République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Superieur et de la Recherche Scientifique
Université Mohamed Khider Biskra

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

THESE

Présentée pour obtenir le grade de
DOCTORAT EN SCIENCE EN INFORMATIQUE
Par

Tarek Ababsa

THEME

Emergence des Structures Evolutionnistes à Base de Robots Cellulaires Auto-Organisés

Emergent Structures in Self-Reconfiguring Cell Robots

Soutenue le: 11/12/2016

Devant le jury composé de:

Pr. Abdelmalik Bachir	Université de Biskra	Président
Pr. NourEddine Djedi	Université de Biskra	Rapporteur
Pr. Foudil Cherif	Université de Biskra	Examinateur
Pr. Abdelouahab Moussaoui	Université de Setif	Examinateur
Pr. Mohamed Benmohamed	Université de Constantine	Examinateur
Pr. Mohamed Chaouki Batouche	Université de Constantine	Examinateur

Abstract:
Modular robots are complex systems composed of a set of simple robotic units. These units can rearrange to change the shape of the whole structure. This ability enables the reconfigurable robots to be extremely adaptive to changes in the environment for performing real-world tasks. Unlike conventional robots such robots are: i) Modular, since they are assembled from numerous robotic modules. ii) Self-reconfigurable, since the modules themselves are able to change how they are combined. iii) Autonomous, since robots control themselves with-out human guidance. Such robots are attractive to study because of the several desirable characteristics they have, such as versatility, reliability and cheapness.

Challenges of enhancing modular robotic system capabilities not only are limited to designing reliable, responsive, and robust hardware, but also include developing software and algorithms that can effectively fulfill tasks through performing fundamental functions like shape-formation, locomotion, manipulation, ... etc. In this thesis, we address these challenges by exploring three central elements of autonomous self-reconfigurable modular robots: adaptation, self-reconfiguration and structure evolution.

Firstly, we tackled the problem of adaptation of modular robots in their environment, we proposed a three layered approach to evolve the structure of the robot. In the first layer a parallel GA engine is used for finding the most suitable morphology, while in the second layer a *PacMan-like* algorithm is used to transform the current morphology into the target morphology identified by the first layer. Secondly, we integrated a hormone signaling system into the model for allowing the system to perform parallel tasks and avoiding the system to get stuck in local minimum. Thirdly, we proposed a genetic programming based mechanism for finding the near-optimal sequence of primitive actions in order to reduce the cost of self reconfiguration.

Keywords : Modular Robots, Self-Reconfiguration, Adaptation, Structure Evolution, Emergent Behaviors.

Resumé:
Les robots modulaires sont des systèmes complexes composés d'un grand nombre d'unités élémentaires. Ces unités peuvent s'arranger pour changer la structure globale du système. Cette capacité permet aux robots modulaires de s'adapter aux changements de l'environnement pour réaliser des tâches relativement complexes. A l'inverse des robots conventionnels, ces robots sont i) Modulaires, puisqu'ils sont composés de plusieurs unités. ii) Auto Configurables, car les modules composant le système sont capables de changer la manière dont ils sont reliés pour pouvoir former plusieurs combinaisons iii) Autonomes, car ils sont capables eux-mêmes de réaliser des tâches sans avoir besoin d'intervention humaine. Cette discipline a attiré plusieurs chercheurs de la communauté scientifique pour réaliser des études dans ce domaine. Les défis d'améliorer les capacités des robots modulaires ne s'arrêtent pas à la conception mécanique et électronique, ils englobent également l'amélioration des algorithmes et des systèmes de contrôle pour que le système robotique se comporte de manière efficace. Dans cette thèse, nous nous proposons de relever ces défis en explorant trois éléments centraux de l'auto-reconfiguration des robots modulaires : l'adaptation, l'auto-configuration et l'évolution de la structure des robots. Dans un premier temps, nous nous sommes intéressés au problème d'adaptation des robots modulaires dans leur environnement. Pour cela, nous avons proposé une méthode en trois couches pour évoluer la structure des robots. Dans la première couche, un algorithme génétique réparti est utilisé pour trouver la structure optimale que le robot doit réaliser, tandis que dans la deuxième couche, nous avons utilisé l'algorithme *PacMan* pour transformer le système en une nouvelle configuration identifiée par la première couche. Dans une deuxième étape, nous avons intégré un système hormonal à l'architecture proposée pour permettre au système de réaliser des tâches en parallèle ainsi pour que le système soit capable d'agir de différentes manières pour ne pas se bloquer dans un seul état. Quant à la troisième étape, nous avons utilisé la programmation génétique pour reconfigurer le système de manière efficace et réduire le coût de la reconfiguration du système.

Mots clés : Robots modulaires, Auto-Reconfiguration, Adaptation, Evolution des structures, Emergence.

ملخص:

الروبوتات الوحدوية (Modular Robots) هي أنظمة معقدة مكونة من عدد كبير من الوحدات العنصرية. هته الوحدات يمكن لها أن تغير التنظيم فيما بينها لتشكل عددا كبيرا من الهياكل المختلفة الوظائف للتماشي مع متغيرات البيئة التي وضعت فيها و ذلك قصد تمكينها من انجاز المهمة الموكلة اليها. خلافا على الروبوتات ذات الهياكل المهندسة, فان الروبوتات الوحدوية تتميز بأنها أ) ذات بنية جزئية: اذ أن تركيبتها مبنية على تداخل العناصر فيما بينها, ب) ذاتية الهيكلة: حيث أن الوحدات المكونة لها قادرة على أن تتعاون فيما بينها لتغيير الهيكل العام للبنية دون تدخل خارجي, ج) ذاتية التصرف: حيث أنه بإمكانها القيام بمهام بشكل مستقل عن القيادة الخارجية. هذه الخصائص و غيرها جعلت من الروبوتات الوحدوية مجالا يستهوي العديد من الباحثين و ذلك لثرائه بالتحديات العلمية المفتوحة و التي لم يتم حلها لحد الساعة.

التحديات التي نتكلم عنها لا تقتصر فقط على الجانب الالكتروميكانيكي بل تتعداه الى ما دون ذلك من خوارزميات و أنظمة تحكم ذات فعالية لتمكينها من أداء مهامها بمردودية عالية. في هذه الأطروحة سوف نقوم بمواجهة بعض هته التحديات من خلال ثلاث عناصر وهي: الهيكلة الذاتية, المسايرة و التأقلم, و أخيرا تطور الهيكلة.

في المرحلة الأولى سوف نقوم بمواجهة مشكلة تأقلم الروبوتات الوحدوية مع بيئتها. لذلك قمنا بتطوير طريقة تعتمد على مرحلتين. المرحلة الأولى التي يتم بها تطوير الهيكلة للتماشي مع المستحدثات, بينما في المرحلة الثانية نقوم باستعمال خوارميات خاصة لتعديل الهيكلة. مواصلة منا بتطوير قدرات الروبوتات الجزئية قمنا بدمج نظام هورموني و ذلك لتمكينها من الانقسام و مزاولة مهامها بطريقة أكثر فعالية. في الأخير قمنا بتطوير طريقة تعتمد على البرمجة الجينية و ذلك لإيجاد سلسلة الأوامر التي تمكنها من اعادة الهيكلة بطريقة أقل كلفة.

الكلمات المفتاحية:

الروبوتات الوحدوية, الهيكلة الذاتية, التأقلم, التطور, السلوكيات الناشئة

Acknowledgments

All praise and thanks to my lord, Allah !

I am grateful to Prof. NourEddine Djedi my academic supervisor and equally to Prof. Yves Duthen, for their support, guidance, assistance and encouragements during this long journey.

I deeply thank Jury members: Prof. Foudil Cherif, Prof. Abdelmalik Bachir, Prof. Abdelouahab Moussaoui, Prof. Mohamed Benmohamed, and Prof. Mohamed Chaouki Batouche, for accepting to evaluate this work. In addition, I would like to express my sincere respect to the anonymous reviewers of the published papers who provided me with helpful critics, suggestions and guidance.

My thanks go to my colleagues at the computer science department of the university of Biskra. I am also grateful for friends who helped me. I sincerely thank them all for their support.

I would like to mention my family (my mother, my father, my wife, my children, my brothers, my sisters and all my relatives). I cannot say enough thankful to them simply because they are the reasons for me to keep on trying.

Finally, to all of my dear friends, who I cannot name all of them since I am afraid of missing someones, I sincerely appreciate their friendships, which have brought different meanings to my life.

Contents

List of Figures

List of Tables

1

Introduction and Motivation

The first chapter introduces the Modular Robotic Systems and highlights the motivation of the research conducted in the thesis

Contents

1.1 Introduction

When we think about the future, one of the most things that come to mind is the *Robot*. A robot is a machine performs useful actions and is capable of making decision about its behavior. Robots are the love children of computer and powerful electromechanical components. They come in all shapes and sizes, each specific to its function.

Since they first come on the see in 1960's, real world robots have slowly worked there where into the industry taking over more and more 3D (*Dirty, Dull, and Dangerous*) jobs for each passing year. In assembly line businesses, such as building cars, the ability of robots to perform complicated tasks over and over is perfect, but getting a robot off of its initial working area has proven far more difficult than science fiction films would have to believe. The robots should master an incredibly complex skill before fitting into our world. For example, walking is one of the major challenges that is very difficult for them to learn but comes naturally to us. In other word, if we want robots to live with us in an environment designed for humans, then they have to get the shape and the skill capacity of human beings.

Tele-operated robotic systems have introduced to fill the skill gaps. However, robotic systems must be smart enough to get the job done, especially if there is a time delay between human action and machine response. The revolution in computing power combined with new approaches for building robot brains has resulted in new best robotic systems ever devised.

Sophisticated robots are already called into the human service. But what of all these robots lack is versatility. Indeed, there is no single robot that is good in everything. The conventional approach in designing robots has been to design their hardware and software in conformance with the tasks they are supposed to do. Conventional robots are successfully used in industry to perform repetitive tasks at much higher speed and precision than humans can ever do. However, these robots are not very flexible and adaptive, and thus applications consigned to them heavily

depend to their physical structure on the one hand and their controller capabilities on the other hand.

Even though robotics have clearly shown its usefulness, it is only used to a limited extent by complex tasks. What we mean here by complex tasks is those tasks where the operating conditions and ability requirements are substantially more challenging than the predictable conditions. One important thing to note is that the problem of adaptive locomotion is indeed so hard that it has not been completely solved by nature. To the best of our knowledge, no animal is able to move in any environment efficiently. For example an albatross is really comfortable in the air because of the size of its wings but not at all on the ground for the same reason. In order to overcome this limitation, robots should be able to dynamically select the best shape according any environment. This idea brought the research community to contemplate autonomous kinematic machines with variable morphology or *self-reconfigurable modular robots*.

Real robots that change their shape, made up of many identical modules have been created and are being studied by a wide variety of groups.[3] These robots promise to be versatile, low cost, and robust. While these systems do not yet behave like liquid metal (possibly many million microscopic modules), systems of the order of 100 modules have been built and promise to be useful in search and rescue or space exploration.

The concept of morphologically variable robots was introduced for the first time by Fukuda and Nakagawa in 1988[4] with the biologically-inspired *CEllular roBOT* (*CEBOT*). They define that a self-reconfigurable robot consists of several cells (or modules), each cell have some measure of intelligence and mechanical capabilities. Modules have their own computational power and are able to communicate with other modules and sense the environment. The modules have connectors and actuated degrees of freedom that allow them to connect to, disconnect from and move relative to each other in order to deliberately change the shape of the whole structure. The design of the modules provides the self-reconfigurable robot with abilities such as failure tolerance, redundancy and the ability to self-repair.

With the electronic revolution and the appearance of 3D printing ma-

chines, the hardware field has dramatically progressed further with working implementations of many modular robot prototypes. These prototypes can be classified into three categories: lattice, chain, and hybrid based systems. In a lattice-based system, modules are restricted to move in a 3D grid. This gives them a regular structure and simplifies the interaction between modules. Discrete motion can occur by the repetitive movement of individual modules among each other, this is a highly flexible approach, and can allow the system to pass through relatively small gaps in obstacles. In a chain-based systems, modules are connected in long chains capable of continuous rather than discrete motion. This makes the connection more difficult, but simplifies the use of motion via high level actuation mechanisms such as walking, sliding, and so on. A hybrid system combines features of the lattice and chain paradigms.

The reasons for studying self-reconfigurable robots are at least two-fold. On one hand, the exploration of such systems may provide insight into the nature and principles of biological organisms: To understand biology, we must understand its basic principles such as modularity, redundancy, emergence, self-organization and self-replication. On the other hand, a reason for studying self-reconfigurable robots is their possible applications. Potential areas of applications include exploration of unknown environments such as space, collapsed building or earthquake areas, entertainment such as educational toys and industry such as flexible robot arms or reconfigurable machine tools. We believe that a strong focus on the basic research of biological principles will uncover new possibilities and open new doors for long-term applications. The study conducted in this thesis is not far from possible applications, while at the same time contains a more fundamental scientific value.

1.2 Motivation of the thesis

Although it is a relatively new field of inquiry, work in modular self-reconfigurable robots has already yielded demonstrations of working systems and significant advances in both hardware and software design.

However, exploiting the adaptive potential of changing morphologies has not been fully investigated.[3] It raises many issues not only in engineering design but also in understanding natural forms of intelligence.

Realizing the full capabilities of modular self-reconfigurable robots will require the development of systems with a very large number of modules. This poses a significant engineering challenge in hardware and software as well. Such for example, as the computation and communications methods associated with a central controller perform poorly at large scales, planning and control methods must be entirely decentralized. Ideally, each module must execute its own copy of the same code, reacting to locally sensed information and the one being passed to it by its neighbors.

One of the major challenges for the modular robotic systems is *Planning and Control Challenge*. Although algorithms have been developed for handling millions of units under specific ideal conditions, challenges to scalability remain both in low-level control and in high-level planning to overcome realistic constraints such as:[3]

- Algorithms for parallel motion for large-scale manipulation and locomotion with and without obstacles.
- Algorithms for optimal (time, energy) reconfiguration planning with and without obstacles.
- Algorithms for robustly handling a variety of failure modes, from misalignments and dead units (not responding, not releasing) to units that behave erratically.
- Algorithms that determine the optimal configuration for a given task and environment.
- Efficient and scalable (asynchronous) communication among multiple units.

Note that the added degrees of freedom make modular robots more versatile in their potential capabilities, but also incur a performance trade off and increased mechanical and computational complexities.

1.3 Problem Statement

Several key technical difficulties stand in the way of progress for modular robotic systems before they would be used in vast numbers for practical applications. In this section, we describe several problems to be addressed in this thesis. These problems can be expressed in four items as follows:

1. **Adaptation to terrain**: Since a score of years, several robotic systems have been proposed for locomotion over natural terrains. In real-world applications, self-reconfigurable robots are required to perform locomotion, manipulation, and self-reconfiguration tasks in the presence of obstacles and in an uncontrolled environment. Based on the different design experiences of such complex systems, we have noticed that a dynamic adaptation of the structure and its behavior is more than useful. The main reason for this fact is that the mechanical solutions for locomotion on rough terrain are multiples and none is ideal for all situations. Rus et al.[5,6] presented general distributed algorithms for adaptive locomotion and shape formation, over a terrain with unknown obstacles. However, in this work, the authors used hand-designed algorithms which took hours of design time to synthesize. Besides, they considered an abstract modular robot and it is not clear how easy it will be to translate the abstract model of sensing to actual hardware. Chih et al.[7] presented a full decentralized algorithm for the self-organization of environmentally-adaptive shapes to form a flexible sheet structure. A chain-based modular robot is used to form a flexible surface. However, the structure has been already designed, meaning that the system complexity is significantly reduced.

   Reinforcement learning has been used in[8] for locomotion and adaptation in modular robots. The proposed approach suffers from local optima where the robot is effectively stuck in a configuration from which it will not be able to move. Besides, the scalability remains to be demonstrated.

   Leading robots to learn through experience for reaching the highest

adaptability grade to the perceived information has been largely explored by searchers[9–12] . The learning process has specially proven their capability of learning for specific situations. The achieved controllers perform well for the trained problems and make the process useful in the search for solutions. However, they do not perform well when tested in environments different from the trained ones.

When the environment is highly disordered, systems that use cognitive architectures or those that use learning process perform poorly in their environment due to the limitation resources for perceiving the world and affecting it. Thus, it is important to look forward into alternative approaches where the autonomous system adjusts itself via perception and interaction with the environment.

2. **Failure recovery**: Whenever we think of keeping all of our systems up and running in an environment, we very often think about what can happen if the systems suffer physical damage following a hardware failure. So we have to think about redundancy and fault tolerance.

   One of the major desirable features of modular robotic systems is their ability to recover from failures by ejecting and replacing damaged or malfunctioning modules. In such a case, the control system should be responsive to the new situation to allow the robot to reach a repair station or even to finish its task without the need to exchange the broken modules (recovery behavior would be created to minimize the consequences of the damage).

   Failure in modular robotic systems is mainly occurred for two reasons: *communication errors* or *joint failures*. Possible solutions to overcome this problem usually mentioned in literature[3,13,14] , some of them need to be adapted for the whole environment, others used the motion planning with the adaptation of motion primitives for failure recovery. It is interesting to study the impact of the failure recovery on the behavior of the system.

3. **Performing parallel tasks**: The most important characteristic of modular robotic systems is that they are composed of a large

number of units. Although, it has been proved in theory that the reconfiguration of modular robots linear scales with the number of modules,[15] it is also interesting to give importance to the self reproduction of the system since the huge number of modules gives material for the modular robot to *replicate* itself in several independent smaller robots with the same basic functionality (but not identical size). Self assembly methods can be applied to aggregate the units again in order to build the original system as the global task is achieved.

Self-replicating robots are useful in tasks where the overall effectiveness and task completion time is improved by parallelism. Thus, it is important to empower modular robots with the ability to split up when they need to explore different directions.[16,17]

4. **Planning efficient self reconfiguration**: In modular robots, the self-reconfiguration problem is difficult for planning because of two reasons. On one hand, the robotic units are obstacles to each other, and on the other hand, the search space (the number of possible sequences of configurations) grows exponentially with the number of modules in the system.

   Since the mechanical actions (expand, contract, attach, detach) performed by the modules are typically the slowest part of the system, it would be desirable to design an optimal algorithm that minimizes the total number of mechanical operations required to reach the final configuration.[15,18]

1.4 Contributions

To address some of the challenges described previously, this thesis makes three major contributions: (i) an approach for evolving the structure of metamorphic robot in a decentralized fashion, (ii) empower the modular robot with the ability to split up in response to environmental needs (iii) find a near-optimal sequence for self reconfiguring modular robots.

1. **Decentralized approach to evolve the structure of metamorphic robots**:[19] In which, we propose a three-layered motion planning approach where at the higher level a parallel genetic algorithm (GA) search is used for finding the most suitable morphology that a lattice based modular robot (particularly *"Crystalline"*) must assume during its Flow among obstacles toward a goal position. The genome data structure in the proposed GA contains coordinates of modules in a grid environment, and the fitness function evaluates the Euclidean distance of the robot's center of mass to the destination position. In such a model, the fitness function does not include any subjective information about *"how to accomplish the task"* but objective information about *"how the task has been accomplished"*.

   Since each module runs the same copy of the genetic algorithm, the population can be divided among modules so that the planning is done in a decentralized manner. Once the next morphology of the modular robot is determined, the lower level plans motions of the modules using a *PacMan-like* algorithm to transform the current morphology to the target morphology identified by the higher level.

   Our model, takes advantage of the computational power of each unit in the system where the majority of the existing systems neglect this capability. In the existing older modular robotic systems, the embedded CPUs were limited just for controlling the actuators and managing the inter-modules communications. However, the last generation of micro CPUs is extremely different from the older micro-controllers and they are much more powerful in term of performance. Thus, we believe that involving the computational power of each module will give the system another dimension and it will open the doors for various quantities of interest.

2. **Splittable Metamorphic Carrier Robots**:[17] Here we improve our previous work as follows: Firstly, we used the concept of positional information which is most commonly thought to be implemented by morphogen gradients. The main reason for that,

is to overcome the U-Shaped obstacles, while the second reason is to allow the robot to respond in a concentration-dependent manner rather than dealing with any global point of information. Secondly, we integrate a hormone signaling system to control the robot behavior. As a result, the system is being able to replicate itself in several smaller parts to perform a parallel task, moreover, the robot is being able to escape the local optima in almost cases.

3. **Genetic Programming-based Self-Reconfiguration Planning for Metamorphic Robot**[18] (accepted paper): In which we use the genetic programming paradigm as an automatic programming tool for finding the near-optimal sequence of primitive actions which are required for reshaping the robot into the desired configuration. The proposed model is intended for both *Crystalline* and *Telecube* robots; however, it can be easily extended to any lattice based modular robot.

1.5 Thesis outline

This thesis contains state of the art study, a description of the proposed contributions, and some conclusion and perspectives. This content is organized in 8 chapters as follows:

- **Chapter 2** presents a background material on: Self Reconfigurable Modular Robots. Some mechanical designs and properties.

- **Chapter 3** presents a background material on: Self Reconfiguration Algorithms which exist in the literature.

- **Chapter 4** Introduces Artificial Life and Morphogenetic Engineering as a novel field of inspiration for developmental robotics.

- **Chapter 5** introduces an approach for evolving the structure of metamorphic robots.

- **Chapter 6** describes how to use a finite state machine coupled with a hormone signaling system to control the behavior of the modular robot, and how to allow the system to split up into several parts to perform the parallel task.

- **Chapter 7** introduces a genetic programming as an automatic programming tool for finding a near-optimal sequence of primitive actions which are required to reform the robot into the desired shape.

- **Chapter 8** concludes the thesis a summary of contributions, and outlines some future research directions.

List of Papers

A Tarek Ababsa, Nouredinne Djedi, Yves Duthen, Sylvain Cussat-Blanc. Decentralized Approach to Evolve the Structure of Metamorphic Robots (regular paper). In : IEEE Symposium on Artificial Life (IEEE-ALife 2013), Singapore, 16/04/2013-19/04/2013, IEEE e-suppor.

B Tarek Ababsa, Nouredinne Djedi, Yves Duthen, Sylvain Cussat-Blanc. Splittable Metamorphic Carrier Robots (poster). In : Artificial Life, New York, 30/07/2014-02/08/2014, The MIT Press e-support.

C Tarek Ababsa, Nouredinne Djedi, Yves Duthen. Genetic Programming-based Self-Reconfiguration Planning for Metamorphic Robot. **accepted paper**. International Journal of Automation and Computing IJAC, 2016.

Part I

State of the Art

2

Self-Reconfigurable Modular Robots

This chapter introduces the concept of the modular robot and highlights the mechanical designs of some existing modular robotic systems

Contents

2.1 Introduction

The concept of *self-reconfigurable robots* was introduced by Fukuda and Nakagawa[4] in the late 1980s, they define that a self-reconfigurable robot is an autonomous robot with a variable morphology that consists of several cells (or modules). The modules have connectors and actuated degrees of freedom that allow them to connect to, disconnect from and move relative to each other in order to morph the whole structure into different shapes without outside assistance. The design of the modules provides the self-reconfigurable robot with abilities such as morphing into the optimal configuration in a given situation, failure recovery and ability to self-repair as well.

There are at least two reasons for studying self-reconfigurable robots: On one hand, the exploration of such systems may provide insight into the nature of biological systems: To understand biology, we must understand its basic principles such as modularity, redundancy, emergence, self-organization and self-replication. On the other hand, a reason for studying self-reconfigurable robots are their possible applications in a variety of domains such as industry (modular manipulators), search and rescue in collapsed buildings (snake robot), or even entertainment (educational toys).

There are several ways of categorizing modular robotic systems. One is based on the geometric arrangement of their units[3] (Lattice vs. chain vs. mobile), and another is based on the methods of moving between the locations for attaching[20] (Stochastic vs. deterministic).

2.2 Lattice Modular Robots

A lattice-based modular robotic system has modules arranged and connected in some regular, three-dimensional pattern, such as a simple cubic or hexagonal grid. In this category, there are discrete positions that a

given module can occupy. In contrast to chain-based architectures where modules are free to move in continuous space, the grid based structure of lattice systems generally simplifies the reconfiguration process. Control and motion can be executed in parallel. Lattice-based modular robotic systems usually offer simpler reconfiguration, as modules move to a discrete set of neighboring locations.

Over a couple of years, a large number of working lattice platforms have been developed. Many module designs have been proposed, ranging from the very simple to relatively complex. A number of different connection mechanisms have been developed for physically joining modules and a variety of different actuation methods have been proposed for controlling and reconfiguring systems. Some of these platforms are described in this section as follows:

2.2.1 Crystalline Robots

First introduced by Rus et al.[21] (Figure 2.1). *Crystalline* robot is a mechanism with some of the same motive properties as biological muscles that can be closely packed in 3D space and that can attach themselves to similar units. Each of the Crystal modules is actuated by expansion and contraction of its four faces. By expanding and contracting the neighbors in a connected structure, an individual module can be moved in general ways relative to the entire structure. Crystal atoms never rotate relative to each other. Their relative movement is actuated by sliding via expansion/contraction. This basic operation has yielded new algorithms for global self-reconfiguration planning. When fully contracted, each atom is a square measuring two inches on each of its sides. When fully expanded, each atom is a square with a four-inch side (twice the contracted size). Crystal robot systems can realize a wide range of geometries. Crystalline robot systems are dynamic structures. They move using sequences of reconfigurations to implement locomotion gaits and undergo shape metamorphosis. The dynamic nature of these systems is supported by the ability of individual modules to move globally relative to the entire structure. Unlike other developed unit modules,

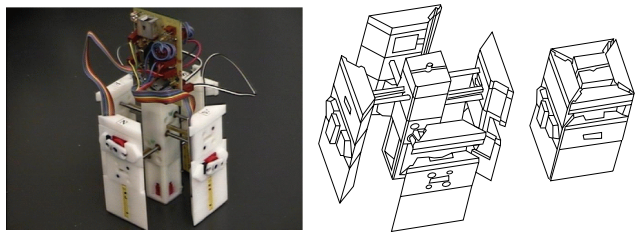

Figure 2.1: The physical prototype for the Crystalline Atom.

in which modules are relocated only by traveling on the surface of a structure, Crystalline atoms can be relocated by traveling throughout the volume of a Crystal. Instead of propagating the module along the surface of the robot structure, the same goal can be achieved using a constant number of internal expansion and compression operations.[21]

The first locomotion experiment was designed to evaluate whether Atoms could work together to effect a reconfiguration. In this experiment (see Figure 2.2), both **a** and **b** were initially contracted. **a** was connected to **0** (at **a.n**) and **b** was connected to **1** (at **b.n**). **a** and **b** were connected together at **b.w**. The Atoms were programmed with state sequences designed to perform a variant of inchworm translation along the fixed connectors.

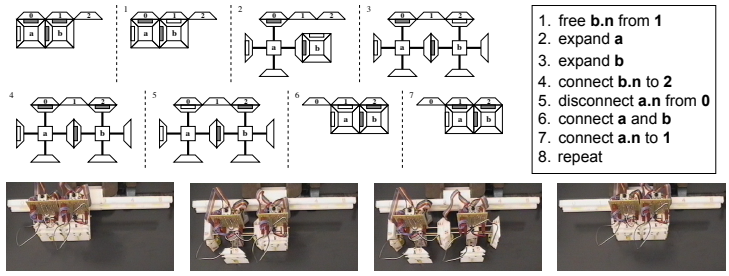

Figure 2.2: Several snapshots of the Atom prototype hardware performing the inchworm experiment.

2.2.2 Molecule Robots

A Molecule robot[22] consists of multiple units called *Molecules* (Figure 2.3). The Molecule is capable of independent movement on a substrate of identical Molecules, including straight-line traversal and 90 degrees convex and concave transitions to adjacent surfaces (Figure 2.4).

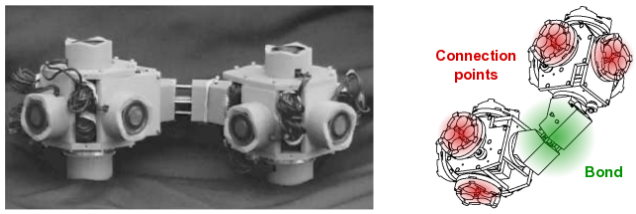

Figure 2.3: Molecule Robot

The Molecule is a 4 degree-of-freedom, small-scale module capable of aggregating with other identical modules to form three-dimensional dynamic structures. The Molecule consists of two atoms linked by a rigid connection called a bond. Each atom has five inter-Molecule connection points and two degrees of freedom. One degree of freedom allows the atom to rotate 180 degrees relative to its bond connection, and the other degree of freedom allows the atom (thus the entire Molecule) to rotate 180 degrees relative to one of the inter-Molecule connectors at a right angle to the bond connection.

The rotating connection points on each atom are the only ones required for Molecule motion. The other connection points are used for attachment to other Molecules in order to create 3D structures. Each Molecule also contains a microprocessor and the circuitry needed to control the servomotors and connectors.

Each individual Molecule has three basic motion capabilities. Linear motion on the top of identical Molecules, convex 90-degrees transitions between two planar surfaces composed of Molecules, and concave 90-degrees transitions between two planar surfaces composed of Molecules.

These primitive motions for a Molecule relative to a substrate can be combined and sequenced by the robot control system to achieve global

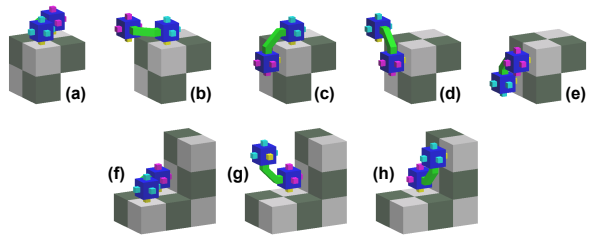

Figure 2.4: (a-e) convex transition sequence, (f-h) concave transition sequence

motions of the entire robot. It has been found that a four-Molecule robot is the smallest one that can move in general ways in the plane.[22]

2.2.3 TeleCube

Similar in design to Crystallyne modules used by Rus et al. in[21] , these cube shaped modules are designed and constructed at Xerox PARC[23] (Figure 2.5). The Telecube module has the ability to independently extend out each of the 6 faces (called *plats*). These extensions and retractions provide the modules particular form of motion. The arms can extend independently up to half of the body length, giving the robot an overall 2:1 expansion ration along each dimension. A latching mechanism on the plats on the end of each arm enables two aligned modules to connect to each other. For power routing, communication and alignment reasons, the modules must remain globally connected to each other at all times.

The modules have the following low level primitives:

- *ExtendArm(Direction)*: if there is a free space to extend the arm in *Direction*, then extend the arm.

- *RetractArm(Direction)*: Retract the arm in *Direction*, attempting first to disconnect connections which are perpendicular to *Direction*.

