed before an upper cavity is formed. For this reason, the time it takes the module furthest from the sink to traverse the configuration is the same as in Equation 5.2. The remaining modules, $|V^{\text{exc}}| - 1$, in the configuration are able to follow the preceding module out with a gap of two time steps between arrivals at the sink. As such the equation for the performance when using `DO-PAS` is

$$T = 2\big(|V^{\text{exc}}| - 1\big) + 2H + W + 2, \tag{5.5}$$

or, more simply as the height remains constant,

$$T = 2|V^{\text{exc}}| + 8 + W. \tag{5.6}$$

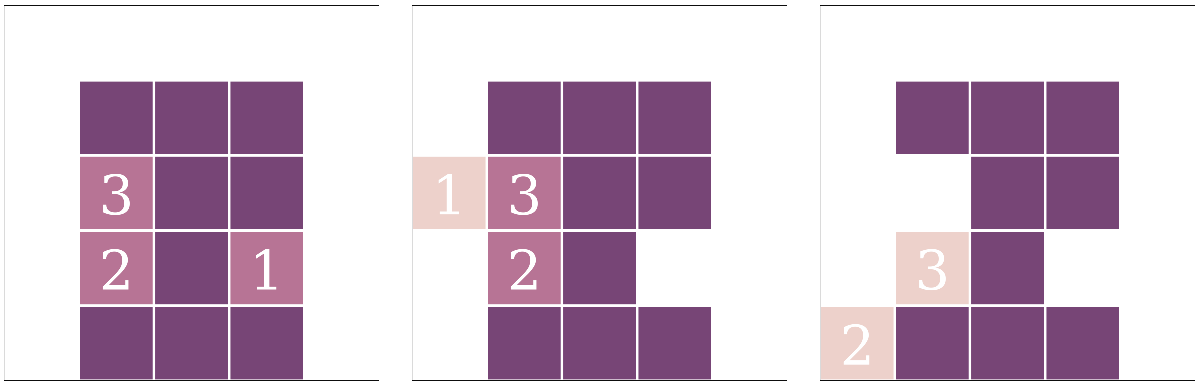
When normalised for the number of excluded modules in the initial configuration, this gives a linear relation between configuration width and time steps required for reconfiguration, as can be observed in Figure 5.14. This demonstrates `DO-PAS` yielding an asymptotically optimal performance of $\Theta(|V_0|)$, as was shown in Theorem 12.

There are situations where the performance of `DO-PAS` does not improve upon `PAS`. This is the case when the amount of time that it takes the module with $P = 1$ to travel to the sink is greater than the benefit of it not having to navigate the cavities left behind by modules closer to the sink were `PAS` to be used. An example of such a configuration can be seen in Figure 5.15. The priority order given by `DO-PAS` is shown in Figure 5.15(a), where it can be seen that the module with $P = 1$ must traverse the entire structure before the module with $P = 2$ can begin moving. This traversal takes 12 time steps, after which the other two modules can arrive two time steps after their predecessor, giving a total reconfiguration time of 16 time steps. Inspecting Figure 5.15(b), the order that is given when using `PAS` can be seen. In this case the module farthest from the sink still begins moving at the first time step, as do the modules closer to the sink. By the time the module labelled 3 reaches the West face of the configuration, the modules labelled 1 and 2 have already removed themselves, leaving a cavity for module 3 to navigate. However, this cavity only increases the total time it takes module 3 to navigate the structure by two time steps. This yields an overall reconfiguration time of 14 time steps, a reduction of 12.5% when compared to `DO-PAS`.

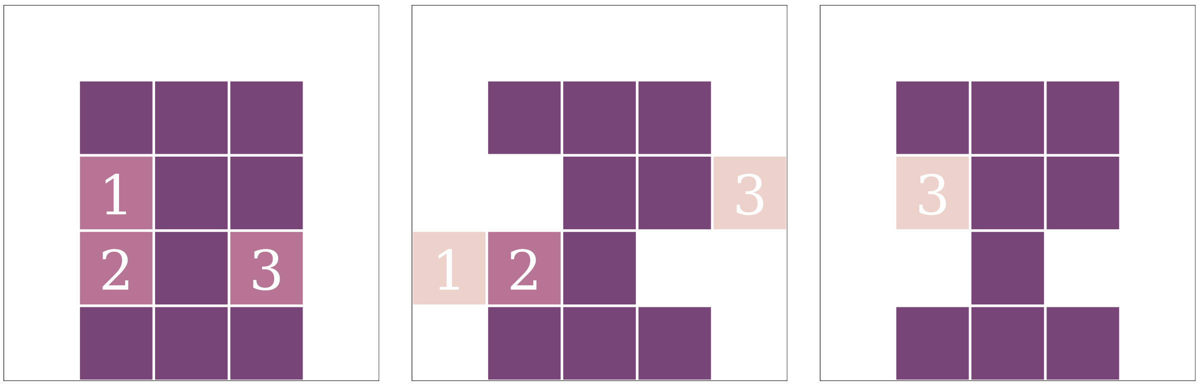
5.6.2 Randomly Generated Configurations

Along with the user defined configurations, randomly generated configurations are simulated, allowing for performance analysis under a wide range of conditions. The effects of both variable initial configuration size, and *inclusion density* are considered. As in Section 4.4.2, inclusion density is the percentage of modules within the initial configuration that are not removed in the formation of the desired configuration, that is, the percentage of included modules. Formally,

$$\rho = \frac{|V^{\text{inc}}|}{|V^{\text{exc}} \cup V^{\text{inc}}|} \times 100\%. \tag{5.7}$$



(a)



(b)

Fig. 5.15 A configuration where using DO-PAS, (a), for reconfiguration requires more time steps than using PAS, (b). DO-PAS requires 16 time steps overall and is shown at time steps 0, 10 and 14, while PAS takes 14 time steps to fully reconfigure and is shown at time steps 0, 2 and 11.

The configurations used for the simulations of DO-PAS and DO-SAS are the same as those generated for SAS and PAS in Section 4.4.2, allowing for direct comparison between the solutions. The way in which these configurations are generated is detailed there. Fig. 5.16 shows an example of a randomly generated configuration of size 20×20 with 50% inclusion density.

The performance of DO-PAS is compared to that of PAS from Chapter 4. Randomly generated configurations for inclusion densities ranging from 10% to 90%, and at initial configuration sizes of 10×10, 20×20 and 30×30 are considered. For each combination of criteria, 100 desired configurations were generated and here the performance of both approaches was simulated and compared, the results of which can be seen in Figure 5.17. Figure 5.18(a) shows box plots of the results for DO-PAS, with line plots of the best-case parallel times overlaid.

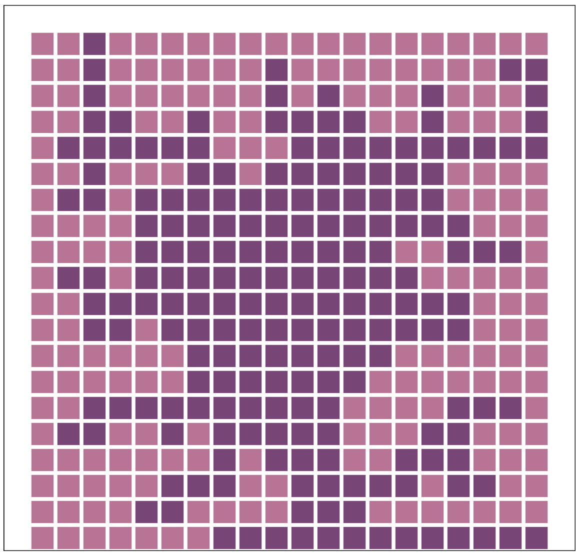


Fig. 5.16 Example of a randomly generated configuration, of size 20×20 and inclusion density (ρ) of 50%.

In Figure 5.18(b), box plots of the performance of `DO-SAS` for the same set of randomly generated configurations are shown, alongside line plots of the median best-case sequential time.

The way in which the performance of `PAS` and `DO-PAS` scale with the configuration size is also studied, considering sizes of 5×5, 10×10, 15×15, 20×20, 25×25, and 30×30. For each case, the inclusion density is 60%. The same 100 randomly generated desired configurations are used for both approaches, and are the same as those used in the similar comparison of Section 4.4.2. The results are shown in Figure 4.10.

5.6.3 Discussion of Results

For every simulation, employing `DO-PAS` or `DO-SAS` resulted in a successful reconfiguration from the initial configuration to the desired configuration, without the configurations becoming unfeasible at any point. As well as this, the time-based performance outclassed `PAS` and `SAS` for the same configurations. Moreover, Figure 5.18(b) clearly demonstrates that the median performance of `DO-SAS` follows the best-case sequential time, as is also observed in Section 5.6.1. This shows that the routes the modules take are optimal. However, the method in determining the routes to the sink remain the same as in Chapter 4, so it can be surmised that the order of removal given by `DO-SAS` is the reason that the modules follow their optimal paths. The same order is also used for `DO-PAS`, meaning that the modules will follow the same optimal

