NTNU

Norwegian University of
Science and Technology

Self-assembly Mechanisms for Evolutionary Robotics

Eirik Jakobsen
Christopher Jeffrey
Tannum

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Keith Downing, IDI
Co-supervisor: Kazi Shah Nawaz Ripon, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In recent study, an increasing amount of time has been spent researching complex systems due to the traditional Von Neumann architecture lacking sufficient efficiency. A specific complex system which has been growing in popularity is a system of swarm robots. Introducing evolutionary principles in such systems makes an attempt at collective computation, from simple elements, to resolve complex problems. To further add functionality to such systems, self-organizing techniques, such as self-assembly, have been introduced. Because of the lack of research literature within this paradigm, there are few comparisons of the different implementations of self-assembly mechanisms. This thesis outlines some mechanisms of self-assembly and explores the positive and negative implications these may have on evolutionary self-assembling robots. The results of the experiments conducted in this thesis show that self-assembly mechanisms such as the connection hardware and introducing local communication can influence the system's ability to self-assemble, and the behaviour of the self-assembled group.

Sammendrag

I nyere forskning har en økende innsats blitt brukt til å forske på komplekse systemer. Grunnen til dette, er et økende behov for å løse komplekse problemer der den tradisjonelle Von Neumann arkitekturen mangler tilstrekkelig effektivitet. Et spesifikt komplekst system som har vært i økende popularitet er et system av sverm roboter. Ved introdusering av evolusjonære prinsipper i slike systemer gjør man et forsøk på kollektiv beregning fra enkle elementer, for å løse komplekse problemer. For å legge til ytterligere funksjonalitet til slike systemer har selvorganiserende teknikker, for eksempel selvmontering, blitt introdusert. På grunn av mangel på forskningslitteraturen innen dette paradigmet, er det få sammenligninger av de forskjellige implementeringer av selvmonterende mekanismer. Denne avhandlingen beskriver noen mekanismer for selvmontering og utforsker de positive og negative konsekvenser disse kan ha på evolusjonære selvmonterende roboter. Resultatene av forsøkene utført i denne avhandlingen viser at selvmonteringsmekanismer, som for eksempel tilkoblingsmaskinvare og innføring av lokal kommunikasjon, kan påvirke systemets evne til å selvmontere, og oppførselen til den selvmonterte gruppen.

Preface

This master thesis is written at the Department of Computer and Information Science at NTNU during the spring semester of 2016. During their 8th semester, students are exposed to a number of different projects which are provided by the IDI faculty and professors at NTNU. These projects are open to interpretation by the students to enable their creativity. This thesis builds on a research project conducted during the autumn of 2015, which had the following project description:

"Self-assembly mechanisms are useful in collective robotics in order to allow a group of robots to cross obstacles that a single robot could not cross. Another use of such mechanism is the drag heavy objects by multiple robots. In the last years, multiple mechanisms have been proposed in the literature. The first task of the student will be to list these mechanisms find the relevant axes to classify the self-assembly mechanisms of the literature. The second task will be to compare these mechanisms. More specifically, an evolutionary algorithm will be selected for its ability to learn behaviours in groups of robots. Additionally, a task will be selected or designed for its proper level of challenge (that can be addressed thanks to self-assembly). Finally, the algorithm will be used on the task with the various mechanisms located, and comparisons will be made on the difficulties to evolve behaviours using self-assembly."

From the researched project during 2015, several interesting research scopes were discovered. An initial experiment was outlined during the project which was elaborated and continued throughout this thesis. Exploring the different mechanisms and behaviours of swarm-robotics became the foundation of this thesis.

Special thanks to:

- Kazi Shah Nawaz Ripon for extensive help in forming this report, supplementing theoretical foundation and imperative guidance.
- Jean-Marc Montanier for comprehensive help during the research project of fall 2015 which prompted the writing of this thesis.
- Pauline Catriona Haddow for supplying this project and supplying contact with excellent supervisors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every year the expansion and development of robots increases. We see robots being used in large military operations in the form of drones, as well as small robots in households that can vacuum your house while you are gone. A lot of this technology has only recently been developed, and it is expected that it will continue to grow and become a part of almost every aspect of society.

As a part of advancing robotics, a lot of the different functionality and behaviour of robots are being explored. These mechanisms include self-replicating machines, artificial intelligence, self-assembly, inter-robot communication, etc. In this study, we will be exploring a robot's ability to self-assemble.

Self-assembly is the autonomous organization of components into patterns or structures without human intervention [60]. We can observe self-assembling processes in nature at a molecular scale [26] as well as at a planetary scale in the form of solar systems; each of which presents many different configurations. There are several reasons to study self-assembly. One reason is, the cells in the human body self-assemble, so understanding this mechanism can help us understand life and the prerequisites for it. In chemistry, it has been discovered that polymers can self-assemble from monomers to form materials that can be used in a wide range of applications[12, 52]. In electronics, the self-assembly of nanoparticles can be used to create certain semiconductors such as solar panels[1]. In either case, understanding and controlling self-assembly is essential.

Self-assembly is also used as a practical strategy for making nanostructures as well as providing beneficial characteristics to the field of robotics. In a more general sense, self-assembly can improve the movement of robots by being able to overcome larger obstacles in the environment and having a large sensory field by grouping the sensors in a self-assembled structure.

According to [62], there are three main characteristics that self-reconfigurable systems benefit from. The first is versatility, systems can self-assemble into new structures based on the specific task it is to solve. For example, the system

may change from a rolling robot to a snake robot depending on the situation. The second characteristic is robustness. Since the robots are usually homogeneous, one can replace broken robots to repair the system. It is also possible to automate this process, leading to self-repair which makes the system more robust than other traditional systems. The last characteristic is the potentially low cost of developing a self-reconfigurable system. Constructing many of the same type of robot generally lowers the overall robot cost. In addition, there are many complex machines that can be built from a reconfigurable system that saves cost through reuse and generality. There are clear benefits of using self-assembly mechanisms in robotics. However, there have been many different approaches to try to achieve self-assembly in different systems of robots.

Self-assembly can form many different configurations based on the topology of the organized structure. In addition, self-assembly systems vary in their ability to reconfigure and disassemble depending on the challenges the environment provides. One class of self-assembly systems is stochastic reconfiguration. With stochastic reconfiguration, the robots do not have movement capabilities themselves. The robots are moved passively by the environment and bind to each other on random collisions[21]. With this form of reconfiguration, the reconfiguration time can only be guaranteed statistically[62]. [7] demonstrates how robots named "programmable parts" can self-organize through passive movements provided by the environment. Systems that use stochastic reconfiguration are most common at micro level environments, and only achieve self-assembly because of the influence of their environment. These characteristics do not conform to what we are trying to study, so this study will not delve further into this topic.

A crucial part of this study is to observe how self-assembly can transpire as an emergent behaviour through machine learning. A robot that can improve its performance based on past observations is said to use some form of machine learning. The main competing algorithms for machine learning in self-assembly robot systems, as of writing, are reinforcement learning and evolutionary algorithms. This study will look at machine learning and self-assembly from an evolutionary algorithms point of view.

This study covers how to implement machine learning in a system that takes advantage of self-assembly. In particular, how self-assembly can be used in conjunction with evolutionary robotics. Evolutionary robotics represents a way to automate the design of control systems for autonomous robots, using algorithms based on Darwinian evolution [56]. Evolutionary robotics is a field of research that lies under the field of artificial intelligence. The main purpose of evolutionary robotics is to develop autonomous robot controllers that solve a given task which is not directly programmed by a human. Evolutionary robotics takes advantage of selective reproduction based on how well the robots solve a certain task. The controller of a robot is most often represented with an *artificial neural network* where the parameters of the neural network are

set by an evolutionary algorithm.

There have been previous studies about using machine learning to promote self-assembly[56, 36, 31]. However, many of these studies use differing self-assembly mechanisms such as the architecture used by the self-assembly system, the actions that are performed by the robots to achieve an organized structure and the docking mechanism hardware. In addition, the studies have not compared the difficulty of environments in regards to the effect it can have on self-assembly. Another element this study aims to explore is adding a local communication module to give the robots the ability to communicate simple messages within a self-assembled structure.

The motivation of this research is to analyse if any of these mechanisms are superior in terms of promoting self-assembly with learned robot controllers and the effect it may have on the self-assembled structure's behaviour.

The experiment will be done in simulation with the Roborobo platform[10]. Roborobo is a light-weight multi-platform simulator for extensive robotics experiments, based on basic robotic hardware setup. The experiment is a simple predator/prey scenario, where the evolved robots(prey) can self-assemble to gain certain advantages over its predators. While creating the simulation, great care has been taken into making most of the parameters of the system configurable. This is done so that testing the different mechanisms is simple, such that we can compare how introducing different mechanisms affect the robot's ability to self-assemble and the behaviour of the robots.

1.1 Thesis objectives

The main focus of this study is to compare the different mechanisms that influence self-assembling robots that are governed by machine learning. Relevant mechanisms are the self-assembly architecture, the assembly protocol, and docking(assembly) mechanisms. The main objectives of this study includes:

- Implement and explain a simulation that uses various self-assembly mechanisms such as a self-assembly architecture and an assembly protocol.
- Compare and analyse the results of the self-assembly mechanism simulations.
- Based on the analysis, make a conclusion as to the advantage of using certain implementations of self-assembly mechanisms and the environment in which they reside.

1.2 Problem statement

The scope of this study covers the self-assembly mechanisms in a system of robots. The grand question this study is trying to contribute to is:

How do the elements present in a system of robots influence the emergence of self-assembly when the robots are given basic learning capabilities?

To get a better understanding of this field, this study will present methods and analysis to answer the following questions:

- In what way does an evolved assembly protocol influence the robots ability to achieve self-assembly?
- How does the connection port configuration, provided by the docking mechanism, influence the systems ability to self-assemble?
- How does changing the robot environment difficulty change the systems ability to self-assemble?
- How does introducing local communication between the robots influence the self-assembled system's behaviour?

1.3 Structure of report

The first chapter following the introduction is chapter 2 presenting the state of the art. It explores the field of self-assembly robotics as well as machine learning with an emphasis on evolutionary algorithms. The goal of this chapter is to present the self-assembly mechanisms which are used to categorize different self-assembly systems. In addition, machine learning is introduced with an in-depth look at off-line and on-line evolutionary algorithms. Chapter 3 contains the system description and implementation, including modifications which were made to the existing *roborobo* framework. Chapter 4 describes the results of the experiment and an analysis and reasoning for their given state. Chapter 5 concludes the study including personal thoughts about future work.

Chapter 2

Background

The purpose of this chapter is to introduce the background and main concepts required to describe and compare self-reconfiguring robotic systems. The beginning of this chapter includes a section of related work comparing similar projects and experiments. Following, this chapter reviews the state-of-the-art of self-assembly mechanisms. Self-assembly mechanisms includes the assembly architecture, protocol and different hardware mechanisms for real world application. This chapter also reviews any additional theory that is required for the implementation and analysis of this experiment such as machine learning and artificial neural networks.

2.1 Related work

Self-assembly in robotics is a relatively new field of research; however, there are still some papers which attempts to discuss and compare the different mechanisms in a self-assembly system. [37] mainly covers the potential of reconfigurable systems, as well as the challenges of self-reconfigurable robotics. Though this article does not conduct any experiment, it does touch upon some of the same issues that this report is trying to outline.

[63] is an article released in 2002 that tries to explain how modular robots can self-reconfigure to solve different tasks. In particular, this article looks at *PolyBot* and how it is able to reconfigure into different shapes. [63] also covers some of the benefits in general of having a self-reconfigurable system, as well as challenges that are still being explored.

[48] discusses the hardware design issues for self-reconfigurable robot systems, and presents two solutions developed in their laboratory. Additionally, the algorithmic challenges of planning reconfiguration from one shape to some desired shape is discussed.

[62] is the more complete and detailed of the articles presented here. In addition to the general benefits and challenges of modular self-reconfigurable

robot system, it also looks at the different architectures the system can take, application areas and history of the field. The article also includes an extensive list containing many self-reconfigurable modular systems and the type of architecture they have.

These papers outline what a modular self-reconfigurable system is and what the benefits and challenges of these systems are. However, there is no direct comparison between the mechanisms themselves. [62] introduces many different systems that work quite differently, but there is no conclusion based on if these systems promote self-assembly or not.