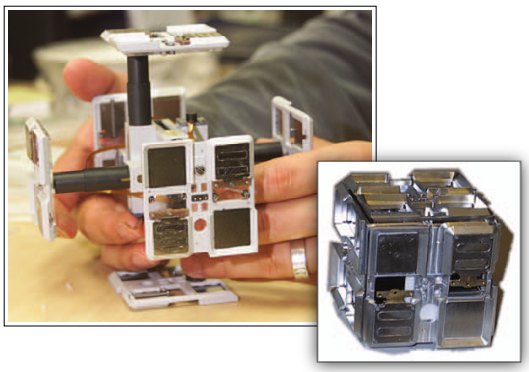

Figure 2.5: Telecube Robot

- *Connect(Direction)*: if there is a neighboring module in *Direction*, then latch to that module.
- *Disconnect(Direction)*: if there is a module is currently latched in *Direction*, then break the connection in between.

Prior to moving, the module must confirm that it has at least one neighbor along the direction of motion on which it can push or pull. Moreover, it must disconnect from all neighbors perpendicular to the direction of movement.

2.3 Chain Modular Robots

A chain based modular robotic system consists of modules arranged in groups of connected serial chains, forming tree and loop structures. Since these modules are typically arranged in an arbitrary point in space, the coordination of a reconfiguration is complex. In particular, forward and inverse kinematics, motion planning, and collision detection are problems that do not scale well as the number of modules increases.

The reconfiguration scheme in the chain based modular robot is achieved by detaching an array of modules from one point of the architecture, and reattaching it at a different point while maintaining the physical connectivity of the entire assembly. This is realized either autonomously,

or under human supervision. Some of the chain based modular robots are presented in this section as follows:

2.3.1 PolyBot

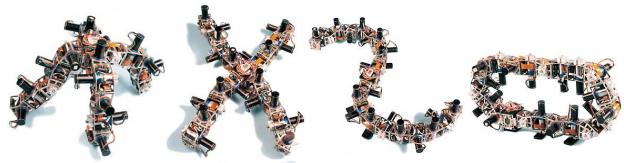

Figure 2.6: PolyBot Robot

PolyBot[24] (Figure 2.6) is one of the earliest examples of a chain-based modular robot. It is composed of two types of modules; a flexible segment that provides actuation with 1 DOF and 2 connection ports, and a static node that supplies power and serves as a branching point within structures.

In this platform, the main design goal is that each module is very simple and cannot do very much by itself. In combination with many others a more complex system can be built to achieve more complex tasks. Another design goal for PolyBot is that for each module should fit within a cube. Two opposing faces of the cube have connection plates. The module's one DOF allows these two faces to be rotated so they are no longer parallel. They can be rotated up to +/- 90 degrees.

As for computational power, each module has embedded processor with external RAM. Each module is equipped with a hall-effect sensors serving both for communication as well as joint position. The modules communicate with each other over a controller area network bus.

The PolyBot systems have demonstrated that n-modular systems can be very versatile by showing several different forms of statically stable locomotion with a variety of characteristics[24] . In addition, they have demonstrated some manipulations as well. Some of the gaits that have been implemented resemble: earthworm locomotion, turning and sinusoid snake like locomotion, rolling track, three legged caterpillar like

locomotion. The platform is notable for being the first to demonstrate the automatic transition between two different modes of locomotion and for its repeated deployment within unstructured environments.[25]

2.3.2 GZ-I

GZ-I[26] is a modular chain robot with manual configuration. It is composed of several modules which provide only one revolute degree of freedom and three connection faces for attaching other modules. These faces provide docking means with adjacent modules in the assembly via bolts and nuts. Experiments showed the ability of these modules to reconfigure into snake and quadruped robots.

The underlying motivation for this system is driven by the prospects of building complex electromechanical structures from simple building blocks. By homogenizing the structure of individual modules, repairing and maintaining faulty components becomes easier and cost-effective as any part can replace any other part in the assembly. However, as opposed to self-reconfiguration, this robotic system requires human intervention any time a reshaping of the morphology is desired.

2.3.3 ModRED

More recently, in 2010, Nelson et al. introduced *ModRED*[27] (Modular Self-reconfigurable Robot for Exploration and Discovery). Each single module of the ModRED is composed of an end bracket to which another module can connect, two parts, each of them contains actuators and drive-train components, and another end bracket for module docking. The two end brackets are connected to the respective half-bodies by revolute joints, and the central axis running the length of the two half-bodies is the common location of 1 prismatic joint and 1 revolute joint, giving the module 4 DOF overall. The two end brackets can rotate ± 90 degrees, and the rotation along the central axis of the module is unbounded. Although this 4-DOF architecture provides a more versatile

set of motion possibilities than many other chain-type designs, the authors have yet to demonstrate autonomous docking with real robotic hardware.

2.4 Hybrid

There have been some modular robotic systems which combined capabilities from both lattice and chain architectures to take advantage of the both architectures. Such systems were investigated for mobile reconfigurable furniture applications (Roombots[28]), as well as for other highly adaptive mobile systems.

2.4.1 M-TRAN

The *M-TRAN* system (Figure 2.7) developed by Murata et al.[29] at AIST/Tokyo Institute of Technology combines the positive capabilities of chain and lattice based systems to implement a highly maneuverable and reconfigurable system. The hybrid-morphology robot consists of semi-cylindrical modules attached together in either a lattice or chain architecture. Each module of the robot is composed of one passive and one active cube that can continuously pivot about the link that connects them and can form chains for performing tasks. However, during reconfiguration, each of a module's two cubes can occupy a discrete set of positions in space when attempting to align with another module and bond for reconfiguration as in a lattice system. The recent generation of M-TRAN modules utilizes a mechanical latch as a bonding mechanism which is considerably faster, stronger and more reliable than the previous generation's magnetic latch. M-TRAN modules are *self-contained*, meaning that all hardware, electronic boards and batteries are housed inside the semi-cylinders and the links. Data communication and power sharing among modules is achieved through electrodes implemented on the mating surfaces. As for the software part, kinematics and dynamic

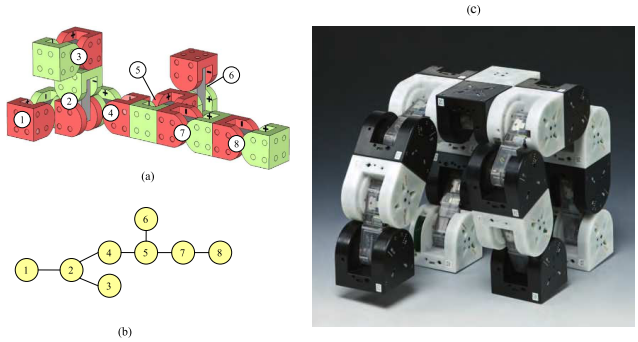

Figure 2.7: M-TRAN Robot

simulator with GUI have been developed to aid the user in planning a reconfiguration or motion sequence of operations before relaying the commands to the real hardware to realize the desired configuration.

2.4.2 Molecube

Molecubes are first introduced by Lipson et al.[30] They are designed and prototyped at the Computational Synthesis Lab to demonstrate a step towards self-reproducing machines. Molecubes are electromechanical cubes which are able to attach and detach from one another using electromagnets on the cube faces (Figure 2.8). The polarity of these electromagnets can be switched electrically between "north", "south", and "off" states so as to attract, repel or act as a neutral element to the neighboring module. Thus each cube has at most $3^6 = 729$ possible states of electromagnetic activation if all six surfaces are equipped with connectors. Switching between states, allows modules to pick up, hold, and drop other modules or groups of blocks, as well as grip and climb over other structures.

The revolute joint that connects the two halves of the body is actuated by an embedded motor that enables one half-cube to swivel around the other, allowing the modules to position themselves in arbitrary, three dimensional way using this swiveling action.

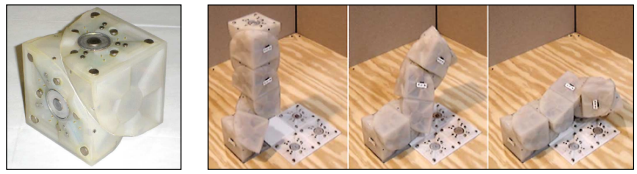

Figure 2.8: Molecube Robot

Each module is a 10-cm cube, split into two halves along an inclined plane perpendicular to the cube's long diagonal. Swiveling two of these blocks relative to each other by 120 degrees causes the entire structure to reconfigure into a Z-shape and then into an L-shape. The cubes are powered through the baseplate and transfer data and power through their faces.

There are a number of ways self-replication may occur in a Molecube space. Looking for possible self-replicating designs Z. Victor et al.[31] explored this space manually and automatically, by evolving self-replication. A number of designs of self-reproducing machines (both structure and control) were found ranging from simplest 4 modules designs to larger and more complex designs. However, for larger machines, it is likely to introduce new power, force, and additional physical constraints that would need to be addressed.

2.4.3 Roombots

Merging technologies from information technology, roomware, and robotics raised *Roombots* as self-reconfiguring modular robots. These robots are constructed by Sproewitz et al.[28] in the hope of developing units which autonomously connect to each other to form different types of adaptive furniture such as stools, chairs, sofas and tables, depending on user requirements (Figure 2.9). The Roombots are also capable of locomotion by using the actuated joints of the modular robots.

Each Roombots module consists of several actuated joints with three degrees of freedom, controllers, and energy supply. Unlike molecubes, Roombots have mechanical connectors for rapid and solid attachment

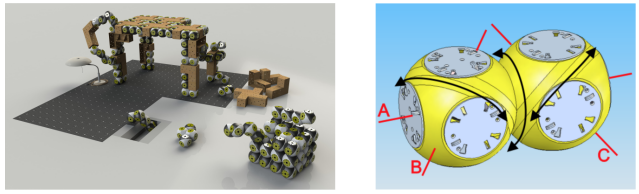

Figure 2.9: Roombot

and detachment between modules to allow the structure to support high loads.

To tackle locomotion control, the authors developed a method for online learning of locomotion by running a gradient-free optimization algorithm in parallel to the central pattern generator (CPG) model. The method produces coordinated patterns of rhythmic activity without any rhythmic inputs from sensory feedback. Besides, it exhibits limit cycle behavior, produces smooth trajectories even when control parameters are abruptly changed, and it is robust against imperfect communication among modules.

2.4.4 SuperBot

An advanced hybrid system is the *SuperBot* of Shen et al.[32] The SuperBot modular robot has been designed and built at the University of Southern California in 2007. The Superbot system improves on the mechanical design of *M-TRAN* (next section) by adding an additional degree of rotational freedom between the two existing rotation axes. To demonstrate the diversity and multifunction of the modules which constitute this robot, both indoors and outdoors experiments have been done include crawling, slithering, walking, moving in sand, climbing ropes between buildings. SuperBot system was designed to be a more robust modular robot, capable of operating in rough environment and real-world situations, perform locomotion, manipulation and self-reconfiguration tasks in the presence of obstacles in an uncontrolled environment. Each SuperBot module consists of three main parts (Figure 2.10): Two end effectors and a rotating central part. This design allows a module to

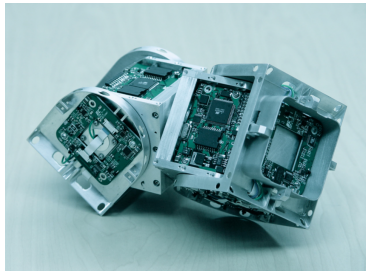

Figure 2.10: SuperBot

have three degrees of freedom in the form of 180 degrees yaw, 180 degrees pitch, and 270 degrees roll. This design gives the SuperBot module the most flexible movements that we know in the literature, and allow a single module to bend and twist into many different shapes and provide the needed flexibility for multimode locomotion.

There are six connectors on each SuperBot module; one on each side of the end effectors. Each of the six connectors of a module can connect to any connectors of another module in all 90 degrees interval orientations. It is through connectors that SuperBot modules are reconfigured into different shapes.

2.5 Mobile

The mobile class of reconfiguration occurs with modules moving in the environment disconnected from other modules. When they attach, they can end up in chains or in a lattice. Examples of mobile reconfiguration devices include multiple wheeled robots that drive around and link together to form trains, modules which float in a liquid or outer space and dock with other modules.

2.5.1 S-Bots

A *SWARM-BOT* entity is composed of many single robots (*S-Bots*) physically interconnected.[33] S-Bots are modular robots which combine collective team-work with reconfigurability accomplished via robotic grippers.

Each module in the S-Bot is autonomous, and is equipped with nine degrees of freedom that operate the mobile tracked-and-wheeled platform, as well as the gripper. In addition to these features, an S-Bot is able to communicate with other S-Bots and physically connect to them in flexible ways. To achieve reconfigurability, modules use their grippers to hold on to one another (Figure 2.11). Partial gripping between two modules allows one to rotate in a horizontal plane with respect to the other, while full gripping restricts this motion, but enables one module to lift the other off the ground in a chain-like formation.

The SWARM-BOT is able to perform exploration, navigation and transport of heavy objects in very rough terrain, where a single S-Bot has major problems to achieve the task. This hardware structure is combined with a distributed adaptive control architecture inspired upon ant colony behaviors.

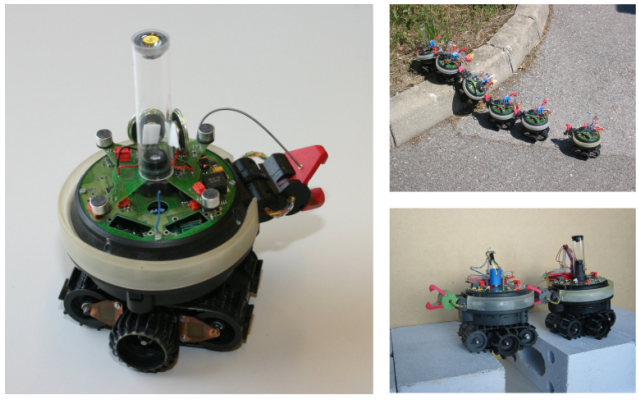

Figure 2.11: Swarm-Bot

The mobility of the system is ensured by a combination of tracks and wheels. The S-Bot can freely move in the environment and easily rotate

on the spot. Each S-Bot is allowed to move in rough terrain, with more complex situations being addressed by SWARM-BOT configurations. This kind of modularity and flexibility to pass large obstacles is very similar to the one developed by self-reconfigurable robots. The main difference consists in the fact that the SWARM-BOT has less 3D capabilities than self-reconfigurable robots, an S-Bot is able to lift only one single S-Bot. This aspect is compensated by exploiting the mobility of each module, which the self-reconfigurable robots lack.

Rigid connections between S-Bots are implemented by a gripper mounted on a horizontal active axis. The gripper can connect to other S-Bots on a T-shaped ring around the main S-Bot body. If not completely closed, the connection leaves some degrees of freedom, which are very important for positioning and physical interaction between robots. If completely closed, the gripper ensures a rigid connection and can be used to lift other S-Bots.

One point to be considered, is that interconnecting robots to build a self-assembling SWARM-BOT differs from interconnecting modules in a self-reconfigurable robot. Self-reconfigurable robots can compute the exact position of each module in order to ensure precise positioning during interconnection.

2.5.2 KiloBots

KiloBot (Figure 2.12) robot is designed and prototyped at Harvard university[34] . This robot is low cost and quick assembly. It has abilities similar to other collective robots. These abilities include: differential drive locomotion, on-board computation power, neighbor-to-neighbor communication, and neighbor-to-neighbor distance sensing. These abilities are achieved at low cost through the use of vibration based locomotion and a simple range sensor. The major issue in designing such a robot is that the robot needs enough functionality to allow it to perform a wide variety of collective behaviors, while at the same time, it must be simple enough to keep the cost low. One important capability of the Kilobot is that it must be able to move in its environment. Two vibration motors

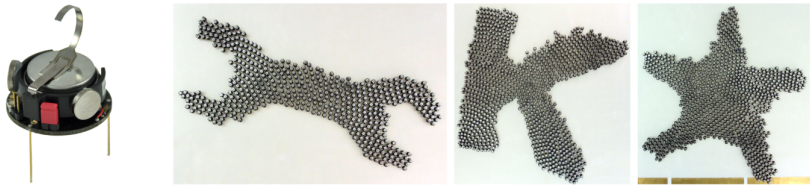

Figure 2.12: kilobot

are embedded in Kilobot for low cost locomotion system. When one of these motors is activated, the centripetal forces generated by the vibrating motor are converted to a forward force on the Kilobot located at the motor's mounting location. The principle of converting the motor vibration to a forward force can be explained using the slip-stick principle. By controlling the magnitude of vibration for the two motors independently in a differential drive manner, the robot can move in a continuous range from clockwise rotation, to straight forward, to counter clockwise rotation.

One major drawback to using this low-cost slip-stick based locomotion is that there is no real form of odometry. This makes moving precisely over long distances or for a long time difficult. Another limitation to this system is that it requires a smooth surface such as a dry erase surface to work and it cannot move over rougher surfaces. While this does limit the environments that Kilobot can operate in, it dramatically reduces its cost, and still allows for the demonstration of many interesting collective behaviors.

2.5.3 Symbiotic robot

To investigate collective intelligence, S. Kernbach et al.[35] developed several independent modules with limited complexity and capabilities (Figure 2.13). These modules are able to connect to each other in different configurations, in order to form a robot with greater capabilities.

From bio-inspired and evolutionary perspectives, novel principles of adaptation and evolution of symbiotic multi-robot organisms has been

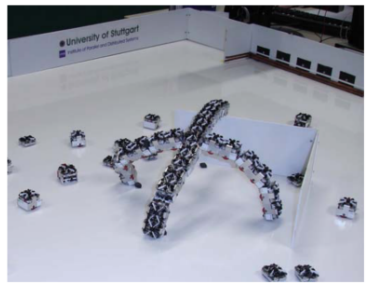

Figure 2.13: Symbiotic robots

developed to allow a large-scale of these units to dock with each other, share energy and computational resources as a single "artificial-life-form". When it is advantageous to do so, these swarm robots would dynamically aggregate into one or many symbiotic organisms and collectively interact with the physical world via a variety of sensors and actuators. The bio-inspired evolutionary paradigms combined with robot embodiment and swarm-emergent phenomena enable the organisms to autonomously manage their own hardware and software organization in order to meet the demands of different working tasks and environments.

2.6 Stochastic

At the micro and nano-scales, biological and physical systems massively rely on parallel stochastic self-assembly, and the system's reorganization is based on passive motion of too small components. This tendency is progressively pronounced as scales decrease.

As scales reduce, deterministic active locomotion becomes extremely difficult due to the limits achieved by the miniaturization of the electromechanical components, whereas stochastic passive motion becomes easier. Over the last years, there have been a considerable effort to build micro-scales modular robotic systems, some of them is described in this section as follows.

2.6.1 M-Block

M-Blocks (Figure 2.14), are magnetically-bonded, angular momentum-actuated modular robot.[36] These cube-shaped robots have no external actuated moving parts, and no active connectors. The modules pivot relative to the edge of each others using inertial force actuation. A flywheel located inside the module, (oriented in the plane of the intended motion), is used to store angular momentum before a braking mechanism is used to decelerate the flywheel and, during a short duration, exert a high torque on the module. If this torque is sufficiently high, the module breaks its magnetic bonds with its neighbors and pivots into a new location.

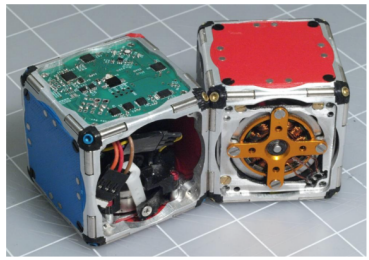

Figure 2.14: M-Block

Thanks to this pivoting locomotion, each individual module can move autonomously in an unstructured environment without any help of other modules. Each module can also move on a 3D lattice of identical modules, achieving a desired trajectory on a planar surface or making convex and concave transitions to other planes. Moreover, the modules can jump over distances up to several body widths wide. This broad range of motions enables the M-Block system to achieve a wide range of shape morphing and locomotion capabilities (Figure 2.15).

In a stochastic system, M-Blocks move in a 2D or 3D environment randomly and form structures by bonding to each other. Modules move in the environment in a passive state. Once a module contacts the substrate or another module, it makes a decision about whether it will bond to the structure or reject a bond. The time that it takes

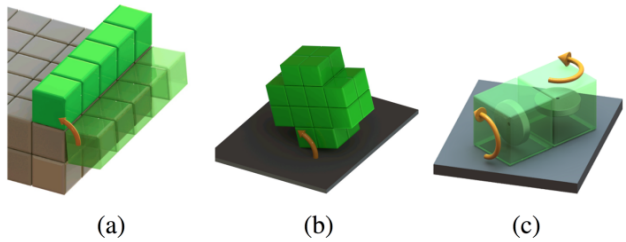

Figure 2.15: Groups of modules can move as rigid assemblies

for the system to reach a desired configuration can be measured only statistically.

2.6.2 Programmable Matter

The fantastic miniaturization of today's electromechanical components is primarily the result of high-volume nanoscale manufacturing. One possible outcome of this technology is the ability to inexpensively produce millimeter-scale units that integrate computing, sensing, actuation, and locomotion mechanisms. A collection of such units can be viewed as a form of *programmable matter*.[37]

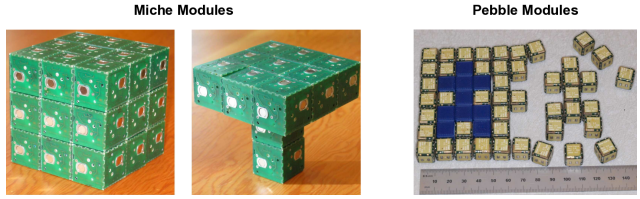

Figure 2.16: Miche and Pebble Robot

A small end of the scale includes the *Miche*[38] and *Pebble*[39] systems (Figure 2.16) developed at the MIT Computer Science and Artificial Intelligence Laboratory. Envisaged as a test bed for future systems of programmable matter, using magnetic connectors, these small, immobile, cube-shaped modules are assembled by hand (they also may self-assemble with the help of an external stochastic force). After the modules in the initial structure use local communication to establish their location,

they cooperatively distribute a user-defined goal configuration using neighbor-to-neighbor communication. Once all modules know whether or not to remain a part of the system, the unnecessary modules disconnect from the system and drop off to create the desired shape. In contrast, a lattice of interconnected smart particles (pebble robots) is used to demonstrate a generic and architecture-independent approach to digital fabrication by distributed shape sensing and duplication.[39]

Other examples from the field of programmable matter include *Catom*[37] and *Fluidic assembly*[40] modules' (Figure 2.17). Catom is a small cylindrical module which uses a set of electromagnets to connect modules on a two-dimensional plane. By coordinating which magnets are on and which are off, modules can reconfigure by revolving around one another. In contrast, Fluidic assembly modules self-assemble with the help of external stochastic force (flow of liquid). Their special design, allows them to channel the flow of liquid through their bodies by adjusting a set of values. The flow of liquid through a structure allows the modules to direct the assembly process.

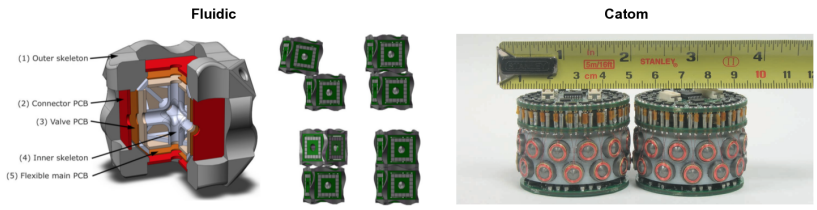

Figure 2.17: Fluidic assembly and Catom robots

Programmable matter systems have shown an interesting idea for shape sensing and duplication, however, there are several larger problems that need to be addressed before they become practical. First, the modules need to be miniaturized so that the resulting objects have acceptable resolution. Second, a distributed duplication in 3D needs to be implemented.

2.7 Deterministic

In deterministic modular robotic systems, modules move or are manipulated directly from one position to another in the lattice or chain architectures. The positions of each module in the system are known at all times and the time that it takes for the system to reach a desired configuration can be accurately measured. A module's reconfiguration mechanism requires a control structure that allows it to coordinate and perform reconfiguration sequences with its neighbors.

2.8 Summary

The field of modular robotic systems has seen a great deal of creativity and innovation from the level of designing physical systems capable of matching shapes to the level of designing algorithms that achieve this capability. The convergence of innovation in hardware design and materials for creating the basic building blocks has resulted in the success of many modular robots which are involved more and more in the study of self organizing systems.

In this chapter, we showcased some self-reconfigurable modular robotic systems that have been developed over the past years. The functionality of these systems varies from simple unactuated structures, to self-contained robots, to robot swarms. These robots either can function in the plane (2D) or in space (3D) and their environments can vary from being on land or submerged in liquid. These systems are classified according to the taxonomy of Yim et al.[3] in four main types: chain, lattice, hybrid and mobile. The different types of platform shown in this chapter are distinguished according to the manner in which they reconfigure themselves, and may be further be classified according to the number of degrees of freedom that the individual modules possess, the type of docking mechanism they utilize and whether reconfiguration is performed in a deterministic or stochastic manner.

Differing in the mechanical design and the basic functionality of their elementary components, each of these robots has specific strengths and weaknesses. For example, at the macro-scales, lattice-based robots perform better on the reconfigurability criterion compared to chain-based robots. The reason is that in the chain-based robots, a chain of modules has to bend and dock with the chain itself. This docking process involves the coordination of several modules and is difficult to control as described in[3] . The big advantage of lattice-type systems is that often only a few modules are involved in the self-reconfiguration process. However, they are still expensive to be constructed in a large number due to the difficulty of reducing the cost of attachment mechanisms. There are thus very few platforms available to buy or which have been released as open hardware projects.

Several bio-inspired platforms have been developed in large number and in micro-scale in order to provide a simple, low-cost systems, which may be used to investigate the interesting properties of self-reconfigurable modular robotics from a simplified level. Some of these systems use external stochastic forces to compensate for the lack of the actuators.

For complexity reason, the design scope considered in this thesis is limited to lattice modular robots able to interact with the environment and further fulfill the general design goals: i) scalable in the number of modules, ii) versatile to solve a large range of different tasks, iii) reliable to tolerate module failures and iv) autonomous to be independent from human guidance.

3

Self-Reconfiguration Algorithms

This chapter highlights the important Self-Reconfiguration Algorithms

Contents

3.1 Self Reconfiguration Problem

The most central characteristic of a modular robot is the ability to change its shape or morphology. Such a change is the subject of the *Self-Reconfiguration Problem* (SRP), which has been emerged as a focus area in the mid-1990s in parallel with the evolution of modular robots hardware design. Self-Reconfiguration is the process of transforming a modular robot from an initial configuration to a desired configuration through a set of primitives and module-level actions while the total number of modules is preserved. Self-Reconfiguration Planning refers to the process of the determination of the sequence of module motions from any given initial configuration to any given final configuration in a reasonable number of moves.[41]

Figure 3.1 shows examples of the reconfiguration process. SRP concerns with two aspects: on one hand, it concerns the algorithmic and optimization aspects, and on the other hand, it deals with kinematic constraints of modules. Hence, SRP is not only a software and hardware challenge, but also a difficult problem that a general solution for it has not been devised yet.[42]

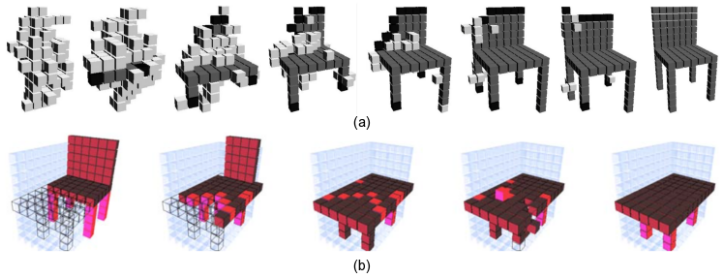

Figure 3.1: Examples of reconfiguration planning: (a) self-reconfiguration from an initial random configuration into a chair;[1] (b) self-reconfiguration in presence of surrounding obstacles (walls).[2]

The SRP is formally defined as the problem of planning a sequence of reconfiguration steps that optimally transforms an initial configuration I into a goal configuration G. It has been proved that SRP is an NP-complete for chain modular robots,[43] therefore, this discipline is being

open to heuristic-based methods. The main challenge in SRP is the exponential growth of possible configurations as the number of modules and degrees of freedom increase. Klarner in 1965 found a typically exponential lower bound of $a_n > \frac{3.6^n}{8}$ for the number of morphologically different possible shapes that can be made by a finite number of cells, in which a_n is the number of arrangements of polygonal-shaped cells to form different configurations for n square-shaped cells.[44] Klarner also found that the size of a_n can be increased further if there is more than one way that two connectors can mate. In his turn, F. Harary et al.[45] found that for hexagonal modules, the number of configurations generated *(N)* is asymptotic to $N = \frac{(2n-1)!}{(n-1)!(n+1)!}(\frac{5}{4})\sqrt{5}$, where n is the number of modules. This, as well as the results of other works suggests a very rapid (non polynomial) growth in the number of different configurations as a function of the number of modules. This fact prompted many researchers to look forward for using metamodules in order to reduce the complexity of the system.

In this section we use the taxonomy of Ahmadzadeh et al.[46] to categorize SRP solution approaches into three classes of Search-based, Control-based, and Bio-inspired approaches.

3.1.1 Search-based approach

Many problems in AI can be solved in theory by intelligently searching through many possible solutions. There have been many search techniques which are widely used in solving constraints satisfaction and optimization problems by representing a problem as a *state space graph*. The graph theory can then be used to analyze the structure and complexity of both the problem and the search procedures employed to solve it.

In the state space model of problem solving, all possible solutions to a problem are represented in the form of a graph $G = (V, E)$ in which each node $v \in V$ represents a solution to the problem and each edge $e(v, w) \in E$ corresponds to the total cost of transition from node v to node w.[47] Search algorithms provide systematic strategies for searching

the graph G until a path that optimizes some quality measures and satisfies particular constraints is found. In the context of SRP, the search space is represented by a *Configuration Graph*, each node of which represents a configuration, and each edge represents an action that transits the robot from a configuration into another. Due to the fact that the branching factor of the configuration graph increases exponentially with the increase of number of modules (section 3.1), the space complexity of search space is intractable. Therefore, it is crucial to take some measures to make the search process tractable, among which, Abstraction methods are used widely and effectively.

3.1.1.1 Abstraction methods

Abstraction techniques are generally used for reducing the size of the search space and speeding up the search process. They must be compatible with search methods, that is, they have to be defined in such a way that provides a concise representation of the state-space for search algorithms. The most prominent abstraction methods are as follows:

Graph Representation : Graph theory has been widely used for representing modular robot topologies, through which the morphology of reconfigurable robots is modeled by a one-dimensional combinatorial topology of edges and vertices, creating a planar graph representation, also known as Connectivity Graph. In such a graph, nodes represent module IDs and edges denote connections between them.[48] One issue with graph representation is that when modules are connected via different connectors, it is very likely to encounter configurations with identical graph topologies but different functionality.