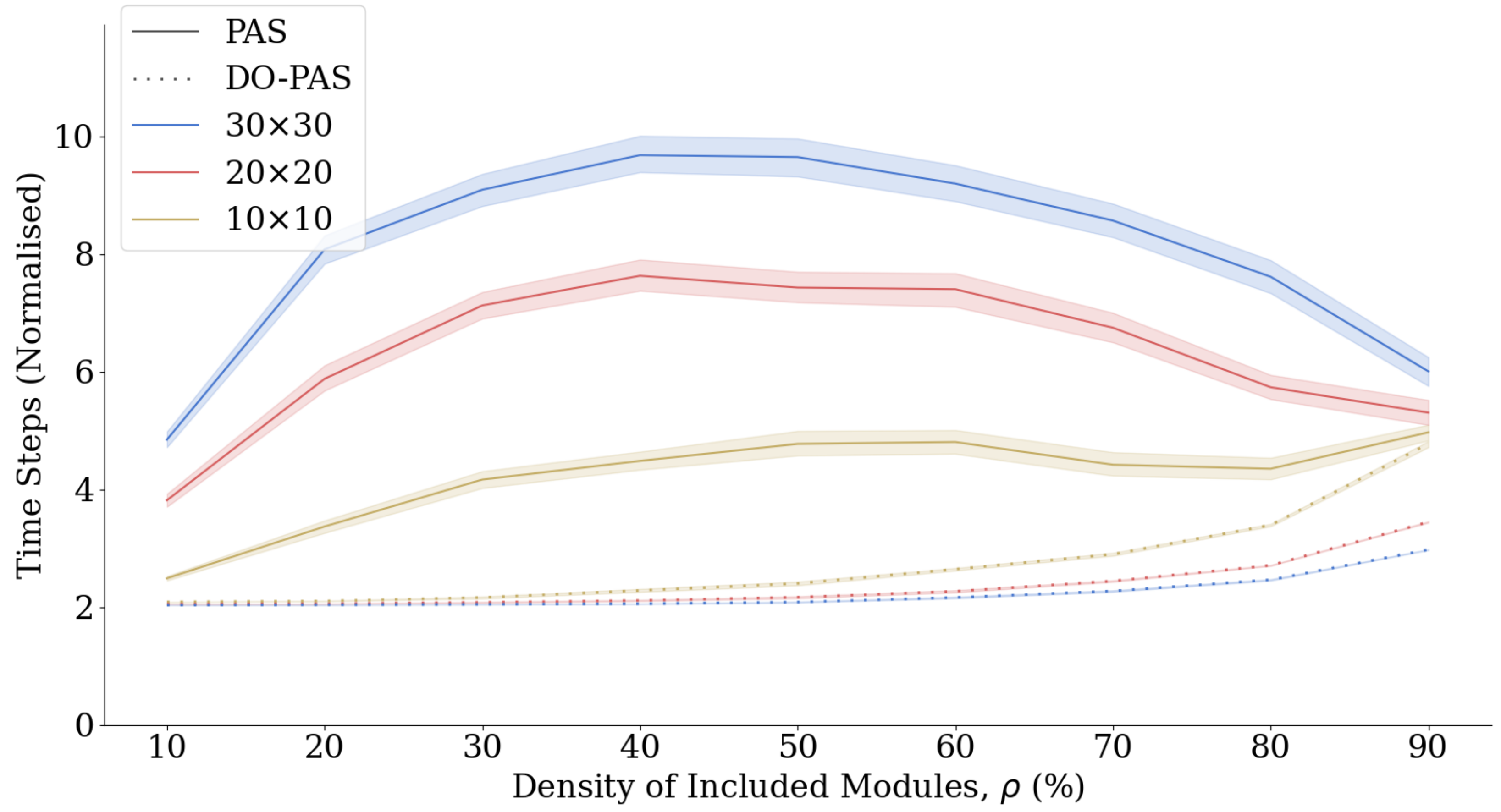
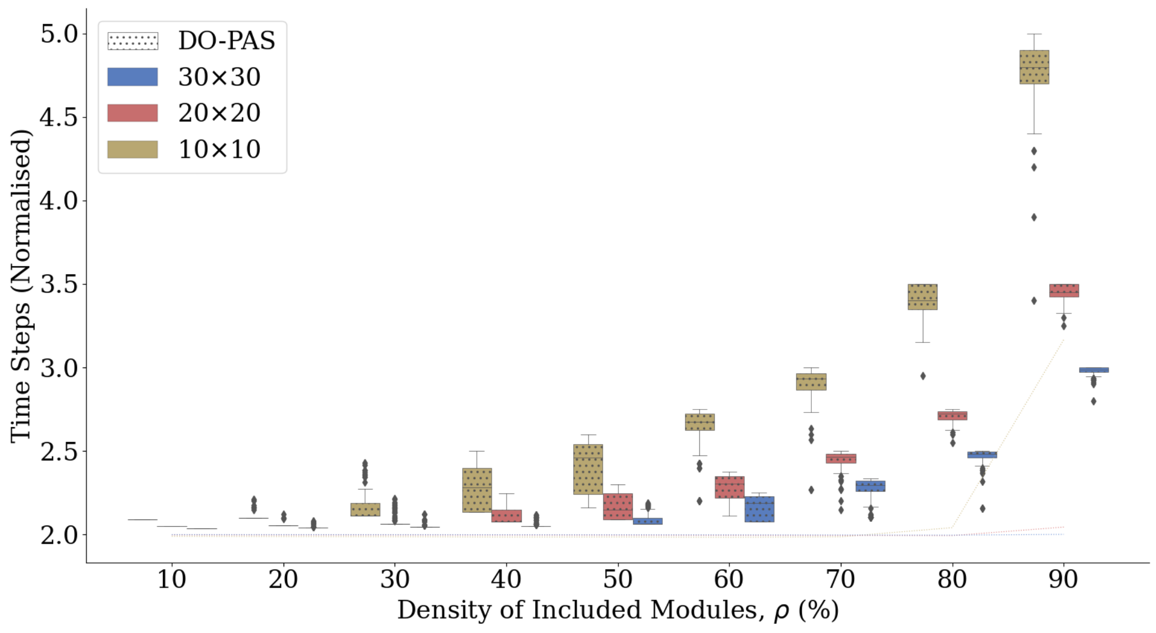
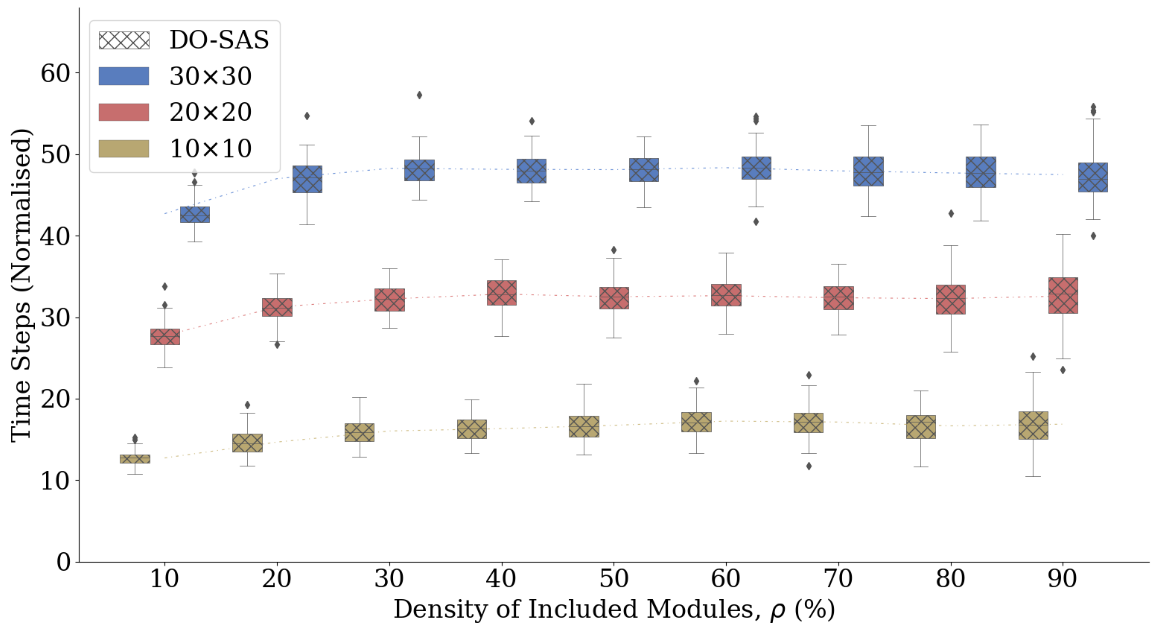


Fig. 5.17 Line plots of the normalised time taken by excluded modules to remove themselves, leaving behind randomly generated desired configurations of various inclusion densities. Shown for `PAS` and `DO-PAS`. 100 configurations are generated for each 10% inclusion density step. Initial configurations of sizes 10×10, 20×20 and 30×30 are simulated.



(a)



(b)

Fig. 5.18 Box plots of the simulated performance of (a) DO-PAS and (b) DO-SAS for the randomly generated desired configurations of inclusion densities between 10% and 90% for initial configurations of sizes 10×10, 20×20 and 30×30. Line plots of the median BCT_P and BCT_S for each scenario are overlaid, respectively.

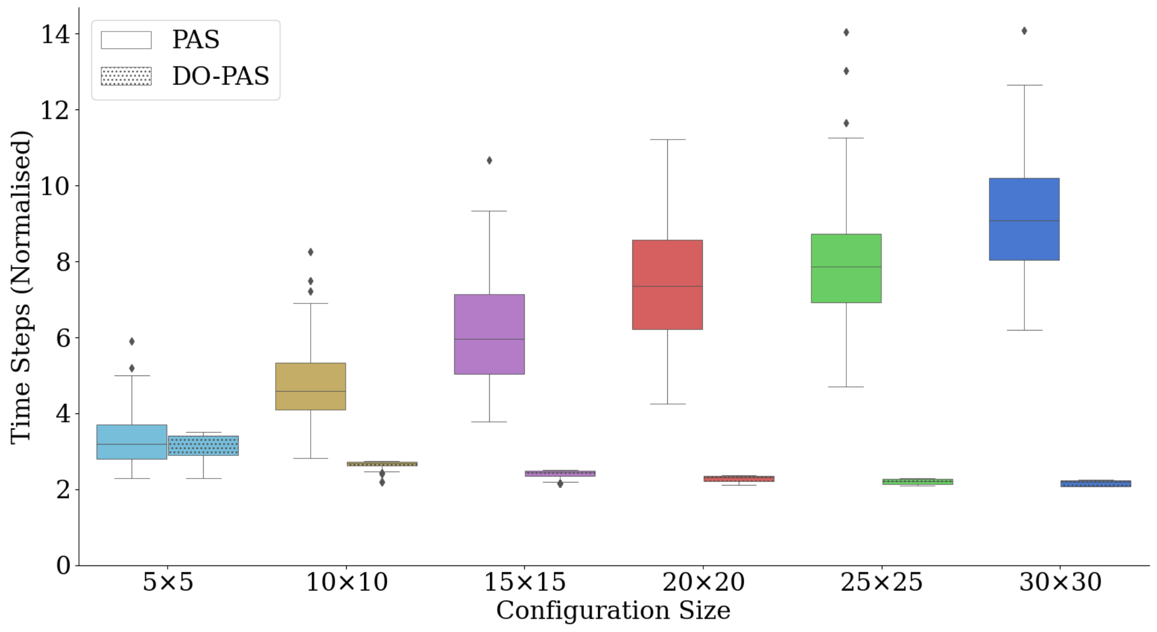


Fig. 5.19 Box plots of the normalised time taken by excluded modules to remove themselves from configurations of various sizes. The desired configurations were generated randomly with inclusion density 60% (100 samples per box). Time steps normalised by the number of excluded modules.

routes when using that approach too. Although the performance shown in Figure 5.18(a) does not follow the BCT_P, it is worth noting that the scale along the y-axis is very small. It can also be seen that the scenarios with the fewest overall excluded modules are the ones where the performance is furthest from BCT_P. This further confirms the previous reasoning for the difference in performance, where the calculations for BCT_P use the module with the shortest path to the sink as the first value, while in practice `DO-PAS` must use the entrance module of the furthest cavity to maintain the optimal routes. Modules arrive two time steps after their predecessor, with the exception of the very first module, which for `DO-PAS` takes many more time steps to reach the sink than the first module in BCT_P. For scenarios with fewer excluded modules overall, this difference will have a much greater effect.