2.2 Complex systems

What separates complex systems from what we determine "simple systems" is that the behaviour of the system cannot be understood by simply looking at individual parts of the system. The way the parts act together and how they are connected is what gives rise to complex behaviour. This concept brings us to emergent behaviour. Emergent behaviour is a process where the system takes on certain patterns, regularities or form larger entities through the interconnections and interactions among simple parts that by themselves do not portray such properties. Further, this report will look at an emergent behaviour known as self-assembly.

In nature, one can find many examples of complex systems displaying self-organizing behaviour. The collectives range from just a few individuals to millions. Some examples are flocking behaviour in birds, fish schooling, and the complex societies seen among social insects(Figure 2.1).

(a) Flocking[4] (b) Schooling[44] (c) Termites[3]

Figure 2.1: Examples of self-organizing behaviour in nature.

Social insects such as termites build large structures featuring complex architectures. These examples show how a collective of simple agents, without a central controller, can accomplish tasks that each individual would be unable

to perform themselves. Together, the actions of these agents form a collective behaviour that can be considered intelligent.

These biological systems display massively parallel, decentralised computation. These systems stand in contrast to conventional computer systems with a central processing unit executing instructions serially. Research into self-assembling robot systems, therefore, take inspiration from biology, using methods such as evolutionary algorithms simulating Darwinian evolution, and artificial neural networks inspired by the way brains process data.

2.3 Self-assembly architectures

Self-reconfiguring robots can assemble into many different shapes, and there are many ways to do so. We define these properties as the self-assembly architecture. Each architecture presents different challenges and benefits. The architectures can in broad terms be categorized either as non-mobile- or mobile-architectures. The non-mobile architectures are characterized by the topologies of the structures they form, while mobile architectures are characterized by having more mobile robots with a higher degree of freedom. This section will describe and define some of the common assembly architectures used in previous projects.

2.3.1 Non-mobile Architecture

Typically these robots have little mobility themselves but can form complex structures capable of a wide range of behavior. One of the long-term visions for these kinds of robots is called "bucket of stuff"[62]. The idea is to have a collection of different robot modules which can be assembled on demand to solve various tasks. One way to classify non-mobile robots is based on their configuration topology. The main classes for configuration topologies are chain and lattice configurations.

Chain topology

The chain topology is a configuration where each robot is connected in a serial architecture. The robots can form many different shapes to represent creatures such as snakes, spiders or other configurations needed to complete its task. One of the challenges with this topology is coordinating movement and decisions as information has to be propagated serially. Examples of chain architecture based robots are Molecubes[64], Yamor[39], and Conro[11]. Yamor is depicted in figure 2.2.

Figure 2.2: Yamor robots configured as a snake[39].

Lattice topology

The Lattice architecture is a configuration where robots are arranged in a grid/lattice topology. The robots can then execute motion and control in parallel which provides a simpler configuration in comparison to the chain architecture. Examples of lattice architecture would be a cubic or hexagonal grid. Due to its simplicity, a lattice architecture is easier to scale. There have been more research groups working on this class of architecture due to its easier implementation [63].

Symmetry is a desirable property for robots that employ the lattice architecture because this makes reconfiguration into new positions in the grid easier. To maintain the symmetry property in three-dimensional space, each robot needs more connection points and actuators than in two-dimensional space[37]. This property complicates the design of the robots. Thus, building these robots with a high enough power/weight ratio is difficult. Examples of lattice architecture based robots which is able to perform self-reconfiguration are ATRON[9], Miche[19], and Catoms[28]. Atron is depicted in figure 2.3.

Figure 2.3: Atron assembled into a four wheeled car [9].

2.3.2 Mobile architecture

Mobile architectures are characterized by having autonomous robots which can self-assemble and reconfigure themselves. Each robot can operate independently or form larger structures when needed, by hooking up with other units. The structures formed may have different topologies and can coordinate their movements to form a larger virtual network[62]. One example of a mobile architecture is presented in [24]. In the experiment depicted in figure 2.4, a group of robots are programmed to move an object which requires cooperation by being too heavy for a single robot to move.

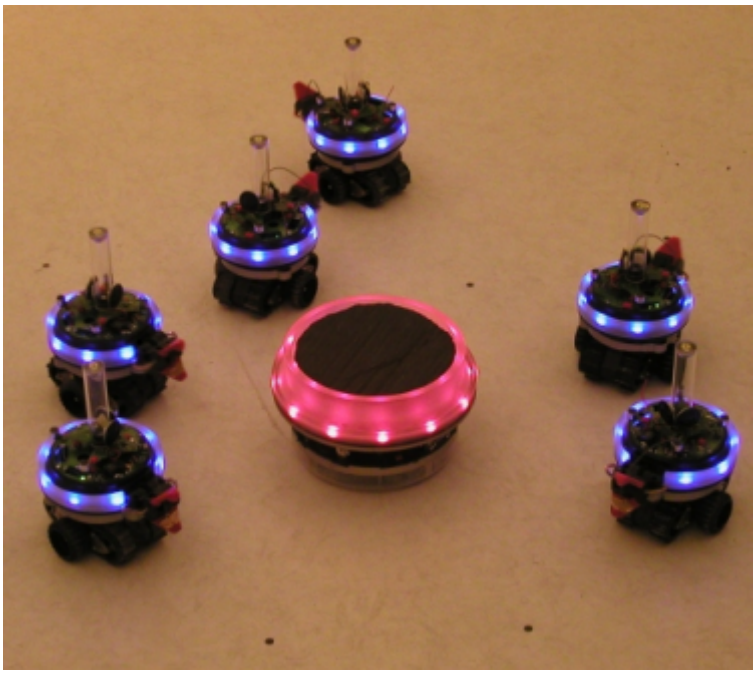


Figure 2.4: A group of swarmbots preparing to transport an object[21].

Since each robot in the architecture is autonomous one challenge is making the individual robots cooperate. Cooperation is especially difficult in cases where the optimal solution to a task involves cooperation, but the robots can complete the task individually. This challenge is demonstrated in [56] where the researchers were able to promote self-assembly to solve the task, but the self-assembly was only present in 2 out of 10 trials.

The high degree of mobility in each robot allows them to form a wide variety of topologies to solve tasks, but this also makes reconfiguration more difficult for large robot swarms. Because of this difficulty, there has been less research into mobile architectures.

Experiments conducted demonstrates robots solving simple tasks such as object transport[21], phototaxis[56], and energy collection[36, 58]. These experiments demonstrate that mobile robots are capable of solving tasks through

self-assembly. However, the tasks presented are trivial problems and do not reflect the complexity of real world problems. Before solving real world problems, future research needs to demonstrate a greater amount of robots cooperating. Additionally, robots that have the ability to take different roles when solving problems as well as cooperating when solving multiple problems presented by the environment.

2.4 Hardware mechanisms

Several robot systems that are capable of self-assembly have been developed over the years[23, 59, 9]. Even though some vary greatly in implementation, all these robots have some implementation of hardware which makes it possible to perform self-assembly. The desired assembly architecture influences the hardware available in each robot. However, some common properties can be identified.

Prerequisites for self-assembly

The robots must have some form of *movement* to reach each other. This movement can be provided by the hardware available or from external sources. Similar to movement, the robots require a *docking mechanism* to physically connect to other robots in order to self-assemble.

The requirements described can be considered as minimum requirements for self-assembly, but for more advanced behaviour, the robots need additional hardware utilities. If the robots are to be self-assembled without human intervention, they need a *discovery mechanism* to detect the presence of each other. To coordinate behaviour, the robots also need a *communication mechanism* capable of receiving and sending information. These requirements are very general and can be achieved in a wide variety of ways. Table 2.1 describes some methods to complete the discovery and communication requirements.

Table 2.1: Various hardware solutions to discovery and communication.

	Chain	Lattice
Discovery	IR[11]	IR[19]
Communication	1-wire bus[64], IR[11], Bluetooth[39]	IR[9, 19]

It is possible to simplify or ignore some of these requirements, but self-assembly is impossible without the docking mechanism. The choice of docking mechanism can be a deciding factor in the performance of a self-assembling robot.

2.4.1 Docking mechanisms

The docking mechanism is an essential component in self-assembly and has been solved in various ways in previous studies. For experiments with self-assembly in simulation, the mechanisms are often simplified since the focus of the experiments usually is the self-assembly behaviour. Therefore, it can be difficult to replicate the results on real hardware. The real world introduces noise, and the mechanisms designed must be able to cope with that imperfection. This is especially true in the context of machine learning as the resulting controllers are rarely optimal. The desired self-assembly architecture also plays a role in designing a docking mechanism.

Male-female connector

This form of connection mechanism is used by the ATRON[9] robots. The male modules are shaped like three hooks that can lock onto female connector bars. Each robot has eight connector modules and is arranged such that every second connector is male and every second is female. The connector modules allow each robot to connect to a maximum of eight other robots. A disadvantage with this type of mechanism is that the robots have to match the male to the female connectors before they can assemble, which eventually complicates the docking procedure.

Gripper and ring

The swarm-bot platform[21] uses this mechanism for self-assembly in the real world. Each robot is equipped with a gripper and is surrounded by a ring matching the gripper. Each robot can initiate one connection to other robots, but multiple robots can connect to each ring. There are several advantages with this mechanism; it allows some misalignment that makes docking simpler, and the ring makes it possible for several other robots to connect. Although many robots can connect to the ring, having only one gripper limits the possible configurations. This disadvantage can be mitigated by adding more grippers.

Magnetic docking mechanisms

These mechanisms typically use one or more magnets to form connections. One desirable property these mechanisms have is that they can tolerate some noise/misalignment since the magnet connectors can compensate by attracting each other. One problem with using magnets is that robots wishing to connect must make sure that they have the magnets in the right polarity to attract each other. Also, if electromagnets are used, they have to power them continuously to maintain the connection which increases power consumption.

Magnetic slip-ring This mechanism is used in simulation by [58]. The magnetic slip ring can be set to either a positive, negative, or neutral state. A connection will be made if at least on of the robots have their slip-ring set to positive, and neither of them has it set to negative. For simulation purposes, it is assumed that the connection immediately becomes rigid.

Surfaces with magnets The Molecubes[55] also use magnets to form connections. The Molecubes has flat connection surfaces with electromagnets which can be turned on, or off to create connections. Each cube has four connector surfaces, which means that each cube can connect to four other cubes.

Permanent magnets and springs [38] describes this mechanism. Unlike the two magnetic mechanisms previously described, this mechanism uses rare earth permanent magnets. The magnets are separated by two springs and a shape memory alloy(SMA) coil. The springs are designed to have a slightly lower force than the magnets when they are compressed. The mechanism detaches by applying an electrical current to the SMA, extending it to the memorized length.

2.5 Assembly protocols

In the context of swarm robotics and this paper, the term assembly protocol is defined as the sequence of actions that has to be performed before a successful self-assembly can occur. For pre-programmed robots, the protocols are determined by the programmer and known in advance. Evolutionary robotics are different in this regard since the controllers are evolved; therefore, the resulting assembly protocols may be unexpected. An overview of assembly protocols used in previous projects is presented in table 2.2. As for a specific model of an assembly protocol, we have two main approaches.

Table 2.2: An overview comparing past research projects of swarm robots based on the robot controller being evolved using an evolutionary algorithm, or pre-programmed for its intended usage.

Project	Pre-determined	Evolved	Reference
CONRO	X		[11]
Miche	X		[19]
SWARM-BOTS project	X	X	[24, 56]
N/A		X	[58]

2.5.1 Pre-determined assembly protocol

The traditional approach is to use an assembly protocol which is determined by the programmer in advance. This means that the assembly protocol is embedded in the controller software and does not change without reprogramming.

An example of a pre-determined protocol is found in [24]. In this experiment, the s-bots uses RGB LEDs to signal their position, and if other s-bots should dock with them. The lights are also used to determine if the self-assembly process is complete.

Using the protocol described, the s-bots are successfully self-assembled in 26 out of 30 trials. The advantage of using a pre-programmed assembly protocol is that it can be tailored to the problem. This often leads to good performance and a predictable outcome of the assembly once the system is deployed. It does, however, carry the disadvantage of being less general in the sense that the protocol can only be changed by implementing new software. Depending on the assembly protocol, it can also be quite complex which in turn, makes the implementation increasingly difficult.