Connector Graph : F. Hou et al.[43] introduced the Connector-graph (C-graph) method to integrate connection ports and orientations in the graph representation of a configuration. Such a method is introduced to resolve the problem of identical graph representations for configurations with different functionalities. In the C-graph

representation, a connection between two modules i and j is represented by the 3-tuple (C_i, OR_i, C_j), in which C_i and C_j refer to the ID of the connectors of modules i and j, respectively, and OR_i is selected from the orientations set denoting the orientation of module j relatively to module i.

Abstract Modules : This general method abstracts modular robots as cubes (or any geometric structure that supports the formation of lattices) with connectors on all faces. These abstract modules are capable to perform general actuation model, presuming that modules can generally move over the surface of a group of modules and convex transition as well.[49] The concept of abstracting modules by cubes had been previously developed by Butler et al.,[50] although it was extended later by Fitch et al.[49] in order to support a class of module hardware instead of one specific modular robot. The use of abstract modules significantly simplifies kinematic constraints of modular robots and decouples the configuration transition complexities from self-reconfiguration algorithms. It allows also the researchers to examine architecture-independent algorithms that can be instantiated for many different systems because an individual abstract module can move in general ways relative to a structure of modules by traveling on the surface of the structure or even through the volume of modular robot's body.

Proteo Modules : This model is introduced by Yim et al.[51] *Proteo* model is defined as a class of three-dimensional metamorphic robotic system capable of approximating arbitrary 3D shapes by utilizing repeated abstract modules, each of which encapsulates both module residence spaces and module motion constraints. This model is intended to be general enough for accommodating a large class of robots with different geometry and motion constraints. The model is applicable to regular-shaped modules that inhabit lattice grid positions and abstracts translational steps of modules into discrete motion steps by the constraint that a module can translate into vacant grid cells only when one of its neighboring modules provides support for its motion.

Meta-Module : *Meta-Modules* are used as a way to reduce motion

constraints and the complexity of shape planning. In this model, groups of modules that act as a unit are attached to one another to perform high-level actions and offer more motion capabilities than single modules.[52,53] As a result, not only is the size of the search space reduced, but also higher manipulability can be achieved through meta-module actions. Due to usual nonholonomic kinematic constraints of modules, meta-modules can be used in composing a group of modules that can act as holonomic units.[54,55] Meta-modules transform the SRP from finding sequences of module-level actions into finding a sequence of meta-module actions.

3.1.1.2 Solution methods

Uninformed Search Methods : *Uninformed Search Methods* like Depth-First, Breadth-First and Lowest-Cost-First search do not take into account the location of the goal. Intuitively, they ignore where they are going until they find a goal and report success. These algorithms are among the most elementary solution methods to SRP. However, the high branching factor of configuration graphs on one hand, and the explicit enumeration of generated and expanded nodes (e.g. the OPEN or CLOSED lists) on the other hand, make the space complexity of uninformed search methods intractable. Therefore, these methods are avoided for solving self-reconfiguration problems in general. In contrast, they can be used for self-reconfiguration planning applications under the following two scenarios: (1) they can be utilized as core search components when the size of configuration graph is manageable, as studied in,[56,57] (2) they can be employed as auxiliary components in finding a path between two positions in a lattice structure, as studied in.[2,15,52,58]

Hierarchical Task Network (HTN) : In AI, *Hierarchical Task Network* is an approach to automated planning in which the dependency among actions can be given in the form of networks. This engineering top-down approach plans for sequences of actions in order to transform the system from an initial state into a goal state.

In other words, it plans a sequence of primitive actions so that a certain goal task satisfying a condition is realized. HTN is based on recursive decomposition of compound tasks into smaller subtasks until reaching primitive tasks performable by planning operators.[59] In modular robotics context, SHOP2 (Simple Hierarchical Ordered Planner 2)[60] as a domain-independent planning system based on HTN has been successfully employed by Bihlmaier et al.[61] for reconfiguration planning of SYMBRION and REPLICATOR mobile modular robots. In this work, a domain description of the SRP is provided to the SHOP2 planning system and a heuristic function that counts the number of docking and undocking actions is used for guiding the SHOP2 searches.

Divide and Conquer : This method simplifies a problem by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved in their own subspaces locally. The solutions to the sub-problems are then combined to give a solution to the original problem. In order to solve SRP, this method plans reconfigurations via intermediate (canonical) configurations. The key challenge here is to characterize intermediate configurations such that not only they should represent most of configurations in the configuration space, but also reconfiguration planning between them should be less complex than planning directly between initial and goal configurations. Casal and Yim[48] adopted this method in solving SRP for a class of closed-chain modular robots in which at first both initial and goal configurations are decomposed into loop and chain substructures using Hierarchical Substructure Decomposition (HSD) rules, and then reconfiguration between intermediate configurations are pre-computed, pre-optimized, and stored in a lookup table. Intermediate configurations in[62] were characterized in such a way that a transition between them requires at most a specific number of reconfiguration actions. Motion planning for many thousands of modules using divide-and-conquer was studied in[63] in which pre-computed reconfiguration plans were reused for recursively reconfiguring groups of modules into meta-modules. Aloupis et al.[64] employed the divide-and-conquer method for reaching goal

configurations through merging canonical forms generated by recursively decomposing lattice surfaces into a hierarchy of square cells.

Dynamic Programming (DP) : *Dynamic Programming* is a major classic optimization method for sequential decision problems that can compute an optimal solution through breaking down a complex problem into a collection of simpler sub-problems in the form of a sequence of decisions over time. Like the divide-and-conquer method, DP solves problems by combining the solutions to subproblems. However, the difference is that the sub-problems in the former are independent while, in the latter, they are not independent, meaning that subproblems themselves share subproblems.[65] A dynamic programming algorithm solves each sub-subproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each sub-subproblem. In the first stage, the algorithm examines all possible ways to solve the problem and then picks the best solution. As a solution to SRP, DP has been used as a backward search policy in[66] where the minimum cost of reconfiguring an open-chain robotic system was found in $O(n^2)$ time and O(n) space complexities. The DP has also been used for implicitly searching the state-action space to learn a reconfiguration policy that maximizes the expected sum of discounted rewards by solving an underlying Markov Decision Process (MDP) through Bellman equations.[67]

Simulated Annealing (SA) : The *Simulated Annealing* method searches the neighborhood of a solution for an iterative improvement of the objective function through hill-climbing, but accepts worse solutions with a probability of $e^{-\triangle E/T}$, in which $\triangle E$ is the change in *energy* (quality) of solutions, and T is the *temperature* parameter controlling the search process.

For employing the SA in SRP, it is necessary to identify neighborhood configurations, and compute either the energy level of each configuration or the difference in system energy level after each reconfiguration. The SA has been employed in SRP in.[68,69] Pamecha et al.[69] computed the energy difference *E(C,G)* between

an arbitrary configuration C and the goal configuration G using the two distance metrics of Overlap and Optimal assignment denoted by δ^c and δ^0 respectively. A configuration C_N is called a neighbor of C if, and only if, $\delta^c = 1$ and $\delta^0 = 1$. During the run, the algorithm randomly chooses the next configuration N from neighboring configurations which satisfy $\triangle E = E(C_N, G)$ - $E(C, G) < 0$. In case that all neighboring configurations lead to higher energy ($\triangle E \geq 0 \ \forall N$) the next configuration will be selected according to the normalized probability of $p_i = \frac{e^{-\triangle Ei/T}}{\sum_{j=1}^{|N|} e^{-\triangle E_j/T}}$, where $|N|$ is the number of neighborhood configurations and T is the current system temperature. Experiments showed that the energy function corresponding to the optimal assignment metric yielded better results than the overlap metric in all cases.

Genetic Algorithm (GA) : The GA method works by initializing a population of solutions in the form of arrays and assigning a fitness value to each individual of the population. Then, at each iteration, a number of individuals are selected according to a selection operator to form *parents*, which are used to reproduce *offsprings* by means of crossover and mutation operators. Finally, a replacement strategy determines which individuals will be passed to the next generation. This process iterates until a stopping criteria is satisfied.

In SRP, the GA-based search have been used for various purposes. For example, it has been: (1) employed particularly for evolving emergent behaviors by modular robots,[70] (2) used for evolving algorithm parameters in complex systems which lack obvious ways of finding their optimal values, (3) used for finding network weights in artificial neural networks,[71] and (4) used for tuning parameters of Central Pattern Generators of gait controllers.[72] Besides, GA can be utilized in evolving bodies that fit on-hand tasks, as Chen[73] employed it to find task-optimal configurations via a task-oriented objective function which was formulated as a combinatorial optimization problem. The procedure was to search among non-isomorphic configurations identified by the help of defining some chromosomes called assembly strings. A hybrid GA–SA

method was introduced in[74] for multi-objective optimization of both generation of configuration and selection of joint variables of the modular robots to achieve a specified orientation and position.

3.1.2 Control-based approach

The Control-based approach tackles the SRP from the Control Theory perspective. It considers the self-reconfiguration as the result of local and distributed acts of controllers.[75] A direct correspondence can be established between the elements of SRP and those of conventional control systems by mapping reconfiguration action sets to the *input*, consecutive configurations to the *output*, and goal configuration to the *reference* elements of a control system. Therefore, these elements should necessarily be characterized in any control-based approach to self-reconfiguration. In the Control-based approach, the role of Abstraction methods is to describe the goal configuration for the controller, while the role of Solution methods is to develop mechanisms for selecting suitable actions to transform a modular robot into its ultimate goal configuration.[46]

3.1.2.1 Abstraction methods

Abstraction methods in the Control-based approach constitute techniques that facilitate presentation of goal configuration either implicitly or explicitly for each module, such that local controllers can easily detect whether or not they have reached the goal configuration. The most straightforward technique is to provide each module with a common-knowledge about geometrical characteristics of the goal configuration. Different techniques have been developed for this purpose:

Coordinates Map : In this technique, the workspace is discretized into a 2D or 3D grid and the coordinates of each module in the goal configuration are explicitly specified and stored. However, this technique, not only consumes huge amount of memory in each

module, but also urges the modules to have the capability of localizing themselves.[76]

Volume Approximation : This method describes the goal configuration as a geometrical expression and requires localization capability of modules. *Volume Approximation* describes the volume of the goal configuration by boxes of same size as studied in,[2,38,55] or by boxes of different sizes called overlapping bricks that can overlap each other to represent complex geometries, as used in.[75,77] The accuracy of approximation is affected by the two parameters of size and number of modules. In fact, the more modules (boxes) with smaller size are available, the finer approximation of the goal geometry can be achieved.[42]

Surface Approximation : The goal configuration can also be geometrically described by its exterior surface, where its boundary faces partition the space into inner and outer sub-spaces.[78] This method is similar to what is done in computer graphics for defining different shapes using small triangular faces. However, a module must be equipped by the capability of detecting whether it is located inside, outside or on the boundary of the approximated volume, which is computation-intensive considering the limitation of onboard processors of modular robots.[42]

3.1.2.2 Solution methods

Random Action : The simplest though least efficient control method is to choose reconfiguration actions at random, where in each step the controller hopes to reach the goal position by chance.[79] The performance of this policy is not predictable and obviously has no guarantee to converge to the goal configuration in a finite number of steps.

Self-evolving Controllers : This method aims to evolve suitable controllers for the tasks at hand either automatically or semi-automatically, while reducing the need for predesigned controllers and a priori information about the environment.[31,74] As evolution

of a controller takes place across several populations, this method is suitable for circumstances in which problem specifications do not change significantly from instance to instance. For example, when defining a goal configuration subjectively, problem specifications do not change much. Thus, self-evolving controllers are mostly adopted to solve problems in which various behaviors (without emphasizing on assuming specific morphologies) must be exhibited by modular robots, such as supporting and balancing functions, in which a modular robot acts as a stanchion to support an unstable object,[71] or locomotion through gait.[80,81]

Gradient-based : A control policy could be as simple as following a gradient which is used to attract wandering modules to an unfilled position. Thereby, the complexity of self-reconfiguration is mostly incorporated into a process which generates the gradients rather than controllers. In the *Gradient-based* method, some modules or cells of a lattice grid become a *source* module and play the role of a *seed*. A seed module acts as a source and sends out a constant integer, representing the concentration of an artificial chemical, to all neighboring cells. *Non-source* modules calculate the concentration of artificial chemicals in their neighborhood and follow them according to the steepest descent scheme.[82]

Gradient-based methods have been employed in growing objects from an initial seed module in.[75,83] In,[75] gradient-based motions are produced for cubic lattice modules by utilizing the attraction between vacant spaces and modules, which reside inside and outside of the goal configuration, respectively. Both modules and holes contribute to generating gradients. Direction and distance information of the movement are implicitly encoded in gradients. Here, the holes play the role of seeds and produce integer concentration values which are propagated to their neighbors. Each neighbor then recursively adds a unit number to the received value and propagates the new value to its own neighbors. In this way, the concentration value is propagated all over the structure. When the gradient propagation front reaches a module, it starts to traverse across the structure by following a decrementing path of concentration values until reaching the source module.

The paths to the unfilled positions always go through or on the surface of the structure. The structure does not contain local minima, because of the staging structure. Therefore, the paths to unfilled positions never contain local minima.

Distributed Planning : In this method, similar to the Gradient-based method, a path is searched from a seed (a target position in the goal configuration) to a *moving* module (a module that can be moved without disconnecting the structure) using a distributed *depth-first* search method. Once a moving module is assigned to a seed, the moving module backtracks the path (i.e. the tree generated by the search algorithm) until reaching the seed.[84,85]

In order to plan motion of modules in a distributed manner, Butler et al.[84] developed the *PacMan* algorithm inspired from a video game with the same name for a system of Crystalline modules, in which *target* modules attract *moving* ones via a motion path drawn for each module by means of special data structure called *pellets*. Their algorithm places pellets in the grid environment such that undesirable situations such as deadlocks, fragmentation, and conflicts are avoided.

3.1.3 Bio-inspired approaches

Developing controllers for self-reconfigurable systems gets more challenges when the systems are supposed to face some undesirable facts such as unpredictable events, sensor noise, and actuator imperfection, under which circumstances classical engineering approaches fail to function efficiency. However, it is interesting to note that biological systems are able to handle such complex situations efficiently in autonomous and decentralized manner. For this reason, biological systems have been a rich source of inspiration for many engineering and mathematical methods, such as bioinformatics and biocomputation.

In the context of modular robotics, bio-inspired approaches take their motivation from the self-organization property of multicellular organisms in such a way that the principles that govern self-organization of natural

systems are translated into algorithms that can exhibit comparable self-organizing behaviors in modular robotic systems.

There are two categories of bio-inspired methods: (1) methods inspired from *morphology development* and *growth process* of multicellular organisms, and (2) methods inspired from the *physiology of organisms* (the means by which organs carry out their chemical or physical functions in an organism). The former is applicable to the process of producing different morphologies in modular robots while the latter is mainly applicable in the process of developing controller for behaviors such as locomotion through gait. In addition, there are other biologically-inspired computing methods such as Genetic Algorithm (GA), Gene Regulatory Network (GRN), Artificial Neural Networks (ANN), and Cellular Automata (CA) which are inspired from natural evolution, brain, and life, respectively.

3.1.3.1 Abstraction methods

Abstraction techniques in Bio-inspired approach assume modules as entities that cooperatively exhibit a particular behavior.

Genome Data Structure : This method was proposed in[86] to store information about controller structure and controller dynamics. A Genome is a collection of genes which can be a simple part of a blueprint, which depicts a part of the final controller. But a gene can also work as a rule, which is used to construct parts of the final controller.

Morphogen Concentration Gradients : It is a technique for abstractly describing target shape information in the methods inspired from the biological morphogenesis process in organisms.[87] Where, a signaling gradient regulates differential gene expression in a concentration-dependent manner, provides a basis for performing many patterning processes.

3.1.3.2 Solution methods

Virtual Embryogenesis (VE) : The process of embryogenetic development and evolutionary adaptation of living organisms (EvoDevo) was the inspiration source in development of the Virtual Embryogenesis (VE) method.[88] The VE method uses artificial evolution to evolve processes which manage assembly of mobile units such that regenerative abilities are exhibited when the robot suffers any damage during the evolution.

Morphogenetic Robotics : Jin and Meng[89] studied the morphogenesis procedure in multicellular organisms in order to tackle the challenges of autonomous adaptation of modular robots morphologies with environmental changes. Morphogenesis is a biological process that causes an organism to develop its shape, during which, gene expression generates various cellular functions. Expression of genes, however, is regulated by their own protein products, as well as by proteins produced by other genes in the same cell or its neighborhood through intracellular and intercellular diffusion. Such interactions form a complex regulation network called *Gene Regularity Network* (GRN). By using GRN, a hierarchical model consisted of a two-layer morphogenetic controller for modular robots has been proposed in,[90,91] where the the first layer is a rule-based pattern generator that generates appropriate patterns for the current environment and assigns tasks, and the second layer is a GRN-based controller that automatically generates reconfiguration plans that converge the current configuration into patterns generated by the first layer.

Protoplasmic Streaming : This method is inspired from slime molds in exhibiting collective behaviors. A cellular slime mold is a primitive organism that fully employs decentralized control for self-organization based on reaction–diffusion of a substance called *Cyclic Adenosine Monophosphate* (AMP). Slime molds are unusual creatures which sometimes live individually as monocellular organisms but at other times form multicellular bodies. As monocellular

organisms, they are amoebae that move individually. When environmental conditions deteriorate (e.g. when food is running out), many amoebae gather to form a slug-like entity, which then moves as a whole.[92]

A decentralized algorithm to control 2D translational DOF modular robot was introduced in[93] through focusing on the primitive organism of slime mold, in which *coupled nonlinear oscillators* were utilized to simulate *Protoplasmic Streaming*. In this method, modules are arranged in a network of passive and real-time tunable springs and covered by an outer skin, and are filled with an incompressible fluid (i.e. protoplasm). The modular robotic system could exhibit supple locomotion similar to adaptive amoeboid locomotion by conforming to the principle of *protoplasmic mass conservation* and by the exploitation of long distance interaction among modules.

Physarum Robotics : *Physarum polycephalum* is a slime mold that inhabits in shady, cool, and moist areas and can exhibit transportation, navigation, and complex behaviors from very simple local interactions. Inspired by such slime molds, Jones et al.[94] studied complex behavior generation from simple components, and developed a particle-based model which mimicked plasmodium (a mass of protoplasm containing many nuclei) of Physarum polycephalum and could spontaneously generate oscillatory patterns. This model can exhibit self-reconfiguration behaviors which are morphologically adaptive, amenable to external influences, and robust to environmental assaults. Through this model modules are able to perform different forms of controllable motions such as linear, rotational, helical, reciprocal, and amoeboid movements.

3.2 Flow methods

Modular robots can mimic the flow of fluids usually through concurrent reconfiguration of modules, similar to the way that spilt water traverses

a terrain toward a sink.[95] Lattice-based modules have the ability to move on the surface of their neighbors along any direction relative to the whole structure, while the connectedness of the whole body is preserved. Therefore, inspired from the way fluids run toward a sink, such modular robots can relocate through simulating this behavior.

In the Flow, modules change their morphology in conformance with the rough environment and the obstacles they meet during locomotion, just like fluids conforming to their stream bed. This way of locomotion is called *water-flow motion* as it simulates the flow of water on the ground, in which modules assume the shape of their underlying terrain while moving forward.[95] While in water-flow motion, Flow is planned at the level of modules, in Cluster-flow it is planned for a block of modules.

The Flow operation is realized through changes in the morphology of the modular robot. Thus it can be considered as a self reconfiguration problem and then solved using the solution methods. However, a delicate point is that consecutive target configurations in Flow are defined subjectively, i.e. they are defined to attain a desired functionality (which is to translate the robot's center of mass along a particular direction) rather than an explicit geometrically-defined shape.[46]

It is important to note that the Flow operation is generally implemented through distributed interactions of modules, thus, it is necessary to adopt synchronization methods to guarantee that the whole cluster of modules remains connected while flowing, and modules are sufficiently coordinated such that overcrowding situations (where several modules intend to enter the same lattice position) are avoided.

3.2.0.3 Control methods

Cellular Automata (CA) : The local nature of transition rules in CA fits well with Flow locomotion, where modules should decide on their next state according to their local state, local configurations of surrounding modules, and limited sensory data about the environment. Xu et al.[96] developed a set of rules for the class of planar lattice modules based on modules local perception about

their immediate neighboring lattice cells. When executed on the Finite-State Machine of modules, the rules can result in flow-like locomotion behavior in presence of obstacles on a 2D grid. Butler et al.[95] proposed a set of transition rules for realizing water-flow motion by MoleCube, M-TRAN, and Crystal modules, in which a modular robot is considered as a particular type of Cellular Automata which runs local rules in each individual cell. Local rule sets are based only on the local configuration around a particular module, and consist of five and eight rules for water-flow in obstacle-free and obstacle-filled environments, respectively. The rules exhibit a behavior similar to a tank tread, by which modules move in turn from the back of a module cluster over its top and eventually place at the front, either on the ground or on another module. Wu et al.[97] developed CA for flow locomotion in presence of obstacles by the M-Cubes lattice modules. In this work, the state of a cell is defined by the ID of modules located in front of its connectors and the next system state is computed by a transition rule generated by a two-layer artificial neural network with seven inputs and one output. Murata et al.[98] employed CA for generating the Flow operation based on an abstract model for a specific meta-module of M-TRAN called *tile* model, in which a planar regular structure is considered as a plane filled with 2 x 2 tiles (cells). Their work considered these cells as objects of control as opposed to conventional CA implementations that consider moving modules as control objects. Obstacle avoidance is realized by setting a vector field in the cell space through a process similar to gradient generation which places sources and sinks next to the cells on edges of the structure. Then, Flow on the contour of the environment is realized by using modules that can detect obstacles in their neighborhood.

Distributed Planning : It is possible to fulfill the Flow operation by parallel execution of locomotion paths planned per each individual module. For example, the *PacMan* algorithm by Butler et al.[84] can be utilized for generating surface-moving systems, and is applicable to unit-compressible systems like the *Crystalline* module, in which paths of modules are planned in parallel using the depth-first search

strategy and path conflicts are resolved in the actuation phase to let modules flow among obstacles or even climb up them. Fitch and Butler[67] employed distributed dynamic programming for planning an individual locomotion path for each module, where modules use local constant-time search and a module-locking scheme in order to ensure physical integrity of the robot, while following their paths toward the goal of locomotion which has been specified by a simple bounding box.

Hierarchical Planning : Flow can be considered as a two-level planning problem: at the higher level gross locomotion of the modular robot's body is planned, and at the lower level detailed local movements of modules, coordination of their actions and conflict resolution issues are addressed. For example, for *cluster-flow* of a class of regular structures and especially M-TRAN modules, Yoshida et al.[99] proposed a centralized two-layered planning method where the upper layer (called global flow planner) plans the overall cluster motion along a desired trajectory, and the lower layer (called motion scheme selector) locally determines low-level module motions by means of a set of *If-Then* rules. As a result, a block of modules from the tail is transferred toward a given heading direction. The lower layer checks whether paths found by the global planner are valid for each module in the block by repeatedly applying rules which include local motion sequences and are defined according to the initial local configuration of the module. Our approach (Ababsa et al.[19]) is another instance of the layered motion planning in which at the higher level a parallel GA search is used for finding the most suitable morphology that a lattice based modular robot must assume during its Flow among obstacles toward a goal position. The genome data structure in the proposed GA contains coordinates of modules in a grid environment, and the fitness function evaluates the Euclidean distance of the robot's center of mass to the destination position. Since each module runs the same copy of the GA algorithm, the population can be divided among modules so that the planning is done in decentralized manner. Once the next morphology of the modular robot is determined, the lower level plans motions of modules using

a PacMan-like algorithm to transform the current morphology to the target morphology identified by the higher level.

Reinforcement Learning : The RL method can also be utilized for exhibiting flow-like motions. Varshavskaya et al.[100] implemented RL on a 2D lattice modular robot with a learning objective of displacing the center of mass towards a particular direction. Varshavskaya[101] also employed *Distributed Reinforcement Learning* on a 2D lattice-based modular system and set the learning objective to move modules in presence of obstacles such that the robot's center of mass is translated toward a given direction while the connectivity of modules is preserved. Consequently, each action that moves the center of mass toward the desired direction gains a reward. The problem was modeled as a *Partially Observable MDP* since each module could identify the presence of obstacles or modules in its eight immediate neighbors. The modules learned proper reconfigurations through centralized and decentralized methods based on direct search of parameterized state-space called *Gradient Ascent in Policy Space*.

3.2.0.4 Synchronization methods

Messaging : Modules can coordinate their motions by passing messages between themselves. For example, messaging can be used to assure that the destination of a module is not occupied by another module, and if so, to ask the blocking module to leave that location. However, such a coordination scheme requires the messages to be exchanged between all modules of a modular robotic system, which not only is bandwidth-intensive, but also needs some kind of centralized controller for coordination. Murata et al.[98] used meta-modules of four modules as an approach for resolving communication and coordination problems as locally as possible. In that method, modules are grouped together and communication paths between two adjacent groups are determined automatically. A module aiming to occupy a particular location within an adjacent

group verifies the vacancy of its destination through messages exchanged at the group level. As a result, by considering meta-modules as groups, flow motion is realized in a distributed manner by means of a hierarchical control structure, in which the lower level controls the motion of meta-modules via coordinating their constituting modules, while the higher-level is responsible for the distributed control of meta-modules.

Locking : Controllers must also care about the arrangement of modules within the robot's structure in order to prevent modules from taking actions that may lead to collisions. The aim of a *locking mechanism* is to synchronize access of modules to the free space in the same way. Critical sections handle simultaneous access to a shared resource in concurrent programming. In modular robotic systems, rather than defining critical sections as blocks of program codes, Lund et al.[102] defined them as regions in the free space, and instead of constraining a critical section to be executed by only one module at a time, they devised a locking mechanism in which the module that wants to use the critical section must lock the module that controls the critical section. The controlling module is a module that has to be accessed exclusively in order to avoid collisions (like a module used as a base for a movement). Fitch et al.[67] utilized the locking mechanism for assuring the connectedness of the modular robot structure during parallel actuation of modules. Connectivity is preserved by locking the modules located on the path searched toward the goal position by the MDP method, and collisions between modules are prevented through locking the immediate vacant destination position of a module in order to stop other modules from occupying it. After that, locked modules are unlocked and allowed to follow their own plans.

3.3 Gait Methods

Although the Flow locomotion is applicable to structures with many modules, they are extremely inefficient in practice due to the substantial time

and energy required for the modules, in order to assist other modules in moving on the faces of each other. Thus, researchers incorporated a more energy-efficient robot locomotion technique into modular robots that is *locomotion through Gait*, in which scheduled rotations of joints and adjustments of joint angles leads to translation of a limbed or limbless body along a desired direction. In Gait motion, since the joints merely rotate at their positions and no attachment/detachment action takes place, the morphology of the modular robot remains fixed during locomotion. Despite the abundance of developed gaits for bipedal, multi-legged, crawler, caterpillar, and snake robots, their direct implementation to modular robots is not straightforward because in most cases, conventional robots are designed to suit a particular locomotion gait while modular robots are normally designed to be general purpose.[46]

In order to realize a particular gait in modular robots, it is necessary to characterize two basic elements of controller and synchronization methods. In fact, a gait is a set of cyclic actions that need a controller to tell each individual module what action must be done at each time-step. On the other hand, synchronization is crucial for creating harmony between movements of modules so that discrete movements of each module lead to a continuous and smooth gait.

3.3.1 Control methods

In general, gait controllers can be devised either manually (by a designer) or automatically (by the controller itself). In the first type, a mapping between states and actions is constructed using lookup tables, periodic functions, and event-driven state machines, etc. as employed in *Control Tables and Phase Automata* methods[44] . However, the difficulties associated with hand-designing of state–action mappings instigated methods that can automatically develop gait controllers, including methods that solely automate the controller development process, such as *Central Pattern Generator* (CPG) and *Neuroevolution*, and methods that automate both the development of body and controller so that optimal gait locomotion is realized, such as *Brain and Body Coevolution*.[9,103]

Control Tables : The most common and obvious approach to control locomotion of a chain-type robot in a specific configuration is a centralized gait control table.[103] This approach assumes that the modules are equipped with a list of predesigned mappings from a set of states (either motion-step or time-step) into a set of actions which tell the module what action to adopt according to its current state. Various gaits for rolling-track, slinky-like, cartwheel, earth-worming, and snake-like locomotion have been implemented by means of mapping from motion-steps to actions, as studied in[44,104,105] . Control Tables is basically an open-loop controller that its adaptation to environmental changes is limited to the flexibility incorporated in the mappings of actions.[42]

Another approach in using Control Tables is to map time-steps of each module to actions through a cyclic function. Time-based control tables are implemented in[106] for sidewinder, rolling, and caterpillar gaits.

Distributed Reinforcement Learning : Christensen et al.[14] developed a *Distributed Reinforcement Learning* strategy for learning simple gait control tables in which the velocity of the whole modular robot is considered as a global shared reward signal to individual learning modules. Each module selects its action from an action set at random based on ε-greedy policy for exploration and exploitation. Although, such a learning strategy is independent of the robot's morphology, it converges slowly to a meaningful behavior. Hence, in order to accelerate the learning process, a heuristic was proposed to bias the exploitation toward the action that has received a reward greater than the maximum expected value of other actions. While this accelerated strategy may converge more quickly, it is prone to remain trapped in local optima conditions longer than the normal ε-greedy policy. These two strategies were experimentally employed in learning gait control tables for realizing typical gaits in ATRON and M-TRAN modules such as snake, walker, and crawler, which each module independently learns what action to do at each time interval.

Central Pattern Generator (CPG) : CPGs are biologically-inspired

neural networks capable to produce coordinated patterns of rhythmic motor actions while being initiated and modulated by simple input signals even in isolation from motor and sensory feedbacks.[107] In[108] a simple CPG is employed in a robot composed of eight modules in which each module is equipped by a sinusoidal CPG equation that controls the rotation angle of each module. Through the interactions between CPGs and by manual tuning of each individual CPG parameters, five different gaits were developed, namely, sinusoidal, turning, rolling, rotating, and lateral shift. In[109] sensory information feedback was incorporated in tuning CPG parameters so that enhanced locomotion movements such as traveling through an obstacle-filled uneven terrain could be achieved. The *Genetic Algorithm* was employed in[72,110,111] for automatic optimization of parameters in a network of interconnected CPG oscillators towards finding a stable walking gait where four state variables belonging to each CPG and the connection weights among CPGs were evolved using GA. Furthermore, CPGs have been successfully implemented in developing adaptive gaits of M-TRAN,[112] YaMoR,[113,114] and Roombots[81] modular robots.

Neuroevolution : The *Neuro-Evolution of Augmented Topologies* (NEAT) is a Genetic Algorithm for evolving *Artificial Neural Networks* (ANNs) by altering both weighting parameters and structures of networks.[115] As an extension to the NEAT, HyperNEAT evolves a particular type of ANN called *Compositional Pattern Producing Network* (CPPN) which in contrast to traditional ANNs can employ a mixture of many activation functions in addition to the widely-used sigmoid function. As a generative encoding description, the HyperNEAT was employed in[116] for generating genotypes that give rise to controllers that work appropriately in different positions of a given organism, and for developing gait controllers for locomotion and obstacle avoidance in a corridor with some bricks randomly placed on the terrain.