The performance demonstrated by the simulation studies, as well as the distributed nature of `DO-PAS` and `DO-SAS`, can be considered to be improvements when compared with `PAS` and `SAS`. However, as showed by the results when simulating the configuration shown in Figure 5.15, in some cases `PAS` still delivers better performance. A combination of the approaches could be considered, centralising the ordering process to a degree, in order to obtain the optimal removal order. Figure 5.20 shows a configuration that is not optimal for `DO-PAS` or `PAS`. If

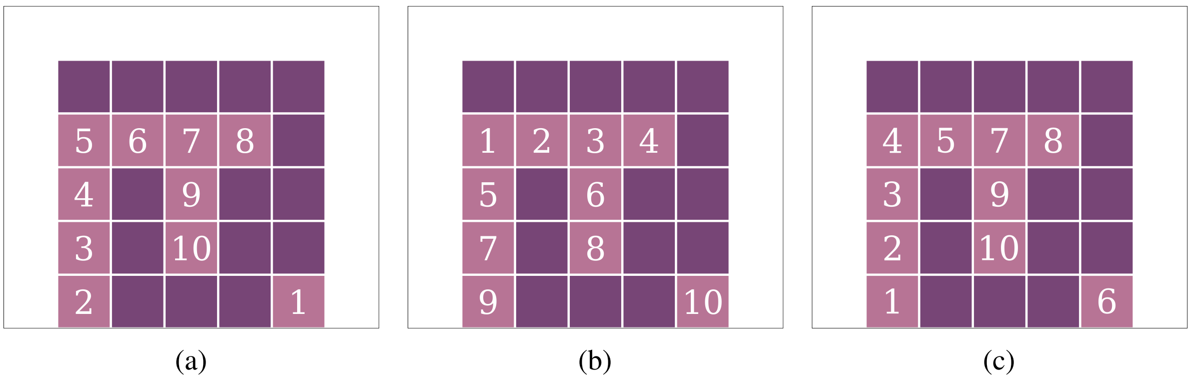


Fig. 5.20 Example of a configuration where the order given by both (a) `DO-PAS` and (b) `PAS` can be improved upon by the creation of (c) an order determined by hand.

`DO-PAS` were to be used, the priority order would be that of Figure 5.20(a), which has the same issue as was explored in Section 5.6.1 with Figure 5.15, where the module with $P = 1$ takes a long time to reach the sink, in which time no other modules can begin reconfiguration. This would take 35 time steps for complete reconfiguration. Yet, by employing `PAS` as shown in Figure 5.20(b), the large cavity in the centre would be evacuated before the module labelled 10 reaches it, increasing reconfiguration time. In this case it takes 30 time steps for reconfiguration. Combining the benefits of each approach, a better order could be determined. By calculating how long it takes the module in the South-East corner to reach the modules in the cavity to the West, and how many more time steps it would take that module to navigate any cavities left by the modules, an optimal order could be derived. This has been manually achieved here for the sake of illustration, and is shown in Figure 5.20(c). The module with $P = 5$ would begin moving at the same time as the module with $P = 1$, and arrive once the modules up to $P = 4$ have already removed themselves. By using this proposed order the performance is improved, taking 29 time steps. The improvement that could be granted through this approach would have to be compared with the cost of computation and the re-centralisation of control before it could be considered a suitable solution to the problem.

5.7 Conclusion

In this chapter, the concept of active subtraction applied to modular, self-reconfigurable robots was expanded upon, introducing a distributed method for determining a removal order for unwanted modules in a configuration. The newly proposed distributed method offers not only the benefit of requiring no centralised controller, but also outperforms the previous active subtractive approach substantially. The correctness of the solution was formally proved, with the run time performance being characterised. Specific simulations confirm the run time performance, and vast numbers of randomly generated simulations demonstrate this performance in a variety of simulated scenarios.

Chapter 6

Active Subtraction on Real World Systems

6.1 Introduction

In Chapters 4 and 5, the concept of reconfiguration via active subtraction was introduced, and algorithmic control solutions for its application to modular robotic systems were presented. However, these control solutions were shown in a simulated environment, using models of the modules, configurations and physics that were idealised in a number of ways.

The implementation of sliding square movement means that modules in the simulation did not need to consider collisions between themselves, as their footprint within the lattice remained constant—the height and width of one module. In practice, however, a more realistic method of movement would be via hinge movement, which could cause the footprint of a module to extend beyond the size of a single module while moving. This would mean that modules may be unable to remove themselves from narrow corridors in structures, that are only the width of one module. Moreover, some modular robotic systems may be incapable of moving individually in practice, requiring the formation of meta-modules, or the assistance of neighbours.

Furthermore, the constraint of gravity was included, and the solutions proven to not collapse, but the effect of gravity on the modules was limited to simply causing modules to fall if not supported, rather than causing strain on modules and their connections. Because of this, in the simulation, a desired configuration that is infinitely wide on the second layer, but supported below by a single module was considered valid; in reality this is clearly not the case.

In this chapter, the feasibility of Sequential Active Subtraction (SAS), Parallel Active Subtraction (PAS), Distributed Ordering Sequential Active Subtraction (DO-SAS) and Distributed

System	Dim.	Foot-print	Movement	MCW	Connection Method	Conn./ Faces	Effort Req.
Idealised System	2D	1×1	Sliding (Planar)	1	N/A	$4/4$	
3D Catoms [52]	3D	1×1	Rolling (Omni)	2	Electro-static	$12/6$	**Med.**
3D M-Blocks [50]	3D	1×1	Pivoting (Planar)	2	Permanent magnets	$6/6$	**Med.**
CHOBIE II [45]	2D Vert.	1×1	Sliding (Planar)	1	Mechanical	$4/4$	**Low**
HyMod [54]	3D	1×1	Hinged (Planar)	2	Mechanical	$4/6$	**Low**
M-TRAN III [10]	3D	2×1	Hinged (Planar)	2	Mechanical	$6/10$	**High.**
Roombots [60]	3D	2×1	Hinged (Diagonal)	3*	Mechanical	$10/10$	**High**

Table 6.1 A table comparing the characteristics of various modular robotic systems with regard to the application of active subtraction as a method of self-reconfiguration. (MCW stands for minimum cavity width.)

Ordering Parallel Active Subtraction (`DO-PAS`) are considered in relation to a number of real-world modular robotic systems. When referring to all of the active subtraction solutions of the previous chapters (`SAS`, `PAS`, `DO-SAS` and `DO-PAS`), they will be denoted by `*AS`. The chosen systems represent a number of unique characteristics within the field of modular robotics, and the analysis presented is designed to highlight the promise of `*AS` as a method of reconfiguration.

The chapter is structured as follows. Section 6.2 presents a comparison of the various characteristics of the chosen modular robotic systems and the relevance when considering the application of `*AS`. Subsequently, Section 6.3 concludes the chapter.

6.2 Comparison of Systems

Table 6.1 presents an overview of certain characteristics of six modular robotic systems that have the potential to employ `*AS` as a form of self-reconfiguration.

The first characteristic compared is the number of dimensions in which the modular robotic systems can operate. As `*AS` is concerned with reconfiguration in a vertical space, the majority of the applicable systems operate in three-dimensions. An outlier is *CHOBIE II*, which is able to operate in two-dimensions in the vertical plane.

Next, the impact of the module footprint is explored, which describes the size of a single module in terms of unit squares it inhabits. For example, the simulated modules of previous chapters would have a module footprint of 1×1 as they each inhabit one unit square. However, modules such as *M-TRAN III* and *Roombots* consist of two cubic halves, so have a module footprint of 2×1. This is an important characteristic of a system, as it means that a number of considerations must be made before `*AS` can be used, such as the construction of the initial structure, the way in which modules can move, and the valid desired configurations that can be formed.

Also considered is how a module is able to move within a configuration. Here the style of movement is important, as it may affect the connectivity considerations. Furthermore, the direction in which a module can move relative to its own facings must be assessed, as it could enable or prohibit more complex movement and configurations.

A related characteristic that must be considered is the minimum cavity width (MCW). The MCW is the minimum width that a cavity must be to allow the modules to extract themselves. This is important to consider as it limits the possible desired configurations.

The methods by which modules form connections are also compared. This facet of the system influences power requirements, the speed of reconfiguration, and the reliability of connections for supporting neighbouring modules. Additionally, any heterogeneous connection mechanisms will add complexity to the planning required to form valid configurations.

The table presents a comparison of the number of connectors a module has compared to the number of candidate faces that could be adjacent to a neighbouring module. This is important as a connection to a neighbour will enable communication and connectivity of a configuration, but may not be possible if a module has limited connectors.

Finally, a rating as to how much effort would be required to map the `*AS` solutions to these systems is presented. This rating is relative between the systems, and more information on the required effort is given in the following sections.

The first row in the table is the idealised system described in Section 3.4 and is shown to make it possible to compare the capabilities of real world systems.

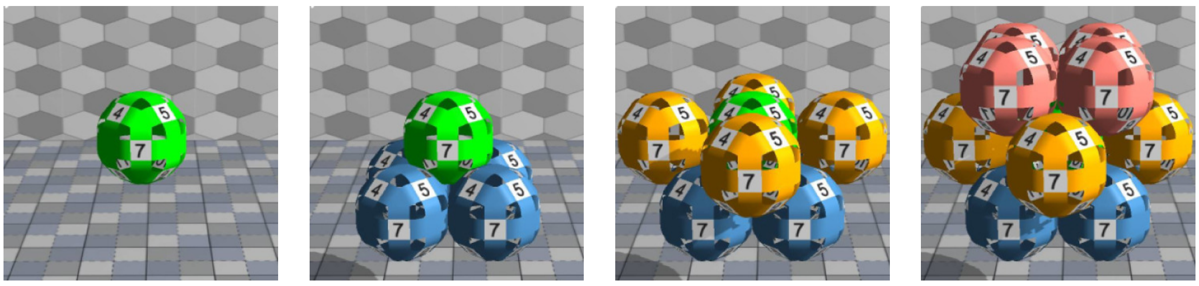


Fig. 6.1 Illustration of *3D Catoms* modules forming a face-centred cubic (FCC) lattice. Reprinted from [135] with permission from Benoît Piranda.

The following provides a discussion of how the aforementioned characteristics affect how good of a candidate each modular robotic system is for the application of *AS.

6.2.1 3D Catoms

The *3D Catoms* are modules developed by Piranda et al. [52] as part of the *Claytronics* project, where the authors aim to create a programmable matter. The *3D Catom* is a quasi-spherical, millimetre scale module that is actuated through electro-static forces. Here, their suitability as a system for utilising active subtraction is assessed.