2.5.2 Learned assembly protocol

In a machine learning setting, we can give the robots docking mechanisms, but leave the entire assembly protocol up to the learned controller. One might ask why machine learning, where the resulting controller rarely is optimal, is a good approach when pre-determined protocols can be developed instead. According to [49], there are three main reasons for why we would want a robot to learn. First, the programmer may not always have the ability to foresee all the situations the robot may encounter. The robot must then learn from its observations if it is to solve the probable new set of problems. Second, the programmer cannot predict all changes that happen over time. If we had some system that tried to anticipate tomorrow's weather, then it must be capable of analysing based on new sets of observation which were unknown at the time of system's design. And finally, the programmer may not even know the solution to the problem themselves, such that the system itself must find the solution. This last reason is especially interesting in accordance with self-assembly as we do not always know when it is appropriate to employ self-assembly for solving a certain problem.

An example of a type of learned assembly protocol is found in [56]. This example uses s-bots each of which uses an *artificial neural network* to control them. The neural network controls all of the input and output signals of the s-bot. This also includes the use of the gripper(docking mechanism) and the loudspeaker(produces sounds that can be used for communication between the robots). Using a simple genetic algorithm, they were able to evolve an assembly protocol in 2 out of 10 trials.

Even though an assembly protocol was evolved, the desired results of using the loudspeaker to signal position and performing self-assembly were inconclusive. The s-bots rather used exploits such as lining up against the wall of the experiment to perform a simpler self-assembly. The results also showed that the loudspeaker was not used optimally and was something they would like to explore in future research. This behaviour is one of the disadvantages of using machine learning to develop assembly protocols. First of all, it may not be able to create a protocol at all, and it is also likely that the end results are sub-optimal. A reason for this is that the environment, in which the robots are evolved, often impacts the type of assembly protocol that emerges.

2.6 Machine learning algorithms

We say that a robot is learning if it can improve its performance on tasks in the future based on past observations. Learning ranges from trivial matters such as writing a word to complex theories of the universe.

The learning algorithms commonly used for the automatic design of controllers can be divided into two main subdomains: reinforcement learning(RL) and evolutionary algorithms[8].

Reinforcement learning is a class of learning algorithms where the agent learns through trial and error, receiving positive and negative feedback for its actions[8]. The agent records the feedback received for performing a particular action in each state. Through this process, the agent discovers which action is optimal for a given state. Applying this form of learning to swarm robotics presents several challenges. One challenge is the fact that a swarm performs actions as a collective, but the learning is done on an individual level and does not reward global behaviour[8]. Additionally, traditional RL requires that the environment can be quantized into a finite number of discrete states[50, 8]. The number of possible interactions between the robots and the environment, and the noise in the real world leads to a high amount of states and makes it difficult to know how these are connected in advance[50]. Techniques such as using an approximator[8] to reduce the state space, and more intelligent reward mechanisms[8], can be used to reduce the impact of these challenges. Several projects[32, 2, 34] have successfully applied reinforcement learning in multi-agent systems. Given the challenges of RL, as the goals and ambition of projects increase, researchers are drawn towards techniques that resemble evolutionary algorithms[50].

Evolutionary algorithms attempt to find solutions to a problem by simulating the mechanisms of Darwinian evolution[56]. In these algorithms, the possible solutions are encoded into genomes and improved iteratively over multiple generations, where well-performing individuals reproduce more and spread their genome.

2.6.1 Evolutionary algorithms

Evolutionary algorithms is a class of algorithm in the category of *bio-inspired algorithms*[6]. Nature has shown that it is an excellent optimizer. When we examine features and phenomenon in nature, it tends to find the optimal strategy. The strategies employed are often very simple, but has great results. Bio-inspired algorithms take inspiration from nature and attempts to mimic the behaviour observed in nature. These algorithms can be categorized by the natural phenomenon they mimic. The three main categories are *evolution*, *swarm-based*, and *ecology*[6]. These algorithms are optimization algorithms and can, therefore, be used to automate the design of robot controllers. Specifically in the field of evolutionary robotics, the focus has been to employ evolutionary algorithms in the design of robot controllers.

There are many variants of evolutionary algorithms, the most common can be categorized by when the evolution takes place[16].

> In *off-line* evolution, the evolutionary development of robot controllers takes place before the robots start their real operation period. *On-line* evolution is the opposite, in that the evolutionary development of robot controllers takes place during the real operation period of the robots(although off-line evolution might precede on-line evolution as an educated initialization procedure)and is an ever-continuing process.[16]

Off-line

The details of how off-line evolutionary algorithms are implemented usually vary, however most implementations perform a variation of the same following steps[13], displayed in figure 2.5.

The first step of an off-line algorithm is to generate some population. Two main procedures can be used to generate the initial population. One can either generate the genomes for the robots completely random or by some function that applies bias. In the biological sense, a genome contains the configuration or blueprint for an individual. Before the genome can be used by the robots, the configuration must be translated into a robot controller. The translation procedure can for example consist of translating the genome into an *artificial neural network*, and uploading it to the robots. The next step after initialization is the robots acting in the environment. After a given set of time steps, the genome is evaluated by a fitness function which scores the robots based on their ability to solve the task. At the end of a generation, some of the genomes will be selected for reproduction based on their fitness. The algorithm then proceeds to mate genomes through crossover and applying mutation on the population. The process is repeated for this new set of genomes until the desired fitness is reached, or some processing time threshold is met. [56, 31]

describes examples of how this process can be applied to create controllers for self-assembling robots.

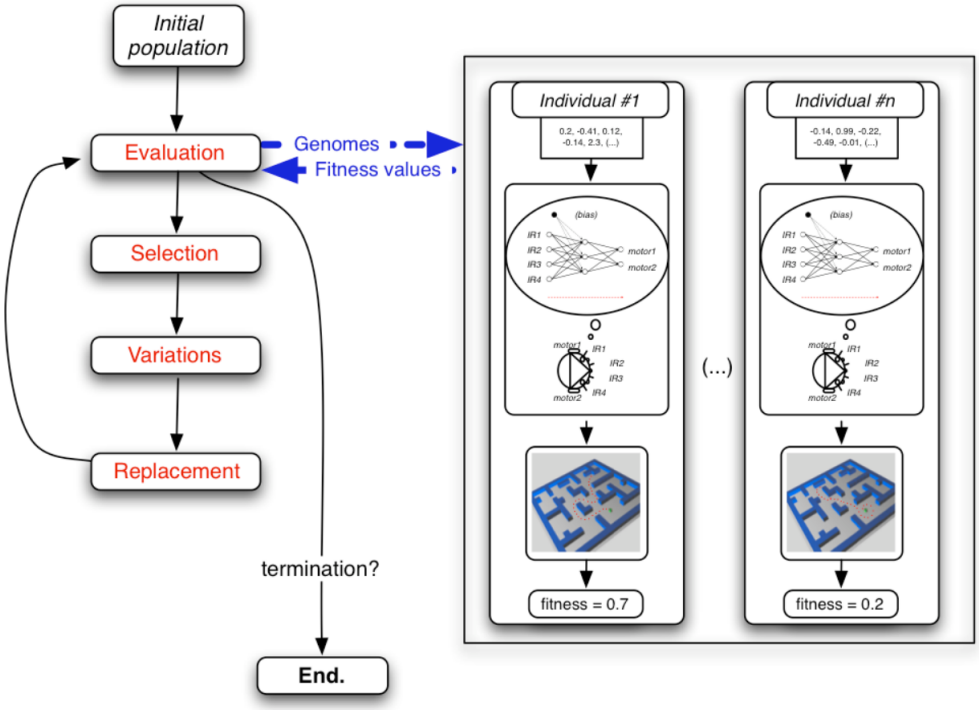


Figure 2.5: Operation of a typical EA[13]

The evolution process requires a lot of trials to evaluate the performance of each genome. Performing these trials on real hardware is, therefore, impractical, time-consuming, and possibly dangerous as the robots may damage themselves or the environment. The evaluation is therefore often performed in simulation[29]. Transferring the controllers evolved in simulation to real hardware has proven to be problematic because the controllers tend to be less efficient once transferred. This *reality gap* remains a critical issue in evolutionary robotics[29].

On-line

On-line evolutionary algorithms, in contrast to the off-line variants, does not share a well-defined series of steps. Several algorithms for this category has been proposed. Among them are mEDEA[36], MONEE[41], and ASiCo[57].

In mEDEA(minimal Environment driven Distributed Evolutionary Adaptation) each robot contains an active genome and a reservoir of received genomes. The robots broadcast their genome in a limited range to other robots. At a pre-defined number of time steps, the robot forgets its active genome and selects a new one at random from the reservoir. The robot becomes inactive if

it runs out of genomes. An inactive robot remains stationary for the following generation and receives genomes broadcasted from other robots. The result is that over time, ineffective genomes are removed from the population while the more successful genomes are able to spread.

MONEE(Multi-Objective and Open-Ended Evolution) can be considered an adaptation of mEDEA. In MONEE the robot has a life cycle consisting of two phases with a fixed duration. The first phase is the *life phase*. In the *life phase*, the robot performs tasks such moving or foraging. When the lifetime of the robot is reached, it enters a rebirth phase and becomes an "egg". The egg is stationary and receives genomes from other robots that are currently in their *life phase*. For the egg to select one genome out of the ones received from the other robots, MONEE introduces the concept of *parental investment*. During the *life phase* the robots receive credits for performing tasks in the environment. When a robot passes its genome to an egg, it passes the number of credits earned. The egg then uses the number of credits to compare and select a new genome for rebirth. If there are multiple tasks defined, the egg uses an exchange rate between the tasks based on their difficulty to compare them. The exchange rate ensures that genomes who specialize in difficult tasks can compete with genomes that perform easier tasks. This system allows genomes to specialize in different tasks and therefore makes MONEE well suited for multi-objective tasks.

ASiCo(Asynchronous Situated Co-evolution) is a co-evolutionary bio-inspired algorithm that in contrast to MONEE and mEDEA uses a fitness function to decide if a robot should reproduce. Like other on-line algorithms, multiple genomes are present among the robots even though this algorithm carries the characteristic of a fitness function from off-line algorithms. The ASiCo algorithm works by having the robots act out some scenario and uses energy gain or loss as a performance gauge. Differing from off-line algorithms, ASiCo continuously evaluates the different robots' fitness asynchronously. If a death condition is met, then the robots are either removed from the system or substituted. If however a situated mating condition is met, then the candidate is evaluated by its fitness. If the candidate is acceptable, then it is allowed to reproduce. In the case where the fitness is not acceptable, it simply continues to live on in the system.

2.7 Artificial neural networks

Computers excel at performing tasks that can be described as a series well-defined steps, and can at these tasks easily outperform humans in doing so. Tasks that humans find easy, such as understating speech or recognizing faces has proven to be difficult to describe a series of unambiguous operations in the conventional computation model. Biological neural networks, such as brains,

have evolved to be excellent at pattern recognition and data classification. Artificial neural networks(ANN) take inspiration from how biological neural networks process data.

ANN's consist of interconnected processing units(neurons) which work in parallel to solve a specific problem. The neurons employ a simplified model of how a biological neuron works. The operation of an artificial neuron consists of two main components, depicted in figure 2.6.

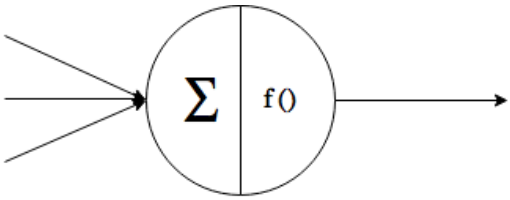


Figure 2.6: Model of an artificial neuron.

First, the neuron sums the action potential from the preceding neurons. The activation function is then applied to the sum which produces a new action potential as the output. The role of the activation function is to transform the action potential in the node to an output that can be passed to the succeeding nodes. These functions often have a thresholding effect where the neuron outputs a low potential until a threshold is reached. A common activation function is the logistics equation(2.1).

$$o_i = \frac{1}{(1 + e^{-v})} \tag{2.1}$$

The ouput of node i with internal action potential v.

The connection strength between neurons is simulated by introducing weights. A neuron with a higher weight on one of its connections will be further influenced by that particular connection. Learning can then be performed by modifying the weights on the connections in the ANN.