Brain and Body Coevolution : Throughout implementing conventional gait control methods in modular robots, a subtle presumption is transferred as well, that is, the body of the robot (the morphology of the modular robot) remains fixed during locomotion. However,

this presumption has roots in conventional robotics where controllers are developed to best fit into the fixed morphologies of robots, which is not the case with modular robots which enjoy a reconfigurable and versatile body. The concept of *Coevolution of Brain and Body* was employed in[117–119] in order to develop more sophisticated gait controllers and to generate optimal locomotion patterns. This method is also used for coevolving both morphology and controllers through bringing flexible robot morphology, controller development, and environment dynamism together. Brain and Body Coevolution results in generation of *Developmental Modular Robots* that change their body and controller automatically in harmony with the situations they encounter. The *Morphogenetic Robotics* method can also be used for locomotion purposes and for modeling neural and morphological development in single and multi-robot systems. Jin and Meng[89] showed how the Brain and Body Coevolution method was used for both guiding the flow of a rectangular block of 16 CrossCube modules through a narrow passage, and forming a vehicle shape with a gait controller which traversed a flat terrain with freely rotating wheels.

Hormone-Based : For all living organisms, the ability to regulate internal *homeostasis* is a crucial feature. This ability to control variables around a set point is found frequently in the physiological networks of single cells and of higher organisms[120] . Shen et al.[121] took inspiration from hormones to design reliable, distributed control algorithms that can deal with dynamic configuration changes. A *Digital Hormone* is, as its biological counterpart, a message that propagates in the cells-network (e.g the organism's body) and triggers different actions from different receivers. In contrast to message broadcasting, a hormone may have a lifetime and can be modified or deleted by cells as it travels through the cells-network.

The *Artificial Homeostatic Hormone System* (AHHS) introduced by Hamann et al.[122] constitutes artificial hormone messages which are diffused in the body of modular robots carrying parameters that control behavior of actuators at each time-step. As a result, movements of modules are synchronized due to diffusion of hormones in the whole body. Moreover, hormone-based synchro-

nization is robust to addition or removal of modules, since (1) the action of each module is basically a reaction to the hormone level it has absorbed, and (2) unlike lookup tables, hormone-based synchronization is independent of modules IDs. Hormones are used in synchronizing controllers either individually, as in,[123] or in conjunction with other techniques, as in[109] , in which a hybrid gait control strategy based on hormone-based messaging and CPGs was proposed. In that work, CPGs are responsible for generating motor primitives while hormones propagate sensory feedback information to CPGs enabling CPG network to achieve complex tasks such as obstacle avoidance and traversing across uneven terrains.

3.4 Self-assembly methods

The Self-assembly operation is a means for fulfilling Shape-formation function, in which modules aggregate spontaneously to a final formation (configuration). More precisely, Self-assembly enables modular robots to transform into desired morphologies and concerns motion planning of several dispersed, initially-detached modules which move freely in the environment and can establish multiple bilateral connections to other modules in such a way that a coherent configuration is built up at the end[124] . Self-assembly can be used in forming final configurations for both self-actuated modular robots, as studied in[86,125–127] , and for modules that lack primary actuation ability, like stochastically-driven modules in liquid environment. Self-repairing can be considered as an extension to Self-assembly, in which a modular robot repels faulty modules autonomously and replaces them with working ones, whether they are constituted in the current configuration or not.

3.4.1 Control methods

The most common self-assembly approach is to grow the final shape from a so-called *seed module*, which is a dedicated module that initiates a Self-assembly operation and guides the growth process by attracting other modules. The final configuration is then achieved as a result of interactions not only between the seed module and other modules, but also between semi-assembled structures (intermediate products) and the modules in the environment. Thus, it is crucial to devise methods that control the interactions between modules (including communication, motion, and connection) according to the state of modules and towards the final configuration so that the likelihood of reaching the desired assemblies is maximized.

Simulated Chemical Kinetics : An analogy between chemical kinetics and dynamics of self-assembling systems was drawn by Hosokawa et al.[128] , which maintains that disassembly processes can be accelerated or decelerated by controlling the rate of assembly reactions through manipulating probabilities that stochastically control the bonding policies of modules. Inspired from this analogy, Miyashita et al.[129] studied the self-assembly behavior of *Tribolon* modules and represented the composition of an intermediate product (called cluster) by a state variable X_i, in which i denotes the number of modules in the cluster. The state transition of the system is then expressed in the form of an easy interpretable chemical reaction (e.g $X_1 + X_2 \rightarrow X_3$) regarding the constraint that no more than two units can aggregate into a cluster at the same time. In[127] reaction rates were used for exhibiting at what speed assemblies were forming or decaying. Also, global performance of the system was tuned and optimized by tuning the probabilities associated with each reaction. Klavins et al.[130] utilized programmed self-assembly for maximizing the assembly yields through tuning the rates of experimentally-determined self-assembly reaction pathways.

Finite State Machines (FSM) : In this method each module is equipped with an internal logic that determines its docking be-

havior according to the sequence of states. Tolley and Lipson[131] suggested an open-loop FSM for self-assembly with the goal of minimizing the required feedback and consisting of four fundamental assembly operations corresponding to the four main states of *Attract*, *Align*, *Latch*, and *Release* in the system. Wei et al.[57] designed a FSM controller for self-assembly of *Sambot* modules and categorized modules based on their role into three types of: (1) *SEED* to denote modules that initiate the Self-assembly operation, (2) *Docking Sambot* (DSA) to mention modules that participate in the growth of the assembly, and, (3) *Connected Sambot* (CSA) to refer to modules that have connected to the assembly. The proposed FSM was devised for DSA modules and included *Wandering*, *Navigation*, *Docking*, and *Locking* states. Experiments show that the devised FSM is scalable and capable of self-assembling into snake, quadruped, H-shape, etc. without any modification in the controller.

Cellular Automata : The CA method is also applicable to the Self-assembly operation and can produce distributed controllers that can generate desired structures using local rules. Kotay and Rus[6] studied the development of self-assembly controllers using CA and devised a set of ten transition rules for creating a cubic assembly with n modules in each dimension. Stoy[1] proposed a seed-based self-assembly system in which gradients in the system were generated by seeds in order to produce growth in the system. Once gradient paths are generated, the automatically generated cellular automata were utilized to guide the growth process. Simulations showed that time to complete a configuration scaled almost linearly with the number of modules.

Gene Regularity Networks (GRN) : Bongard[132] employed GRNs for automatic evolving assemblies of hypothetical cylindrical agents by employing transcription factors that affect the expression of genes along the genome. In this work, 23 pre-defined phenotypic transformations such as increasing the length, dividing a unit into two, and deleting or modifying the properties of the agent's neurons or synapses were initiated. Kernbach et al.[124] utilized the GRN method and its algorithmic inspirations to propose a

self-assembly scheme which can produce functional assemblies from heterogeneous modules. In this work, regularity networks were employed to generate environment-dependable topologies (Φ_S) with a process analogous to gene expression process. Then, the expressed topologies were optimized subject to the number of on-hand modules in the assembly, availability of docking ports, assembling dynamics (e.g. the number of collisions). Scalability tests showed that this approach can scale well to systems with 5 to 30 modules.

3.5 Summary

In the previous chapter we have seen that self-reconfigurable robots are useful because of their robustness, versatility, scale extensibility, and adaptability. In order to realize this vision it has to be supported by the hardware as well as the underlying control algorithms.

In this chapter the discussion is focused on the complexity of the self reconfiguration problem, and on which classes of control systems hold the promise to realize the vision of self-reconfigurable robots. It is observed that Self-reconfiguration of lattice-based modular robots is most-addressed subject in the development of modular robotic systems algorithms. Both biological-inspired and artificial intelligence-based techniques are widely used for exploiting the potential relationship between the robot and its environment to produce control policies that would cause the robot to reshape and exhibit new behaviors. A major challenge in self reconfiguration problem is to produce simultaneous motions of modules while it is guaranteed that there is no chance for deadlock conditions, collision between modules, and risk of fragmentation in the modular robot.

Due to the high complexity of the self reconfiguration problem, *Search-based methods* are not competent in finding optimal solutions, but only suboptimal ones. Indeed, challenges of self reconfiguration problem are beyond finding an optimal sequence of attachment and detachment

actions between modules. Therefore, a planner must accurately produce module-level actions so that undesirable conditions that are likely to emerge in modular robotic systems are avoided (e.g *Fragmentation* in which modular robot's body gets separated into several parts). In addition, the *Overcrowding* situation mostly occurs in local and distributed search methods, as several modules traverse concurrently on the surface and may block each other[51] . Also, falling into local minima is a very common problem when local search methods are employed. This issue can be addressed by adding *randomness* or *turbulence* to the search methods.

In contrast, *Control-based methods* are mostly applicable to goal configurations that are defined subjectively based on their functionalities because guaranteeing convergence to a particular geometry is relatively more difficult than guaranteeing convergence to a behavior. Control-based methods generally decide upon next actions based on local communications between modules. A main issue in generating controllers is their convergence to the goal configuration as they are usually developed based on local interactions and communications between modules.

Bio-inspired methods of self-reconfiguration originate from biological self-organizing systems and aim to realize self-reconfiguration through *Emergence*, which means to exhibit complex behaviors through simulating simple local interactions between individuals in the absence of a central commander. Unlike the conventional top-down approach in which a high-level problem is decomposed into simpler subproblems solvable by known algorithms, an emergent behavior is realized through a bottom-up approach. A challenge that must be addressed in developing Bio-inspired methods is to control the Emergence phenomenon such that the desired behavior is achieved. Although, emergent approaches are desirable for the control of self-reconfigurable systems, they all have different advantages and disadvantages, and none of them can be said to have solved the problem by providing a method which is at the same time robust, systematic, efficient, and provably convergent.

4

Artificial Life and Morphogenetic Engineering

This chapter introduces the concept of Artificial Life and Morphogenetic Engineering in the context of modular robotic systems

Contents

4.1 Introduction

The term Artificial Life (also known as *ALife*) was originally coined by Cristopher Langton in the 1980s with the aim to shed light on life as it could be possible in a vast number of settings and ways[133] . The two most important qualities of artificial life are that it focuses on the essential rather than the contingent features of living systems and that it attempts to understand living systems by artificially synthesizing extremely simple forms of them. Three different kinds of synthetic methods are used in artificial life, theses are: *Soft artificial life*, which creates computer simulations or other purely digital constructions that exhibit life-like behavior; *Hard artificial life*, which produces hardware implementations of life-like systems; and *Wet artificial life*, which involves the creation of life-like systems in a laboratory using biochemical materials.

In computer science, One of the first attempts to replicate life-like phenomena was the Conway's *Game of Life*[134] . A simple game in which every cell of a lattice could switch from alive to dead following some basic rules (the roots of this work return back to Von Neumann and his cellular automata). This field has flourished in a rich and interesting manner, covering many disciplines spanning from biology to social sciences. A lot of tools have been used include: cellular automata, neural networks, agent simulation . . . ect, aiming to study different scientific phenomena that share a common trait in being the emergent outcome of dynamical interactions among simple agents at a lower scale.[135]

4.2 Artificial Life Techniques

Three main classes to which refers almost every topic in artificial life. These classes are related to the fundamental characteristics of living systems which are *evolve*, *learn* and *reproduce*. Each of these classes contains several systems which can be devised as following:

4.2.1 Reproductive Systems

One of the fundamental functions that provides living systems with mechanisms for allowing them to survive under various environmental conditions is their ability to multiply by self-reproduction. This mechanism is necessary for living systems because it enables them to create offspring and continue their population.

Self-Reproductive Systems are systems that have the ability to produce copies of themselves. Biological organisms are of course the most familiar examples of such systems. However, around 1950 mathematicians and computer scientists began studying artificial self replicating systems in order to gain a deeper understanding of complex systems and the fundamental information processing principles involved in self replication. In Computer Science, the quest for *artificial self-reproduction* dates back to the end of the 1940's and started with the work of John Von Neumann on self-reproducing artificial machines. Since that, the logic necessary for self-reproduction has been formally investigated in many work-related issues, which can be classified in the following three categories:

4.2.1.1 Reaction-Diffusion

Reaction-Diffusion systems are systems involving constituents locally transformed into each other by chemical reactions and transported in space by diffusion in order to push the system to the state of chemical equilibrium. It has long been realized that the approach to equilibrium can be both in the form of a simple exponential decay, or more involved transient behaviors associated with non-trivial space dependencies.

The most familiar quantitative description of reaction-diffusion systems is based on the evolution of the macroscopic variables (e.g (u_1, u_2) in equation 4.1) and the dynamics at the molecular level, which provides the values of a set of phenomenological parameters. The concentration of the macroscopic variables $u_1(r)$ and $u_2(r)$ are governed by the following master equations:[136]

$$\begin{cases} \frac{\partial u_1}{\partial t} = r_1(u_1, u_2) + D_1 \bigtriangledown^2 u_1 \\ \frac{\partial u_2}{\partial t} = r_2(u_1, u_2) + D_2 \bigtriangledown^2 u_2 \end{cases} \tag{4.1}$$

where the following assumptions are made:

1. The reaction terms, r_1 and r_2, are assumed to be a function of present and local concentration u_i only $(i = 1, 2)$, but not explicitly on space and time.

2. The diffusion coefficient Di do not explicitly depend on space and time.

Reaction diffusion systems arise, quite naturally, in chemistry and chemical engineering but also serve as a reference for the study of a wide range of phenomena encountered beyond the area of chemical science such as environmental and life sciences.

4.2.1.2 Cellular-Automata

For investigating the logic of the fundamental properties of living systems, especially self-reproduction and the evolution of complex adaptive structures, the mathematician Von Neumann designed the first artificial life model when he created his famous self-reproducing, computation-universal *Cellular Automata*. A cellular automaton is a regular spatial lattice of cells, each of which can be in one of a finite number of states, and each cell's state is updated on every time step according to a deterministic rule based on the values of the neighboring cells and the value of the cell being updated.

A cellular automata is a model of a system of cell objects with the following characteristics:

1. The cells live on a grid.

2. Each cell has a state. The number of state possibilities is typically finite.

3. Each cell has a neighborhood. This can be defined in any number of ways, but it is typically a list of adjacent cells.

This simple model is used to describe a universal constructing machine, which can read assembly instructions of any given machine, and construct that machine accordingly. The copying mechanism is similar to the replication of living cells whereby the DNA-instructions are first copied by cells preceding cell division.

4.2.1.3 L-Systems

L-systems are parallel rewriting systems which were originally introduced as a mathematical tool for modeling the development of simple filamentous organisms.[137] The central concept of L-systems is that of rewriting. Therefore, the development happens in parallel everywhere in the organism. This means, from the point of view of rewriting, that everything has to be rewritten at each step of the rewriting process.

An L-system consists of an alphabet of symbols, an initial sequence of symbols used to begin a construction, a set of production rules that expand individual symbols into strings, and a set of rules that match terminal symbols to drawing functions in order to translate generated strings into geometric structures. To construct an L-system, the production rules are iteratively applied, starting from the initial string. In each iteration all rules are applied in parallel.

In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or productions.

4.2.2 Evolutionary Computation Systems

Evolutionary Computation Systems can technically be considered as global optimization methods with a stochastic optimization character. They are characterized by the maintenance of candidate solutions (a population of search points), rather than just iterating over one point in the search space. Alternatively, evolution is often seen as a process

of adaptation. From this perspective, the fitness is not seen as an objective function to be optimized, but as an expression of environmental requirements.

Over the past years, there have been many different variants of *Evolutionary Algorithms (EA)* which have been proposed to solve complex problems. However, the common underlying idea behind all these algorithms is the same: given a population of individuals, the environmental pressure causes natural selection (survival of the fitness) and this causes a rise in the fitness of the population. In an EA a number of artificial creatures search over the space of the problem. They compete continually with each other to discover optimal areas of the search space. It is hoped that over time the most successful of these creatures will evolve to discover the optimal solution.

There have been three main independent implementation instances of EAs, each of them provides a framework for effectively sampling large search spaces. These implementation instances are discussed in turn as follows:

Genetic Algorithms (GA) : The effective beginnings of *Genetic Algorithms* can be traced back to the work done in the late 1960s at the university of Michigan under the direction of John Holland, following-up to past research undertaken by several biologists, all of whom have used computers for simulations of biological systems.[138]

A genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy. It is devised to model adaptation process, mainly operates on binary strings and uses genetic operators which are *Sellection, Crossover,* and *Mutation*.

Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a fitness function that allows each candidate to be qualitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random.[139]

Evolution Strategies (ES) : Evolution Strategies were developed in the 1960s at the technical university of Berlin by Rechenberg and Schwefel[140] . From the beginning, ES have been developed as a method to solve parameter optimization problem. The search space is the continuous domain, $\mathbb{R}^n$, and solutions in search space are *n-dimensional* vectors.

In Evolution Strategies, the selection method and the population are described by two variables μ and λ. The variable μ gives the number of parents (corresponding to the population size) while λ descibes the number of offspring produced in each generation. The selection scheme is deterministic and the notation $(\mu + \lambda)$-ES describes an algorithm where parents and offspring together compete to reach the next generation with a quantity of μ best individuals.

Genetic Programming (GP) : Another interesting approach was developed relatively recently by Koza[141] . In that work, Koza developed a new methodology named *Genetic Programming*, which provides a way to run a search in the space of possible computer programs in the hope to find the best one (the most fit).

Genetic Programming is one of a number of evolutionary algorithms that solves problems without being explicitly programmed[141] . It achieves this goal of automatic programming by genetically breeding a population of computer programs using biologically inspired operations (stochastic *selection, crossover* and *mutation*). On each generation, the fitness of each individual in the population is evaluated. Once the termination criterion for the run is satisfied, the best single individual obtained during the run is considered as the result of the run. Such a process, stochastically transforms a population of random computer programs into new, more optimized generation of programs.

Unlike genetic algorithm's individuals which are typically encoded as linear bit-strings, GP usually uses tree-based expressions as nonlinear structures for hierarchically structuring and manipulating individuals. The shape, and the contents of these individuals can dynamically change during the process.

4.2.3 Learning Systems

The term *learning system* is very broad, and often misleading. In the context of Artificial Intelligence (AI), a learning system is considered to be any system which uses information obtained during one interaction with its environment to improve its performance during future interactions. The idea behind learning is that percepts should be used not only for acting, but also for improving the system's ability to act in the future.

In AI landscape, learning in machines is the subject of *Machine Learning*. It refers to a system capable of acquiring and integrating the knowledge automatically. The capability of the systems to learn from experience, training, analytical observation, and other means, results in a system that can continuously self-improve and thereby exhibit efficiency and effectiveness. The field of Machine Learning seeks to answer the question: How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes. To this end, many researchers think it is the best way to make progress towards human-level AI.[142] The field of machine learning has traditionally been divided into the following three sub-fields:

Supervised Learning : *Supervised learning* is the machine learning task of inferring a function from a set of labeled examples called *training dataset*. This function can be used to assign a value (make predictions) for all unseen observations. In literature, this is the most common scenario associated with classification, regression, and ranking problems.

In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value. Given a set of training examples of the form $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, a supervised learning algorithm analyzes the training data and produces an inferred function $g : X \to Y$ where X is the input space and Y is the output space. The function g is called a *classifier* if the output is discrete, or it is called *regression function* if the output is continuous. The inferred function should predict the

correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a reasonable way.[142]

Unsupervised Learning : Unlike the supervised learning, the dataset does not include output vector or a known outcome. Only the input vector is available during the unsupervised learning process. This class of algorithms seems much harder; the goal here is to have the computer learn how to do something that we don't tell it how to do.

There are actually two approaches to *unsupervised learning*. The first approach is to teach the learner not by giving explicit categorizations, but by using some sort of reward system to indicate success. This approach nicely generalizes to the real world, where learners might be rewarded for doing certain actions and punished for doing others. A second type of unsupervised learning is called *clustering*. In this type of learning, the goal is not to maximize a utility function, but simply to find similarities in the training data.[142]

Reinforcement Learning (RL) : *Reinforcement learning* is learning how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In Supervised learning, the learner is learning from examples provided by an external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems the machine interacts with its environment by producing actions $\{a_1, a_2, \ldots, a_n\}$. These actions affect the state of the environment, which in turn results in the machine receiving some scalar rewards $\{r_1, r_2, \ldots, r_n\}$. The goal of the machine is to learn to act in a way that maximizes the future rewards it receives over its lifetime. In such a case it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the learner has to act.[143]

4.3 Emergent Properties in Modular Robotic Systems

Emergent properties represent one of the most significant challenges for the engineering of complex systems. They can be thought of as unexpected behaviors that arise through interactions among smaller parts that alone do not exhibit such properties.[144]

In some contexts, emergent properties can be beneficial. They can also be harmful if they blow up important safety requirements. There is, however, considerable disagreement about the nature of emergent properties. Some include almost any unexpected properties exhibited by a complex system. Others refer to emergent properties when an application exhibits behaviors that cannot be identified through functional decomposition (*the system is more than the sum of its component parts*).

4.3.1 Complex Systems Engineering

Future large and robust engineering tasks will require dramatically larger scales of design. To meet these challenges, we need to incorporate techniques from complex systems science into engineering[145] .

A *complex system* is often defined as a system composed of a large number of interconnected parts that as a whole exhibits one or more properties that are not obvious from the properties of the individual parts.[146]

It is the topic of Complex Systems Research to identify and to understand indirect effects. Problems that are difficult to solve are often hard to understand because the causes and effects are not obviously related. Theories need to be developed to capture the interaction of different temporal or spatial scales, the interplay between the individual history of a system and the universal features, the self-organized coordination of different elements or parts, the emergence of collective phenomena on the basis of local and nonlocal interactions, and so on. Complex

systems research, therefore, encompasses the interaction between general principles and methods on the one hand and the detailed investigation of concrete complex systems on the other.[146]

While complex systems cannot readily be studied with a reductionist paradigm, in principle, it provides a number of sophisticated tools, some of them are *concepts* that help us to think about these systems, some of them are *analytical* for studying these systems in greater depth, and some of them are *computer-based* for describing, modeling or simulating these systems.[147]

4.3.2 The importance of being emergent

Many macro-level phenomena that arise out of complex, adaptive systems are called *emergent* by complexity science researchers[148–151] . Such emergent phenomena are considered to be unpredictable and irreducible. Understanding of this macro-level phenomena can only be derived by studying the consequences of the behavior of functioning micro-level agents.

Emergence appears in a variety of fields and it can be studied by complex adaptive systems. With complex adaptive systems we refer to a group of *locally interacting agents*, who act and react to actions of other agents. The key feature of such systems is that the cooperative interactions of the individual components determine the emergent functionalities, which individually do not exist.

In the context of complex systems, emergent strategies are more suitable since they don't require a design engineering for the system, which significantly reduces the complexity of its implementation. Moreover, Emergent systems could be studied by using computationally efficient models like Cellular Automata, since it is capable of demonstrating rich emergent behavior from a handful of simple rules based on local information only. Emerging techniques are actually driving a revolutionary change. This allows, for the first time, to quantitatively address many long-standing questions about the mechanisms of pattern-formation

and self-organization from Physics, Chemistry and Material Science, to Biology and Medicine. Progress requires seamless collaboration across all disciplines and with it a new breed of scientists that are knowledgeable both in the field of complex systems and their specific field of science.[152]

4.3.3 Emergent Behaviours

In the context of robotic systems, emergence is taken to mean that a robot's behavior has become something not explicitly defined in its controllers, but something that has arisen as a consequence of its interaction with its environment[153,154] (*the system is more than the sum of its parts*). The point here is to focus into the coupling of perception and action in a way that ensures robots respond in a timely manner to moving and working in dynamic, unstructured, and at least partially unknown environments. This approach is more suited to the complex on-line context than the classical hierarchical and decision-based approach.[155,156]

Some points in the design of an autonomous robot system that are deemed favorable in encouraging emergence are:[157]

1. Any emergent behaviour should arise out of a large number of parallel and not too closely coupled processes.
2. The system must be redundant.
3. The system needs to employ principles of self-organization.
4. The system should employ cheap and simple design as much as possible, and exploit its environment.
5. Intelligence should be regarded in the context of sensory-motor coordination.

It is clear that multi-agent systems provide excellent ground for studying emergent behavior, especially with the appearance of easily monitored and highly convenient multi-agent simulations, which constitute a very

good framework for the study of emergence, as they allow for easy highlighting and discovery of phenomena that would signify emergence.[158]

A group of robots can be seen as embodying the *"more than the sum of its parts"* concept when it becomes a robotic team once it shows some degree of specialized aptitude of performing a task cooperatively, in the sense that the group provides better performance than its individual components would, by taking advantage of its distributed sensing and acting capability to carry out complex tasks while also taking into consideration increased fault tolerance thanks to agent redundancy and group cohesion obtained from formation-keeping algorithms and related trajectory calculations and motion planning.[159]

Another emergent phenomenon is collective intelligence, the result of two or more agents engaged in global behaviors meaning that an intelligent multi-robot system arises from a group of mobile robots that cooperate, communicate, and dynamically reconfigure their group during their attempts to solve a complex task.[160]

It is important to note that emergence can have both a positive or a negative effect on system performance and task achievement. For instance in computer networking, open desktop computing grids provide a framework for unrestrictedly joining in. However, openness and heterogeneity present serious challenges to the overall system's stability and efficiency since uncooperative and even furtive participants are free to join.[161]

4.4 Artificial Evolution and Artificial Ontogeny

Taking the morphology into account is a necessity and would be a big step forward if we are interested in developing *morpho-functional* machines. The field of morpho-functional machines is still in its infancy. Exploiting the adaptive potential of changing morphologies has not been systematically investigated. Many related issues are raising up not only in engineering design but also in understanding natural forms of intelligence.[162]

While conventional evolutionary robotics starts from a given morphology, variable morphology has to be taken into account when studying morpho-functional machines. In most approaches, the morphology is parameterized in one way or another.

Lichtensteiger and Eggenberger[162] have experienced a very simple morphology parameterization which consists of the angles at which the facets are positioned. Sims[9,10] used length, width, and depth of the limb and body segments, types of sensors and types of joints to describe artificial creatures. Lipson and Pollack[163] evolve robots consisting of rods which are physically characterized by diameter, length, and material constants, as well as types of joints. The implication of this is that no really complex morphological structures can emerge. Complex structures like muscles or other types of organs cannot emerge. By contrast, Bongard and Pfeifer[162] provide a model of artificial ontogeny which is based on structural units equipped with genetic regulatory networks that have the potential of growing morphologies that are more complex and incorporate more variety.

While artificial ontogeny is able to produce a large variety of morphologies (creatures with flexible deformation), materials and soft deformable surfaces have not yet been included in the systems. In contrast the developed systems are parameterized and sufficiently flexible to allow inclusion of this aspect as well. In the experiments conducted so far, typically no completely rigid morphologies have emerged.[162]

4.5 Morphogenetic Robotics

The past decade has witnessed rapid theoretical and technical advances in evolutionary developmental biology[164] (also known as *evo-devo*) and systems biology in understanding molecular and cellular mechanisms that control the biological *morphogenesis*. These advances have not only helped us in understanding biological processes such as human deceases, but also provided us new powerful tools for designing robust

engineered systems. Indeed, biological morphogenesis has shown a surprising degree of robustness. *Morphogenetic robotics* generally refers to the methodologies that address challenges in robotics inspired by biological morphogenesis. Due to the attractive properties that biological morphogenesis exhibits, much attention has been paid to employing genetic and cellular mechanisms for designing robotic systems, in particular for self-organizing swarm robotic systems and self-reconfigurable modular robots. In addition, many researches have been performed in artificial life and robotics to design the morphology plan and neural controller of robots using an evolutionary developmental approach[89] . For instance, *Developmental robotics* (also known as *epigenetic robotics*) is a rather new emergent area of research in the field of robotics that employs simulated or physical robots to understand natural intelligence on the one hand, and to design better robotic systems using principles of biological development, on the other hand[165–168] . Lungarella and Metta[169] have mentioned that there are two driving forces for developmental robotics:

1. Engineers are seeking novel methodologies for construction of advanced robots which are more autonomous, adaptable and sociable robotic systems. In that sense, studies of cognitive development can be used as a valuable source of inspiration.

2. Robots can be employed as research tools for the investigation of embodied models of development.

Jin and Meng[89] have suggested the term *morphogenetic robotics* to denote research efforts dedicated to the application of morphogenetic mechanisms to robotics, which belongs to developmental robotics. According to their perspective, morphogenetic robotics may include the following three main topics:

4.5.1 Morphogenetic Swarm Robotic Systems

Distributed robotic systems are often considered particularly appropriate for inhospitable and rapidly changing environments[170] . It is important

for teams of distributed robots operating in space, in search and rescue conditions, or even inside the human body to work together in order to overcome the physical limitations of individual robots. For these systems to display efficient physical cooperation, it is a prerequisite that they are able to form larger composite robotic entities with morphologies appropriate to the tasks and environmental conditions.

The basic idea in applying genetic and cellular mechanisms in biological morphogenesis to self-organized control of *swarm robots* is to establish a metaphor linking *Swarm Robotic Systems* to *Multicellular Systems*[89] . It is assumed that the movement dynamics of each robot can be modeled by the regulatory dynamics of a cell[171–173] . Christensen et al.[174] experimented *morphogenesis* with self-assembling robots. The authors have proposed *SWARMORPH*: a system that enables the construction of arbitrary two-dimensional morphologies with self-propelled autonomous robots. Both, low-level and higher-level control logic are developed to allow the Swarm-bot robotic platform to form desired morphologies. In[175,176] , the authors proposed a morphogenetic approach based on the Gene Regulatory Network (GRN) to control swarm robots. This approach has the following advantages. First, the global behavior (the target shape in the context of pattern formation) can be embedded in the robot dynamics in the form of *morphogen gradients*. The GRN model can then generate implicit local interaction rules automatically to generate the global behavior, which can be guaranteed through a rigorous mathematical proof. Second, the morphogenetic approach is robust to perturbations in the system and in the environment. Third, it has also shown that the morphogenetic approach can provide a unified framework for multi-robot shape formation and boundary coverage.[177]

4.5.2 Morphogenetic Self Reconfiguration

Morphogenetic approach is used again in[177,178] for reconfiguring modular robots. Similar to morphogenetic swarm robotic systems, each unit in modular robots can be seen as a cell. There are similarities in

control, communication and physical interactions between cells in multi-cellular organisms and modules in modular robots. For instance, the decentralized control is the main aspect that characterizes both modular robots and multi-cellular organisms. Besides, the global behavior of both modular robots and multi-cellular organisms emerges through local interactions of the units, which include electro-mechanisms in modular robots, and chemical production and diffusion of different proteins trough neighboring cells as well as cellular physical interactions such as adhesion in multicellular organisms. Therefore, it is a natural idea to develop control algorithms for self-reconfigurable modular robots inspired from biological morphogenetic mechanisms.