From Table 6.1, and the name of the modules, it can be seen that the system is able to form shapes in three-dimensions. Existing work with *3D Catoms* is concerned with face-centred cubic (FCC) lattices, where layers on a plane are regular, densely populated square lattices, but each layer in the vertical direction is offset from the one below, meaning modules have twelve neighbours in a dense, three-dimensional shape [8, 52, 135]. An example of a small FCC lattice formed of simulated *3D Catoms* modules can be seen in Figure 6.1. Taking a vertical slice of one of these configurations for use with *AS would not work, as modules in alternating layers would only be directly connected to a module in the layer above and below, not any neighbours in the same layer. However, a regular, densely populated square lattice could be formed vertically, fulfilling this requirement for *AS.

The next characteristics shown in Table 6.1 are the module footprint and MCW. As each *3D Catoms* module is quasi-spherical and occupies one square in a square lattice, is has a footprint of 1×1.

The reason that *3D Catoms* modules are quasi-spherical, is to enable rolling movement. This gives each module the ability to move across the modular robotic structure unaided by neighbouring modules, a very desirable characteristic when considering active subtraction. The fact that this movement is possible in all directions means that the system may be able to apply a future iteration of active subtraction in three-dimensions, although the previously discussed FCC lattice issues may prohibit this.

A truly spherical module in a square lattice would be capable of navigating a cavity one module wide. However, the *3D Catoms* modules are not entirely spherical, requiring flat surfaces for the connectors. As the lattice size is determined by the distance between opposite faces on a module, this means that the lattice formed by *3D Catoms* is of a slightly smaller scale than their spherical bounding box, preventing movement in a cavity one module wide. As such, the MCW for *3D Catoms* is two. This is less desirable than an MCW of one, as it means that shapes formed by *3D Catoms* through `*AS` will be limited, as well as the way in which these shapes must be formed.

The way the modules move, and how they form connections, is through electrostatic actuation. As the scale of the *3D Catoms* modules is so small, using electrostatic forces allows for connections to be made. Although the movement has not been physically demonstrated in this way, the connectivity has been shown by Misumi et al. [82]. The adhesive strength of the connection is demonstrated in this work, but as the final weight of a *3D Catoms* module is unclear, it is not known how large the structures that use this type of connection could be, a vital consideration for the deployment of `*AS`.

In order to form connections with neighbours, the modules must have enough connectors in the correct places to do so. In the case of *3D Catoms*, the number of connectors is much larger than necessary for a two-dimensional square lattice. Each module is equipped with twelve connectors, but in a two-dimensional lattice, they will only require four of these. In considering three-dimensions, the fact that the modules are designed to form an FCC lattice means that the connectors are not correctly arranged to form a cubic lattice. This means that, despite the modules being equipped with twelve connectors, only four of these would be useful when employing `*AS` in three-dimensions, something to consider for future iterations of `*AS`.

6.2.2 3D M-Blocks

3D M-Blocks are cubic modules developed by Romanishin et al., whose novelty comes from their method of micro-locomotion [50]. The system holds promise for the application `*AS`, and the modules can be seen in Figure 2.6(e).

As the name would again suggest, *3D M-Blocks* are capable of forming structures in three-dimensions, enabling them to form the vertical structures required for `*AS`. The modules also have a footprint of 1×1, so they can form initial configurations without any need for adjustment.

As previously mentioned, the *3D M-Blocks* and their predecessor *M-Blocks* locomote by flipping forward using a transfer of momentum. The modules have been demonstrated pivoting to adjacent spaces in a configuration, and even around corners. It is also possible for the modules to jump multiple spaces in one movement, depending on how the braking to the flywheel is applied. The progress from *M-Blocks* to *3D M-Blocks* comes from the introduction of a system to realign the flywheel within the module, allowing movement in different directions. The flywheel is locked into place facing one of the three axis of the cube, and can rotate in either direction, meaning the modules are able to move in any planar direction depending on the alignment of the flywheel. This allows the modules to perform `*AS`, as well as be a potential platform for future, three-dimensional active subtraction solutions. However, when the modules pivot to move they rotate end of end, causing them to be out of alignment with the cubic lattice, and hence requiring an MCW of two units. Movement across multiple neighbours with one motion may require an even greater MCW, but this is not a limiting factor for current versions of `*AS`.

3D M-Blocks use permanent magnets as their connection method, which helps with aligning the modules once they have pivoted. This is a desirable attribute for any form of reconfiguration, including `*AS`. However, it is not a very strong method of connection, limiting the shapes that can realistically be formed. Many shapes formed in Sections 4.4 and 5.6 would not be possible with *3D M-Blocks* because of gravitational forces overcoming the magnetic ones.

The connectors are present on every face of the modules, and are non-gendered connectors, so *3D M-Blocks* can attach to any face of a neighbouring module. This is useful in that the movement algorithm used in `*AS` (Algorithm 1) could be deployed directly. Furthermore, the connections on all faces coupled with the ability to move in any planar direction, would enable a transition to three-dimensional active subtraction relatively easily from a movement perspective.

6.2.3 CHOBIE II

The next modular robotic system to be considered is *CHOBIE II* by Koseki et al. [45].

The *CHOBIE II* system operates in two-dimensions, unlike the other systems presented in this chapter. The plane that *CHOBIE II* modules inhabit is the vertical one, meaning that they are suitable for forming shapes via `*AS`. Furthermore, the modules occupy a 1×1 area, granting easy applicability of `*AS`.

The unique concept behind the system is that each module is able to slide along the surface of its neighbours to locomote, and was introduced by Inou et al. with *CHOBIE* [136]. To

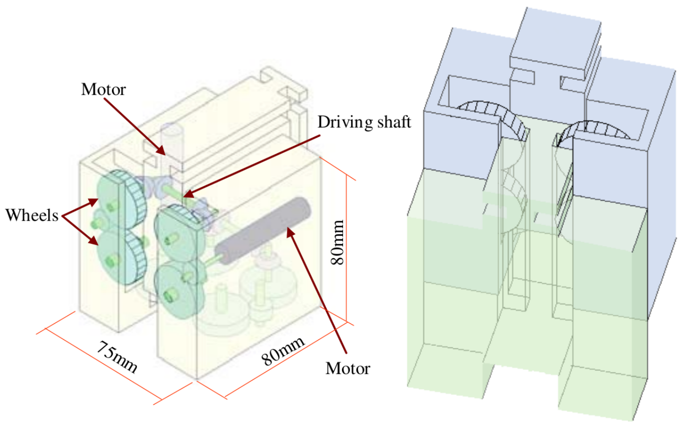


Fig. 6.2 Illustration of a *CHOBIE II* module. Reprinted from [45][1].

achieve this, each module is equipped with a groove on two faces, and a matching set of wheels that fit into the groove on the other two faces, as seen in Figure 6.2. When driven, the wheels create friction on the groove of the neighbour, and cause the unanchored module to slide along the relevant face. This has been demonstrated to work with individual modules locomoting across configurations, as well as locomotion while carrying other modules. Although very similar to the sliding square movement of `*AS`, the *CHOBIE II* modules are unable to perform diagonal movement around corners in the way that is shown in Figure 3.1(b). However, if a neighbouring module is available to assist, it could be used as a supporting neighbour module, enabling the modules to navigate corners, as shown by Koseki et al. [45]. The authors also demonstrate the modules shifting entire layers of modules, which could be useful in completing reconfiguration through active subtraction, but may have little applicability to the solutions of `*AS`.

Thanks to the sliding movement employed by *CHOBIE II* modules, they require an MCW of one, the narrowest of any modules considered for `*AS`. This is highly desirable as it means that even the most intricate shapes can be formed by the system. In fact, a cavity width of one aids *CHOBIE II* modules in their movement, as it means a neighbour will always be present on at least two sides of the module, so, when inside a cavity, an active excluded module will be able to perform diagonal movement around a corner without needing any neighbours to move.

The aforementioned wheel and groove mechanism that provides locomotion to the module also acts as the connector between them, providing a mechanical link. The connection, while strong, causes high levels of friction between the wheels and grooves when multiple modules

[1]Reprinted by permission from Springer Nature: Springer 6th International Symposium on Distributed Autonomous Robotic Systems, “Cellular robots forming a mechanical structure,” M. Koseki, K. Minami and N. Inou, © 2007.

are supported by one connection. Inou et al. equip each module with strain gauges to measure these effects, deducing that many modules can be supported, but it can be difficult for a module to move while carrying others [136]. In the case of `*AS`, this type of movement is not necessary, so only the benefit of a strong mechanical connection needs to be taken into account.

Although every face that can make connections in the two-dimensional lattice is equipped with a connector, it should be noted that the connection mechanism is a heterogeneous one, meaning that the modules must match a face with a groove to a neighbouring face with a wheel and vice versa. As the modules cannot rotate or change their orientation in any way, provided that all modules are assembled in the same orientation at initialisation, the correct matching of connectors will be maintained throughout reconfiguration using `*AS`.

6.2.4 HyMod

Another system that holds promise for the application of `*AS` is *HyMod* by Parrott et al. [54], a module of which can be seen in Figure 2.4(c).

HyMod modules are capable of operating in three-dimensions, able to extend vertically in free space, or through the use of the surface extension, also developed by Parrott et al. [54]. For the purposes of employing `*AS`, the surface extension would provide a stable base for any configurations, and allow communication between modules that are connected only by the ground. Each module has a spherical bounding box within the cubic lattice that the modules create, so the footprint of each *HyMod* module is 1×1.

The modules are equipped with joints that can infinitely rotate on two sides, that form wheels. However, in terms of movement within the considerations of `*AS`, the continuous movement granted by the wheels would not be utilised. Instead, the central hinge of each module would provide movement for the modules, rotating $\pm 90°$ inline with the central axis of the modules, creating planar movement. A single module actuating its own hinge does not move, rather, *HyMod* modules require the collaboration of neighbours to produce movement within a lattice formation. This can be achieved through the use of meta-modules, where two connected *HyMod* modules are considered a single entity that is capable of locomoting across the configuration. In this case, the MCW is two, as the meta-module would require enough space to walk end over end through the cavity. This approach limits the granularity of the system, effectively turning the footprint of a single "module" from 1×1 to 2×1. To avoid this, movement algorithms could be developed to enable neighbouring *HyMod* modules to aid one another in their movement. In this case, the MCW remains at two, as a module manoeuvring across the configuration would need to be lifted by its neighbours in order to locomote.

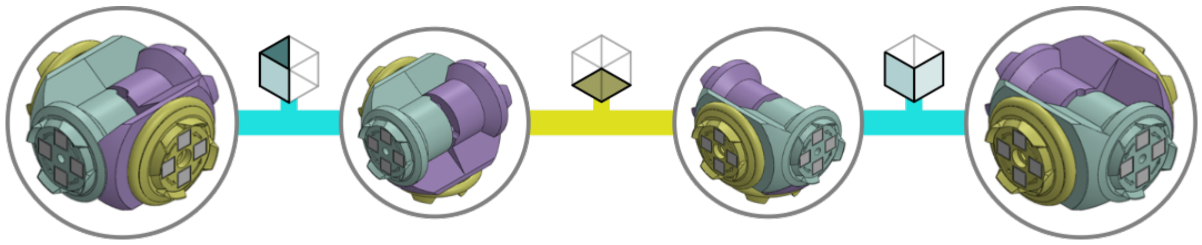


Fig. 6.3 Example of a model of a *HyMod* module reorienting itself, and the connections required to complete each movement. Adapted from [55].