The neurons in ANN's are organized into multiple layers, where each neuron is connected to the neurons in the preceding layers. Typically there is an input layer, an output layer, and zero or more hidden layers between them.

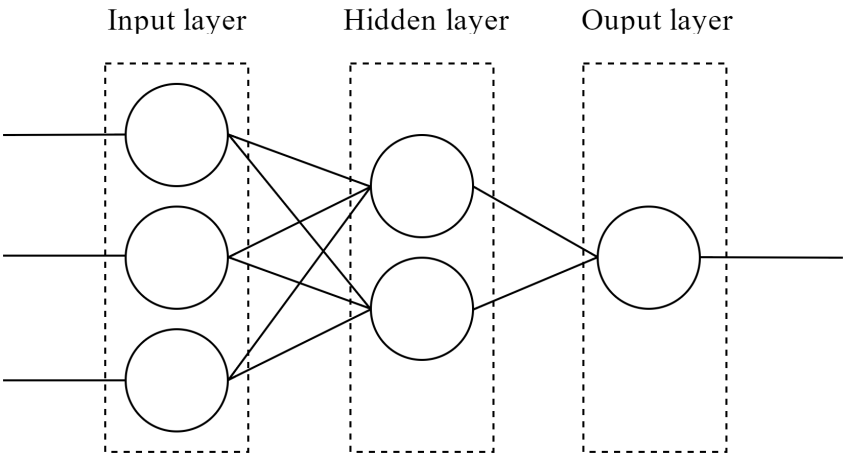


Figure 2.7: A feed forward network.

Figure 2.7 depicts a simple feed-forward network with one hidden layer. The inputs are propagated from the input layer, on the left, to the output layer. More complex network topologies introduce connections within the layers and connections back to preceding layers.

Creating efficient robot controllers can be difficult as the number of sensors and actuators grow. The designers may know what desired behavior for the robots is, but expressing this in the conventional computing model as a series of instructions can be difficult. Instead, with machine learning an ANN can learn from examples of desired behavior provided by the controller designers. This approach allows designers to create the desired controllers by providing examples instead of knowing the exact solution.

Neural networks have been applied to design robot controllers in several[56, 36, 9] self-assembling robot systems. These robot systems all use different approaches in how neural networks are applied to create the controller. [36] uses a simple feed forward network, while [56] uses a CTRNN, and finally [9] uses multiple neural networks which decide different parts of the behavior. This shows that neural networks are a versatile tool for designing robot controllers for self-assembling robot systems.

Chapter 3

Methodology

For this study, an experiment was implemented using the *roborobo* simulator[10]. Even though *roborobo* contains some functionality for running robot simulations, a lot of modifications were needed for the simulator to fit the needs of this experiment. This chapter will include a description of *roborobo*, the modifications that were made to the existing software, the experiment in general and the motivation for its implementation.

3.1 Experiment motivation

The introduction lists several research questions that this project is aiming to contribute. To not trivialise the environment and the interaction between the robots, care was taken when designing this experiment. The thought behind the experiment is to see collective behaviour using an evolutionary algorithm. Caution was taken to implement an unbiased system in regards to the influence it imposes on the algorithm to achieve correct and analysable results. For example, the evaluation function in section 3.7.6 is quite simple and does not add any guiding parameters to promote self-assembly.

One of the questions concerns how collective decision-making influences the emergence of self-assembly. Swarm behaviour found in nature displays how simple agents collectively can make a decision. The goal of this experiment is, therefore, to design a similar environment to promote this form of collective behaviour. In this environment, the robots have to form swarms for survival, and collectively decide if they should self-assemble or attempt an escape.

By design, the robots are given a local communication module that can be used to send an array of floating point numbers to its neighbours in the self-assembly structure(see section 3.5.3). The influence of the communication module and the values that the communication packets take are entirely governed by the neural network(see section 3.6).

The system has been made to be highly configurable. The focus of these

configurations are the evolutionary algorithm parameters as well as general environmental setup. By varying the parameters of the experiment, it is possible to observe and analyse the problem statements which were discussed in chapter 1. Depending on the configuration used when running the simulation, it should be possible to see how different assembly protocols and behaviours emerge, and how the self-assembly behaviour of the robots differ from similar trials and different configurations. Chapter 4 portrays the results and analysis of this experiment.

3.2 Experiment setup

For this experiment a micro-organism environment is imagined, where the organisms find themselves in some liquid pool. The micro-organisms are the agents in our system and are represented as robots. The main functions of the robots are movement, foraging and self-assembly.

The robots find themselves in a pond. In the pond there are nutrients which the robots can feed on to replenish their energy. At the same time the pond also contains larger predator organisms who intend to prey on the robots. It is imagined that the robots can assemble to form larger structures. The robots can protect themselves from the predator organisms by forming a larger structure which the predator is unable to eat. When the robots are assembled they may eat the larger predator to replenish energy as well. To gain an advantage the robots must form a large enough structure to prevent them from being consumed by the predators. Maintaining the assembled structure cost energy, which gives the robots an incentive to disassemble once the structure is no longer required.

3.3 Environment

The imagined pond environment can be represented as a rectangular box(figure 3.1), containing the robots, nutrients and predators. Initially, the environment is populated with nutrients, robots, and predators placed at random positions in the environment. When a nutrient is consumed, a new one is placed at a random position in the environment.

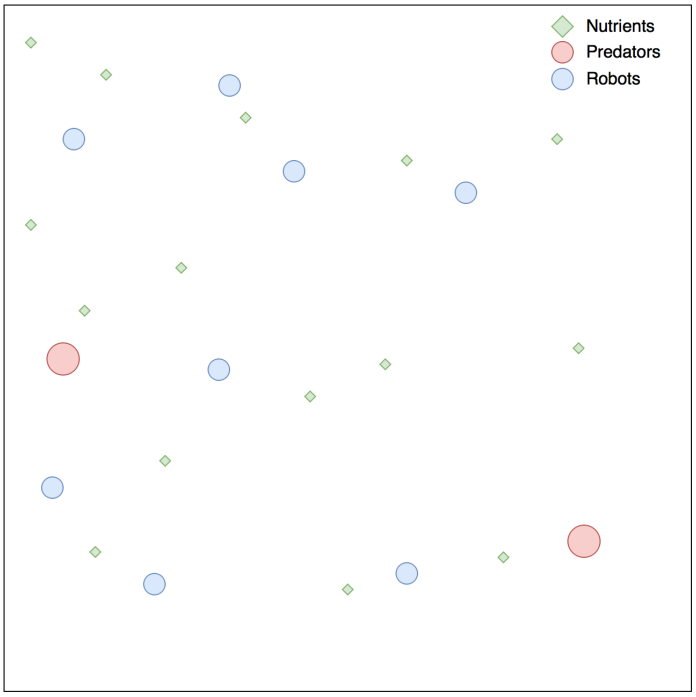


Figure 3.1: Initial configuration of the environment.

3.4 Roborobo overview

The programming interface for roborobo is divided into three components(*World observer, Agent observers, Agent controllers*)[10]. Each robot in the simulation receives an instance of the *agent controller* and the *agent observer*. The philosophy of the roborobo framework is that the programmer should implement the robot behaviour in the *agent controller*, and use the *agent observer* and *world observer* to access states about the given agent and the world. This relationship is visualised in figure 3.2.

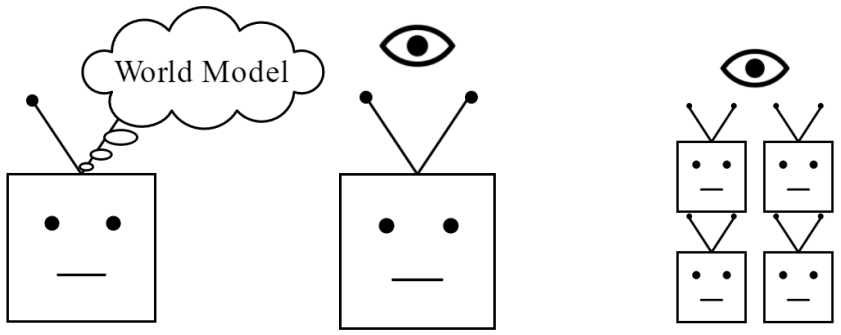


Figure 3.2: The relationship between a) the agent controller, b) the agent observer, and c) the world observer.

Each robot in the simulation also receives a component called the *world model*. The world model contains an agent specific representation of the outside world. The model includes states such as sensors readings and current velocity.

The observers can be used to perform tasks that are not directly related to the robot behaviour. In each simulation step the observers are run before any of the *agent controllers* are run. This means that the observers have a stable snapshot of the world between each update. The observers are therefore useful for performing tasks that should be carried out between each update of the world. Examples of useful applications of the observers are updating the agent's world models, monitoring, logging, computing fitness, and managing evolutionary algorithms. In the standard case, understanding the components described in this section is sufficient to perform a wide variety of simulations.

The depicted experiment in this chapter is not realizable with the basic roborobo framework. The necessary modifications to the framework are described in section 3.5.

3.5 Roborobo modifications

As mentioned in section 3.4, roborobo does not support self-assembly out of the box. Self-assembly support, therefore, requires adding additional abstractions to roborobo. This section includes the different modifications that were made to the roborobo framework to support self-assembly, local communication, and a configurable system.

3.5.1 Robot Groups

The high-level abstraction that encapsulates the behaviour for connected robots is called a *Robot group*. All robots that are capable of self-assembly are robot groups. Single robots that are not connected are simply robot groups with

one member. The responsibilities of the robot group are to take care of group movement, handling collisions, connecting and disconnecting.

Movement

In the roborobo framework, the robot's movement is decided by the robot controller by requesting a certain angular and translational velocity. The movement of a group is decided by first converting the desired direction and velocity into a translation vector.

$$\vec{v_{xy}} = v \cdot [\cos(\theta), sin(\theta)] \tag{3.1}$$

The combined movement of the group is then decided by averaging the translation vectors from each member in the group.

$$\vec{v_t} = \frac{1}{n} \sum_{i=0}^{n} \vec{v_i} \tag{3.2}$$

Once the translation vector for the group is computed, it can be applied to each member of the group by converting it back to the format of direction and velocity that roborobo uses.

Collisions

Roborobo already performs collision detection for robots, but in robot groups, some additional logic is required. The collision behaviour for robots is that if they collide with a solid object, they stop. This behaviour is a problem for groups of more than one robot because if one robot collides it may get left behind by the rest of the group. This issue is solved by backing up the position of each robot in a group before applying the computed translation. If a robot in the group collides with something then the robots in the group are reverted to their original position.

Connections

The connections between robots in a group can be considered as a graph, where the robots are nodes and connections are edges. Connecting robots can then be treated as simply adding an edge between the two nodes representing the robots. Similar to connecting, disconnecting consists of removing the edge between the nodes representing the robots. In the case that a robot has multiple connections in the group extra care has to be taken because removing one edge may split the graph into two smaller sub-graphs. If this occurs, the two sub-graphs must now be treated as two new separate groups. It can be determined if a robot can simply be disconnected, or if we have to split the group by finding out if there is a cycle that leads back to the disconnecting robot.

The existence of a cycle is determined by first removing the edge representing the connection, and then performing a depth first search.

3.5.2 Docking mechanism

As described in section 2.4.1, there exists a wide variety of docking mechanisms used in earlier projects. The docking mechanism implemented is therefore highly configurable, to support a wide variety of different mechanisms. The configurable properties are as following: the number of ports, the placement of ports, and different genders for the ports.

Connection validation

The robots can attempt a connection at any time; this requires a procedure to make sure the attempted connections are valid. Three requirements have to be met before a connection can be established between robots. First, the ports have to be spatially sound. Here, the distance between the ports has to be less than a threshold value.

$$|\vec{p_1} - \vec{p_2}| < \epsilon \tag{3.3}$$

Next, the position of the ports has to be geometrically sound. Here, the angle between the connection ports has to fit within a threshold range.

$$180 - \epsilon < |\theta_1 - \theta_2| < 180 + \epsilon \tag{3.4}$$

Finally, the gender of the ports must be compatible, meaning one of them is female while the other is male or at least one of the connection ports being universal.