Meng and Jin[90] experimented the attraction and repellent behaviors of the modules which are regulated by a GRN-based controller. As biological cells, the modules produce different artificial proteins which diffuse into neighboring modules and decay over time to create morphogen gradients. Morphogens are artificial chemical substrates which are used to define the target configuration. The dynamic of this approach is modeled by a finite state machine which includes the following five states, namely, *stable*, *unstable*, *attracting*, *repellent*, and *repelled*. The transition between these states is controlled by a single GRN in two layers. Where one layer defines the desired configuration of the modular robots while the other layer organizes the modules autonomously to achieve the desired configuration. Similar to biological gene regulatory networks, such a hierarchical structure makes it possible to separate the control mechanisms for defining a target configuration from those for realizing it[90,179] . In response to the environment changes, the layer for defining the robot configuration is able to adapt the target configuration, based on which the second layer can reorganize the modules autonomously to realize the target configuration.

4.5.3 Morphogenetic Brain-Body Design

Traditionally, evolutionary robotics is concerned with the design of robot controllers using evolutionary algorithms and morphological computation

was employed for connecting brain, body and environment in robot design. Unfortunately, it turned out that morphological computation has put too much emphasis on the hard design of the morphology of robots in shaping intelligent robotic behaviors and has not paid sufficient attention to the developmental aspects. Indeed, the role of neural and morphological development in designing intelligent robots has largely been neglected in evolutionary robotics[90,180] , although co-evolution of brain and body has long been recognized both in *co-evolutionary robotics*[180] and *artificial life*[181] .

In the research field of artificial life, there has been much attention payed to Brain-body co-evolution since the seminal work of Karl Sims[9] . The most attractive aspect of the work is that a developmental model using a directed graph has been adopted for both neural controller and body plan. No significant progress in the understanding of biological principles have been achieved since Sims' work due to the following facts. First, there are a lack of knowledge about the developmental mechanisms in biology and a lack of physically realistic environment[182] . Second, the influence of artificial development on the systems performance is not well understood. Although it is believed that the developmental mechanism offers the possibility to evolve complex systems, the performance advantage of such developmental systems over non-developmental ones remains unclear. Finally, necessary hardware, which is of particular importance in robotics, such as growing materials, adaptable structures, adaptable sensors and actuators are still lacking.

Genetic Regulatory Network models have been proposed for the development of a nervous system and body plan of primitive creatures for elaborating upon the cellular growth model for structural design[183–186] . The importance of co-evolving the development of morphology and control in robotics is twofold. First, object manipulation with a robot is in itself a challenging task as such systems are usually highly redundant. Existing work focuses on the design of the controller for a given morphology, which becomes inefficient when the shape of the objects changes noticeably. A better approach is to co-design the morphology and control in a developmental manner. Second, co-evolution of the morphology and control in a computational environment gives us means for understanding the *phylogenetic* changes in evolution of creatures

structures. It is also important to note that brain-body co-evolution in computational environments has led to findings regarding the organizational principles of nervous systems and the emergence of bilateral symmetry in neural configuration.[170]

4.6 Conclusion

The field of *morpho-functional machines* is still in its infancy. Exploiting the adaptive potential of changing morphologies has not been systematically investigated. It raises many issues not only in engineering design but also in understanding natural forms of intelligence.

It is important to look at meta-morphing machines not as a problem in isolation, but in the context of a complete machine that has to perform a set of tasks in the real world. If morpho-functional machines are viewed in a traditional way where everything needs to be controlled, the attempt of designing them will not be met with much success due to the high complexity of the system. However, if morphology and materials are appropriately exploited, many problems may turn out to be much simpler.

In this chapter we showed the examples of emergent morphology formation techniques in multi-robot system, which showed that, to build a distributed morphological structure by a number of elements, there are at least two fundamental actions, *repulsion* action and *attraction* action among the elements. One such technique, which holds promise for the creation of large and robust designs, is artificial development. It concerns the inclusion of a mid-step between the representation and a pattern in an automated design task, inspired by biological *embryogenesis*. Through the emulation of this biological phenomenon, desirable properties of biological organisms can be included into the machine-learned designs, resulting in increased evolvability in the underlying search space.

To achieve a realistic control in multi-robot system, it is important to find a way to generate the position or direction information for each

individual robot. Indeed, each robot has neither information planned in advance nor global information about its environment. In this thesis we are strongly interested in paying attention to the generation mechanism and the characteristics of the emergent behaviors of morphological formation by multi-robots. Note that such emergent behaviors of multi-robot system are caused by only primitive interaction among the robots themselves and between robot and its environment. We also have to pay attention to the fact that such emerging behaviors of the multi-robot system are based on the simple mechanical function and very small control algorithm. This may offer us a possibility to utilize such behavior of a multi-robot system to achieve a given task.

Part II

Contributions

5

Decentralized Approach to Evolve the Structure of Metamorphic Robots

In this chapter we present a decentralized method for evolving the structure of compressible modular robot

Contents

5.1 Overview

The objective of this chapter is to present our decentralized method that can evolve the structure of a metamorphic modular robot using the computing capability of each module in the system. The method is experimented within a crystalline based modular robot whose goal is to surround a known target object dropped in the environment. The task requires a set of reconfiguration to the robot shape in order to emerge *water-flow* like behavior to go through a tunnel that separates the target position from the initial position of the robot. Our approach is divided in two layers in which at the higher level a parallel GA search is used for finding the most suitable morphology that a lattice based modular robot must assume during its *Flow* among obstacles toward a goal position. The genome data structure in the proposed GA contains coordinates of modules in a grid environment, and the fitness function evaluates the *Euclidean distance* of the robot's center of mass to the destination position. Since each module runs the same copy of the genetic algorithm, the population can be divided among modules so that the planning is done in a decentralized manner. Once the next morphology of the modular robot determined, the lower level plans motions of modules using a *PacMan-like* algorithm to transform the current morphology to the target morphology identified by the higher level.

5.2 Discovering the Topology of the Robot

The topology of metamorphic robots is defined by both the spatial location of each module and the physical links between modules. Before evolving the morphology of the modular robot, we need to discover the initial robots' topology by distributing the perceived information over all the units to emerge a global vision of the whole structure and initialize the morphology generator that will evolve this structure into a more suitable one.

The structural discovering must use only local communications, therefore the modules have to keep the whole structure in a connected state (no fragmentation must be occurred) during all the time steps of the simulation. We consider here the crystalline units to form our metamorphic robot where the number of units is known in advance.

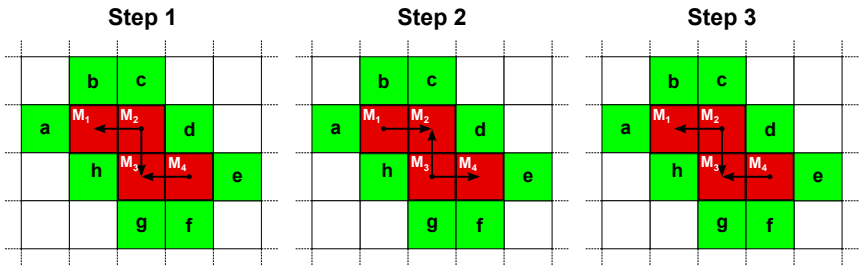

Figure 5.1: Propagation of local perception over modules of the metamorphic robot. The red cells represent 4 connected modules (M_1, M_2, M_3, M_4), while the green cells represent empty cells perceived by the modules and the white cells represent the unperceived cells in the environment

Each module (i) has an adjacency matrix (M_i) with a size of (n^2) where (n) is the number of modules forming the robot. To discover the robot topology, modules use message propagation mechanism based on two primitives of asynchronous communication "Send(M_i, Destination)" and "Receive()" (black arrows in Figure 5.1) to achieve the next constraints:

1. The local vision (perception) of the module (k) is encoded in an adjacency sub-matrix that encodes the next three values : M_k[i,j] = 0 if cell (i,j) is empty, M_k[i,j] = 1 if cell (i,j) is occupied by a module and finally M_k[i,j] = 2 if cell (i,j) is occupied by obstacle.

2. Each module perceives the environment to get local vision about both environment and robot morphology by extending free arms of the module. If the extension is done without physical contact then the first cell in the same direction is empty, otherwise the physical contact identify if there is a module or an obstacle.

3. If sub-matrix M_k that encodes the local vision of the module (k) is updated then the module should send the updated matrix to neighbor modules.

Algorithm 1: Broadcast

```
1  L_r : list of received messages;
2  L_s : list of broadcasted messages;
3  L_n : list of imediate neighbor modules;
4  while true do
5      if L_r ∩ L_s ≠ ∅ then
6          for each u ∈ L_n do
7              Send_u(L_r ∩ L_s);
8          end
9          L_s ← (L_r ∩ L_s);
10         L_r ← ∅;
11     end
12 end
```

Algorithm 2: Receive

```
1  r : new received request;
2  ρ : request being processed;
3  π : local fitness;
4  if (ρ = Null ∧ π > r_π) then
5      Respond("Not OK", r_id);
6      Initiate Domination by means of local state;
7  end
8  if (ρ_π > r_π) then
9      Respond("Not OK", r_id);
10 end
11 if (r_π > π) then
12     Wait.stop(ρ_id);
13     ρ ← r ∧ π ← r_π;
14     Broadcast(ρ);
15     Wait(ρ_id);
16     Respond("OK", r_id);
17 end
18 if (r_id = ρ_id) then
19     Respond("Request Being Processed", r_id)
20 end
```

Algorithm 3: Initiate Domination

```
1 r : domination request;
2 Broadcast(r);
3 Wait(r_id);
```

Algorithms 1,2,3 show the implementation of these constraints to ensure the propagation of local perceptions over the modules. The two algorithms (1 and 2) explain the dynamic of the propagation of messages through the modules. A global vision of the metamorphic robot emerges from this mechanism, and gives each module in the system the ability to have a full description of the robots' topology and its surrounding environment using only a set of simple local communications.

5.3 Evolving The Structure of the Robot

In this work, we have adopted a decentralized approach to evolve the structure of metamorphic robots. The robot consists of a set of autonomous units that have the same properties, so the nature of the system does not imply the existence of a supervisor module: all modules have the same functional level and they can all contribute in the evolution of the whole structure.

A genetic algorithm is used to find the successive time step configurations. Looking to involve the computing capacity of each module in the system, we attempt to parallelize the GA in order to reduce the computation time. In fact, *Parallel Genetic Algorithms* (PGA) have been developed to reduce the large execution times that are associated with simple genetic algorithms for finding near-optimal solutions in large search spaces. They have also been used to solve larger problems and to find better solutions. PGAs have considerable gains in terms of performance and scalability. PGAs can easily be implemented on networks of heterogeneous computers or on parallel mainframes.

Mainly, there are two models of parallel genetic algorithms; *master-slave*

model and *island* model[187] . In our work the results are obtained using our proper framework developed in JAVA. This framework makes a heterogeneous computer network (where each machine runs java virtual machine) looking as a supercomputer with a shared memory and multiple processing units that can run a set of parallel algorithms (the efficacy of our framework is not the subject of this work).

A copy of the same genetic algorithm is implemented in each unit in the system to perform the distribution of the algorithm according to the island model. The initial population is divided into several sub-populations that are processed separately by a set of interconnected computing units (CPUs embedded in each unit).

At the beginning, we create (n) occurrences of the same process that performs a copy of the genetic algorithm and progress in separate machines within the required parameters. Each process evolves its local population independently and from time to time it migrates some genomes (a quantity proportional to the size of the local population) into a selected process. The receiver process adds the new genomes to its population and eliminates the worst genomes (the size of local population must be respected).

This strategy of parallelization makes it easy to perform a genetic algorithm within a large population, and gives result in a reasonable time step. It is observed that a proper size for sub-population must be correctly selected because a subdivision of too small size leads to non-reliable genetic algorithms. Indeed, a population must also contain enough diversified genomes so that the search space can be well explored and the result returned is more interesting.

To reduce the system complexity, we consider a static environment modeled by a lattice of 2D cells, where each cell has the interior architecture shown in Figure 5.2 and may be in one of the following states:

Empty : and it can be filled by a single module as it goes along.

Occupied by a module : and it become empty if the inside module moves into one of the neighbor cells.

Occupied by an obstacle : while considering static environment, this state remains unchanged during all time steps of the simulation.

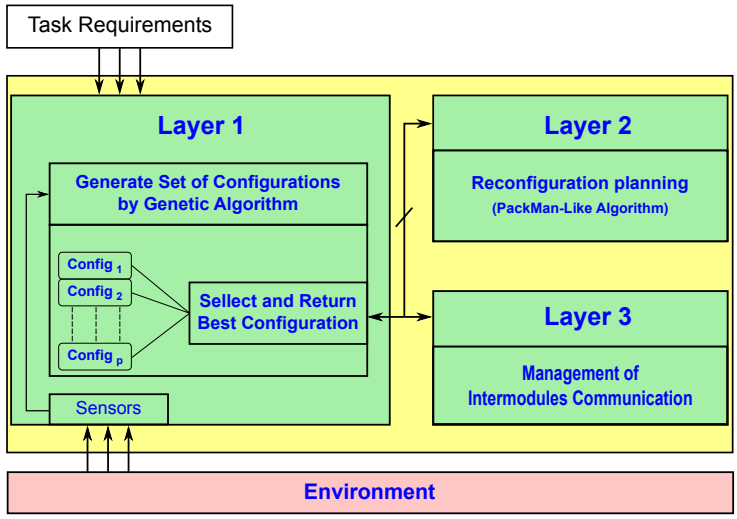

Figure 5.2: The diagram of our evolutionary approach

The proposed approach is shown in Figure 5.2. It consists of the next three stages: evolving modules configuration, domination of new structural information and reconfiguration to the new pattern.

5.3.1 Evolving modules configuration using GA

At this stage, each module uses its computational capacity to run a distributed genetic algorithm to find the next step configurations which are better adapted to the environment where the best one is the configuration that maximizes the fitness function. Here the fitness function is defined as the euclidean distance between the center of mass of the robot and the target object that has to be surrounded.

Initially, each module in the system have the following genetic information: initial population where the genomes encode both the links between

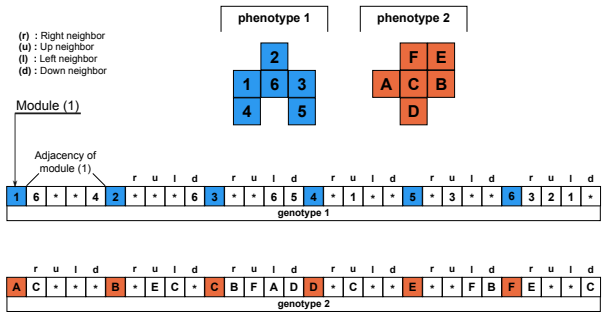

Figure 5.3: Encoding individuals by using character strings

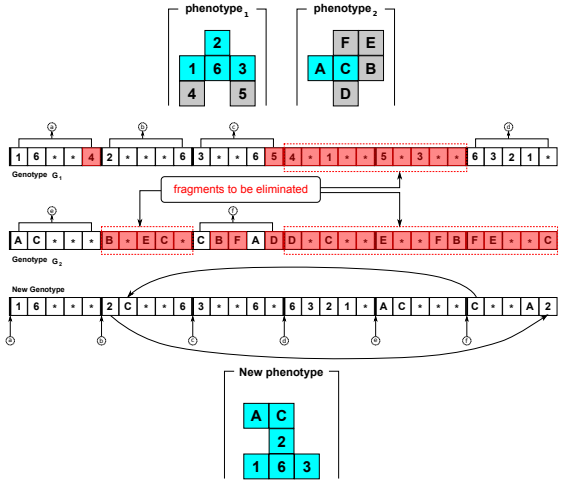

Figure 5.4: Crossover operation, the two candidate configurations $(phenotype_1, phenotype_2)$ are encoded by character string

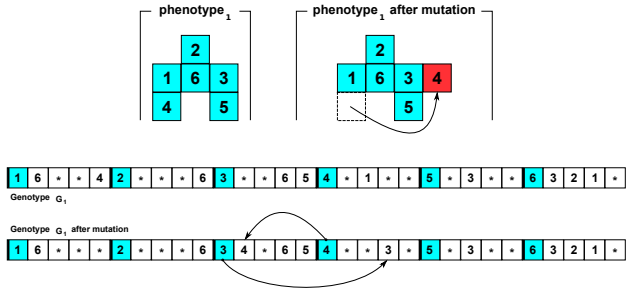

Figure 5.5: Mutation operation, the candidate configuration $(phenotype_1)$ is encoded by character string

modules and the position of each module in the environment. The fitness function is defined to calculate the importance of each genome.

Considering a metamorphic robot of (n) modules, a set of genomes with a size of (n) genes are randomly created and diversified as much as possible. Each genome should also be well-formed (it must encode non fragmented robot structure) where each gene of the genome contains the next two integer fields (see Figure 5.3):

- Identifier field (id): to identify each module in the system.
- Discrete coordination field (i, j): to encode the discrete positions in the lattice based environment.

The links between modules are deduced by using neighbor rules (all the neighboring modules are linked together). The genetic operations (*crossover, mutation*) are applied only for discrete coordination field, while the identifier field remains always unchanged.

Crossover : The crossover operation is used for producing offspring from individuals which are most successful in each competition, thus each successive generation will become more suited to the environment. Commonly, a *crossover site* along the character strings is randomly chosen, then the values of the two strings are exchanged up to this point. However, this operation does not work directly with the genomes shown in Figure 5.3. The next four additional actions are required:

1. From the first parent G_1, eliminate $(n - m)$ modules so that $m < n$ and $\forall M_i \in G_1$ / $Adjacency(M_i) \neq \phi$. (n is the number of modules)
2. From the second parent G_2, eliminate $(n - k)$ modules so that $n = m + k$ and $\forall M_i \in G_2$ / $Adjacency(M_i) \neq \phi$.
3. From G_1 (respectively G_2), select two modules x_1, x_2 that have not a full adjacency list $\{\forall genome_i \ \exists$ module $x_j \in genome_i$ / $\| \ Adjacency(x_j) \ \| < 4\}$ so that the translation of every module of the remaining part in G_2 by the vector $\overrightarrow{p_{x_1} p_{x_2}}$ (p_x: denotes the position of the module (x), Figure 5.4) should respect the next constraint: $\forall \ x_j \in SubGenome_2$

$\nexists\ x_i \in SubGenome_1\ /\ p_{x_i} = Translate_{\overrightarrow{p_{x_1}p_{x_2}}}(p_{x_j})$ Where, $SubGenome_i$ is the remaining part of $Genome_i$.

4. Translate all the modules (update the p vectors) of $SubGenome_2$ by means of the vector $\overrightarrow{p_{x_1}p_{x_2}}$, then update the adjacency list of (x_1, x_2) and integrate them together into new children genome as shown in Figure 5.4.

Mutation : The *mutation* operation is applied to a single genome from which a gene is chosen to be mutated. Except the part that encodes the module position, the remaining parts of the selected gene should not be mutated. This operation proceeds as following:

1. Randomly select a genome A from the population.
2. Randomly select two genes (g_1, g_2) from the genome A so that g_2 has not a full adjacency list and the translation of the module specified by g_1 to fill a random free neighbor of the module specified by g_2 should not produce a fragmented phenotype.
3. Perform the translation and update the neighbor lists of (g_1, g_2) as shown in Figure 5.5.

Selection : In selection, the individuals producing offspring are chosen by means of their reproduction probabilities. Each individual receives a reproduction probability depending on its own fitness value and the fitness values of all other individuals. To select a genome we used a *roulette wheel* selection method. When choosing a parent to produce offspring, the probability ρ for a genome g to be selected is evaluated by means of equation 5.1.

$$\rho(g) = \frac{\nabla(g)}{\sum_{i=1}^{n} \nabla(g_i)} \tag{5.1}$$

5.3.2 The domination of new structural information

After a number of iterations (fixed in advance), each module in the system retrieves the best genome of its local population, this genome encodes

the current best configuration evolved by this module. A message of domination request that contains this genome is created and sent to the neighbor modules that will process it using the fitness function to measure the importance of the encapsulated genome, and in particular they deal with the following five situations:

1. If the extracted genome has a higher fitness than the best one in the local population then, first suspend the genetic algorithm progressing inside the module, next add this genome to the local population and diffuse the same message to neighbor modules that will do the same operation. At this moment each sender module must wait for the acknowledgment answer from the contacted modules in order to answer positively the domination request of the initiator.

2. If the fitness is smaller than the best one in the local population then, create a message of *genome migration*, in which the best genome of the local population is encapsulated. Next, respond negatively to the domination request with this message. This operation is executed without suspending the progress of the genetic algorithm.

3. If the answer to the domination request is negative, then extract the encapsulated genome from the received message (this genome is the best individual of the local population at the moment of the domination request), add the genome to the local population, and send the acknowledgment message to the initiator module (the module that initially requests the domination).

4. If $(n-1)$ acknowledgment messages are gathered within a particular module then it must report the domination of the new configuration using propagation of the dominated message to allow the modules starting the reconfiguration stage.

As a module receives a message of domination (this message contains the geometric description of the conventional best configuration) it cancels the processing of other messages, sends this message to the neighbor modules, and starts the reconfiguration process.

5.3.3 Reconfiguration to the target pattern

The reconfiguration to the new pattern is the last stage in our approach. At this stage, we use a PacMan-Like algorithm[85] since it is a parallel planning algorithm used to reconfigure crystalline robots. In fact, the PacMan algorithm was inspired by the video game of the same name. This algorithm is parallelized and reused by Bulter et al.[85] It uses a specific data structures called "*pellets*" as a way of marking the path that each module should follow to perform its part of reconfiguration. An example of application of the PacMan algorithm is illustrated in Figure 5.6. Once the pellets are distributed, the modules start their asynchronous virtual locomotion where each module switches its identifier with the neighbor module found in the direction along its path of the reconfiguration.

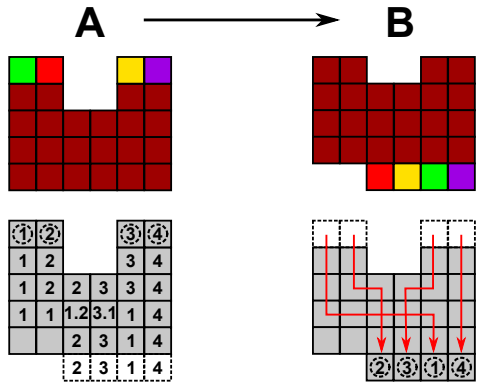

Figure 5.6: An example of using PacMan algorithm to transform the configuration (A) into the configuration (B)

The figure 5.6 shows that the PacMan approach lets several modules to do simultaneous moves, which make it possible for the occurrence of a deadlock or structure fragmentation. To recover this problem, the following constraints must be applied:

- The switching of identifiers between modules must be executed in mutual exclusion.

- At the end of its displacement, each module must diffuse a message that indicates the end of PacMan algorithm to all the modules, to inform them that it successfully reaches its final destination.

5.4 Experimental Results

Initially, the crystalline units are aggregated in an entire connected structure (no fragment must be occurred), where each module runs a genetic algorithm in order to evolve the whole structure. To ensure the autonomy of modules, we used a grid of 4x4 computers to perform the computation of the parallel island genetic algorithm, where each machine acts as the computing unit of a module. A mechanism of inter-modules communication is implemented using the SOCKETs of BERKLEY. The results shown in Figure 5.7 are obtained by using the experimental setup shown in table 5.1.

Table 5.1: experimental setup parameters

Parameters	Values
Selection	Roulette Wheel
Mutation rate	5%
Crossover rate	60%
Sub-population size	60 genomes
Migration rate	5%

To demonstrate our approach, we assumed that the robot should surround an object in the environment. Knowing the object position we define the fitness function as the euclidean distance (D) between the object and the center of the mass of the robot. So the quantity (D) must be minimized as much as possible to ensure the best surrounding of the object.

The environment is simulated as a 2D lattice matrix composed from 23x8 cells as shown in figure 5.7, where 23 cells represent obstacles (black cells) and the metamorphic robot is represented by 4x4 red cells. The

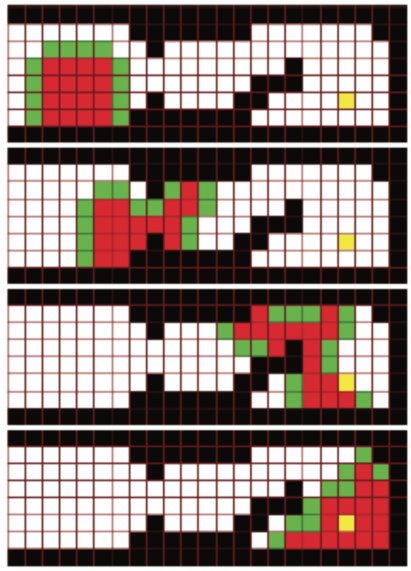

Figure 5.7: The evolution of a metamorphic self-reconfigurable robot during its movement in a tunnel from left to right to surround the yellow square

green calls represent the perceived cells while the yellow cell represents the object to be surrounded. Figure 5.7 shows instances of simulation of the metamorphic robot evolution that moves from left to right while moving through a tight tunnel to surround the object represented by the yellow square.

During this simulation, we can clearly observe in all time steps the changes in the morphology of the robot while performing its evolution. It adapts to the environment to achieve the goal at hand. This evolution is emerged by the cooperation of all the modules that evolve population of feasible configurations and make an agreement (by using domination request) to apply the best one.

We can also observe a kind of *liquid-like* locomotion behavior of the whole structure that emerges from the overlap of successive configurations of the metamorphic robot. This locomotion is expected because the search space of the genetic algorithm integrates and disintegrates dynamically the perceived cells for each configuration which create not only a local vision for each module (the green cells in Figure 5.7 represent the vision

field of the modules) but also a local vision of the whole evolutionary structure(by propagating information) thus the emerging behavior is well-adapted to the environment because the evolved structure is strongly influenced by the information perceived from the environment.

5.5 Conclusion

In this chapter, we presented a decentralized approach that evolves the configuration of metamorphic robot which consists of crystalline modules. This approach mainly uses *PacMan-like* algorithm coupled with a genetic algorithm.

In fact, the propagation of the local information trough modules emerges a full-description of the structural topology of the metamorphic robot in each module. This description is indispensable to perform a distributed genetic algorithm using modules as computational units. Each unit in the system is an autonomous module that has a computational capacity, though limited but it is sufficient to perform genetic operations "*crossover, selection, mutation*" because these operations are relatively simple. This first stage occurs prior to an evolutionary process which is distributed over the modules of the robot, where the perceived information is shared between all modules. The second stage uses a reconfiguration algorithm (*PacMan* algorithm which is devised for compressible units) to reshape the robot structure from the current configuration to the one defined in the first stage.

The evolutionary algorithm used in this work is decentralized, which ensures theoretically the continuity of the evolutionary process even if there are faulty modules. A *liquid-like* movement behavior emerges from the different reconfigurations of the metamorphic robot. This behavior can be studied in a future work to get the most possible hidden potential of evolutionary structures.

6

Splittable Metamorphic Carrier Robots

In this chapter we improve our approach and we experiment it with a more complex task

Contents

6.1 Overview

In this chapter, we show how to evolve the structure of a metamorphic self-reconfiguring modular robot to perform the task at hand by taking advantage of the computational power of the individual modules. The research presented in this chapter is grounded in our previous work (see chapter 5) in which we showed how simple local sensing, local communication and control rules achieve useful emergent behaviors of crystalline robots.

In this chapter, we are interested in evolving the configuration of metamorphic modular robots to transport sliding objects from their current positions to specific target positions. This process can be subdivided in three successive steps: (1) evolve dynamically the structure configuration to find and surround objects, we use morphogen gradients to locate these objects, (2) evolve the current structure configuration to be able to transport the surrounded objects, (3) release the sliding objects whenever the final position is achieved. These steps are coded in a simple finite-state machine (FSM) that denotes all the states which may be taken by every unit in the system, and the possible transitions that can be performed according to the required conditions. A basic hormone system is used as well to control the inputs of the FSM.

6.2 The Modular Robot and its Environment

To reduce the simulation's complexity, we consider a static environment modeled with a lattice (*grid*) of 2D cells, where each of these cells may be in one of the following states: *empty*, *occupied with a single module*, *occupied with an obstacle*. Basically, all the modules have the same size. We use the same software architecture discussed in chapter 5, with a slight change in the sensing system.

Actually, the cell-robot has no idea about the positions of the target objects, however it can sense some particular informations called *morphogens* diffused by these objects. We assume that the environment contains different morphogens. They are gradually spreading through the grid so that the variation on their concentrations between the neighbor cells emerges a guidance system that gives an implicit information about directions that should be followed to reach the source of these morphogens. This change improves the control model to take advantages not to use global information and makes the system more realistic. We use a basic diffusion algorithm to spread morphogens on each cell in the environment with respect to the following rules:

1. Each morphogen has a unique identifier.
2. Each of the target objects is considered as a morphogen source, it produces and diffuses a unique morphogen in the environment.
3. The concentration of $morphogen_i$ is maximal at the position of its source.
4. The concentration of $morphogen_i$ changes across the grid, and constantly decays through the environment.

Indeed, we used morphogen gradient in the hope to improve our previous model in which we used euclidean distance to locate target objects and to drive the system evolution. In such a model, all the units are supposed to know (as a global information) the exact position of all the target objects. Besides, using euclidean distance makes the metamorphic robot unable to overcome some situations, in particular avoiding U-shaped obstacles (obstacles which are parabolic in shape). To resolve this problem, we introduce f_{moving} fitness (equation 6.1) that should be performed by the robot to acquire morphogens as much as possible. Maximizing f_{moving}, the robot will track the morphogens concentrations from low to high level of concentration.

$$f_{moving} = \sum_{i=1}^{m} \sum_{j=1}^{n} Morphogen_i(m_j) \tag{6.1}$$

In this equation, $Morphogen_i(m_j)$ denotes the concentration of $Morphogen_i$ perceived by the module m_j and n denotes the number of

modules, m denotes the number of morphogens sources.

Experimental studies showed us that when we use f_{moving} as a fitness, the GA may have a tendency to converge towards local optima in which it is not defined how to sacrifice short-term fitness to gain longer-term fitness. As a result, the robot gets stuck in an *equipotential* area from which it can not get out anymore. This particular circumstance strongly depends on the shape of f_{moving} landscape that depends itself on the way of spreading morphogens (the decay function).

In order to alleviate this problem, we introduced an activator/inhibitor coefficient δ_i to control the fitness f_{moving} as shown in equation 6.2.

$$F_{Moving} = \sum_{i=1}^{m} \delta_i \sum_{j=1}^{n} Morphogen_i(m_j) \tag{6.2}$$

In this equation δ_i is used to activate or inhibit $Morphogen_i$.