As detailed in Chapter 2, the connectors used on the *HyMod* modules are called *HiGen* [77]. These are high-speed mechanical connectors, capable of making connections in 0.2 seconds. They utilise extending hooks to connect, and the strength of the connection allows a *HyMod* module to lift 1.8 modules. The limiting factor is the material that the current generation of *HiGen* connectors are made from, a brittle plastic. If future iterations are machined from metal or a stronger plastic, then the connections would be much stronger and prove a suitable basis for applying `*AS`. Another advantage of *HiGen* connectors is the fact that they are genderless. This means that only one side of the connection is required to actuate to make or break a connection. As such, a *HyMod* module that has failed can be removed, even if it is unable to retract its own connectors, increasing the reliability. It also allows for passive *HiGen* connectors that can be used in other ways, such as in the aforementioned surface extension, which could provide a surface on which to perform `*AS`.

Each *HyMod* module is equipped with four *HiGen* connectors, leaving two faces without a connector. However, thanks to the previously described central hinge, the faces that do not have connectors can be altered. Furthermore, the *HyMod* modules possess the ability to perform in-place rotation, as their bounding box is a sphere within the cubic lattice. This means that not only can the modules use the central hinge to change the location of the faces with connectors, but they can rotate as well. Through investigation, it has been found that a *HyMod* module is able to reorient itself to have connectors on specific faces between any two orientations, given it has at least two neighbours, and that at least one neighbour is connected to at least one face in the initial and end orientations. An example of the steps required to achieve a reorientation can be seen in Figure 6.3, using a simulated model of a *HyMod* module for demonstration. The ability to reorient itself in place, means that the *HyMod* module, while not possessing a connector on every face, still proves a promising candidate for the deployment of `*AS`.

6.2.5 M-TRAN III

M-TRAN III by Kurokawa et al. [10] should also be considered as a potential platform for the realisation of *AS.

These modules lend themselves to chain type configurations, given that they are longer than most modules on this list and feature heterogeneous connectors on their end faces. However, the modules have also been demonstrated forming lattice based configurations in three dimensions.

Similar to the suggested meta-modules of the previous section, each *M-TRAN III* module has a footprint of 2×1. It is made of two cubic halves, connected in the middle by a joint, around which each half can pivot by $\pm 90°$. This larger footprint means that the granularity of the configurations that *M-TRAN III* modules can form must be considered, as well as the fact that each module is not uniform in dimensions. It could be possible to create initial and desired configurations where all modules are oriented the same way, but this limits the configurations possible. Furthermore, each half of the module features either active or passive connectors, making the connection method heterogeneous. As such, the gender of each connector must be taken into account when designing the initial and desired configurations. One solution to enable uniform units within the configurations, could be to use a meta-module as a unit. Parada et al. propose a meta-module design that could be formed by *M-TRAN III* modules [128]. It would create a uniform unit for the lattice, and enable unique movement options that they term *scrunch* and *relax*. Although promising, this type of movement has not been considered for current iterations of *AS, and the size of the meta-modules is much larger than a single *M-TRAN III* module, so may not be suitable.

The hinge in the centre of a module, around which the two halves can pivot, allows a single *M-TRAN III* module to locomote across a configuration by moving end over end. This process means that the modules require an MCW of two, further limiting the configurations that the system can form using *AS.

As previously discussed, the connection method used by *M-TRAN III* modules is a heterogeneous one, consisting of an active and passive side. The active side extends small hooks, that are able to latch onto holes on the passive side. This method provides a strong connection, and *M-TRAN III* modules have been demonstrated lifting multiple other modules thanks to it. However, the heterogeneous nature of the connection method means that the active and passive halves must be adjacent throughout reconfiguration. Furthermore, the design of the hinge between the two halves mean that only three faces of each cubic half are equipped with connectors. This gives the system a total of six connectors per module, three active and three

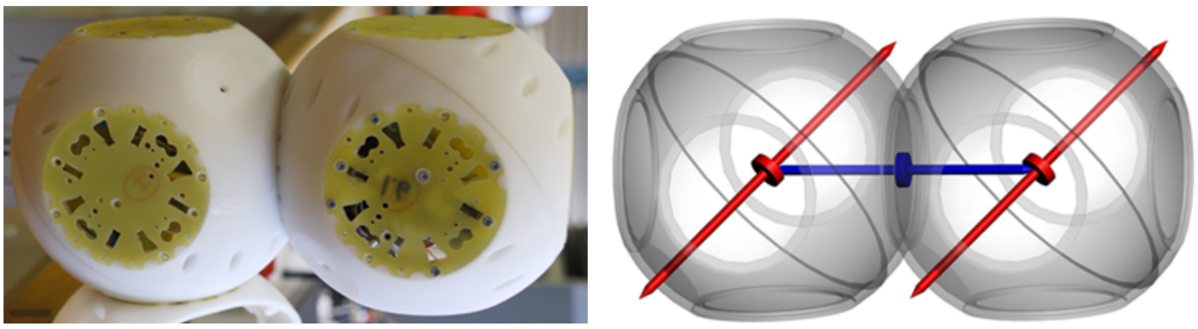


Fig. 6.4 Depiction of a *Roombots* module and an illustration of the rotational axis of the three joints. Reprinted from [60] © 2009 IEEE.

passive, out of a possible ten[2]. In practice, the *M-TRAN III* modules would be able to use even fewer connectors if employing `*AS`. As the modules are unable to rotate in any way, they must face the direction of travel at all times. In the case of the two-dimensional reconfiguration of `*AS`, this is parallel to the configuration. The layout of the connectors on the modules mean that four of the six connectors would be facing outward in this case, unable to connect to any neighbours. This renders the modules entirely unsuitable for reconfiguration via `*AS`, as the number of connections between the modules in the system would be too few to allow any movement to occur.

6.2.6 Roombots

The final active modular robotic system to be considered in Table 6.1 is *Roombots*, by Sproewitz et al. [60]. This is a system whose primary reason for development was to create adaptable furniture, that would reconfigure to meet the needs of the user.

The *Roombots* modules can exist horizontally or upright, and have been demonstrated creating three-dimensional configurations in free space, as well as in a structured space [87] as with *HyMod*. Similarly to *M-TRAN III*, each *Roombots* module consists of two cubic halves joined together in the centre, creating a module footprint of 2×1. However, the two halves of the *Roombots* modules are connected with a rotational joint that allows each cubic half to rotate with respect to the other, as can be seen by the blue joint in Figure 6.4. This means that the central hinge is not used for the movement of a module.

Instead, each module locomotes by rotating the diagonal joint in one half of the module, providing that half is fixed in place using a connector. These are the red joints in Figure 6.4.

[2]Each module could have ten connectors that face neighbouring modules as the modules are comprised of two cubic halves, which would have six faces each, but one face from each half is attached to the other half.

This allows the module to transition from a horizontal position to an upright one, and vice versa. However, the movement extends beyond the two dimensional plane, so it is harder to categorise the requirements in terms of MCW. The movement requires a cavity width of three, as the connecting faces are flat, so when the modules rotate the diagonal joints, the faces extend slightly beyond the module footprint. Moreover, the movement extends into the third dimension, so the MCW is insufficient to fully characterise the space required. If the configurations remain in two-dimensions, with free space either side, then this does not affect the *Roombots* modules' ability to perform `*AS`. If future iterations of `*AS` form the basis for self-reconfiguration in three-dimensions, then this type of movement and the space required could be a limiting factor in the applicability of the *Roombots* system.

As previously mentioned, the connection method employed by *Roombots* modules uses flat faces, equipped with hooks and holes. Each connector is able to accommodate both the necessary hooks and the respective holes, creating a homogeneous connection method. This means that modules can connect to any face of a neighbouring module. It is also possible for passive connectors to be used, either in the modules or as a structured surface to which the modules can attach [87]. As with *M-TRAN III* modules, each *Roombots* module has a maximum of ten locations at which a connector could be located. In this case, all ten do house connectors, as the hinge mechanism between halves is entirely self-contained, unlike the hinge in *M-TRAN III* modules. This gives the system great flexibility with how configurations can be formed, as the modules do not have to align male and female connectors, nor consider which faces do not have connectors.

6.3 Conclusion

In this chapter, a number of modular robotic systems and their attributes relating to the realisation of `*AS` were considered. Each of the modular robotic systems offer both benefits and hindrances when considering the application of `*AS`. For example, *CHOBIE II* seems promising in that it is able to most closely replicate the sliding square method of movement, and requires an MCW of only one, but is unable to navigate the corners of a configuration without collaboration with a neighbour. Similarly, *HyMod* is capable of in place rotation, so would prove logical for manoeuvring through the system, but also requires the assistance of its neighbours to locomote. *M-TRAN III* and *Roombots* are able to locomote as individual modules, but their footprint is irregular, *M-TRAN III* can only use two connectors on each module and *Roombots* is unable to move in only two-dimensions. Comparatively, *3D Catoms* and *3D M-Blocks* have uniform, minimal module footprints and are able to move individually, but require an MCW of two and their connection mechanisms may not be sufficient to support neighbouring modules.

It is clear that, while `*AS` provides a sound and effective self-reconfiguration solution in simulation, further work must be undertaken before it can be deployed on real world systems.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis has presented a number of control algorithms to enable self-reconfiguration via active subtraction with modular robots. The first algorithms, presented in Chapter 4, utilised centralised control to achieve this, having either a leader module or external controller inform modules when they should begin removal. Subsequently, a distributed algorithm was presented in Chapter 5, enabling modules to determine a removal order, and be informed of when to begin moving, entirely through message passing, The solutions granted successful self-reconfiguration when modules moved either sequentially or in parallel, as demonstrated by formal proofs and extensive simulations. The application of these solutions to real world systems was then assessed in Chapter 6.

In order to facilitate active reconfiguration, a decentralised movement algorithm was first devised. By prioritising movement towards a predefined sink location, the behaviour was akin to a wall following algorithm, and enabled modules to locomote towards the sink irrelevant of starting location and the morphology of the configuration. Subsequently, centralised algorithms were developed to determine a removal order for given starting and desired configurations. The centralised controller used a top down approach, informing the free module in the highest row and closest to the sink to begin moving when necessary. For sequential movement, this was each time the preceding module reached the sink. To introduce parallel movement, planning simulations were performed so that the central controller could ensure no modules would collide with one another. A requirement for successful self-reconfiguration is that the modules maintain connectedness throughout reconfiguration, something that was formally proved for the centralised solution with both sequential and parallel movement. Also formally proved was the worst-case time, which increased quadratically with configuration size. The formal analysis was complemented by extensive simulations, demonstrating that parallel movement yields better time-based performance than sequential movement, as would be expected. However, the requirement to perform extensive planning simulations for parallel movement means that a more capable controller is required, and this trade-off must be considered when deciding whether parallel movement is possible to realise on a system.