3.5.3 Local communication

The robots are equipped with a communication module that allows local communication with their connected neighbours. The message format is very simple; the robots can only broadcast message packets of floating point numbers. The size of the packets is equal to the number of connection ports. At the receiving end, the packets sent by neighbours are aggregated into a single packet. The aggregation is performed by adding the components of each message packet with the corresponding components from the other message packets. Figure 3.3 illustrates this process.

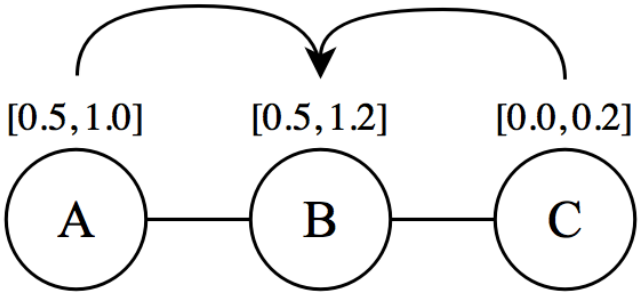


Figure 3.3: Robot B aggregating the messages received from robot A, and C.

In the robot controller, the messages from the communication module are fed into dedicated message input neurons, and the value of the message output neurons are broadcast to the neighbours.

3.5.4 Predators

A crucial part of the environment in the simulation are the predator robots. These robots are not under the influence of the evolutionary algorithm and rather uses a much simpler pre-programmed controller. The predator robots have two basic actions that can affect the system: the predators can either eat prey or explore the environment.

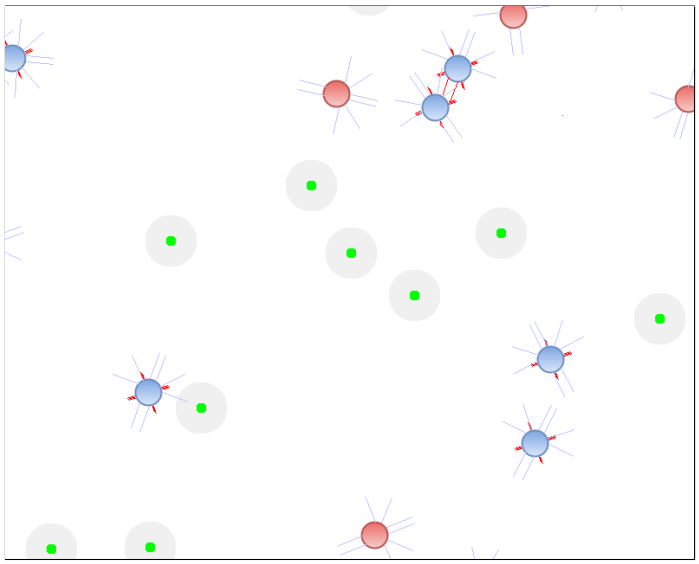


Figure 3.4: A screenshot of a simulation. The predators are marked with red and the robots guided by the evolutionary algorithm are marked with blue

As seen in figure 3.4, the predators also uses six sensors which they use to decide if they are colliding with an object and what that object is. Concerning exploration, the predator controller uses a simple object avoidance tactic. If

the predator robots sense an object on any of its side sensors, it will change its rotational velocity to avoid this object. In practice, this means that if the predator senses an object on the right, it will steer left, and vice versa. A slight random rotational velocity is added such that its movement becomes more dynamic in the case where there is no collision. This object avoidance behaviour is always employed as long as it does not sense a robot(prey). In this case, it will have the opposite behaviour and turn towards the object, as it intends to consume the prey.

In addition to movement, the predators can also consume a robot. Predators consume a robot when there is a collision between a normal robot and a predator robot. If a robot is consumed, it is removed from the environment and robot's lifetime is recorded for use in the fitness function of the evolutionary algorithm. For a predator to be able to eat a robot, the robot must not be part of a robot group of a size less than some configurable threshold. The default threshold is set to two for the simulations run in this experiment. The reason for this is to give the regular robots some advantage of self-assembling in hopes of potentially evolving this behaviour.

3.5.5 Energy drain

In addition to the environmental hazards of predators, the robots also face the issue with having a limited energy supply. If a robot loses all if its energy, then it will die and be removed from the environment. Five main configurations handle the energy parameters of the simulation:

- *gEnergyMax*, is a constant which represents the maximum amount of energy the robot can hold. If a robot picks up more energy from an energy source and the total energy exceeded *gEnergyMax*, then the total energy will simply be *gEnergyMax*.
- *gEnergyInit*, is a configurable parameter which sets the initial energy of a robot when spawned into the environment.
- *gEnergyItemDefaultInit*, is the store of energy an energy item in the environment holds.
- *gPassiveEnergyDrain*, is the leak value of energy of a single robot. At each time step, a robot loses *gPassiveEnergyDrain* amount of energy.
- The final energy parameter is *gConnectionEnergyDrain*, which is an addition drain for being self-assembled with another robot.

The reason for adding an energy hazard for the robots in this experiment is to give them a non-trivial self-assembling scenario. If predators were the

only environmental obstacle, then the robots would most likely develop a self-assembly strategy and hold this construction for the remaining time of the simulation. It is intended that the robots would have an environment which allowed strategies such as dissembling the self-assembly structure to evolve or perhaps not even self-assemble at all. This reasoning is also the justification for giving the robots additional energy drain when assembled.

3.6 CTRNN

For this project, an artificial neural network that has gained a lot a popularity in regards to robotics known as Continuous-Time Recurrent Neural Network (CTRNN) was implemented. The CTRNN was researched and developed by Randall Beer[5] and adds two new properties to the standard artificial neural network. The properties that are introduced in CTRNN is a time constant and a gain constant.

As most neural networks, the simple integration of the inputs from all neighbouring neurons are added giving us the first standard equation for neural networks:

$$s_i = \sum_{j=1}^{n} o_j w_{i,j} + I_i \tag{3.5}$$

Here, o_j represents the output(after activation) of neuron j, $w_{i,j}$ is the weight from neuron j to neuron i and I_i is the sum of all the external inputs to node i.

$$\frac{dy_i}{dt} = \frac{1}{\tau_i}[-y_i + s_i + \theta_i] \tag{3.6}$$

In order to preserve the previous state of the ANN, the internal state is stored. y_i denotes the internal state of neuron i. To derive the next internal state, Beer computes $\frac{dy_i}{dt}$ as a combination of the following inputs and the current internal state of the node, where a time constant τ_i decides the rate of drain. θ_i is a term added for a neuron-specific bias. It is only added here for mathematical soundness as it is simply added as a bias node during implementation and hence incorporated in s_i. The time constant τ_i is what gives CTRNNs the ability to produce rich functionality and convincingly sophisticated cognition. If τ_i has a low value, then we will have a high degree of drain and hence having its new input dominate the next state. However, if τ_i has a high value, then we have a higher degree of memory because its previous state will dominate the next state.

$$o_i = \frac{1}{1 + e^{-g_i y_i}} \tag{3.7}$$

To convert the internal state to an output o_i, Beer typically uses a sigmoidal activation function. A sigmoidal activation function is typical in other neural networks as well, but Beer also employs a gain term g_i, to influence the activation of the neuron.

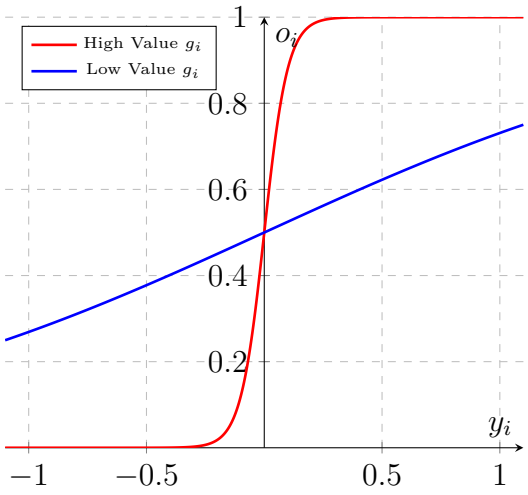


Figure 3.5: The difference in the activation function with respect to y_i based on a low and high value of g_i

As seen in figure 3.5, the value of the gain parameter can influence the activation function in such a way that it can be a smooth, almost linear, function or it can take on the behaviour of an activation function resembling a switch. Having the ability to change the activation function on a neuron to neuron basis, is the second reason(the first being time constants) CTRNNs can give rise to complex and rich behaviour.

3.6.1 Topologies

In the robot controller, the neural network is responsible for making decisions. The role of the robot controller is to forward inputs to the input layer of the neural network and interpret the output. The information that the neural network is provided with is as follows: sensor readings, the connection status of each docking port, messages from the communication module, and finally the current energy level of the robot. Once the inputs are processed the robot controller reads the output layer to decide the values for the motor functions, control of the connection ports, and the message to be sent via the communication module. The number of hidden layers and the number of neurons in each of them is configurable at runtime.

The robots have to be able to detect, and distinguish between predators, other robots, food, and the environment. The robots are equipped with six sensors to achieve this. To represent this two different input topologies for the sensors have been implemented.

Dense

The dense topology uses 24 nodes to represent the inputs where there is one node per sensor for each of the four different objects that can be detected. With this configuration, the robot can distinguish between multiple types of objects at the same time. The motivation for designing another input topology is the concern that by using 24 nodes the search space for the evolutionary algorithm may become too large to find a good solution within a reasonable time.

Sparse

The sparse topology is a compromise between the number of nodes and the robots ability to distinguish between multiple objects at the same time. Here the input layer has six nodes for each of the sensors, and four additional nodes that indicate whether a particular type of object is currently detected by one of the sensors. This modification brings the number of input nodes for the sensors down to 10, instead of 24. With this configuration, the robot can still detect nearby robots, but it can not differentiate between multiple types of objects at the same time.

3.7 Evolutionary algorithm

The following section describes the implementation of the evolutionary algorithm for this project. In addition, in some of the sections, multiple implementations are mentioned. We see an occurrence of this in section 3.7.3 where both random and incremental mutation operators being explained. The reason multiple implementations were done is due to the initial implementation yielding poor results and hence there was made an attempt at improving the results.

3.7.1 Genotype

The genotype contains the weights, gains and time constants for the neural network used in the robot controller. During run-time, the genotype size is fixed and depends on the size of the neural network. The size of the neural network is configurable (topology and size of each layer) in the simulation configuration file, and hence, the genotype can change size depending on the simulation run. The structure of the genotype is an array of double precision

floating point numbers in the range $[-1, 1]$. Each number in the genotype represents a specific weight, gain, or time constant in the neural network. Before the genotype can be used to configure the neural network the values are mapped into the suitable range for each type of parameter(see figure 3.6). The selected range, $[-1, 1]$ used for the values in the genotype is therefore arbitrary and holds no particular significance.

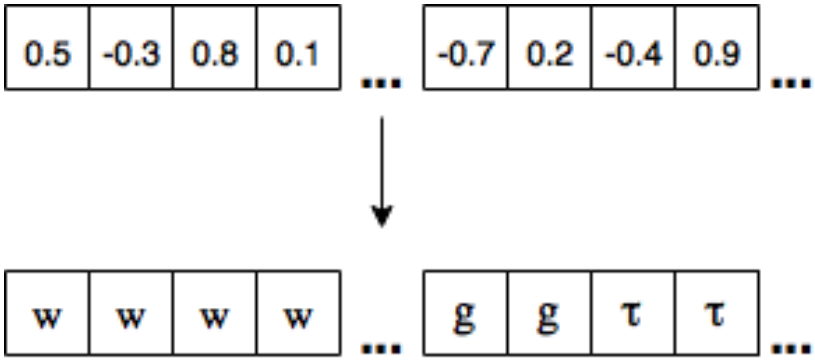


Figure 3.6: Mapping the genotype into weights, gains, and time constants.

3.7.2 Initialization

As seen in theory, there are two ways to initialize a population. It can be done completely random, or there can be a more particular initialization with bias. In the case of this experiment, a purely random initialization was implemented. The reason for this is that there is no trivial functionality that these robots should have for it to reach the most optimal solution.

Since this experiment is about researching self-assembly, one might think that initializing our robots with a behaviour to promote this, could be beneficial. There are two main issues with an approach like this. First, self-assembly in this complex system is an emergent behaviour(see ch. 2.2). This means that there is no trivial configuration of the neural network that gives rise to self-assembly. It is rather, emergent from the robots' behaviour and their connections with each other in the environment. As explained in chapter 2, there are mechanisms such as the assembly protocol, the assembly architecture, and also the hardware mechanisms used. None of these mechanisms can simply be coded into a neural network.