Using F_{Moving}, the robot can perform the task at hand either sequentially by activating a single morphogen at a time, or in parallel by dividing the whole structure into m parts, where $m > 1$ is the number of morphogens sources (target objects). To divide the whole structure, we used the following three rules:

1. Cluster the modules into m classes, using their perceived morphogens as an input data, and assign each of the modules the appropriate class-identifier.

2. Each of the modules keeps the link with the same-class modules and disconnect from the others.

3. For each class C_i, δ_i takes value as shown in equation 6.3, where id_{C_i} is the identifier of C_i.

$$\delta_i = \begin{cases} 1 & if(i = id_{C_i}) \\ 0 & if(i \neq id_{C_i}) \end{cases} \tag{6.3}$$

Once the structure divides, each part behaves as an entirely autonomous modular robot in which only one morphogen is activated and the others

are inhibited. In such a case, no equipotential area appears and each part will track and surround a unique morphogen source.

6.3 Generate Cyclic Locomotions

The GA used in this work is basically designed for evolving the structure of modular robots as discussed in chapter 5. It outputs only the next configuration that improves the fitness at hand. However, it can be used to develop facilities for generating locomotions.

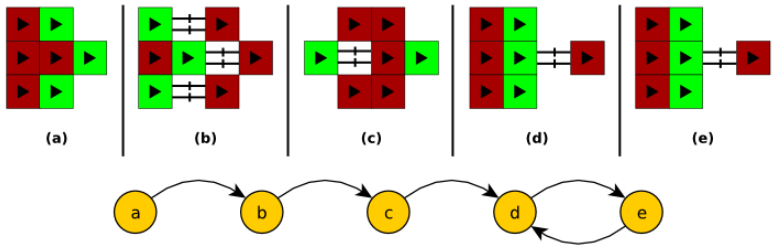

Figure 6.1: Cyclic locomotion

Actually, the modules can generate various motions as a combination of each module micro-movement. In particular, they are able to generate an *earthworm-like* locomotion as a loop of simple cyclic locomotion. Figure 6.1, shows a loop of simple cyclic locomotion that can be generated using the following rules:

1. Define a direction for the movement.
2. Arrange the modules so that each of them can disconnect from its neighbor modules that are perpendicular to the movement direction without leaving any isolated module.
3. The module is able to move one step position(green modules in Figure 6.1) once the following conditions are satisfied:
   - The neighbor cell to the direction of movement is empty.

- The disconnection from neighbor modules that are perpendicular to the direction of its movement should not leave any isolated module.
- All faces of the module are contracted.

The direction of movement is defined by using the variation of morphogen concentration around the space occupied by the modules, while the modules arrangement is defined by using GA since it is a particular configuration.

6.4 Encapsulation into the modules

As mentioned in section 6.1, we are interested in evolving the configuration of metamorphic modular robot for transporting sliding objects from their current positions to specific target positions. This process can be subdivided in three successive steps:

Track and surround target objects : The modules run a GA for evolving the whole structure in order to acquire morphogens as much as possible (using F_{Moving} as a fitness where $\delta_i = 1 \ \forall i = 1..m$). This evolution drives the modular robot towards an equipotential area in which it gets stuck and can not completely converge towards any of the target objects ($\Delta \ F_{Moving} \simeq 0$). Each module in the system can either produce two artificial hormones H^1 and H^2 (equations 6.5) or diffuse an amount of them (H_r^1 , H_r^2) to control their desires to switch between the three steps (see Figure 6.3).

Once the modular robot reaches an equipotential area, each module starts to lose H^1 since $\Delta \ F_{Moving} \simeq 0$. An inter-modular compensation is triggered (equation 6.7) to ensure the homogeneity of H^1 through all the modules. While $\Delta \ F_{Moving} \simeq 0$, H^1 keeps decreasing until it reaches a lower threshold. At this moment, a clustering method is used for determining a division scheme using the informations perceived by the modules as an input data. The whole structure is then divided into several parts where each part

will be attracted by only one target object that matches with its class identifier.

Transport target objects : As a result of step (1), each target object is surrounded by several modules of the same class. Again, F_{Moving} achieves a maximum level and $\Delta_{F_{Moving}}$ converges towards 0, at this moment, every module of the class starts to lose H^2. The modules that are in interaction with the target object lose H^2 faster than the others (coefficient φ in equation 6.5).

Once H^2 gets lower, the modules define the direction of movement using the variation in concentration of the morphogen that denotes the final position, then the GA is used to evolve a structure that can generate a cyclic locomotion to the defined direction.

Each substructure can push ahead the sliding object or pull it from the back while the sliding object moves from lower concentration level to higher morphogen concentration level. Otherwise, the modules in interaction with the sliding object stop the movement and diffuse a hormone H^3 to switch to $Step_2$ and redefine a new direction.

Release the transported objects : Once the sliding object arrives at the final position, it is expelled as is an obstacle or a failed module.

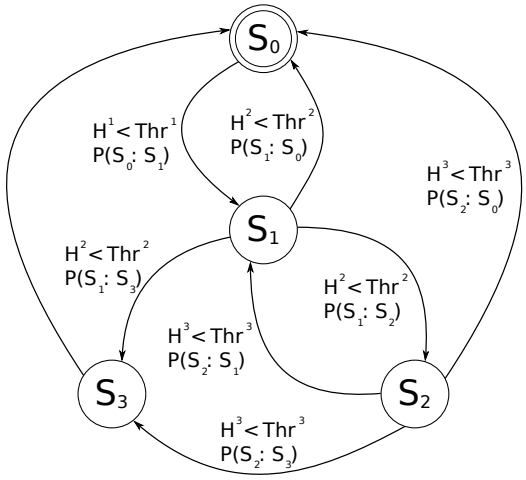

Figure 6.2: FSM modeling the global task

The dynamic of these steps can be modeled by a finite state machine as shown in Figure 6.2, where: S_0: denotes the initial state of the system. States (S_1, S_2, S_3): denote respectively steps 1,2,3. (Thr^1 , Thr^2): denote respectively thresholds of hormones (H^1, H^2). We integrated a highly simplified model of biological hormone system to generate inputs for FSM.

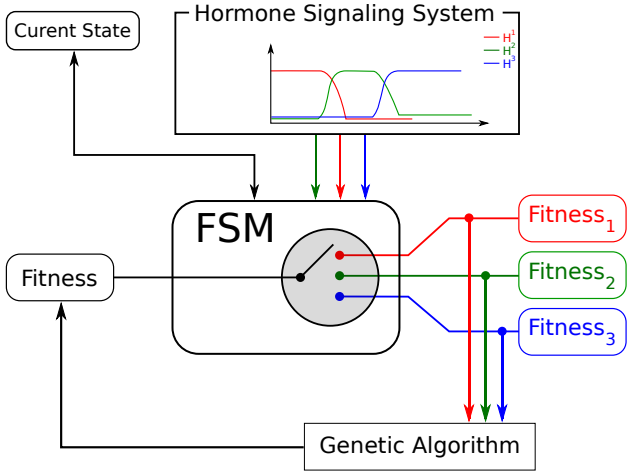

Figure 6.3: The interaction between the FSM, the hormonne system, and the GA engine

The dynamics of the hormones H^1 and H^2 are modeled as follows:

$$H^1 = H_r^1 + \alpha_1 \triangle_{fit} - \beta_1 f(\triangle_{fit}) \tag{6.4}$$

$$H^2 = H_r^2 + \alpha_2 \triangle_{fit} - \varphi\beta_2 f(\triangle_{fit}) \tag{6.5}$$

$$f(x) = \begin{cases} 0 & if x \neq 0 \\ 1 & if x = 0 \end{cases} \tag{6.6}$$

Where, $\Delta_{fit} = |F(t) - F(t-1)|$: denotes the change of the fitness over the time. (α_1, α_2): are two coefficients used to accelerate the production of H^1 and H^2. (H_r^1, H_r^2): represent the received hormones. (β_1, β_2): are

two coefficients used to decelerate the production of H^1 and H^2. φ: is a coefficient used to enhance the deceleration of producing H^2.

The function $D^i_{x,y}(t)$ models the diffusion of the hormone H^i at time t as described in the following equation 6.7, where d_i represents the diffusion coefficient of H^i:

$$D^i_{x,y}(t) = d_i|x - y|H^i(t) \tag{6.7}$$

6.5 Clustering The Modules

Clustering is the task of finding natural groupings among objects in such a way that objects in the same group (called a cluster) share similar values[188] . In our work we use clustering for partitioning modules into several groups so that the whole structure of the modular robot will be able to split up in order to cope with environmental variations or to perform the task at hand in parallel. From this point of view, the perceived information of each module is considered as a data point.

For clustering perceived morphogens, we used the well known algorithm *K-means* which is commonly used to solve clustering problems. This algorithm is based on minimizing the overall sum of the squared errors between each pattern and the corresponding cluster center. This can be written as minimization of the following objective function:

$$E = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - m_i\|^2 \tag{6.8}$$

where x is a data point, and m_i is the mean of the data points.

6.6 Experimental Results

The following experimental parameters are used to set up the simulation:

Max morphogen concentration = 50, hormone accelerators (α_1, α_2) = (0.8,0.8), hormone decelerator (β_1, β_2) = (0.3,0.3), hormone diffusion coefficient d_i=0.5, the cumulation of any hormon H^i can not exceed 10 (otherwise the additional value is ignored except for the non divided structure). This setup has been empirically determined, through a set of tests.

In this experiment, the environment is modeled as a 2D grid composed of 25x10 cells as shown in Figure 6.5, where, the shaded cells represent obstacles, the blue cells represent sliding objects, and the modular robot consists of 16 units which are represented by the red cells. Each of the sliding objects produces a unique morphogen which spreads and decays on the environment. The mission should be performed by the modular robot is to track the sliding objects that are randomly dispersed in the environment and to transport them into a predefined final position (we assume that the final position diffuses a unique morphogen called $Morph_{Final}$).

During the convergence of the genetic algorithm, it is interesting to observe the capacity of the system to evolve the structure towards the best solution. As shown in Figure 6.5, the modular robot starts form its initial position and evolves its structure to acquire morphogens as much as possible. As a result, the modular robot converges more and more towards the most concentrated cells (red-brick cells in Figure 6.5 (a.b.c)), at this moment, the structure is not yet divided, and only H^1 is produced. Once the structure reaches the equipotential area (Figure 6.5 c), $\triangle_{F_{Moving}}$ approaches to 0 and H^1 starts decreasing (blue curve in Figure 6.4). When the H^1 concentration gets down under a threshold of 2.5 (empirically determined), the structure is believed to be steady for a long time and is ready to be divided into several parts.

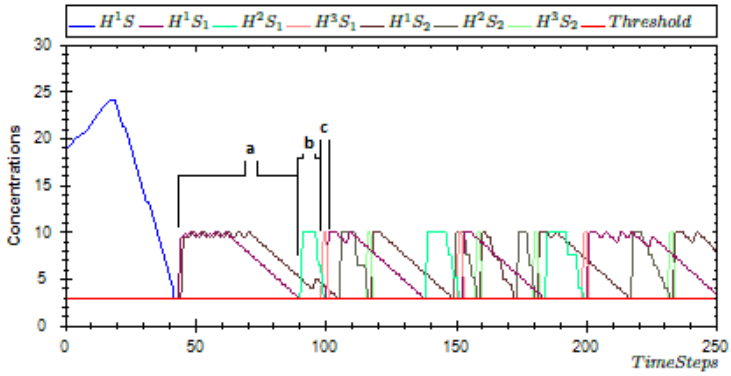

Figure 6.4: Hormones concentrations during the time of simulation, (a,b,c) are the three parts of the global task

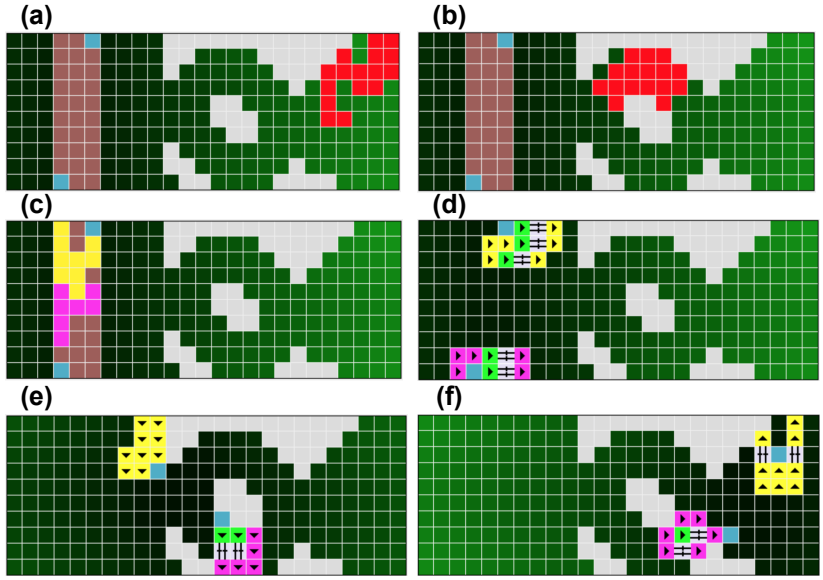

Figure 6.5: The modular robot during the evolution

A k-means algorithm is used to cluster the dataset (informations perceived by the modules) and to assign each module to a class for preparing it to the separation process. Next, each module tests its neighbors and disconnects from modules which have not the same class identifier. As a result, the modules are separated in two classes and the structure is divided in two parts (Figure 6.5 c), where these parts should not be divided anymore and each of them converges only to the object attracted by it, then it surrounds this object and evolves its configuration to be able to perform a cyclic locomotion to the direction by which the concentration of $Morph_{Final}$ is being increased (subscript b in Figure 6.4), otherwise, a hormone H^3 is diffused to evolve an other configuration and move toward a new direction. As a result, the sliding object gets closer to the final position.

The global system dynamic is already modeled by a FSM that is encapsulated in every module to switch between steps (parts of the global task) a,b,c in Figure 6.4 while the hormone system produces H^1, H^2, H^3 to generate inputs to this FSM. Coupling between the hormone system and the FSM is illustrated as following:

- $(State = Start \wedge H^1 < 2.5) \mapsto (State \leftarrow A, H^2 \leftarrow Max)$.
- $(State = A \wedge H^2 < 2.5) \mapsto (State \leftarrow B)$.
- $(State = B \wedge H^3 > 0) \mapsto (State \leftarrow A, H^2 \leftarrow Max)$.

Observing these rules, we notice that the second and the third rules, create such an interesting cycle that can be used to create a generalized process for more complex tasks.

6.7 Conclusion

In this chapter, we presented a decentralized approach that evolves the ability of metamorphic robots according to the task being performed. This approach is based on our previous work (chapter 5) in which we used a GA coupled with a PacMan-like algorithm for evolving the structure of a modular robot. In this study the genetic algorithm is used to

generate the next better configuration of the modular robot, while the PacMan-like algorithm is used to drive the self-reconfiguration process. Switching between generating better configuration and reforming to the new configuration emerges an adaptive locomotion for the modular robot. A more complex task is considered in this work, and a new improvement is presented.

The first improvement we can talk about is the ability of our approach to perform the task at hand in parallel by dividing the modular robot into several parts where each part will focus on one single routine. We note also that the modular robot is driven into the target objects without being stuck by obstacles that are parabolic in shape. In fact, using a gradient of morphogens instead of euclidean distance is not just a natural choice since the modules are not supposed to have global informations but it gets also a significant improvement to the quality of objects-tracking in our system.

The experiment presented in this chapter shows the capacity of the artificial hormone system to control the finite state machine (FSM) that schedules the steps to perform the global task. To do that, the dynamic of the global task is modeled by a FSM, then an artificial hormone system is used to control transitions between the FSM states. As a result, the robot's behavior is controlled by the hormone system.

An other interesting property of our approach is the ability to generate a separation strategy for the modules according to some circumstances. It would be also interesting to investigate the feasibility of biological cell division techniques as well as the multi-objective techniques for generating such strategy.

7

G-Programming-based Self-Reconfiguration Planning

In this chapter we introduce a genetic programming based engine for generating a near-optimal sequence of primitive actionts to reconfigure a classe of modular robots

Contents

7.1 Overview

Performing motion primitives in a coordinated way is the subject of the self-reconfiguration problem. The question is how to change connectivity among modules to transform the robot from the current configuration into the desired configuration within the restrictions of physical implementation[3,58,189,190] . One of the major challenges in such a problem is the number of possible configurations that increases exponentially with the number of modules and the number of their degrees of freedom. Chirikjian et al.[191] and Pamecha et al.[192] were proved that the reconfiguration of modular robots is NP-Complete problem. Thus, determining efficient sequences of modules operations that transform a modular robot from one configuration into another is a pressing need for efficient reconfiguration process.

Most available lattice-based modular robotic systems have only basic locomotion controllers to reshape the robot structure to a few predefined patterns by following predefined sequences and rules. These sequences have usually been hand-coded (by human operators), combined, generalized and optimized. However, in most cases the predefined sequences cannot predict all the possible situations that may face modular robots under dynamic environments. Further, since many modules may perform moves simultaneously, it is useful to increase the number of parallel steps to reduce the reconfiguration time.

In this chapter, we present an experimental study of self-reconfiguration planning using an automatic programming engine. The planner evolves and optimizes sequences of low-level actions which are required to transform the robot geometric structure from initial configuration to the target one. We assume that during reconfiguration the total number of modules and their connectedness must be preserved. This aspect ensures that the overall structure does not fragment during actuation due to module disconnections. The proposed planner is intended for both Crystalline and TeleCube modules which are designed and prototyped in hardware as cubical compressible units.

It is important to note that the mechanical actions (*attach, detach, expand, contract*) performed by the units are the slowest part of the system. Thus, we are also interested in reducing the total number of atomic operations required to achieve the desired reconfiguration.

7.2 Overview of the Simulator

Figure 7.1 shows some 2D compressible units in different configurations. The units are cubic in shape (squares in 2D) with an extensible arm on all six faces. Each unit is completely autonomous with computation, power and communication onboard. Besides, each unit can expand each of its six arms independently up to a factor of two times its fully contracted configuration.

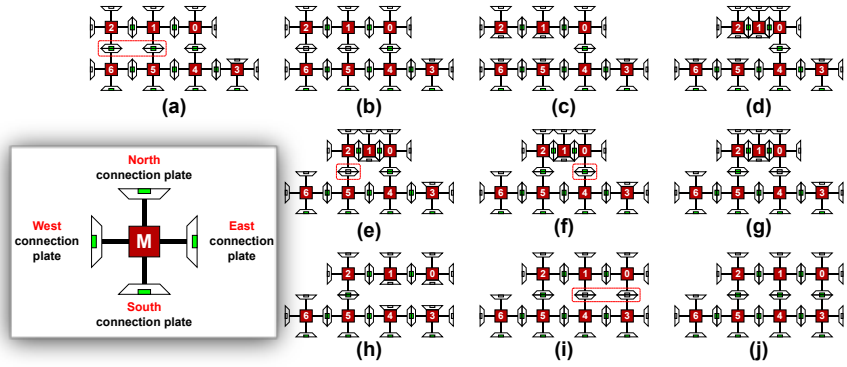

Figure 7.1: Compressible units in different configurations

A set range for contracting and expanding will then allow two neighboring modules to contract to half their normal size and fill a single grid space in the lattice. A single grid gap is then created for other modules to be pushed into. Each face of the unit (called a connection plate) is equipped with an electromechanical functionality, allowing the unit to clamp onto the neighboring unit's connection plate. The mechanical design of compressible modular robots is particularly well-suited to take advantage since modules are allowed to perform motion through the

interior of a robot structure rather than only along the exterior surface. The reader is invited to see[193] for more hardware details.

7.2.1 Unit-Compressible Motion

The unit-compressible motion is controlled by attaching the unit to a neighboring one and actuating the expansion or contraction mechanism. First, the unit disconnects from all neighbors in directions perpendicular to the direction of motion. Then the actual move is performed by expanding and contracting the back and the front arm. The unit slides one step in the desired direction with respect to some substrate modules. A simple module is unable to move by its own without interacting with neighboring modules, however it can carry other attached modules as long as the target positions for the carried modules are unoccupied. When the units combine these operations in a coordinated way, they move relative to one another, resulting in a reconfiguration of the robot as shown in Figure 7.1.

Crystalline and TeleCube modular robots have units that are arranged and connected in regular, three-dimensional pattern "*cubic grid*". As modules actuate the expansion or contraction mechanism each module performs linear motions, which can be described as a set of functions $f_i(x)$ where:

$$f_i(x) = x + \frac{1}{2}\mu_i \ , \ \begin{cases} x = \frac{1}{2}k \ , \ k \in \mathbb{Z}^3 \\ \mu_i \in \{(1,0,0),(0,1,0),(0,0,1)\} \end{cases} \tag{7.1}$$

7.2.2 Vocabulary of Module Actions

In what follows, the unit to which we refer as *parent* is the unit that executes a given action, while the *dangling* modules include all the units that are attached to the parent. During the robot's reconfiguration, each module can perform the following low-level primitives:

- ***ExpandArm(Direction)***: Before expanding arm in *Direction*, the following conditions must be guaranteed: (1) either, the parent or the dangling modules must be labeled as moving blocks, meaning that one of them is not attached to a fixed block of units, otherwise the sum of all forces is equal to zero and the motion fails, (2) collision should not be occurred when performing expansion. This primitive returns *true* if the expansion succeeds, and *false* otherwise.

- ***ContractArm(Direction)***: Before contracting arm in *Direction*, the following conditions must be guaranteed: (1) either, the parent unit or the dangling modules must be labeled as moving block, (2) collision should not be occurred when performing contraction. This primitive returns *true* if the contraction succeeds, and *false* otherwise.

- ***Connect(Direction)***: If there is an immediate neighboring module in *Direction*, activate the appropriate connection plate in order to bond with that module. Then update the neighboring list of the new connected modules.

- ***Disconnect(Direction)***: If there is a module linked in *Direction*, break the link with it only if the system remains connected after disconnection, then update the neighboring list of the new disconnected modules.

By combining these primitives, we can build more complicated actions for height-level manipulation of the robot. We can also build a complete program that reconfigures a modular robot to take several shapes.

7.2.3 Compressible Units Simulator

For our experiments, we implemented a simulator for two types of compressible modular robots based on Crystalline and TeleCube units. The units are square in shape and they all have the same size, they are located in a regular grid and act in a two dimensional discrete environment. We suppose that next to each connector there are a

transceiver and a set of appropriate sensors, which allow modules to communicate with connected neighbor modules and sense the state of the nearby grid as well. A single square module is embedded in a discrete coordinate system, occupies one grid location in the world when its four arms are contracted back to the body. In contrast, it occupies an amount of 1/2 of grid size in addition for each expanded arm. The four facing directions (*East, North, West, South*) of each module are respectively encoded as integer values in [0,3]. Moreover, the states of the four actuators are encoded as a Boolean array $S_{i\in[0,3]}$ where $S_{i\in[0,3]} = true$ if $arm_{i\in[0,3]}$ is expanded and $S_{i\in[0,3]} = false$ otherwise.

Basically, the simulator executes a given sequence of native actions and rewards each action with *true* if it is successfully performed and with *false* otherwise. For reason of simplicity, no complex physical constraints except collision-detection and Newton's three laws of motion are taken into account. When expanding and contracting arms, a unit exerts force (*Push* or *Pull*) upon connected neighboring units. By using Newton's laws of motion, it is possible to determine which part in the system is pushed or pulled against which other and in what direction. We suppose that the module's actuators (mechanisms that control arms) have constant strength (i.e., each unit can push and pull a constant number of other units when expanding and contracting). We suppose also that module arms can only occupy two states, fully expanded and fully contracted. When a module sticks somewhere to the floor, the simulator assumes infinite strength between the module and the floor to avoid slipping (at least one module must be connected to the floor or any stationary obstacle when performing motion).

7.3 GP-Based Reconfiguration Planning

Genetic Programming is one of a number of evolutionary algorithms that solves problems without being explicitly programmed[141] . It achieves this goal of *automatic programming* by genetically breeding a population of computer programs using biologically inspired operations (stochastic selection, crossover and mutation). On each generation, the fitness of

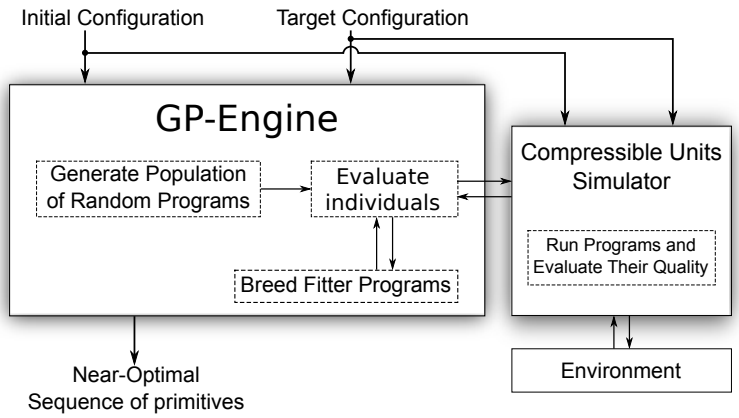

Figure 7.2: The general scheme of the planner. The planner takes initial and target configurations as inputs and outputs a near optimal sequence of primitives required to perform the corresponding transformation.

each individual in the population is evaluated. Once the termination criterion for the run is satisfied, the best single individual obtained during the run is considered as the result of the run. Such a process, stochastically transforms a population of random computer programs into new, more optimized generation of programs. Unlike genetic algorithm's (GA) individuals which are typically encoded as linear bit-strings, GP usually uses tree-based expressions as nonlinear structures for hierarchically structuring and manipulating individuals. The shape, and the contents of these individuals can dynamically change during the process.

In the area of complex systems, GP has demonstrated its potential in evolving programs for a wide range of applications including classification and pattern recognition, symbolic regression and circuit design[141,194] . In this work we used GP as an automatic programming engine to solve the reconfiguration planning problem. To do that, the GP computes a feasible plan that compiles down an abstract move into a sequence of valid native moves that, given a source robot S and a target robot T, reconfigures S into T. Both S and T are robots composed of n modules arranged in 2D grid. For the purpose of reliability, we require that the computed plan must maintain the system-connectivity on every time-step of the reconfiguration process.

We used tree-based GP-engine that was introduced by Koza[141] . In his

model, Koza has mentioned five major steps in preparing to use genetic programming paradigm to solve a problem. These are (1) choosing the set of terminals, (2) the set of functions, (3) the fitness measure, (4) the values of the numerical parameters and qualitative variables for controlling the run, (5) the criterion for designating a result and terminating a run.

7.3.1 Target Shape Description

Performing a successful reconfiguration requires the units to move toward target positions without being stuck behind each other or behind obstacles that exist in the environment. The goal of the units is to find target location and all move as close to it as possible.

Defining a fitness function as the sum of each module's distance to the target location is particularly an attractive choice in a number of situations. However, it seems to be useless when the environment contains concave obstacles[17] . Thus we used the concept of *Morphogens* for representing the target configuration.

Morphogens are long-range signaling molecules that act as graded positional cues to control cell differentiation in a concentration-dependent manner. This concept has brought to light the potential for signaling gradients to provide positional information for many developing multicellular organisms[195] . In previous work[17] we used morphogens to guide a metamorphic modular robot toward target objects without having any explicit information about their locations. In the present work we use the same strategy to provide positional information for the reconfiguring modular robot.

$$C_s(a) = C_0 e^{-\beta L_{(s,a)}} \tag{7.2}$$

We assumed that the environment contains different morphogens that are gradually spreading on the grid by means of the decay function indicated in equation 7.2, where C_0: is the morphogen concentration at

the location of a producing cell s, β is a positive decay rate and $L_{(s,a)}$: is the full length of the shortest path between the producing cell s and a given position a. Morphogens diffuse only through the empty grid with respect to the obstacles that exist in the environment. During their spreading, the substances follow the very easy way to move from the high concentrated grid to the low concentrated one.

The target pattern of the modular robot is defined by morphogen values of each grid. The morphogen is released from producing cells, which occupy each target grid. The morphogen value can be either positive or negative. A positive value means that the grid should be occupied by a module, while a negative gradient suggests that the module in the grid, if any, should be removed. A higher value of morphogen quantity indicates a higher priority for the grid to be filled by a module.

The distribution of morphogen gradients creates a virtual landscape that helps the modular robot to decide which direction to move to while flowing toward the target locations just like gravity does for fluids while traversing a terrain toward a sink.

7.3.2 Representation of GP Individuals

The set of all possible individuals that GP can generate includes all the trees that can be recursively created by compositions of a set of function symbols $F = \{f_1, f_2, ..., f_n\}$ and a set of terminal symbols $T = \{t_1, t_2, ..., t_m\}$. Each function in the function set F takes a specified number of arguments. Functions are derived directly from the description of the hardware capabilities, they include the four mechanical primitives that can be performed by each single module (*attach, detach, expand, contract*), each of which requires both the identity of the module by which it will be performed and the module's actuator (*east, north, west, south*) that will be used to perform this function.

The configuration space generated by a system of n lattice-based modules is $C = Z^{2n}$. It includes all configurations that may be achieved by the collection of these modules. Any configuration *(c)* is a subset of the

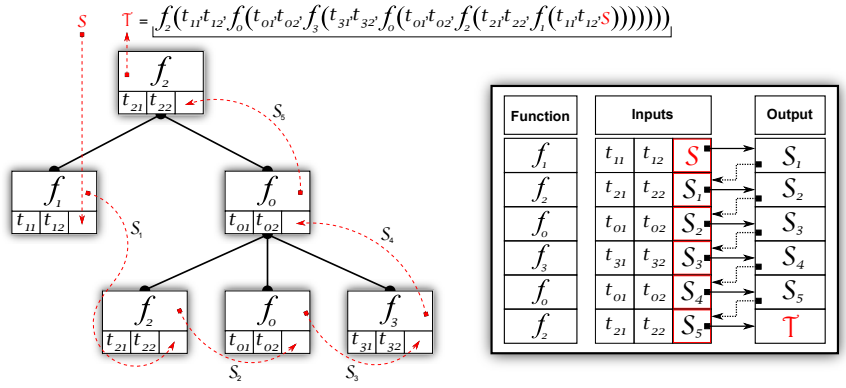

Figure 7.3: Hierarchical representation of a composition of functions.

configuration space ($c \subset Z^{2n}$) that refers to a particular arrangement of connectivity between modules.

As a given function f_i is applied to an arbitrary configuration c, the configuration c is either remaining intact or turning into a new configuration $\acute{c}$ as expressed in equation 7.3.

$$f_i(t_1, t_2, c) = \begin{cases} c & \text{if the function } f_i \text{ fails} \\ \acute{c} & \text{if the function } f_i \text{ succeeds} \end{cases} \tag{7.3}$$

A function $f_i(t_1, t_2, c)$ fails if one of the following reasons is occurred:

- Collision is detected: Due to module expansion and contraction by means of arguments (t_1, t_2), collision can occur either between modules or between any module and the obstacles that exist in the environment.
- The overall structure is fragmented : The robot structure may be divided into several parts due to module disconnections performed by means of (t_1, t_2).