In an effort to alleviate the high computation required by a centralised controller, and to create a more robust system without a single point of failure, a distributed control algorithm was also created. Utilising the same distributed movement algorithm as the centralised solution, the modules used message passing to determine a removal order, and when to begin moving given that order. Given the connectedness of an initial configuration, these messages could be passed through the system so that each module is aware of its place in the removal order before any movement begins. Following this, the modules begin removing themselves, informing the next in the order to begin moving afterwards, either by a direct message for parallel movement, or by flooding a message through the configuration upon reaching the sink location for sequential movement. The correctness of this approach was formally proved, and again the time-based performance assessed, here yielding a linear increase in time with configuration size. The same simulation environment was used to confirm the performance, and provide a comparison to the centralised approach. In almost all scenarios the distributed approach was able to outperform the centralised solution, only being slower in specific situations. This demonstrates that an even faster method of self-reconfiguration could be devised by combining the approaches to determining removal order, but it is posited that this is not possible through solely distributed methods. As well as being a faster method of self-reconfiguration, the distributed approach also removes the need for a centralised controller. Although each module must be able to send and receive messages, as well as react accordingly when receiving a message, the computational requirements for this are dwarfed by those of the central controller for the previous solution.

The practical implementation of the self-reconfiguration algorithms requires that a suitable system be identified. Through extensive research and analysis, a comparison of a number of potential candidate systems was presented. The information was summarised in the table, and focussed on a number of specific characteristics. A detailed analysis of each system was also presented, highlighting the potential, and possible caveats, of transferring the current iteration of active subtraction control algorithms directly to real-world systems.

It is believed that the research presented is meaningful within the broader context of self-reconfiguration for modular robotic systems. The demonstration of a novel method of self-reconfiguration in simulation, and considerations for deployment on physical systems, will hopefully inspire future work, either expanding on this particular solution, or with new, innovative solutions to the problem.

7.2 Future Work

While the results of the work discussed in the previous section demonstrate active subtraction to be a promising solution to the self-reconfiguration problem, they also show that more work is necessary before it can be considered generally applicable to modular robotic systems.

One such area for expansion could be the integration of more thorough physics constraints. A number of modular robotic systems that are equipped with force sensors have been designed, such as the *Blinky Blocks* by Kirby et al. [137] or the *Force-Aware Robots* by Bray and Groß [138]. These systems have been demonstrated giving real-time feedback on the stability of a configuration [138, 139], which has also been shown to be something that can be calculated on a distributed system [140]. By integrating this work with the algorithms contained in this thesis, a method of active subtraction that maintains the physical stability of a real system throughout the reconfiguration process could be developed.

Another possibility for further research could be to analyse the efficacy of the message passing used during reconfiguration. Because of the comparatively short amount of time required to send a message compared to the time required to perform a movement, messages are assumed to be instantaneous in the work presented here. However, if a configuration has a high number of modules, and many messages are required to be passed, such as when flooding the system in `DO-SAS`, then the time and power required may not be negligible. Analysis of this type has been undertaken when considering distributed reconfiguration algorithms in the past [126, 141]. Further investigation could be concerned with the effects of information loss with direct messages compared to messages flooded through the system. This would provide insight into the trade off between robustness and efficiency in terms of message passing.

A key area that holds much promise for future work is by progressing the control solutions to operate in three-dimensions, where the initial configuration would be a cuboid. This would present a number of challenges if directly addressing a three-dimensional approach. For example, the movement algorithm currently used would no longer work, as it would not be guaranteed that modules would be heading towards the sink, and it would be impossible to know which direction is the correct one using only local knowledge. One solution to this is to partition the configurations into two-dimensional vertical slices, and apply the algorithms presented in this thesis to these sub-configurations. The reconfiguration would be far from optimal in this case, but has been proven correct, so would provide a suitable starting point. Furthermore, the goal configurations would be severely limited, as each vertical slice would need to fulfil the requirements of a feasible configuration for the algorithms to work; a module could be supported by a neighbour that is in a separate vertical slice, so would appear unsupported when assessing each slice separately. Nonetheless, the reconfiguration of three-dimensional configurations would be key to proving the general applicability of active subtraction as a method of self-reconfiguration.

A goal for self-reconfiguration is the ability to reconfigure between arbitrary shapes. The problem of finding the optimal method of reconfiguring between two arbitrary shapes is computationally intractable [13, 14], so finding a feasible way to reconfigure between arbitrary shapes requires a heuristic. One such solution is to use an intermediate shape, such as a simple chain demonstrated by Casal and Yim [142], or a “melted” structure, demonstrated by Rus and Vona [44]. The same idea could be implemented using the active subtraction algorithms presented in this thesis, where the intermediate structure is the initial configuration. By using a centralised planner to simulate active subtraction from the initial configuration to a desired configuration that is actually the starting state for the arbitrary reconfiguration, the steps required by each module can be generated and then reversed, so the modules form what has been referred to as the initial configuration. From this it has already been seen that it is possible to form arbitrary configurations. This is just one possibility for achieving self-reconfiguration between arbitrary shapes.

References

[1] M. D. Hall, A. Özdemir, and R. Groß, “Self-reconfiguration in two-dimensions via active subtraction with modular robots,” in *Robotics: Science and Systems XVI*, RSS Foundation, 2020.

[2] A. Özdemir, M. Gauci, A. Kolling, M. D. Hall, and R. Groß, “Spatial coverage without computation,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9674–9680, IEEE, 2019.

[3] T. Fukuda and S. Nakagawa, “Dynamically reconfigurable robotic system,” in *1988 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1581–1586, IEEE, 1988.

[4] E. Bonabeau, M. Dorigo, and G. Théraulaz, *Swarm intelligence: From natural to artificial systems.* Oxford University Press, 1999.

[5] S. Chennareddy, A. Agrawal, and A. Karuppiah, “Modular self-reconfigurable robotic systems: A survey on hardware architectures,” *Journal of Robotics*, vol. 2017, pp. 1–19, 2017.

[6] K. Støy and R. Nagpal, “Self-reconfiguration using directed growth,” in *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 1–10, Springer, 2004.

[7] A. Sproewitz, P. Laprade, S. Bonardi, M. Mayer, R. Moeckel, P.-A. Mudry, and A. J. Ijspeert, “Roombots – towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1126–1132, IEEE, 2010.

[8] P. Thalamy, B. Piranda, and J. Bourgeois, “Distributed self-reconfiguration using a deterministic autonomous scaffolding structure,” in *18th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 140–148, IFAAMAS, 2019.

[9] A. L. Christensen, R. O’Grady, and M. Dorigo, “SWARMORPH-script: A language for arbitrary morphology generation in self-assembling robots,” *Swarm Intelligence*, vol. 2, no. 2-4, pp. 143–165, 2008.

[10] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, “Distributed self-reconfiguration of M-TRAN III modular robotic system,” *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 373–386, 2008.

[11] K. Støy, W.-M. Shen, and P. M. Will, “A simple approach to the control of locomotion in self-reconfigurable robots,” *Robotics and Autonomous Systems*, vol. 44, no. 3-4, pp. 191–199, 2003.

[12] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu, “Miche: Modular shape formation by self-disassembly,” *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 345–372, 2008.

[13] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff, “Evaluating efficiency of self-reconfiguration in a class of modular robots,” *Journal of Field Robotics*, vol. 13, no. 5, pp. 317–338, 1996.

[14] Z. Ye, M. Yu, and Y.-J. Liu, “NP-completeness of optimal planning problem for modular robots,” *Autonomous Robots*, vol. 43, no. 8, pp. 2261–2270, 2019.

[15] M. Gauci, R. Nagpal, and M. Rubenstein, “Programmable self-disassembly for shape formation in large-scale robot collectives,” in *13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 573–596, Springer, 2016.

[16] K. Gilpin, A. Knaian, and D. Rus, “Robot Pebbles: One centimeter modules for programmable matter through self-disassembly,” in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2485–2492, IEEE, 2010.

[17] M. D. Hall, “RSS 2020, spotlight talk 14: Self-reconfiguration in two-dimensions via active subtraction with modular robots.” https://youtu.be/RgGi_mc4tOQ, 2020.

[18] C. J. J. Paredis, H. B. Brown, and P. K. Khosla, “A rapidly deployable manipulator system,” in *1996 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1434–1439, IEEE, 1996.

[19] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, H. Raja, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi, “Multi-robot organisms: State of the art,” in *2010 IEEE International Conference on Robotics and Automation (ICRA) Workshop on“Modular Robots: The State of the Art”*, pp. 1–10, IEEE, 2010.

[20] A. Lyder, R. F. M. Garcia, and K. Støy, “Mechanical design of Odin, an extendable heterogeneous deformable modular robot,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 883–888, IEEE, 2008.

[21] D. Schmitz, P. Khosla, and T. Kanade, “The CMU reconfigurable modular manipulator system,” *Carnegie Mellon University Reseach Showcase*, 1988.

[22] T. Matsumaru, “Design and control of the modular robot system: TOMMS,” in *1995 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 2125–2131, IEEE, 1995.

[23] A. Yun, D. Moon, J. Ha, S. Kang, and W. Lee, “ModMan: An advanced reconfigurable manipulator system with genderless connector and automatic kinematic modeling algorithm,” *IEEE Robotics and Automation Letters (RAL)*, vol. 5, no. 3, pp. 4225–4232, 2020.

[24] L. Kelmar and P. K. Khosla, “Automatic generation of kinematics for a reconfigurable modular manipulator system,” in *1988 IEEE International Conference onRobotics and Automation (ICRA)*, pp. 663–668, IEEE, 1988.

[25] J. Han, W. K. Chung, Y. Youm, and S. H. Kim, “Task based design of modular robot manipulator using efficient genetic algorithm,” in *1997 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 507–512, IEEE, 1997.

[26] C. Nainer, M. Feder, and A. Giusti, “Automatic generation of kinematics and dynamics model descriptions for modular reconfigurable robot manipulators,” in *IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 45–52, IEEE, 2021.

[27] S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, D. Laneri, L. Winkler, T. Schmickl, R. Thenius, P. Corradi, and L. Ricotti, “Symbiotic robot organisms: REPLICATOR and SYMBRION projects,” in *8th Workshop on Performance Metrics for Intelligent Systems*, pp. 62–69, ACM, 2008.