The second issue with implementing a bias towards self-assembly is that this might not be the best survival strategy in this experiment. Even though the experiment is designed to give the robots a slight benefit from self-assembling, this may not be the case in reality. Due to these two reasons, it was decided to have a strictly random initialization.

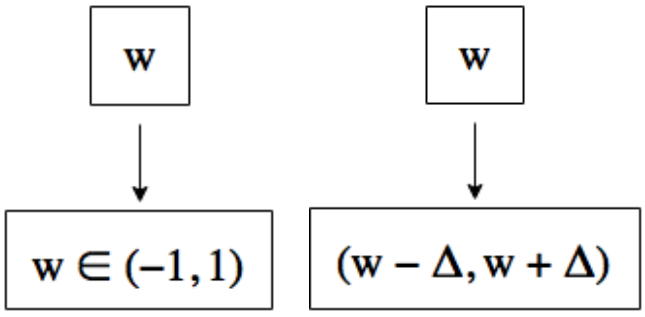
3.7.3 Mutation operators

As seen in section 2.6.1, the proper step after mating genotypes is applying a mutation to the resulting genome. Mutation is done using a mutation operator. During the implementation of the experiment, two different mutation operators were implemented, the random mutation operator and the incremental mutation operator.

During the first iteration of implementation, a standard random mutation operator was implemented. As explained in 3.7.1, the genotype is modelled using doubles, so a random mutation operator for this implementation simply re-rolls the selected double, x, to a new double where $x \in [-1.0, 1.0]$. The number of weights to be mutated is configurable and is represented as some percentage, *mutation rate*.

Incremental

During initial trials, it was observed that the impact of completely changing the value of a weight in the neural network could potentially drastically change the behaviour of the robots. The change in behaviour yielded poor results of the evolutionary algorithm because of constant destabilisation of the emergent behaviours. The results from the preliminary trials caused an incremental mutation operator to be implemented. The incremental mutation operator gave better results than the random mutation operator, so it was decided that the random mutation operator should be removed from the system. The incremental mutation operator works in a similar manner as the random operator, where a weight is chosen for mutation at some percentage, but instead of completely re-rolling its value, it only differs from its original value by a certain threshold. The incremental mutation still causes a mutation to occur, but it will not be as drastic, yielding smoother results. The difference is depicted in figure 3.7.



(a) Random mutation operator (b) Incremental mutation operator

Figure 3.7: Displays the mutation strategies for the random and incremental mutation operators.

3.7.4 Selection mechanism

The selection step of an evolutionary algorithm is responsible for selecting which individuals should become parents. In other words, it applies selection pressure. If the selection pressure is too high, the algorithm may converge prematurely, and if it is too low, the search may take more time than necessary. For the experiment, two different selection mechanisms have been implemented.

The first selection scheme that was implemented in this project was *proportional selection*. In this selection scheme individuals are selected for reproduction with a probability proportional to their fitness compared to the total fitness of the population. It is common to apply a scaling function on the fitness in the population to adjust selection pressure. A scaling function that is often used with proportional selection is *Sigma scaling*[20], which was also initially used in this experiment:

$$S(f, \overline{f}, \sigma, s) = s + \frac{f - \overline{f}}{2\sigma} \tag{3.8}$$

where f is the fitness of an individual, $\overline{f}$ is the mean fitness of the generation, σ is the fitness variance, and s is a scaling factor that can be used to adjust the selection pressure.

Sigma scaling modifies the selection pressure introduced in the raw fitness by using the populations fitness variance as a scaling factor. The scaling has the effect of dampening the selection pressure when there are a few individuals with an exceedingly higher fitness than the rest and increasing the selection pressure when the population has a low variance.

The values obtained from the scaling can now be used to select parents by sampling the distribution. *Roulette wheel selection*(RWS)[20] is one such method. RWS can be visualized as giving each potential parent a sector with size relative to the scaled fitness in the roulette wheel(figure 3.8). The parents are then selected by spinning the wheel until the desired number of parents are picked.

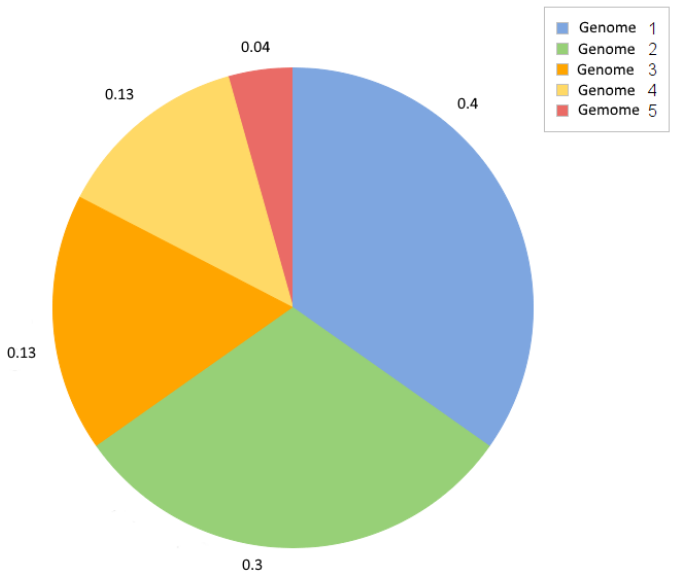


Figure 3.8: A roulette wheel where the size of each sector represents the probability of picking a particular parent.

The roulette wheel implementation gave less compelling results than expected, which caused tournament selection to be implemented which gave better results and caused the roulette wheel selection mechanism to be removed from the system.

Tournament selection

In contrast to proportional selection the individuals do not compete with the entire population but instead, compete within groups selected at random. Tournament selection performs parent selection by picking individuals at random into a group. From the group the fittest individual is selected as the parent. This process is repeated until the desired number of parents have been selected. The group size varies between implementations, but typical implementations compare two[20] individuals at a time, depicted in figure 3.9

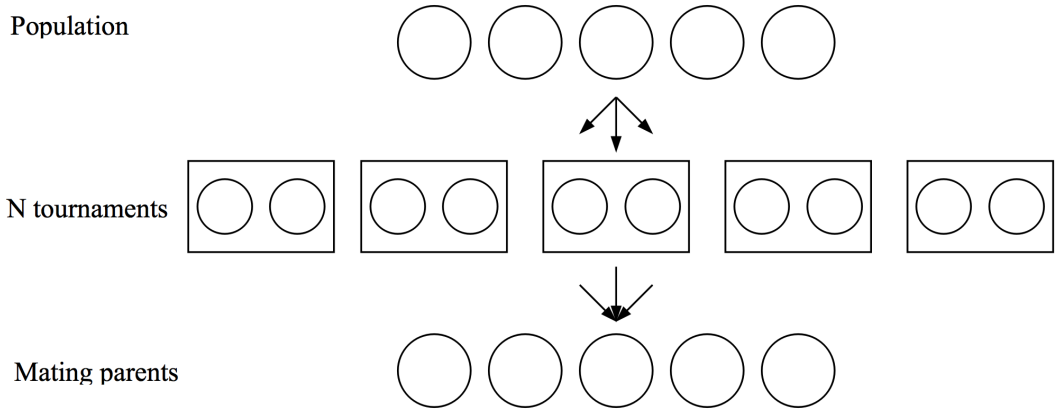


Figure 3.9: Selecting parents using a binary tournament.

The described process introduces a high selection pressure since it selects the fittest individual from each group. It is common to include an acceptance threshold, t, into the selection to modify the selection pressure[20]. Each time an individual is to be selected a random number $r \in (0, 1)$, is generated. If $r < t$, then the fittest individual is chosen. Otherwise, the less fit individual is chosen. The selection pressure can then be tuned by increasing or decreasing the acceptance threshold. A lower value for t decreases the selection pressure, while a higher value increases the selection pressure.

3.7.5 Elitism

In each step of the evolutionary algorithm, the previous generation is replaced by the offspring of the selected parents. Completely replacing the previous generation with the new generation has the undesirable effect where mutations in the children can lead to the loss of good partial solutions. A configurable level of elitism has therefore been implemented into the evolutionary algorithm. At the end of each generation, the n fittest individuals are promoted to *elites*, who are then allowed entry into the next parent selection unchanged. Elitism allows multiple generations to build upon the partial solutions in the elites and ensures that the fittest genomes are preserved between generations.

3.7.6 Evaluation

The evaluation function was kept fairly simple as to restrain from guiding the robots. Primarily, the evaluation function was only calculating how many robots survived the trial.

$$f(G) = \frac{G(\bar{R_{tot}})}{R_{tot}} \tag{3.9}$$

where f is the fitness score of some genome G. $G(\bar{R}_{tot})$ is the number of robots that died, $\bar{R}_{tot}$, in the trial with genome G, and R_n is the total number of robots in the system.

This fitness function, however, gave poor results mainly due to the fitness graph being discrete and failing to differentiate enough between different genomes. Therefore, a different approach was implemented where the lifetime of a robot used is instead.

$$f(G) = \frac{\sum_{i=1}^{n} G(L_i)}{nL_{max}} \tag{3.10}$$

The fitness of a genome is calculated in a similar way as eq. 3.9. The difference here is is that L_i represents the lifetime of robot i during simulation, n is the total number of robots, and L_{max} is the maximum lifetime of a robot. It is also possible to add additional reward for the ability to aggregate such that self-assembly(or at least proximity) is promoted, but as mentioned earlier, the purpose was to limit the amount of evolutionary guiding. In the case where self-assembly does not occur, a more sophisticated objective function could be implemented.

3.8 Data gathering

The statistics logged for post-processing, and later analysis is as follows:

- Fitness
- Group size
- Number of groups
- Energy items collected by individual robots
- Energy items collected by robot groups
- Number of predators eaten by robot groups
- Number of robots starved
- Number of robots eaten by predators

These statistics are logged at the end of each scenario for every genome in all the generations. These statistics are recorded by the agent observers and the world observer, the only exception being energy collected which required some additional modification of roborobo. This modification consisted of performing the logging when energy is rewarded for eating food. Logging the group sizes, and the number of groups is done a bit differently than the rest of the statistics. The logging is performed by taking a snapshot which contains the number of groups and their sizes every 50 iterations of the simulation. The reason for this is that they change in value over the lifetime of a simulation, and this change should be recorded.

Chapter 4

Results and Discussion

This chapter shows the results of the simulations done with the framework shown in chapter 3 as well as discussing these results. The chapter is mainly divided into two sections, the first explaining the configuration of the simulation as well as results depicted by graph data. The second section focuses on discussing the results regarding the reason they have certain values and graphs, as well as comparisons between the different simulations and discovery of correlations and behavioural similarities.

4.1 Experimental Results

As stated in the chapter introduction, this section covers the results of the simulations. Even though the simulator supports a wide variety of different configurations, three main simulation groups were chosen for this particular study.

- The first group considers different connection port configurations for the robots.
- The second simulation group varies the difficulty of the environment.
- The third and final simulation group observes the effect of local communication between the robots in a self-assembly structure(the implementation is shown in sec. 3.5.3).

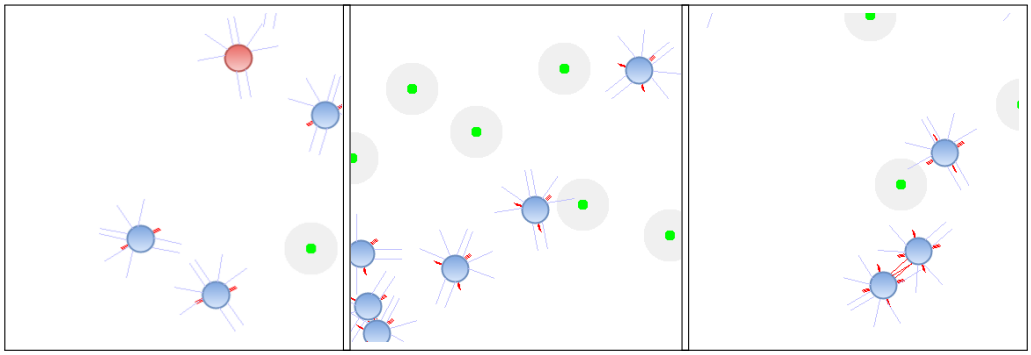
The two first simulation groups targets to contribute to the three main self-assembly mechanisms discussed in chapter 2(Self-assembly Architectures, Hardware Mechanisms and Assembly Protocols). The local communication simulation group aims to study the effect of giving the robots the ability to perform simple communication between the robots in a group.