Whatever a function $f_i \in F$ is succeeded or failed, it always results in a configuration $c \in C$, so that the next function can be applied to this result. The closure property is respected since each of the functions in the function set is able to accept, as one of its arguments, any value and data type that may possibly be returned by any function in the function

set (Figure 7.3). A sequence of consecutive native actions applied on a given configuration can be viewed as a composition of functions which is implemented as shown in Figure 7.4. The result returned by this composition of functions is the final configuration returned when the entire sequence of functions is executed while traversing the program tree by depth-first recursive process.

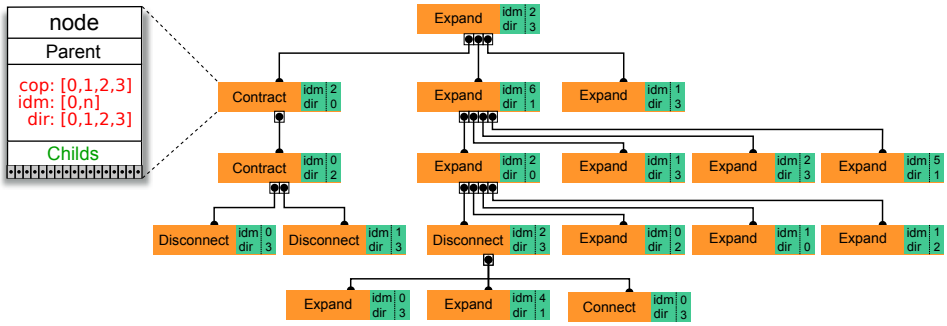

Figure 7.4: The Genotype implementation. Each node encodes a single primitive "C_{op}" and its arguments ("id_m": for the module identifier and "dir": for the actuator)

The set of terminal symbols T and the set of function symbols F are respectively represented as follow:

$T = \{t_1, t_2, c\}$

- The first terminal symbol $t_1 \in \{0, 1, ..., n-1\}$ is used to identify the module M_{t_1} that will perform a given action $f_i \in F$.
- The second terminal symbol t_2 takes value in $\{0, 1, 2, 3\}$. It is used to specify which one of the module's actuators (east, north, west, south) will be used to perform the given action.
- The third terminal symbol $c \in C$ represents the configuration on which the selected action f_i will be applied using arguments (t_1, t_2).

$$F = \{f_0, f_1, f_2, f_3\} \begin{cases} f_0(t_1, t_2, c) = & attach(t_1, t_2, c) \\ f_1(t_1, t_2, c) = & detach(t_1, t_2, c) \\ f_2(t_1, t_2, c) = & expand(t_1, t_2, c) \\ f_3(t_1, t_2, c) = & contract(t_1, t_2, c) \end{cases}$$

7.3.3 Fitness Evaluation

Fitness is the driving force of natural selection in both conventional genetic algorithms and genetic programming. It represents the degree to which a given individual solves the problem of interest. This measure is used to control the genetic operations (*selection, crossover, mutation*) in the artificial population and to determine the probability that the individual survives up to the next generation.

Since the fitness function has a great impact in guiding GP to obtain the best solutions within a very large search space, it is important to give it the best design as possible in order to help the GP to explore the search space more effectively and efficiently. Our experience applying GP to solve the reconfiguration problem suggests that the fitness function should take into account the following three metrics:

- The sum of all distances between the location of the units and the target locations where these units are supposed to be. However, instead of direct measuring these distances, we use the concept of positional information which is most commonly thought to be implemented by morphogen gradients. In general terms, assuming that the production of morphogens at each of the target locations, the perceived morphogen δ_M will increase as the modular robot gets closer to the target configuration. In contrast, responding in a concentration-dependent manner might result in driving the reconfiguration process to output an improperly shaped robot that does not match the desired configuration. This situation is occurred when the most concentrated grids do not bound the target configuration space. It is thus useful to consider other metrics such as the bounding volumes.

- Another such metric is the overlap metric δ_V which gives the volume remaining outside the target configuration $\delta_V^1 = V - V_T$. Where V is the space occupied by the achieved configuration, and V_T is the space occupied by the target configuration. In turn, this metric is expressed in term of the perceived negative morphogens which are equally spread outside the target configuration.

- The third metric δ_V^2 is the difference between the bounding boxes of both the achieved configuration and the desired configuration. $\delta_V^2 = V_T - V$

Let $x = \delta_M$ and $y = \delta_V^1 + \delta_V^2 = V + V_T - 2(V \cap V_T)$. The fitness function $f(x, y)$ we proposed is calculated by means of δ_M, δ_V^1 and δ_V^2. It takes x and y as arguments and returns the degree to which the modular robot matches the desired shape.

$$(\frac{\partial f}{\partial x} > 0) \wedge (\frac{\partial f}{\partial y} < 0) \tag{7.4}$$

In general, we assume the fitness of an individual increases with the perceived morphogens ($\frac{\partial f}{\partial x} > 0$), decreases with both the volume remained outside the target configuration and the difference between the bounding boxes of both the final configuration and the desired configuration ($\frac{\partial f}{\partial y} < 0$).

$$f(x, y) = \frac{1}{2^y}(1 + \frac{x}{x+1}) \tag{7.5}$$

$$Fitness = \frac{1}{2^{(\delta_V^1 + \delta_V^2)}}(1 + \frac{\delta_M}{\delta_M + 1}) \tag{7.6}$$

There are many possible functions satisfying the conditions given in equation 7.4. The fitness function that we designed is shown in equations 7.5 and 7.6. This function increases with the value of x and quickly loses value as y gets bigger (placing modules in a proper location is more important than getting closer to the target configuration), furthermore this function is injective and the gradient $\bigtriangledown f$ has always a non-zero value ($\bigtriangledown f(x, y) \neq (0, 0)\ \forall (x, y) \in \mathbb{R}^2$), which significantly helps the evolutionary algorithm to explore the research space without being stuck in critical points.

The use of arbitrary-length representations in genetic programming often presents the bloat challenge to the search process. This challenge is

occurred due to uncontrolled and unbounded code growth without a corresponding improvement in fitness. Bloat slows the evolutionary search process and significantly consumes memory. The fitness function expressed in equation 7.5, allows individuals to constantly grow during the evolution. Hence the search process has the potential to become inefficient due to the evaluation of big programs that usually suffer from the proliferation of *introns* (parts of the program that do not contribute to the problem resolution). To overcome this problem, one simple solution consists of punishing individuals in some way for being large. However, reducing constantly the size of the individuals carries the risk that the number of possible nodes in a tree of maximum depth could not be enough to represent a solution for the given problem. Instead, a Gaussian-weighted penalty is given to the fitness for controlling the influence of the individual's size on its performance to solve the problem at hand. Whatever an improvement of the fitness is occurred or not (equation 7.8), the weighted sum expressed in equation 7.7 receives more or less influence for the individual size.

We used the fitness function expressed in equation 7.7 to avoid setting hard limits on program sizes. The z argument represents the individual size in term of the sum of all nodes (the number of the encoded primitives).

$$f_t(x, y, z) = \frac{1}{2^y}(1 + \frac{x}{x+1}) - (1 - \frac{\delta_t}{\delta_0})\sqrt{z} \tag{7.7}$$

$$\delta_t = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-t^2}{2\sigma^2}} \ , \ t = \begin{cases} t/2 & if(\Delta f_t \neq 0) \\ t+1 & if(\Delta f_t = 0) \end{cases} \tag{7.8}$$

$$\Delta f_t = f_t(x, y, z) - f_{t-1}(x, y, z) \tag{7.9}$$

As the fitness function increases (meaning that the current population evolves), the parameter t decreases by half and it keeps decreasing by a factor of $\frac{1}{2^s}$ in every time step s while $\Delta f_t \neq 0$. At this stage, it is useful to lower the penalty for individuals to grow in order to allow them to get more nodes (genetic material) while there is a corresponding

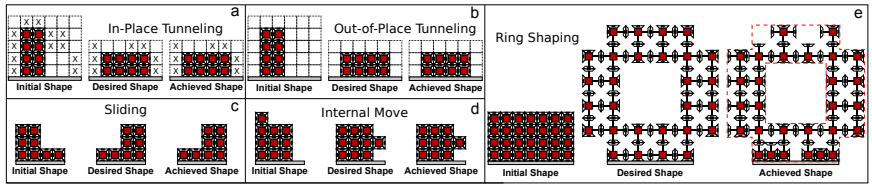

Figure 7.5: The five different reconfigurations considered for testing the planner. In-Place-Tunneling reconfiguration is constrained by obstacles "X symbol"

improvement in their fitness. The properties of this specification can be formulated as follows:

$$\lim_{s\to\infty} \delta_t = \delta_0 \Rightarrow \lim_{s\to\infty} f_t(x,y,z) = \frac{1}{2^y}(1+\frac{x}{x+1}) \tag{7.10}$$

The evolution should benefit from the creation of new genetic material, that should give the necessary amount of diversity for GP to evolve individuals.

In contrast, when $\Delta f_t = 0$ (no evolution is occurred in the considered population) the parameter t increases and keeps increasing in every time step s while $\Delta f_t = 0$. After awhile ($\delta_t \simeq 0$), we believe that the evolution stagnates thus it is useful to start optimizing the population (individuals that have less size replace those having the same performance for solving the problem at hand). This specification can be formulated as follows:

$$\lim_{s\to\infty} \delta_t = 0 \Rightarrow \lim_{s\to\infty} f_t(x,y,z) = \frac{1}{2^y}(1+\frac{x}{x+1}) - \sqrt{z}. \tag{7.11}$$

At this stage, individuals that exceed a certain threshold "Thr" in size are allowed to loos some genetic materials in the hope of reducing the amount of physical memory usage as well as the processing time required to run and evaluate each individual.

7.4 Experimental results

In our experiments, we used the standard GP mutation and recombination operators for trees. The mutation operator replaces a subtree with a randomly created subtree and the crossover operator swap subtrees rooted at two randomly chosen crossover points to generate two new offspring trees. The execution parameters for the GP are shown in table 7.1. We used roulette-wheel selection to select individuals for performing genetic operations. The population was initialized with random trees (programs) with maximum tree depth of 6. A population size of 500 individuals is used to achieve the results depicted in Figure 7.6. The simulator is synchronous, which means that each module performs its action in turn. This assumption is unrealistic, we are trying to relax it for future work.

Table 7.1: The set of all parameters used for the experiments

Parameters	Values
Population size	500
Crossover rate	90%
Mutation rate	40%
Probability of mutating C_{op}	20%
Probability of removing the selected node	20%
Probability of swapping a selected subtree with new one	20%

During the evolution, each individual must be run separately to measure how well it performs the task. The result of the composition of functions which are encoded in a given individual is the achieved configuration when the entire sequence of these functions is carried out on the initial state of the system. The fitter individual is the program that transforms the initial configuration into a configuration which corresponds most closely to the goal.

We empirically tested the proposed method on five different reconfigurations which are *In-Place Tunneling*, *Out-of-Place Tunneling*, *Sliding*, *Circle-Shaping* and *Internal-Move*. The reconfigurations are depicted in

Figure 7.5. Some of them require multiple waves of actuation, others require modules to perform corner turns or internal movements. The results depicted in Figure 7.6 are based on 24 runs. They show the progression over time of some system parameters while performing each task. They also show how the fitness function forces the GP to select genomes that best guide the modules to move toward positive morphogen gradient and away from negative morphogens.

The first observation we can make about these results is the capacity of the genetic programming to generate near-optimal sequences of primitives for compressible based modular robots to perform self reconfiguration. In 7.6.c, we notice that the initial solution is achieved in generation 300 with a size of 604 total operations. This solution contains a feasible sequence of functions that reshapes the modular robot into the desired pattern. However, the encoded expression is complicated by unnecessary duplication of functions "*introns*". For this reason, the GP starts to recombine the genetic material of each individual in order to pick-up the finest short-length expressions without what they loos their performance to solve the desired reconfiguration. By only executing effective sequences when testing the fitness function, evaluation can be accelerated significantly. In Figure 7.6.c a near-optimal expression is achieved in generation 1080 with a size of 15 total operations.

We noticed also that GP performs well for reconfigurations that require moving modules toward target location which are aligned with the direction of the move (Figure. 7.5.c). This kind of reconfiguration may be achieved just by involving a limited number of modules which execute some periodic sequences of actions. Here, it is important to note that some modules (dangling modules) will not contribute to the reconfiguration. In contrast, for reconfigurations that require orthogonal turns (Figure 7.5.a and 7.5.b) or those that require maneuvering modules on interior lines (7.5.d), the problem seems more complex to be solved by GP. The reason for which it takes more generation before the first feasible sequence is achieved. In these cases, the problem returns to the physical complexity of turning corners under unit-compressible actuation because the disconnections required to turn a corner (as well as to maneuver modules on interior lines) need a minimum amount of surrounding structure that must be present (some snapshots of interior

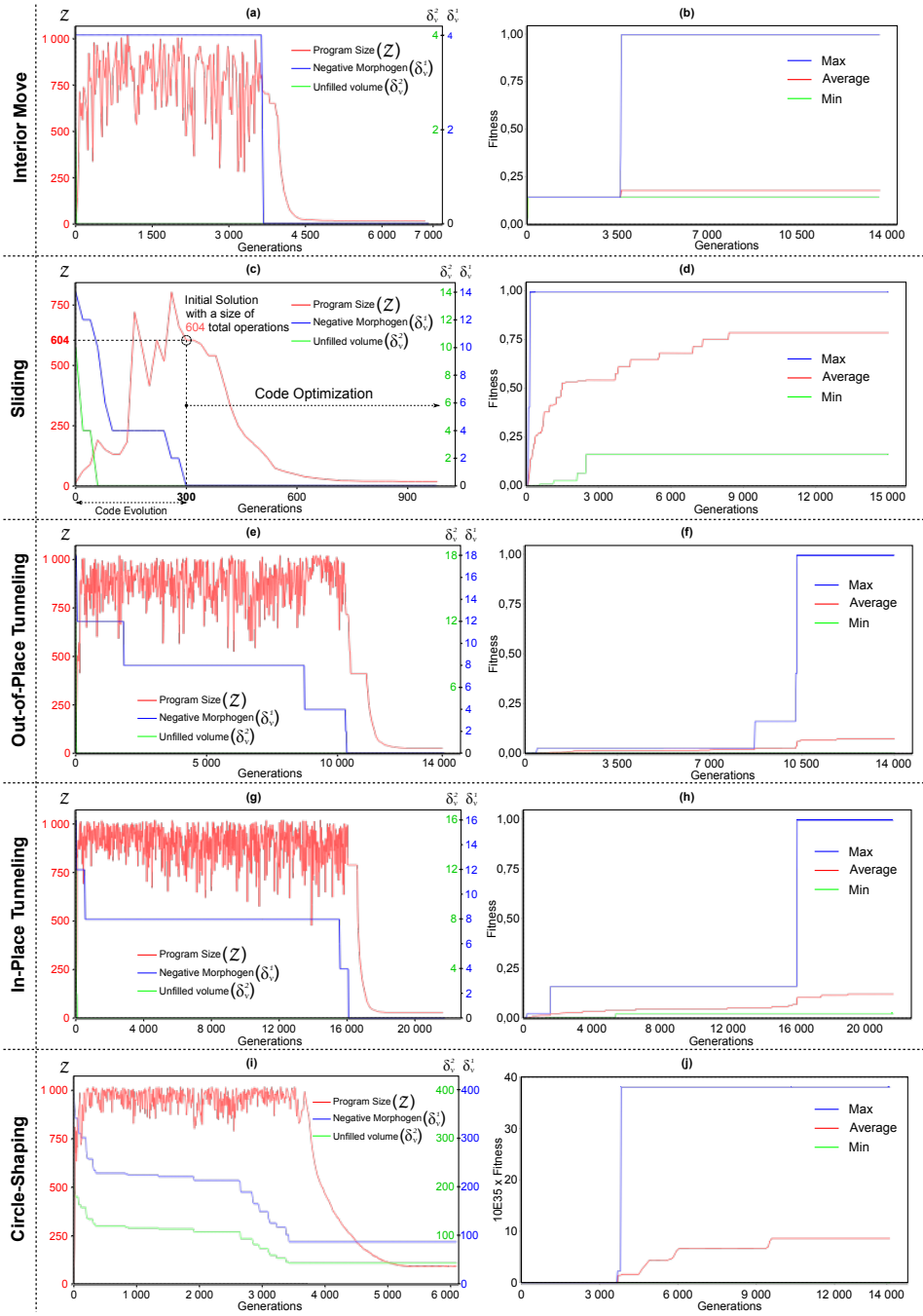

Figure 7.6: *Left-Hand* (a,c,e,g,i) the evolution over time of the size (z) of best individuals. *Right-Hand* (b,d,f,h,j) the fitness function against generations

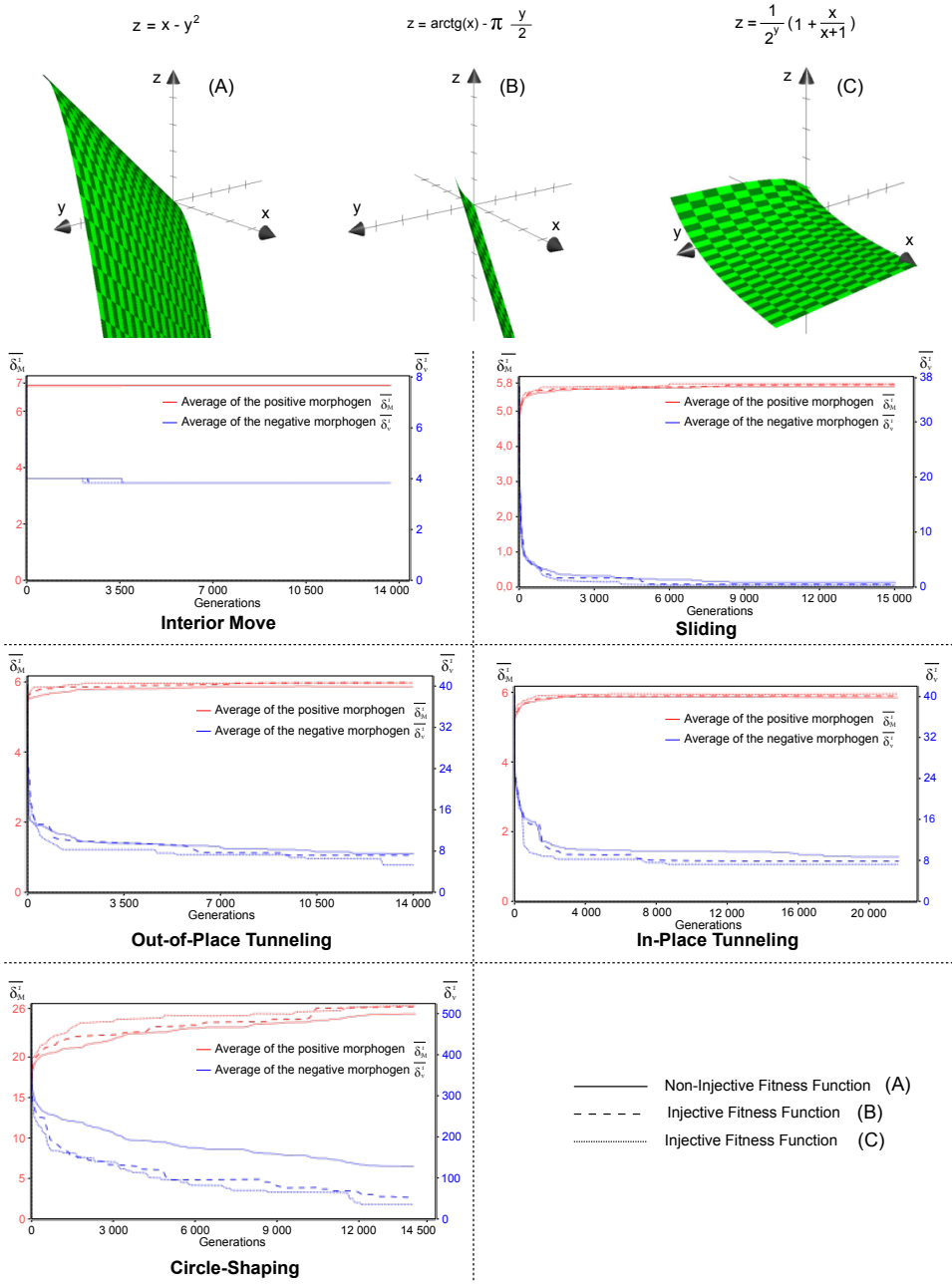

Figure 7.7: The evolution of the perceived positive and negative morphogens when using injective and non-injective fitness function

Table 7.2: Summary of experimental results

Reconfigurations	$n(modules)$	$F-Seq$	$Opt-Seq$	$Ratio(\%)$
Sliding	8	604	15	97.52 %
$Tunneling(Out)$	8	761	27	96.45 %
$Tunneling(In)$	8	995	30	96.98 %
Internal-move	10	803	17	97.88 %
Circle-Shaping	28	903	98	89.15 %

move reconfiguration are depicted in Figure 7.8).

Figure 7.7 shows how the different fitness function affect the optimal process. When the fitness function that associates each genome to one solution is not injective (function A in Figure 7.7), different genomes can encode the same solution and the representation is said to be degenerated. It's not intrinsically too serious problem to get a slight degeneracy. However, this phenomenon can sometimes badly affect the search process, mostly because if several genomes can represent the same phenotype. Which may make some kind of confusion in the search.

Table 7.2 shows the ratio between the average size of the first achieved sequences "*F-Seq*" and the size of the near-optimal sequences "*Opt-Seq*" for each reconfiguration. The optimized sequences have the capability to solve the reconfiguration in reduced steps where each sequence is executed primitive by primitive in a centralized fashion.

Although, it has been executed in a centralized fashion, the evolved program could also be executed in a decentralized fashion as explained in the following steps:

- Propagate the program to each unit in the system by message passing algorithm.
- Each unit keeps both the code segments in which it is involved and the corresponding predecessor for which it should wait before starting the execution of the next primitive.
- The first unit is the unit which is involved in the first line of the program. This unit starts the execution and propagates a termination signal at the end of each single execution.

- As a unit receives a termination signal from the appropriate predecessor, it starts the execution and propagate a termination signal at the end of each single execution.

Here, the termination signal is used for synchronizing the execution of primitives over multiple modules. It contains the module identifier which has just finished executing the current sequential primitive. As a unit receives this signal from a neighbor, it compares the signal identifier with that of the module it is waiting for (the module which executes the predecessor of its current primitive). If both identifiers are identical the unit starts executing its current primitive, otherwise it sends the received signal toward the neighboring units.

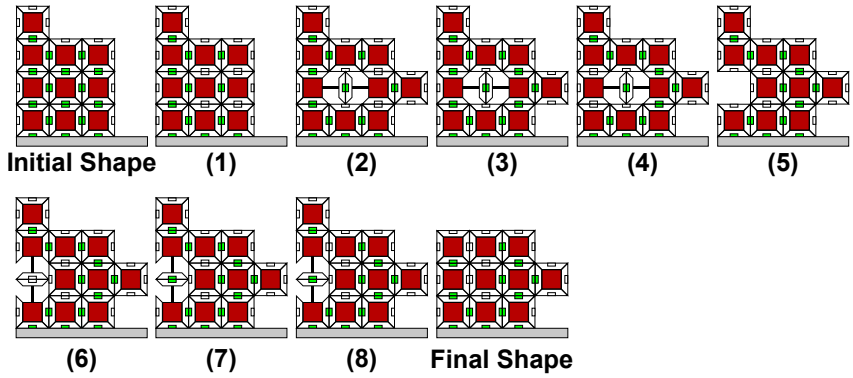

Figure 7.8: Snapshots of Interior Move Reconfiguration

7.5 Conclusion

In this chapter, we have used genetic programming to solve the self-reconfiguration problem for modular robots which are based on compressible units. We see the self reconfiguration as a set of functions recursively applied to a given configuration. These functions are designed according to the mechanical actuators (motion primitives) of each unit in the system.

Here, genetic programming has been used as an automatic programming tool to find a near-optimal sequence of module motion primitives that transform the start configuration into the goal. Our approach does not require any extra effort to breaking down the problem into subproblems as it must be done when using dynamic programming method. In this work, the evolution is divided into two phases. First, a population of random programs is evolved toward better programs (sequence of motion primitives that make the initial configuration more like that target configuration). Once a feasible sequence is achieved, the next phase of evolution will focus on minimizing the total number of primitives performed over all units in the hope of reducing the total power consumption.

We have tested this technique on both in-place and out-of-place reconfigurations. As was noted in[55] , in-place reconfigurations are harder to be achieved than out-of-place reconfigurations. For the first one, the modules tend to suffer from overcrowding due to the restricted available space while for the second the modules are more $free$ to actuate the reconfiguration. The experimental results also show that the system works well for generating primitive sequences for both Slide and Tunnel reconfigurations, however, it does not converge very well for planning reconfiguration that needs interior move of some modules.

For future work, we are continuing to write algorithms for automatically identifying and encapsulating potentially useful subtrees that serve as high-level primitives for different levels of granularity.

8

Conclusions and Future Work

In this chapter we summarize our research and we outline the future work

Contents

8.1 Summary

Self-reconfiguring robots are able to adapt to the operating environment and required functionality by changing their shape. They consist of a set of identical robotic modules that can autonomously and dynamically change their aggregate geometric structure to suit different locomotion, manipulation, and sensing tasks. However, providing robots with self-reconfiguration capabilities is a serious challenge, now being met through new designs for reconfigurable systems and new ideas about algorithmic planning and control that confer autonomous reconfigurability.

This thesis addressed some challenges related to the *adaptation*, *self-replication* and *self-formation* problems of modular robotic systems. These robotic systems will constitute long-lived distributed systems, all the supporting hardware and software will have to be robust, long-lasting, fault-tolerant, scalable, and self-healing. The hardware will have to rely on simple and robust actuation. The units will have to be powered for long periods of time. Adding and removing units into the system will have to be incremental, in that, these changes should affect the overall system only locally. When units break down, the system should be able to repair itself without altering overall global functionality. The units will have to be networked with a reliable communication infrastructure and control will have to be highly parallel, scalable, and distributed.

To develop such systems, designers have to improve their understanding of the general properties that confer modular robots with self-reconfiguration capabilities, as well as more generic (rather than architecture-specific) solutions to control and planning. These issues are being addressed by the research community, motivated by the exciting vision of versatile robots achieving the same level of flexibility as biological systems of cells.

8.2 Future Work

As future work, the present thesis suggests various future research directions related to the modular robotic systems, in this section we outline some of these directions.

8.2.1 Improvement on the proposed approaches

- Extend the developed simulator to support more physical constraints for more realistic modular actuations.
- Adapt the developed framework to be run in parallel computing devices (Clusters, GPU).
- Tackling the issue of code-bloat problem in genetic programming for more efficient research.
- For the developed GP-Engine, it would be interesting to write algorithms for automatically identifying and encapsulating potentially useful subtrees that serve as high-level primitives for different levels of granularity

8.2.2 Evolving efficient Communication system for modular robots

The broadcast communication is not allowed for modular robots since it is very expensive in term of energy and messaging management. Thus, it should be interesting to evolve an efficient communication system for reducing the message passing traffic through the modules.

Bibliography

[1] Stoy Kasper. Using cellular automata and gradients to control self-reconfiguration. *Journal of Robotics and Autonomous Systems*, 54:135–141, 2006.

[2] Fitch Robert, Zack Butler, and Daniela Rus. Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 117–124, 2005.

[3] Mark Yim, Shen Wei-Min, Salemi Behnam, Rus Daniela, Moll Mark, Lipson Hod, Klavins Eric, and Chirikjian Gregory. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics and Automation Magazine*, 14(1):43–52, 2007.

[4] Toshio Fukuda and Seiya Nakagawa. Approach to the dynamically reconfigurable robotic system. *Journal of Intelligent and Robotic Systems*, 1(1):55–72, 1988.

[5] Daniela Rus, Zack Butler, Keith Kotay, and Marsette Vona. Self-reconfiguring robots. *Communications of the ACM*, 45:39–45, 2002.

[6] Kotay Keith and Daniela Rus. Generic distributed assembly and repair algorithms for self-reconfiguring robots. *Proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2362–2369, 2004.

[7] Yu Chih-Han, François-Xavier Willems, Donald Ingber, and Radhika Nagpal. Self-organization of environmentally-adaptive shapes on a modular robot. *Proceeding IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2353–2360, 2007.

[8] Paulina Varshavskaya. Distributed reinforcement learning for self-reconfiguring modular robots. *Ph.D. dissertation, Massachusetts Institute of Technology (MIT)*, 2007.

[9] Karl Sims. Evolving virtual creatures. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques, ACM*, pages 15–22, 1994.

[10] Karl Sims. Evolving 3d morphology and behavior by competition. *Proceedings of artificial life IV*, pages 353–372, 1994.

[11] Berlanga Antonio, Pedro Isasi, Araceli Sanchis, and Jose M. Molina. Neural networks robot controller trained with evolution strategies. *Proceedings of the IEEE Congress on Evolutionary Computation-CEC99*, 1:413–419, 1999.

[12] Ijspeert Auke Jan. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21:642–653, 2008.

[13] Wei-Min Shen Salemi, Behnam and Peter Will. Hormone-controlled metamorphic robots. *In IEEE International Conference on Robotics and Automation (ICRA)*, pages 4194–4199, 2001.

[14] Christensen David Johan, Ulrik Pagh Schultz, and Kasper Stoy. A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots. *Journal of Robotics and Autonomous Systems*, 61:1021–1035, 2013.

[15] Larkworthy Tom and Subramanian Ramamoorthy. An efficient algorithm for self-reconfiguration planning in a modular robot. *IEEE International Conference in Robotics and Automation (ICRA)*, pages 5139–5146, 2010.

[16] Butler Zack, Satoshi Murata, and Daniela Rus. Distributed replication algorithms for self-reconfiguring modular robots. *Distributed Autonomous Robotic Systems, Springer Japan*, 5:37–48, 2002.

[17] Tarek Ababsa, NourEddine Djedi, Yves Duthen, and Sylvain Cussat Blanc. Splittable metamorphic carrier robots. *Artificial Life 14, MIT press*, pages 801–808, 2014.

[18] Tarek Ababsa, NourEddine Djedi, and Yves Duthen. Genetic programming-based self-reconfiguration planning for metamorphic robot. *International Journal of Automation and Computing (IJAC)*, xx:xx–xx, 2016.

[19] Tarek Ababsa, NourEddine Djedi, Yves Duthen, and Sylvain Cussat Blanc. Decentralized approach to evolve the structure of metamorphic robots. *IEEE Symposium on Artificial Life*, pages 74–81, 2013.

[20] Yim Mark, Paul White, Michael Park, and Jimmy Sastra. Modular self-reconfigurable robots. *Encyclopedia of complexity and systems science, Springer New York*, pages 5618–5631, 2009.

[21] Rus Daniela and Vona Marsette. Self-reconfiguration planning with compressible unit modules. *Proceedings of the IEEE International Conference on Robotics and Automation*, 4:2513–2520, 1999.