[28] S. Kernbach, F. Schlachter, R. Humza, J. Liedke, S. Popesku, S. Russo, T. Ranzani, L. Manfredi, C. Stefanini, R. Matthias, C. S. F. Schwarzer, B. Girault, P. Alschbach, E. Meister, and O. Scholz, “Heterogeneity for increasing performance and reliability of self-reconfigurable multi-robot organisms,” *arXiv preprint arXiv:1109.2288*, 2011.

[29] A. Faíña, F. Bellas, F. López-Peña, and R. J. Duro, “EDHMoR: Evolutionary designer of heterogeneous modular robots,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 10, pp. 2408–2423, 2013.

[30] J. Baca, M. Ferre, and R. Aracil, “A heterogeneous modular robotic design for fast response to a diversity of tasks,” *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 522–531, 2012.

[31] H. S. Ahn, Y. M. Beak, I.-K. Sa, W. S. Kang, J. H. Na, and J. Y. Choi, “Design of reconfigurable heterogeneous modular architecture for service robots,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1313–1318, IEEE, 2008.

[32] A. M. Romanov, V. D. Yashunskiy, and W.-Y. Chiu, “SABER: Modular reconfigurable robot for industrial applications,” in *IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 53–59, IEEE, 2021.

[33] Y. Mohan and S. G. Ponnambalam, “An extensive review of research in swarm robotics,” in *2009 world congress on nature & biologically inspired computing (NaBIC)*, pp. 140–145, IEEE, 2009.

[34] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, “Modular self-reconfigurable robot systems,” *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[35] K. Støy, D. Brandt, and D. J. Christensen, *Self-reconfigurable robots: An introduction*. The MIT Press, 2010.

[36] K. Gilpin and D. Rus, “Modular robot systems,” *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 38–55, 2010.

[37] M. Yim, *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University, 1994.

[38] A. Castano, W.-M. Shen, and P. Will, “CONRO: Towards deployable robots with inter-robots metamorphic capabilities,” *Autonomous Robots*, vol. 8, no. 3, pp. 309–324, 2000.

[39] M. Yim, D. G. Duff, and K. D. Roufas, “PolyBot: A modular reconfigurable robot,” in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 514–520, IEEE, 2000.

[40] L. Pfotzer, S. Ruehl, G. Heppner, A. Rönnau, and R. Dillmann, “KAIRO 3: A modular reconfigurable robot for search and rescue field missions,” in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 205–210, IEEE, 2014.

[41] S. Sankhar Reddy CH., Abhimanyu, R. Godiyal, T. Zodage, and T. Rane, “2DxoPod – a modular robot for mimicking locomotion in vertebrates,” *Journal of Intelligent & Robotic Systems*, vol. 101, no. 23, pp. 1–16, 2021.

[42] M. Yim, Y. Zhang, and D. Duff, “Modular robots,” *IEEE Spectrum*, vol. 39, no. 2, pp. 30–34, 2002.

[43] S. Murata, H. Kurokawa, and S. Kokaji, “Self-assembling machine,” in *1994 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 441–448, IEEE, 1994.

[44] D. Rus and M. Vona, “Self-reconfiguration planning with compressible unit modules,” in *1999 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 2513–2520, IEEE, 1999.

[45] M. Koseki, K. Minami, and N. Inou, “Cellular robots forming a mechanical structure,” in *6th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 139–148, Springer, 2007.

[46] R. Oung, A. Ramezani, and R. D’Andrea, “Feasibility of a distributed flight array,” in *48th IEEE Conference on Decision and Control, held jointly with the 28th Chinese Control Conference (CDC/CCC)*, pp. 3038–3044, IEEE, 2009.

[47] M. J. Doyle, X. Xu, Y. Gu, F. Perez-Diaz, C. Parrott, and R. Groß, “Modular Hydraulic Propulsion: A robot that moves by routing fluid through itself,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5189–5196, IEEE, 2016.

[48] D. Rus and M. Vona, “A physical implementation of the self-reconfiguring crystalline robot,” in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1726–1733, IEEE, 2000.

[49] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji, “A 3-D self-reconfigurable structure,” in *1998 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 432–439, IEEE, 1998.

[50] J. W. Romanishin, K. Gilpin, S. Claici, and D. Rus, “3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1925–1932, IEEE, 2015.

[51] J. Neubert and H. Lipson, “Soldercubes: A self-soldering self-reconfiguring modular robot system,” *Autonomous Robots*, vol. 40, no. 1, pp. 139–158, 2016.

[52] B. Piranda and J. Bourgeois, “Designing a quasi-spherical module for a huge modular robot to create programmable matter,” *Autonomous Robots*, pp. 1–15, 2018.

[53] G. Liang, H. Luo, M. Li, H. Qian, and T. L. Lam, “FreeBOT: A freeform modular self-reconfigurable robot with arbitrary connection point-design and implementation,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[54] C. Parrott, T. J. Dodd, and R. Groß, “HyMod: A 3-DOF hybrid mobile and self-reconfigurable modular robot and its extensions,” in *13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 401–414, Springer, 2016.

[55] C. Parrott, *A hybrid and extendable self-reconfigurable modular robotic system*. PhD thesis, University of Sheffield, 2016.

[56] E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund, “Design of the ATRON lattice-based self-reconfigurable robot,” *Autonomous Robots*, vol. 21, no. 2, pp. 165–183, 2006.

[57] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, “M-TRAN: Self-reconfigurable modular robotic system,” *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[58] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, “M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar,” in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2454–2459, IEEE, 2003.

[59] B. Salemi, M. Moll, and W.-M. Shen, “SuperBot: A deployable, multi-functional, and modular self-reconfigurable robotic system,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3636–3641, IEEE, 2006.

[60] A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert, “Roombots – mechanical design of self-reconfiguring modular robots for adaptive furniture,” in *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4259–4264, IEEE, 2009.

[61] V. Zykov, W. Phelps, N. Lassabe, and H. Lipson, “Molecubes extended: Diversifying capabilities of open-source modular robotics,” in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on “Self-Reconfigurable Robotics”*, pp. 22–26, IEEE, 2008.

[62] J. Davey, N. Kwok, and M. Yim, “Emulating self-reconfigurable robots-design of the SMORES system,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4464–4469, IEEE, 2012.

[63] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "An end-to-end system for accomplishing tasks with modular robots.," in *Robotics: Science and Systems XII*, p. 25, RSS Foundation, 2016.

[64] M. D. M. Kutzer, M. S. Moses, C. Y. Brown, M. Armand, D. H. Scheidt, and G. S. Chirikjian, "Design of a new independently-mobile reconfigurable modular robot," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2758–2764, IEEE, 2010.

[65] K. C. Wolfe, M. S. Moses, M. D. M. Kutzer, and G. S. Chirikjian, "M^3 Express: A low-cost independently-mobile reconfigurable modular robot," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2704–2710, IEEE, 2012.

[66] D. Bie, I. Sajid, J. Han, J. Zhao, and Y. Zhu, "Natural growth-inspired distributed self-reconfiguration of UBot robots," *Complexity*, vol. 2019, pp. 1–12, 2019.

[67] G. J. Hamlin and A. C. Sanderson, "Tetrobot: A modular system for hyper-redundant parallel robotics," in *1995 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 154–159, IEEE, 1995.

[68] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Variable topology truss: Design and analysis," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2717–2722, 2017.

[69] S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, no. 6, pp. 99–101, 2005.

[70] M. Shimizu, A. Ishiguro, and T. Kawakatsu, "Slimebot: A modular robot that exploits emergent phenomena," in *2005 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2982–2987, IEEE, 2005.

[71] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. Jan Ijspeert, "Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with bluetooth interface," *Industrial Robot*, vol. 33, no. 4, pp. 285–290, 2006.

[72] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 423–443, 2008.

[73] H. Zhang, J. Gonzalez-Gomez, Z. Me, S. Cheng, and J. Zhang, "Development of a low-cost flexible modular robot GZ-I," in *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 223–228, IEEE, 2008.

[74] G. G. Ryland and H. H. Cheng, "Design of iMobot, an intelligent reconfigurable mobile robot with novel locomotion," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 60–65, IEEE, 2010.

[75] A. Castano and P. Will, "Mechanical design of a module for reconfigurable robots," in *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2203–2209, IEEE, 2000.

[76] W.-M. Shen, R. Kovac, and M. Rubenstein, “SINGO: A single-end-operative and genderless connector for self-reconfiguration, self-assembly and self-healing,” in *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4253–4258, IEEE, 2009.

[77] C. Parrott, T. J. Dodd, and R. Groß, “HiGen: A high-speed genderless mechanical connection mechanism with single-sided disconnect for self-reconfigurable modular robots,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3926–3932, IEEE, 2014.

[78] D. Marbach and A. J. Ijspeert, “Online optimization of modular robot locomotion,” in *2005 IEEE International Conference on Mechatronics and Automation (ICMA)*, vol. 1, pp. 248–253, IEEE, 2005.

[79] V. Zykov, E. Mytilinaios, M. Desnoyer, and H. Lipson, “Evolved and designed self-reproducing modular robotics,” *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 308–319, 2007.

[80] M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L. M. Gambardella, “The SWARM-BOTS project,” in *1st International Workshop on Swarm Robotics*, pp. 31–44, Springer, 2004.

[81] J. W. Romanishin, K. Gilpin, and D. Rus, “M-Blocks: Momentum-driven, magnetic modular robots,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4288–4295, IEEE, 2013.

[82] K. Misumi, G. Ulliac, N. Usami, B. Piranda, Y. Mita, A. Higo, and J. Bourgeois, “Micro-scale electrostatic attach-detach device for self-reconfigurable modular robotic system,” in *2020 Symposium on Design, Test, Integration & Packaging of MEMS and MOEMS (DTIP)*, pp. 1–4, IEEE, 2020.

[83] A. Brunete, A. Ranganath, S. Segovia, J. P. de Frutos, M. Hernando, and E. Gambao, “Current trends in reconfigurable modular robots design,” *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, pp. 1–21, 2017.

[84] A. Bhattacharjee, Y. Lu, A. T. Becker, and M. Kim, “Magnetically controlled modular cubes with reconfigurable self-assembly and disassembly,” *IEEE Transactions on Robotics*, 2021.

[85] K. Gilpin, K. Koyanagi, and D. Rus, “Making self-disassembling objects with multiple components in the Robot Pebbles system,” in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3614–3621, IEEE, 2011.

[86] C.-A. Chen, A. Kamimura, L. Barrios, and W.-M. Shen, “Dynamic power sharing for self-reconfigurable modular robots,” in *14th Towards Autonomous Robotic Systems (TAROS)*, pp. 3–14, Springer, 2013.

[87] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A. J. Ijspeert, “Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot,” *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1016–1033, 2014.