4.1.1 Port Configuration

The first group of simulations that were done was varying the number of connection ports of the robots and their configuration. These simulations were done such that one could record and discuss the impact of connections between the robots. In more detail, this group of experiments was conducted such that one could record:

- How the number of ports affected the general performance of the system.
- If there is any noticeable difference in the self-assembly architecture.
- In what way the port configuration promotes self-assembly, both regarding the sizes of the different robot groups, and the number of groups.
- How the port configuration affects the fitness of the experiment, regarding convergence and the final result.

The goal of running these simulations is to make correlations between its results to show how configuring the hardware mechanism can affect a self-assembly system. Snapshots of simulations using different port configurations can be viewed in figure 4.1.



(a) 2-ports (b) 3-ports (c) 4-ports

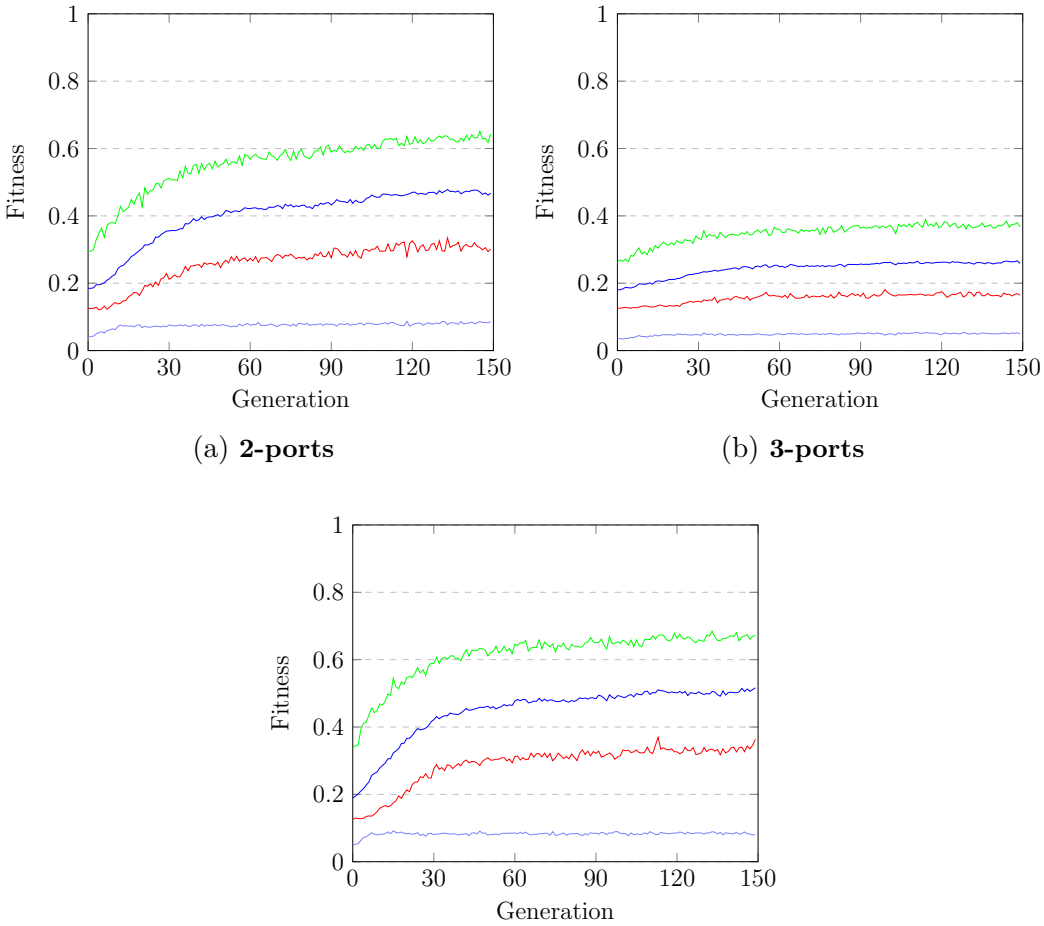
Figure 4.1: The three different port configurations used in simulation

Common configuration parameters of importance for the simulations are listed in table 4.1.

Table 4.1: The simulation parameters for the environments.

Parameter	Value
Number of Robots	20
Iterations per Generation	10000
Scenarios	3
Generations	150

For this group of simulations, the most relevant statistics will be the difference between group actions versus single robot actions. The following pages show the recorded data for simulations using, two connection ports, three connection ports, and four connection ports(see figure 4.1).



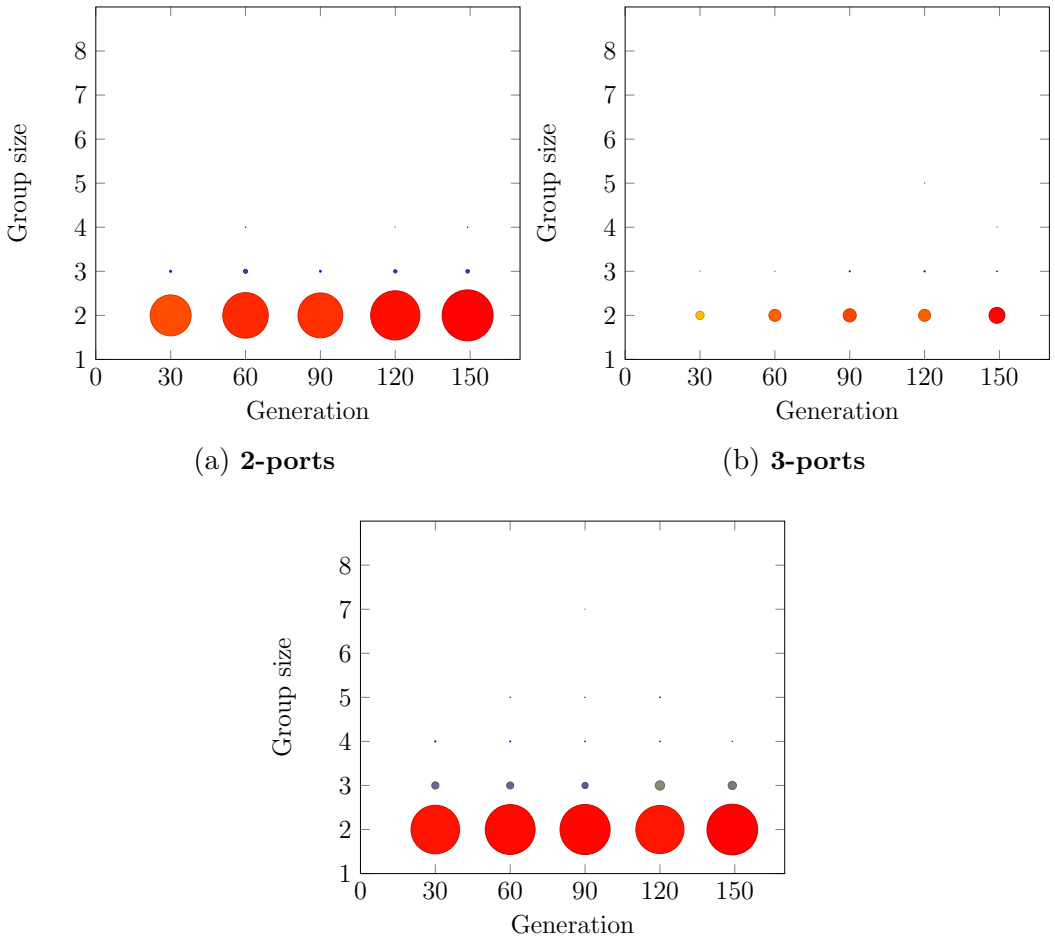
(a) **2-ports**

(b) **3-ports**

(c) **4-ports**

Figure 4.2: The **fitness** from connection port simulations

Figure 4.2 show the results for achieved fitness for the port configuration simulations. The two and four connection port results are very similar where the four connection port performs slightly better with an average fitness of about 0.2 on generation 1 rising to about 0.5 on generation 150. The three connection port simulation performs worse, concerning fitness, in every aspect compared to the other port configurations.



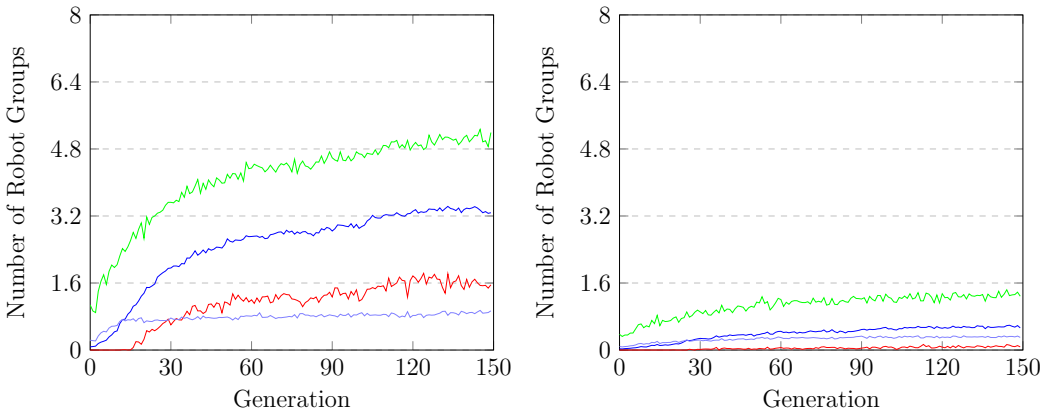
(a) **2-ports**

(b) **3-ports**

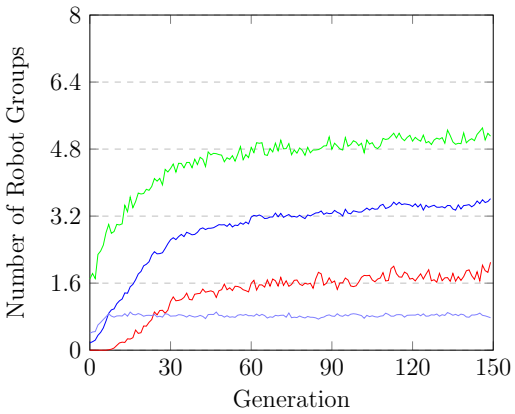
(c) **4-ports**

Figure 4.3: The **group distribution** from connection port simulations

Figure 4.3 represents the distribution of group sizes formed during simulation. The size of the circles indicates the number of groups formed. The two and four connection port simulations have more groups of all sizes with the exception of a single group of size five which was formed from the three connection port simulation. The four connection port simulation formed larger groups than the two connection port simulation.