[22] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule: Design and control algorithms. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 424–431, 1998.

[23] Vassilvitskii Serguei, Jeremy Kubica, Eleanor Rieffel, John Suh, and Mark Yim. On the general reconfiguration problem for expanding cube style modular robots. *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 1:801–808, 2002.

[24] Mark Yim, David G. Duff, and Kimon D. Roufas. Polybot: a modular reconfigurable robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1:514–520, 2000.

[25] Yim Mark, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Sam Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 2:225–237, 2003.

[26] Li Yong, Houxiang Zhang, and Shengyong Chen. A four-legged robot based on gz-i modules. *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, pages 921–926, 2009.

[27] Hossain S.G.M., Carl A. Nelson, and Prithviraj Dasgupta. Modred: A modular self-reconfigurable robot for autonomous extra-terrestrial exploration and discovery. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 747–754, 2011.

[28] Sproewitz Alexander, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert. Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4259–4264, 2009.

[29] Akiya Kamimura, Yoshida Eiichi, Murata Satoshi, Kurokawa Haruhisa, Tomita Kohji, and Shigeru Kokaji. A self-reconfigurable modular robot (mtran) — hardware and motion planning software. *Distributed Autonomous Robotic Systems, Springer Japan*, pages 17–26, 2002.

[30] Victor Zykov, Andrew Chan, and Hod Lipson. Molecubes: An open-source modular robotics kit. *Presented at the Proceeding of International Conference of Intelligent Robots Systems*, pages 3–6, 2007.

[31] Zykov Victor, Efstathios Mytilinaios, Mark Desnoyer, and Hod Lipson. Evolved and designed self-reproducing modular robotics. *IEEE Transactions on robotics*, 2:308–319, 2007.

[32] Chiu Harris CH, Michael Rubenstein, and Wei-Min Shen. Multifunctional superbot with rolling track configuration. *IROS 2007 Workshop on Self-Reconfigurable Robots and Systems and Applications*, pages 50–53, 2007.

[33] Sahin Erol, Thomas H. Labella, Vito Trianni, J-L. Deneubourg, Philip Rasse, Dario Floreano, Luca Maria Gambardella, Francesco Mondada, Stefano Nolfi, and Marco Dorigo. Swarm-bot: pattern formation in a swarm of self-assembling mobile robots. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 145–150, 2002.

[34] Rubenstein Michael, Christian Ahler, and Radhika Nagpal. Kilobot: A low-cost scalable robot system for collective behanviors. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3293–3298, 2012.

[35] Kernbach Serge, Eugen Meister, Florian Schlachter, Kristof Jebens, Marc Szymanski, Jens Liedke, and Davide Laneri. Symbiotic robot organisms: Replicator and symbrion projects. *Proceedings of the 8th workshop on performance metrics for intelligent systems, ACM, New York*, pages 62–69, 2008.

[36] Romanishin John W., Kyle Gilpin, and Daniela Rus. M-blocks: Momentum-driven, magnetic modular robots. *Proceedings of International Conference on Intelligent Robots and Systems*, pages 4288–4295, 2013.

[37] Goldstein Seth Copen, Jason D. Campbell, and Todd C. Mowry. Programmable matter. *Computer*, 38:99–101, 2005.

[38] Gilpin Kyle, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *International Journal of Robotics Research*, 27:345–372, 2008.

[39] Gilpin Kyle, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2485–2492, 2010.

[40] Neubert Jonas, Abraham P. Cantwell, Stephane Constantin, Michael Kalontarov, David Erickson, and Hod Lipson. A robotic module for stochastic fluidic assembly of 3d self- reconfiguring structures. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2479–2484, 2010.

[41] Gregory Chirikjian and Amit Pamecha. Bounds for self-reconfiguration of metamorphic robots. *Proceedings of IEEE International Conference of Robotics and Automation*, 2:1452–1457, 1996.

[42] Stoy Kasper, David Brandt, David J. Christensen, and David Brandt. Self-reconfigurable robots: An introduction. *The MIT Press*, 2010.

[43] Feili Hou and Wei-Min Shen. On the complexity of optimal reconfiguration planning for modular reconfigurable robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2791–2796, 2010.

[44] Yim Mark. Locomotion with a unit-modular reconfigurable robot. *Ph.D. thesis, Stanford University*, 1994.

[45] Frank Harary and Ronald C. Read. The enumeration of tree-like polyhexes. *Proceedings of the Edinburgh Mathematical Society*, 17:1–13, 1970.

[46] Hossein Ahmadzadeh and Ellips Masehian. Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization. *Artificial Intelligence*, 223:27–64, 2015.

[47] Judea Pearl. Heuristics: Intelligent search strategies for computer problem solving. *Addison-Wesley Longman Publishing Co.*, 1984.

[48] Arancha Casal and Mark H. Yim. Self-reconfiguration planning for a class of modular robots. *International Symposium on Intelligent Systems and Advanced Manufacturing (SPIE)*, pages 246–257, 1999.

[49] Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 3:2460–2467, 2003.

[50] Butler Zack, Keith Kotay, Daniela Rus, and Kohji Tomita. Generic decentralized control for a class of self-reconfigurable robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1:809–816, 2002.

[51] Mark Yim, Ying Zhang, John Lamping, and Eric Mao. Distributed control for 3d metamorphosis. *Autonomous Robots*, 10:41–56, 2001.

[52] Kotay Keith D. and Daniela L. Rus. Algorithms for self-reconfiguring molecule motion planning. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2184–2193, 2000.

[53] Zhang Liang, Jie Zhao, and He Gao Cai. A substructure based motion planning method for a modular self-reconfigurable robot. *Proceedings of the IEEE International Workshop on Robot Motion and Control (RoMoCo)*, pages 371–376, 2004.

[54] Dewey Daniel J., Michael P. Ashley-Rollman, Michael De Rosa, Seth Copen Goldstein, Todd C. Mowry, Siddhartha S. Srinivasa, Padmanabhan Pillai, and Jason Campbell. Generalizing metamodules to simplify planning in modular robotic systems. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1338–1345, 2008.

[55] Greg Aloupis, Sebastien Collette, Mirela Damian, Erik D. Demaine, Robin Flatland, Stefan Langerman, Joseph O'Rourke, Suneeta Ramaswami, Vera Sacristan, and Stefanie Wuhrer. Linear reconfiguration of cube-style modular robots. *International Symposium on Algorithms and Computation (ISAAC), Springer Berlin Heidelberg*, pages 208–219, 2007.

[56] Liu JinGuo, YueChao Wang, Bin Li, ShuGen Ma, and DaLong Tan. Center-configuration selection technique for the reconfigurable modular robot. *Science in China Series F: Information Sciences*, 50:697–710, 2007.

[57] Wei HongXing, HaiYuan Li, JinDong Tan, and TianMiao Wang. Self-assembly control and experiments in swarm modular robots. *Science China Technological Sciences*, 55:1118–1131, 2012.

[58] Daniela Rus and Marsette Vona. Crystalline robots: self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10:107–124, 2001.

[59] Ghallab Malik, Dana Nau, and Paolo Traverso. Automated planning: Theory and practice. *Elsevier*, 2004.

[60] Nau Dana S., Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2: an htn planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, 2003.

[61] Bihlmaier Andreas, Lutz Winkler, and Heinz Worn. Automated planning as a new approach for the self-reconfiguration of mobile modular robots. *The 9th Workshop on Robot Motion and Control (RoMoCo)*, pages 60–65, 2013.

[62] Unsal Cem, Han Kiliccote, and Pradeep K. Khosla. A modular self-reconfigurable bipartite robotic system: implementation and motion planning. *Autonomous Robots*, pages 23–40, 2001.

[63] Bhat Preethi, James Kuffner, Seth Goldstein, and Siddhartha Srinivasa. Hierarchical motion planning for self-reconfigurable modular robots. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 886–891, 2006.

[64] Greg Aloupis, Sebastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristan, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using o(log n) parallel moves. *International Symposium on Algorithms and Computation (ISAAC)*, pages 342–353, 2008.

[65] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. *MIT Press, Cambridge.*, 6, 2001.

[66] Ali Nourollah and Mohammadreza Razzazi. Minimum cost open chain reconfiguration. *Discrete Applied Mathematics*, 159:1418–1424, 2011.

[67] Robert Fitch and Zack Butler. Million module march: scalable locomotion for large self-reconfiguring robots. *International Journal of Robotics Research*, 27:331–343, 2008.

[68] Amit Pamecha, Imme Ebert-Uphoff, and Gregory S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13:531–545, 1997.

[69] Chih-Jung Chiang and Gregory S. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10:91–106, 2001.

[70] Sunil Pranit Lal, Koji Yamada, and Satoshi Endo. Emergent motion characteristics of a modular robot through genetic algorithm. *Proceedings of thr International Conference on Intelligent Computing, Springer Berlin Heidelberg*, 5227:225–234, 2008.

[71] Christensen David Johan. Evolution of shape-changing and self-repairing control for the atron self-reconfigurable robot. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2539–2545, 2006.

[72] Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita E. Toshida, Satoshi Murata, and Shigeru Kokaji. Automatic locomotion pattern generation for modular robots. *Proceedings of th IEEE International Conference on Robotics and Automation (ICRA)*, 1:714–720, 2003.

[73] I.M. Chen. Theory and applications of modular reconfigurable robotic systems. *PhD thesis, California Institute of Technology, Pasadena, CA.*, 1994.

[74] Bo Dong and Yuanchun Li. Multi-objective-based configuration generation and optimization for reconfigurable modular robot. *Proceedings of the IEEE International Conference on Information Science and Technology, ICIST.*, pages 1006–1010, 2011.

[75] Kasper Stoy. How to construct dense objects with self-recondfigurable robots. *European Robotics Symposium, Springer Berlin Heidelberg*, pages 27–37, 2006.

[76] Gregory S. Chirikjian. Kinematics of a metamorphic robotic system. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 449–455, 1994.

[77] Kasper Stoy and Radhika Nagpal. Self-repair through scale independent self-reconfiguration. *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, 2:2062–2067, 2004.

[78] Robert Fitch and Rowan McAllister. Hierarchical planning for self-reconfiguring robots using module kinematics. *Distributed Autonomous Robotic Systems, Springer Berlin Heidelberg*, pages 477–490, 2013.

[79] Satoshi Murata, Haruhisa Kurokawa, and Shigeru Kokaji. Self-assembling machine. *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 441–448, 1994.

[80] Heiko Hamann, Jurgen Stradner, Thomas Schmickl, and Karl Crailsheim. Artificial hormone reaction networks: towards higher evolvability in evolutionary multi-modular robotics. *arXiv preprint arXiv*, 2010.

[81] Soha Pouya, Jesse Van Den Kieboom, Alexander Sprowitz, and Auke Jan Ijspeert. Automatic gait generation in modular robots: to oscillate or to rotate? that is the question. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 514–520, 2010.

[82] Kasper Stoy. Controlling self-reconfiguration using cellular automata and gradients. *Proceedings the 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, pages 693–702, 2004.

[83] Kasper Stoy and Radhika Nagpal. Self-reconfiguration using directed growth. *Proceeding of the International Symposium on Distributed Autonomous Robotic System, Springer*, 6:3–12, 2007.

[84] Zack Butler and Daniela Rus. Distributed motion planning for 3d modular robots with unit-compressible modules. *Algorithmic Foundations of Robotics V, Springer Berlin Heidelberg*, pages 435–452, 2004.

[85] Zack Butler and Daniela Rus. Distributed planning and control for modular robots with unit-compressible modules. *International Journal of Robotics Research*, 22:699–715, 2003.

[86] Serge Kernbach, Eugen Meister, Florian Schlachter, Kristof Jebens, Marc Szymanski, Jens Liedke, and Davide Laneri. Symbiotic robot organisms: Replicator and symbrion projects. *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems (PerMIS), ACM*, pages 62–69, 2008.

[87] Hilary L. Ashe and James Briscoe. The interpretation of morphogen gradients. *Development 133*, 3:385–394, 2006.

[88] Ronald Thenius, Markus Dauschan, Thomas Schmickl, and Karl Crailsheim. Regenerative abilities in modular robots using virtual embryogenesis. *Adaptive and Intelligent Systems, Springer, Berlin Heidelberg*, pages 227–237, 2011.

[89] Yaochu Jin and Yan Meng. Morphogenetic robotics: an emerging new field in developmental robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41:145–160, 2011.

[90] Yan Meng and Yaochu Jin. Morphogenetic self-reconfiguration of modular robots. *Bio-Inspired Self-Organizing Robotic Systems, Springer, Berlin Heidelberg*, pages 143–171, 2011.

[91] Yan Meng, Yuyang Zhang, and Yaochu Jin. Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model. *IEEE Computational Intelligence Magazine*, 6:43–54, 2011.

[92] Satoshi Murata and Haruhisa Kurokawa. Self-organization of biological systems. *Self-Organizing Robots, Springer, Tokyo*, pages 19–35, 2012.

[93] Takuya Umedachi, Taichi Kitamura, Koichi Takeda, Toshiyuki Nakagaki, Ryo Kobayashi, and Akio Ishiguro. A modular robot driven by protoplasmic streaming. *Distributed Autonomous Robotic Systems, Springer, Berlin Heidelberg*, 8:193–202, 2009.

[94] Jeff Jones, Soichiro Tsuda, and Andrew Adamatzky. Towards physarum robots. *Bio-Inspired Self-Organizing Robotic Systems, Springer, Berlin Heidelberg*, 8:215–251, 2011.

[95] Zack Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *International Journal of Robotics Research*, pages 919–937, 2004.

[96] George M. Whitesides and Bartosz Grzybowski. Self-assembly at all scales. *Science*, 5564:2418–2421, 2002.

[97] Qiu-xuan Wu, Ya hui Wang, Guang yi Cao, and Yan qiong Fei. Locomotion control of distributed self-reconfigurable robot based on cellular automata. *Proceeding of the International Conference on Intelligent Computing (ICIC), Berlin Heidelberg*, 5564:179–188, 2005.

[98] Satoshi Murata and Haruhisa Kurokawa. Robotic metamorphosis. *Self-Organizing Robots, Springer, Tokyo*, 5564:131–171, 2012.

[99] Eiichi Yoshida, Satoshi Matura, Akiya Kamimura, Kohji Tomita, Haruhisa Kurokawa, and Shigeru Kokaji. A self-reconfigurable modular robot: reconfiguration planning and experiments. *International Journal of Robotics Research*, 21:903–915, 2002.

[100] Paulina Varshavskaya, Leslie Pack Kaelbling, and Daniela Rus. Automated design of adaptive controllers for modular robots using reinforcement learning. *International Journal of Robotics Research*, 27:505–526, 2008.

[101] Paulina Varshavskaya. Distributed reinforcement learning for self-reconfiguring modular robots. *Ph.D. thesis, Massachusetts Institute of Technology*, 2007.

[102] Lund Henrik Hautop, Rasmus Lock Larsen, and Esben Hallundbaek Ostergaard. Distributed control in self-reconfigurable robots. *Proceeding of the International Conference on Evolvable Systems, Springer, Berlin Heidelberg*, pages 296–307, 2003.

[103] Gregory S. Hornby and Jordan B. Pollack. Body-brain co-evolution using l-systems as a generative encoding. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 868–875, 2001.

[104] Kimon Roufas Mark Yim, Sam Homans. Climbing with snake-like robots. *Workshop on Mobile Robot Technology (IFAC)*, pages 21–22, 2001.

[105] Mark Yim, Craig Eldershaw, Ying Zhang, and David Duff. Limbless conforming gaits with modular robots. *Experimental Robotics IX, Springer, Berlin Heidelberg.*, pages 459–468, 2006.

[106] Kasper Stoy, Wei-Min Shen, and Peter Will. Global locomotion from local interaction in self-reconfigurable robots. *Proceedings of the 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, pages 309–316, 2002.

[107] Fred Delcomyn. Neural basis of rhythmic behavior in animals. *Science*, 4469:492–498, 1980.

[108] Gonzalez Gomez, Houxiang Zhang Juan, Eduardo I. Boemo, and Jianwei Zhang. Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules. *The 9th International Conference on Climbing and Walking Robots*, 2006.

[109] Rodrigo Moreno and Jonatan Gomez. Central pattern generators and hormone inspired messages: a hybrid control strategy to implement motor primitives on chain type modular reconfigurable robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1014–1019, 2011.

[110] Satoshi Murata, Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, and Shigeru Kokaji. Self-reconfigurable robots: platforms for emerging functionality. *Embodied Artificial Intelligence, Springer, Berlin Heidelberg*, pages 312–330, 2004.

[111] Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, Akiya Kamimura, Satoshi Murata, and Shigeru Kokaji. Self-reconfigurable m-tran structures and walker generation. *Journal of Robotics and Autonomous Systems*, pages 142–149, 2006.

[112] Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, Shigeru Kokaji, and Satoshi Murata. Distributed adaptive locomotion by a modular robotic system, m-tran ii. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2370–2377, 2004.

[113] Rico Mockel, Cyril Jaquier, Kevin Drapel, Elmar Dittrich, Andres Upegui, and A. Ijspeert. An autonomous modular robot with bluetooth interface for exploring adaptive locomotion. *Climbing and Walking Robots, Springer, Berlin Heidelberg*, pages 685–692, 2006.

[114] A. Sproewitz, R. Moeckel, J. Maye, M. Asadpour, and A. J. Ijspeert. Adaptive locomotion control in modular robotics. *Workshop on Self-Reconfigurable Robots/Systems and Applications (IROS)*, pages 81–84, 2007.

[115] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10:99–127, 2002.

[116] Evert Haasdijk, Andrei A. Rusu, and A. E. Eiben. Hyperneat for locomotion control in modular robots. *International Conference on Evolvable Systems, Springer, Berlin Heidelberg*, pages 169–180, 2010.

[117] Olivier Chocron. Evolving modular robots for rough terrain exploration. *Mobile Robots: The Evolutionary Approach, Springer, Berlin Heidelberg*, pages 23–46, 2007.

[118] Soha Pouya, Ebru Aydin, Rico Mockel, and Auke Jan Ijspeert. Locomotion gait optimization for modular robots; coevolving morphology and control. *Procedia Computer Science*, pages 320–322, 2011.

[119] Andres Faina, Francisco Bellas, Fernando Lopez-Pena, and Richard J. Duro. Edhmor: evolutionary designer of heterogeneous modular robots. *Engineering Applications of Artificial Intelligence*, 26, 2013.

[120] Thomas Schmickl, Heiko Hamann, and Karl Crailsheim. Modelling a hormone-inspired controller for individual-and multi-modular robotic systems. *Mathematical and Computer Modeling of Dynamical Systems*, 17:221–242, 2011.

[121] Wei-Min Shen, Yimin Lu, and Peter Will. Hormone-based control for self-reconfigurable robots. *Proceedings of the fourth international conference on Autonomous agents*, pages 1–8, 2000.

[122] Heiko Hamann, Jurgen Stradner, Thomas Schmickl, and Karl Crailsheim. A hormone-based controller for evolutionary multi-modular robotics: from single modules to gait learning. *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2010.

[123] Wei-Min Shen, Behnam Salemi, and Peter Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE transactions on Robotics and Automation*, 18:700–712, 2002.

[124] Serge Kernbach, Benjamin Girault, and Olga Kernbach. On self-optimized self-assembling of heterogeneous multi-robot organisms. *Bio-Inspired Self-Organizing Robotic Systems, Springer, Berlin Heidelberg*, pages 123–141, 2011.

[125] J. Bishop, Samuel Burden, Eric Klavins, R. Kreisberg, W. Malone, Nils Napp, and T. Nguyen. Programmable parts: a demonstration of the grammatical approach to self-organization. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3684–3691, 2005.

[126] Elio Tuci, Roderich Grob, Vito Trianni, Francesco Mondada, Michael Bonani, and Marco Dorigo. Cooperation through self-assembly in multi-robot systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, pages 115–150, 2006.

[127] Eric Klavins. Programmable self-assembly. *IEEE Control systems Magazine*, pages 43–56, 2007.

[128] Kazuo Hosokawa, Isao Shimoyama, and Hirofumi Miura. Dynamics of self-assembling systems: analogy with chemical kinetics. *Artificial Life*, pages 413–427, 1994.

[129] Shuhei Miyashita, Marco Kessler, and Max Lungarella. How morphology affects self-assembly in a stochastic modular robot. *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3533–3538, 2008.

[130] Eric Klavins, Samuel Burden, and Nils Napp. Optimal rules for programmed stochastic self-assembly. *Robotics: Science and Systems II*, pages 9–16, 2007.

[131] Tolley, Michael T., and Hod Lipson. Programmable 3d stochastic fluidic assembly of cm-scale modules. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4366–4371, 2011.

[132] Josh C. Bongard. Evolving modular genetic regulatory networks. *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 17–21, 2002.

[133] Christopher G. Langton. *Artificial Life.* MIT Press, 1989.

[134] Martin Gardner. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[135] Marco Villani, Roberto Serra, P. Ingrami, and Stuart A. Kauffman. Coupled random boolean network forming an artificial tissue. in: Cellular automata. *International Conference on Cellular Automata, Springer, Berlin Heidelberg*, 4173:548–556, 2006.

[136] Edgar Knobloch, Michael Cross, and Henry Greenside. Pattern formation and dynamics in non equilibrium systems. *Cambridge University Press*, pages 567–572, 2010.

[137] Karel Culik and Aristid Lindenmayer. Parallel graph generating and graph recurrence for multi-cellular development. *International Journal of General Systems*, 3:53–66, 1976.

[138] David E. Golberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[139] John H. Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. *University of Michigan Press*, 1975.

[140] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press, 1996.

[141] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT press, Cambridge, Massachusetts, 1992.

[142] King-Sun Fu. Learning control systems—review and outlook. *IEEE transactions on pattern analysis and machine intelligence*, 8:327–342, 1986.

[143] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction.* Cambridge: MIT press, 1998.

[144] Alex B Novikoff. The concept of integrative levels and biology. *American Association for the Advancement of Science*, 101:209–215, 1945.

[145] Taras Kowaliw and Wolfgang Banzhaf. Mechanisms for complex systems engineering through artificial development. *Morphogenetic Engineering, Springer, Berlin Heidelberg*, pages 331–351, 2012.

[146] Hubert Anton Moser. *Systems Engineering, Systems Thinking, and Learning: A Case Study in Space Industry.* Springer, 2013.

[147] Kim A. Kastens and Cathryn A. Manduca. How geo-scientists think and learn. *EOS, Transactions, American Geophysical Union*, pages 265–266, 2009.

[148] James B Grimbleby. Automatic analogue network synthesis using genetic algorithms. *IEEE Proceedings-Circuits, Devices and Systems*, 3:319–323, 2000.

[149] D. Steinberg, N. Monmarche, M. Slimane, and G. Venturini. Discovery of cluster in numeric data by an hybridization of an ant colony with the kmeans algorithm. *International Journal of Automation*, 3:182–2015, 1999.

[150] Clare Bates Congdon. A comparison of genetic algorithm and other machine learning systems on a complex classification task from common disease research. *Ph.D. thesis, University of Michigan*, 1995.

[151] F. Della Croce, G. Menga, R. Tadei, M. Cavalotto, and L. Petri. Cellular control of manufacturing systems. *European Journal of Operations Research*, 69:498–509, 1993.

[152] Mikhail Prokopenko. *Advances in applied self-organizing systems*. Springer-Verlag London Limited, 2008.

[153] Soichiro Fujiki, Shinya Aoi, Takehisa Kohda, Kei Senda, and Kazuo Tsuchiya. Emergence of hysteresis in gait transition of a hexapod robot driven by nonlinear oscillators with phase resetting. *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 13:1638–1643, 2012.

[154] Yukie Nagai, Yuji Kawai, and Minoru Asada. Emergence of mirror neuron system: Immature vision leads to self-other correspondence. *Proceedings of the IEEE International Conference on Development and Learning (ICDL)*, 2:1–6, 2011.

[155] Georg Martius, Katja Fiedler, and J. Michael Herrmann. Structure from behavior in autonomous agents. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 858–862, 2008.

[156] Arnaud Revel and Pierre Andry. Emergence of structured interactions: From a theoretical model to pragmatic robotics. *Neural networks*, 2:116–125, 2009.

[157] Christian Scheier Rolf Pfeifer. *Understanding intelligence*. MIT press, 2001.

[158] Michail Maniadakis, Marc Wittmann, and Panos E. Trahanias. Time experiencing by robotic agents. *11th European Symposium on Artificial Neural Networks*, 2:429–434, 2011.

[159] Francisco Bellas, Richard J. Duro, Andres Faina, and Daniel Souto. Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. *IEEE Transaction on Autonomous Mental Development*, pages 340–354, 2010.

[160] Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, pages 173–202, 1999.

[161] Jeffrey C. Mogul. Emergent (mis)behavior vs. complex software systems. *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, 40:293–304, 2006.

[162] Fumio Hara and eds Rolf Pfeifer. *Morpho-functional machines: The new species: Designing embodied intelligence*. Springer Science and Business Media, 2003.

[163] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of artificial lifeforms. *Nature*, 406:974–978, 2000.

[164] Scott F. Gilbert. The morphogenesis of evolutionary developmental biology. *International Journal of Developmental Biology*, 47:467–477, 2003.

[165] Max Lungarella, Giorgio Metta, Rolf Pfeifer, and Giulio Sandini. Developmental robotics: A survey. *Connection Sciences*, 15:151–190, 2003.

[166] Yan Meng, Yuyang Zhang, and Yaochu Jin. Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model. *IEEE Computational Intelligence Magazine*, 6:43–54, 2011.

[167] Tsai, Tony Yu-Chen, Yoon Sup Choi, Wenzhe Ma, Joseph R. Pomerening, Chao Tang, and James E. Ferrell. Robust tunable biological oscillations from interlinked positive and negative feedback loops. *Science*, 321:126–129, 2008.

[168] Oros, Nicolas, Volker Steuber, Neil Davey, Lola Cañamero, and Rod Adams. Evolution of bilateral symmetry in agents controlled by spiking neural networks. *IEEE Symposium on Artificial Life*, 321:116–123, 2009.

[169] Max Lungarella and Giogio Metta. Beyond gazing, pointing, and reaching: A survey of developmental robotics. *Third International Workshop on Epigenetic Robotics*, pages 81–89, 2003.

[170] Rene Doursat, Hiroki Sayama, and eds Olivier Michel. *Morphogenetic engineering: toward programmable complex systems.* Springer, 2012.

[171] Guo, Hongliang, Yan Meng, and Yaochu Jin. A cellular mechanism for multi-robot construction via evolutionary multi-objective optimization of a gene regulatory network. *BioSystems*, 98:193–203, 2009.

[172] Yaochu Jin, Hongliang Guo, and Yan Meng. Robustness analysis and failure recovery of a bio-inspired self-organizing multi-robot system. *Proceedings of the IEEE International Conference on Self-Adaptive and Self-organizing Systems, IEEE Press*, 98:154–164, 2009.

[173] Yan Meng, Hongliang Guo, and Yaochu Jin. A morphogenetic approach to flexible and robust shape formation for swarm robotic systems. *Journal of Robotics and Autonomous Systems*, 61:25–38, 2013.

[174] Rehan O'Grady, Anders Lyhne Christensen, and Marco Dorigo. Swarmorph: Morphogenesis with self-assembling robots. *Morphogenetic Engineering, Springer Berlin Heidelberg*, pages 27–60, 2012.

[175] Chih-Han Yu, Kristina Haller, Donald Ingber, and Radhika Nagpal. A self-deformable modular robot inspired by cellular structure. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Academic Press, IEEE Press*, 98:3571–3578, 2008.

[176] Yan Meng, Y. Zheng, and Yaochu Jin. A morphogenetic approach to self-reconfigurable modular robots using a hybrid hierarchical gene regulatory network. *International Conference on the Synthesis and Simulation of Living Systems (ALIFE XII)*, pages 765–772, 2010.

[177] Pauline C. Haddow and Johan Hoye. Achieving a simple developmental model for 3d shapes: Are chemicals necessary? *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation GECCO'07*, pages 1013–1020, 2007.

[178] Douglas H. Erwin and Eric H. Davidson. The evolution of hierarchical gene regulatory networks. *Nature Reviews Genetics*, 10:141–148, 2009.

[179] Yaochu Jin, Lisa Schramm, and Bernhard Sendhoff. A gene regulatory model for the development of primitive nervous systems. *Processing of the International Conference on Neural Information, Springer, Berlin Heidelberg*, 10:48–55, 2008.

[180] Jordan Pollack, Hod Lipson, Pablo Funes, Sevan Ficici, and Gregory Hornby. Coevolutionary robotics. *Proceedings of the IEEE First NASA/DoD Workshop on Evolvable Hardware*, 10:208–216, 1999.

[181] Tim Taylor and Colm Massey. Recent developments in the evolution of morphologies and controllers for physically simulated creatures. *Artificial Life*, 7:77–87, 2001.

[182] Rolf Pfeifer and Alois Knoll. Intelligent and cognitive systems. *ERCIM News*, 64, 2006.

[183] Lisa Schramm, Yaochu Jin, and Bernhard Sendhoff. Emerged coupling of motor control and morphological development in evolution of multi-cellular animates. *European Conference on Artificial Life, Springer, Berlin Heidelberg*, pages 27–34, 2009.

[184] Frank Sengpiel and Peter C. Kind. The role of activity in development of the visual system. *Current Biology*, 12:818–826, 2002.

[185] Nicholas C. Spitzer. Electrical activity in early neuronal development. *Nature*, 444:707–712, 2006.

[186] Jean-Philippe Thivierge. How does non-random spontaneous activity contribute to brain development? *Neural Networks*, 22:901–912, 2009.

[187] S.N. Sivanandam and S.N. Deepa. *Introduction to genetic algorithms*. Springer Science and Business Media, 2007.

[188] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31:264–323, 1999.

[189] Jeremy Kubica, Arancha Casal, and Tad Hogg. Complex behaviors from local rules in modular self-reconfigurable robots. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 360–367, 2001.

[190] Behnam Salemi, Mark Moll, and Wei-Min Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, 2006.

[191] Gregory Chirikjian, Amit Pamecha, and Imme Ebert-Uphoff. Evaluating efficiency of self reconfiguration in a class of modular robots. *journal of Robotic Systems*, 13:317–338, 1996.

[192] Amit Pamecha and Gregory Chirikjian. A useful metric for modular robot motion planning. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 442–447, 1996.

[193] Marsette Vona Daniela Rus. A physical implementation of the self-reconfiguring crystalline robot. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1726–1733, 2000.

[194] Topon Kumar Paul and Hitoshi Iba. Genetic programming for classifying cancer data and controlling humanoid robots. *Genetic Programming Theory and Practice IV, Springer US*, pages 41–59, 2007.

[195] Till Steiner, Jens Trommler, Martin Brenn, Yaochu Jin, and Bernhard Sendhoff. Global shape with morphogen gradients and motile polarized cells. *IEEE Congress on Evolutionary Computation (CEC)*, pages 2225–2232, 2009.