[88] P. J. White, K. Kopanski, and H. Lipson, “Stochastic self-reconfigurable cellular robotics,” in *2004 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, pp. 2888–2893, IEEE, 2004.

[89] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen, “Programmable parts: A demonstration of the grammatical approach to self-organization,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3684–3691, IEEE, 2005.

[90] J. A. Escalera, M. Doyle, F. Mondada, and R. Groß, “Evo-bots: A simple, stochastic approach to self-assembling artificial organisms,” in *13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Springer, 2016.

[91] P. J. White and M. Yim, “Scalable modular self-reconfigurable robots using external actuation,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2773–2778, IEEE, 2007.

[92] M. T. Tolley, M. Kalontarov, J. Neubert, D. Erickson, and H. Lipson, “Stochastic modular robotic systems: A study of fluidic assembly strategies,” *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 518–530, 2010.

[93] J. Neubert, A. P. Cantwell, S. Constantin, M. Kalontarov, D. Erickson, and H. Lipson, “A robotic module for stochastic fluidic assembly of 3D self-reconfiguring structures,” in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2479–2484, IEEE, 2010.

[94] K. D. Chu, S. G. M. Hossain, and C. A. Nelson, “Design of a four-DOF modular self-reconfigurable robot with novel gaits,” in *2011 ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 747–754, ASME, 2011.

[95] G. Qiao, G. Song, J. Zhang, H. Sun, W. Wang, and A. Song, “Design of Transmote: A modular self-reconfigurable robot with versatile transformation capabilities,” in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1331–1336, IEEE, 2012.

[96] W. Wang, G. Qiao, G. Song, Y. Zhang, and Y. Wang, “Design and implementation of a new intelligent modular reconfigurable robot,” in *2013 IEEE International Conference on Information and Automation (ICIA)*, pp. 799–804, IEEE, 2013.

[97] J. Baca, S. G. M. Hossain, P. Dasgupta, C. A. Nelson, and A. Dutta, “ModRED: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration,” *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1002–1015, 2014.

[98] Z. Butler, K. Kotay, D. Rus, and K. Tomita, “Generic decentralized control for lattice-based self-reconfigurable robots,” *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 919–937, 2004.

[99] D. J. Christensen, E. H. Østergaard, and H. H. Lund, “Metamodule control for the ATRON self-reconfigurable robotic system,” in *8th Conference on Intelligent Autonomous Systems (IAS)*, pp. 685–692, 2004.

[100] J.-A. Leal-Naranjo, S. Fichera, and P. Paoletti, “Towards a modular robotic platform for construction and manufacturing,” in *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 1197–1202, IEEE, 2021.

[101] M. J. Doyle, J. V. A. Marques, I. Vandermeulen, C. Parrott, Y. Gu, X. Xu, A. Kolling, and R. Groß, “Modular fluidic propulsion robots,” *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 532–549, 2020.

[102] W.-M. Shen, F. Hou, M. Rubenstein, H. Chiu, and A. Kamimura, “Recent progress of SuperBot,” in *2010 IEEE International Conference on Robotics and Automation (ICRA) Workshop on“Modular Robots: The State of the Art”*, pp. 13–16, IEEE, 2010.

[103] K. Støy, W.-M. Shen, and P. M. Will, “Using role-based control to produce locomotion in chain-type self-reconfigurable robots,” *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 410–417, 2002.

[104] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw, “Connecting and disconnecting for chain self-reconfiguration with PolyBot,” *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 442–451, 2002.

[105] K. Payne, J. Everist, F. Hou, and W.-M. Shen, “Single-sensor probabilistic localization on the SeReS self-reconfigurable robot,” in *9th International Conference on Intelligent Autonomous Systems (IAS)*, pp. 207–216, 2006.

[106] W. Liu and A. F. T. Winfield, “Implementation of an IR approach for autonomous docking in a self-configurable robotics system,” in *13th Towards Autonomous Robotic Systems (TAROS)*, pp. 251–258, 2009.

[107] S. Pouya, J. Van Den Kieboom, A. Spröwitz, and A. J. Ijspeert, “Automatic gait generation in modular robots: “To oscillate or to rotate; that is the question”,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 514–520, IEEE, 2010.

[108] A. Ranganath, J. Gonzalez-Gomez, and L. M. Lorente, “Morphology dependent distributed controller for locomotion in modular robots,” in *2012 Post-Graduate Conference on Robotics and Development of Cognition (RobotDoC-PhD)*, pp. 44–47, Citeseer, 2012.

[109] J. Liedke, R. Matthias, L. Winkler, and H. Wörn, “The collective self-reconfigurable modular organism (CoSMO),” in *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1–6, IEEE, 2013.

[110] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, “An integrated system for perception-driven autonomy with modular robots,” *Science Robotics*, vol. 3, no. 23, p. eaat4983, 2018.

[111] Y. Zhu, D. Bie, S. Iqbal, X. Wang, Y. Gao, and J. Zhao, “A simplified approach to realize cellular automata for ubot modular self-reconfigurable robots,” *Journal of Intelligent & Robotic Systems*, vol. 79, no. 1, pp. 37–54, 2015.

[112] K. Hosokawa, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo, “Self-organizing collective robots with morphogenesis in a vertical plane,” in *1998 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2858–2683, IEEE, 1998.

[113] E. Yoshida, S. Murata, S. Kokaji, A. Kamimura, K. Tomita, and H. Kurokawa, “Get back in shape! A hardware prototype self-reconfigurable modular microrobot that uses shape memory alloy,” *IEEE Robotics and Automation Magazine*, vol. 9, no. 4, pp. 54–60, 2002.

[114] G. S. Chirikjian, “Kinematics of a metamorphic robotic system,” in *1994 IEEE International Conference on Robotics and Automation*, pp. 449–455, IEEE, 1994.

[115] J. W. Suh, S. B. Homans, and M. Yim, “Telecubes: Mechanical design of a module for self-reconfigurable robotics,” in *2002 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 4095–4101, IEEE, 2002.

[116] C. Ünsal, H. Kiliççöte, and P. K. Khosla, “I(CES)-cubes: A modular self-reconfigurable bipartite robotic system,” in *1999 SPIE Conference on Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, pp. 258–270, SPIE, 1999.

[117] D. Hjelle and H. Lipson, “A robotically reconfigurable truss,” in *1st ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots (ReMAR)*, pp. 73–78, ASME, 2009.

[118] C. Ünsal, H. Kiliççöte, and P. K. Khosla, “A modular self-reconfigurable bipartite robotic system: Implementation and motion planning,” *Autonomous Robots*, vol. 10, no. 1, pp. 23–40, 2001.

[119] K. C. Prevas, C. Ünsal, M. O. Efe, and P. K. Khosla, “A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system,” in *2002 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 787–792, IEEE, 2002.

[120] K. Kotay and D. Rus, “Algorithms for self-reconfiguring molecule motion planning,” in *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2184–2193, IEEE, 2000.

[121] J. E. Walter, J. L. Welch, and N. M. Amato, “Distributed reconfiguration of metamorphic robot chains,” in *19th Annual ACM Symposium on the Principles of Distributed Computing*, pp. 171–180, ACM, 2000.

[122] M. Yim, Y. Zhang, J. Lamping, and E. Mao, “Distributed control for 3D metamorphosis,” *Autonomous Robots*, vol. 10, no. 1, pp. 41–56, 2001.

[123] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji, “Distributed formation control for a modular mechanical system,” in *1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1090–1097, IEEE, 1997.

[124] W.-M. Shen, B. Salemi, and P. Will, “Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots,” *IEEE transactions on Robotics and Automation*, vol. 18, no. 5, pp. 700–712, 2002.

[125] B. Piranda and J. Bourgeois, “A distributed algorithm for reconfiguration of lattice-based modular self-reconfigurable robots,” in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pp. 1–9, IEEE, 2016.

[126] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, “A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots,” in *IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 254–263, IEEE, 2016.

[127] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, “Autonomous self-assembly in swarm-bots,” *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1115–1130, 2006.

[128] I. Parada, V. Sacristán, and R. I. Silveira, “A new meta-module for efficient reconfiguration of hinged-units modular robots,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5197–5202, IEEE, 2016.

[129] M. Rubenstein, C. Ahler, and R. Nagpal, “Kilobot: A low cost scalable robot system for collective behaviors,” in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3293–3298, IEEE, IEEE, 2012.

[130] M. T. Tolley and H. Lipson, “On-line assembly planning for stochastically reconfigurable systems,” *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1566–1584, 2011.

[131] M. D. Hall, “Modular Active Subtraction Simulator for 2-Dimensions (MASS2D).” https://gitlab.com/natural-robotics-lab/mass2d, 2021.

[132] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, “A time synchronization protocol for large-scale distributed embedded systems with low-precision clocks and neighbor-to-neighbor communications,” *Journal of Network and Computer Applications*, vol. 105, pp. 123–142, 2018.

[133] M. D. Hall, “Supplementary animations for Active Subtraction: A Viable Method of Self-Reconfiguration for Modular Robotic Systems.” https://youtu.be/RgGi_mc4tOQ, 2022.

[134] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[135] P. Thalamy, B. Piranda, and J. Bourgeois, “Engineering efficient and massively parallel 3D self-reconfiguration using sandboxing, scaffolding and coating,” *Robotics and Autonomous Systems*, vol. 146, no. 103875, pp. 1–18, 2021.

[136] N. Inou, K. Minami, and M. Koseki, “Group robots forming a mechanical structure - Development of slide motion mechanism and estimation of energy consumption of the structural formation,” in *2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 2, pp. 874–879, IEEE, 2003.

[137] B. T. Kirby, M. Ashley-Rollman, and S. C. Goldstein, “Blinky blocks: a physical ensemble programming platform,” in *CHI ’11 extended abstracts on human factors in computing systems*, pp. 1111–1116, AMC, 2011.

[138] E. Bray and R. Groß, “Distributed self-assembly of cantilevers by force-aware robots,” in *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 110–118, IEEE, 2021.

[139] B. Piranda, P. Chodkiewicz, P. Hołobut, S. P. A. Bordas, J. Bourgeois, and J. Lengiewicz, "Distributed prediction of unsafe reconfiguration scenarios of modular robotic programmable matter," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2226–2233, 2021.

[140] P. Hołobut and J. Lengiewicz, "Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning," in *2017 IEEE International Conference on Robotics and Automation (ICRA),*, pp. 2103–2109, IEEE, 2017.

[141] T. Tucci, B. Piranda, and J. Bourgeois, "A distributed self-assembly planning algorithm for modular robots," in *17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 550–558, International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[142] A. Casal and M. Yim, "Self-reconfiguration planning for a class of modular robots," in *1999 SPIE Conference on Sensor Fusion and Decentralized Control in Robotic Systems II*, vol. 3839, pp. 246–258, SPIE, 1999.