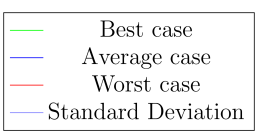
(a) **2-ports** (b) **3-ports**

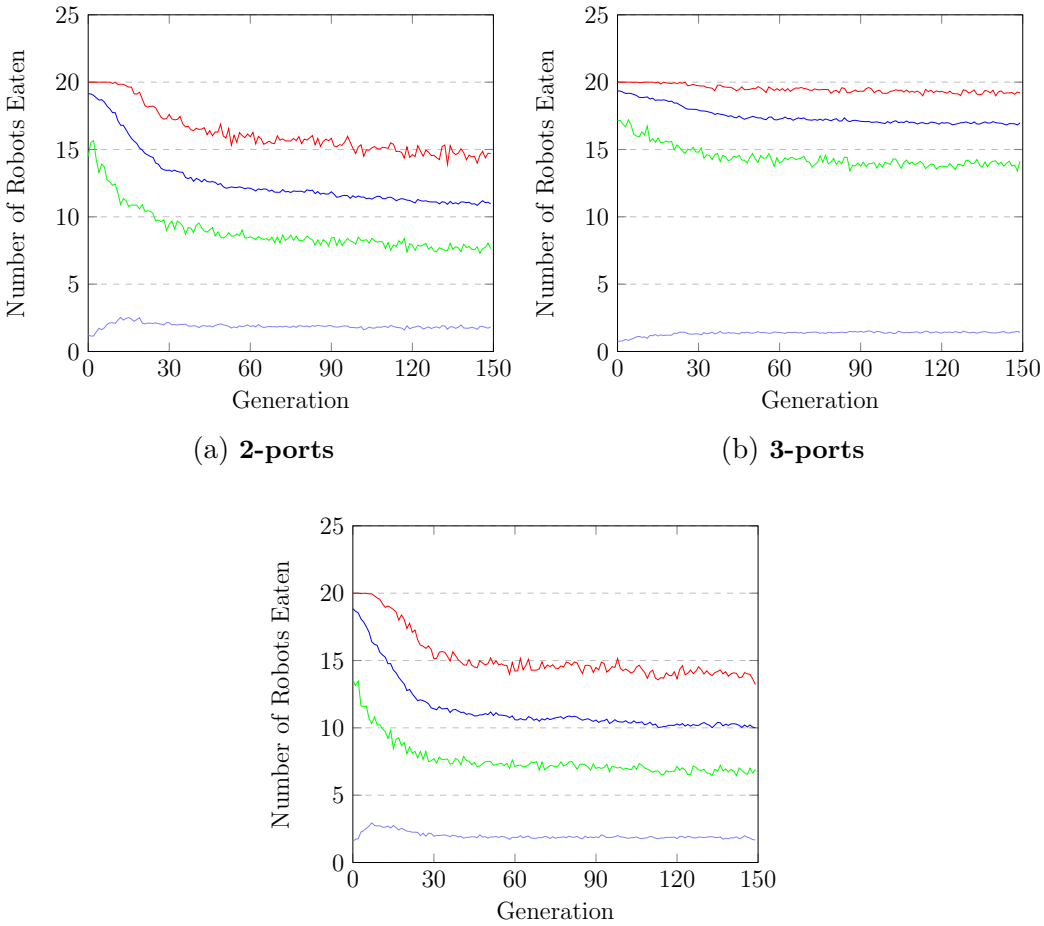


(c) **4-ports**

Figure 4.4: **Number of groups** from connection port simulations

Figure 4.4 shows the number of groups which were formed in the different simulations. It is seen that the results of the two and four connection port simulations are very similar where the only notable difference is that the four connection port results seem to converge at a faster rate. The three connection port results are quite poor, having few groups throughout the trial.





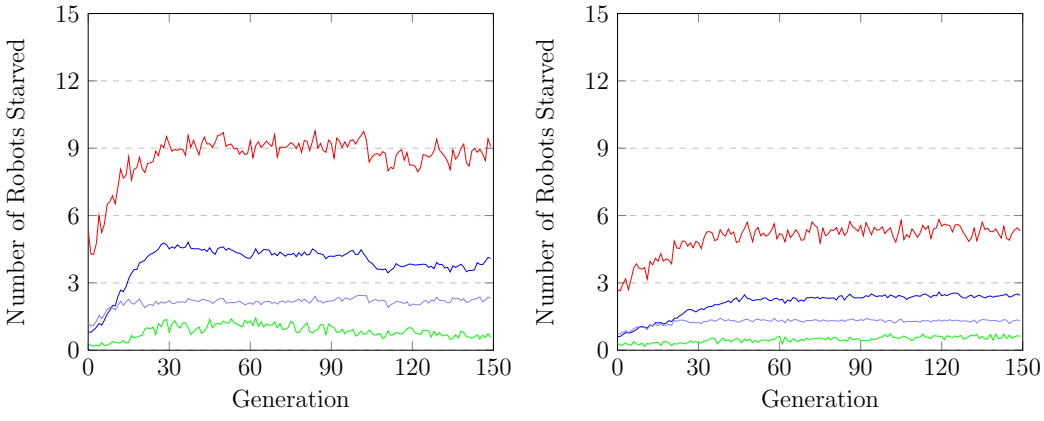
(a) **2-ports**

(b) **3-ports**

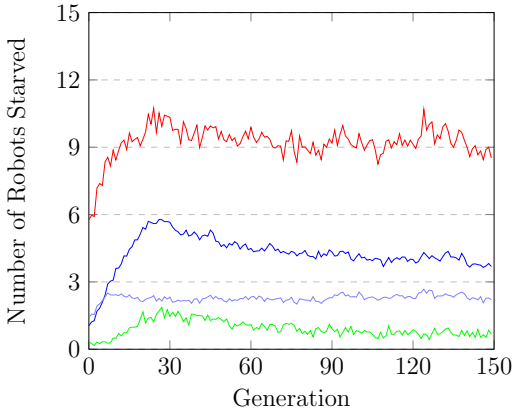
(c) **4-ports**

Figure 4.5: **Number of robots eaten** from connection port simulations

Figure 4.5 shows the number of robots which were eaten by predators during simulation. The four connection port simulation performs best, but only slightly better than the two connection port robots. The four connection port results converges faster and has slightly better results at the end of the simulation. The three connection port results are quite poor in comparison where a lot more robots are consumed by predators.



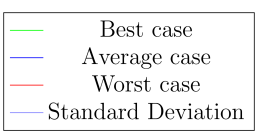
(a) **2-ports** (b) **3-ports**

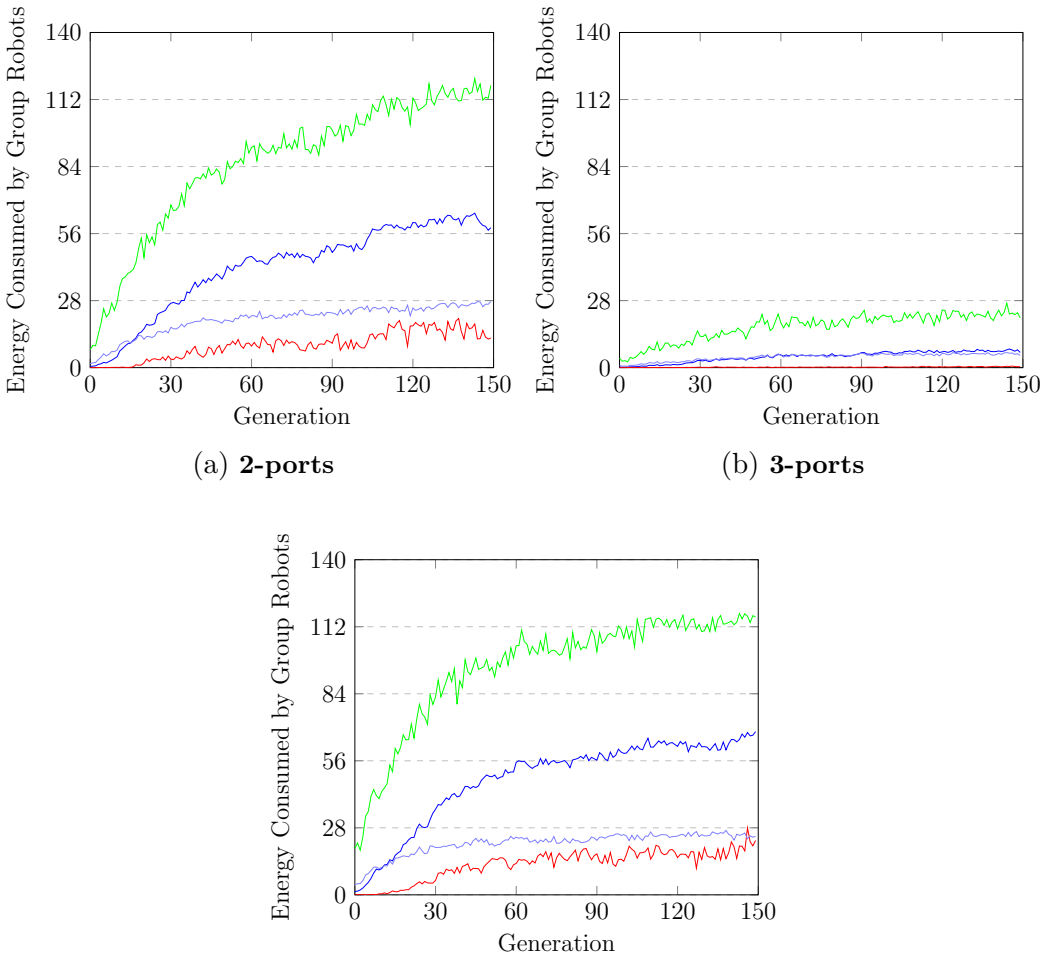


(c) **4-ports**

Figure 4.6: **Number of robots starved** from connection port simulations

Figure 4.6 shows the number of robots starved each generation. The results for two and four connection port simulations are very similar where the average number of robots starved is about four at the final generation(150). The three connection port simulation performs slightly better on these results where the average number of starved robots is slightly less than 3. The three connection port simulation performs best on this result because most of the robots have been consumed by a predator before they die of starvation.



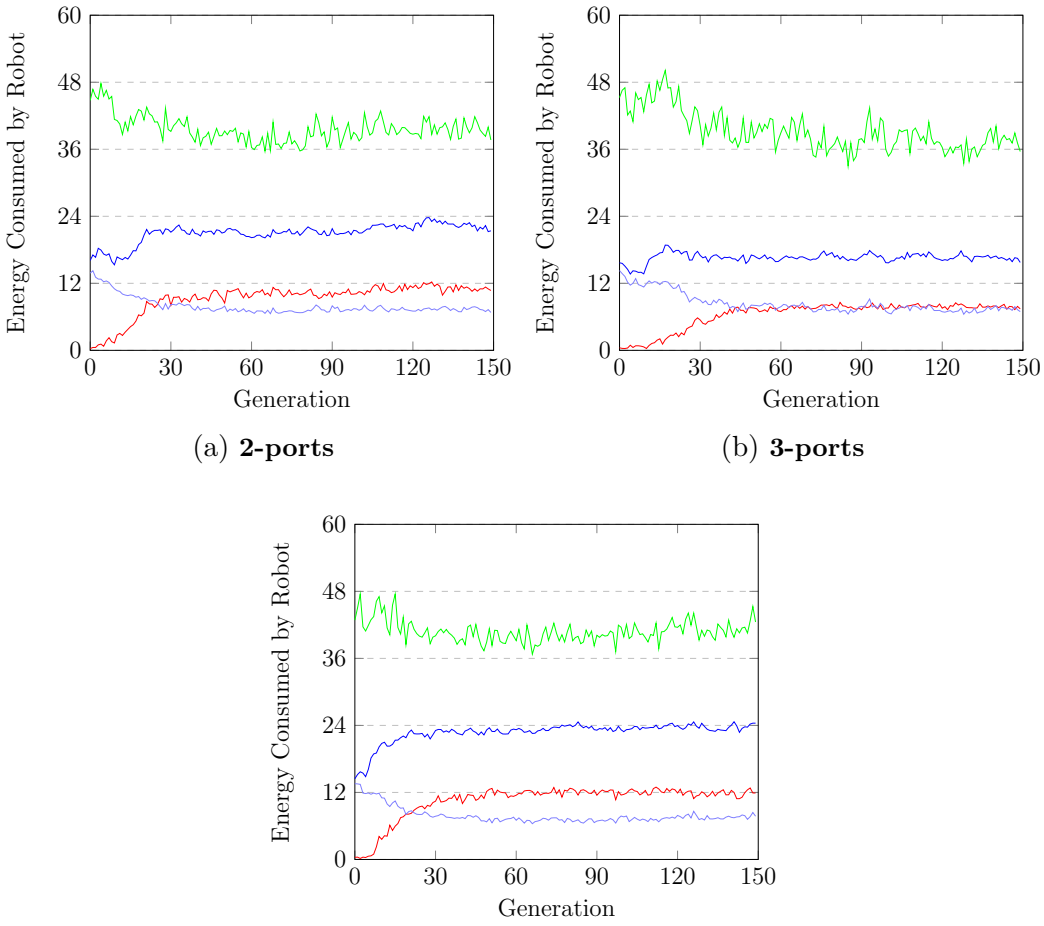


(a) **2-ports** (b) **3-ports**

(c) **4-ports**

Figure 4.7: **Energy consumed by group** from connection port simulations

Figure 4.7 shows the energy consumed by groups of robots during simulation. From these results, it can be viewed that the results containing 2 and four connection ports are very similar with an average result of about 60 energy items consumed at generation 150. The three port configuration performs a lot worse with a result of about ten energy items consumed at generation 150. The results are correlated with the results obtained from the number of groups formed in the simulation(figure 4.4b).



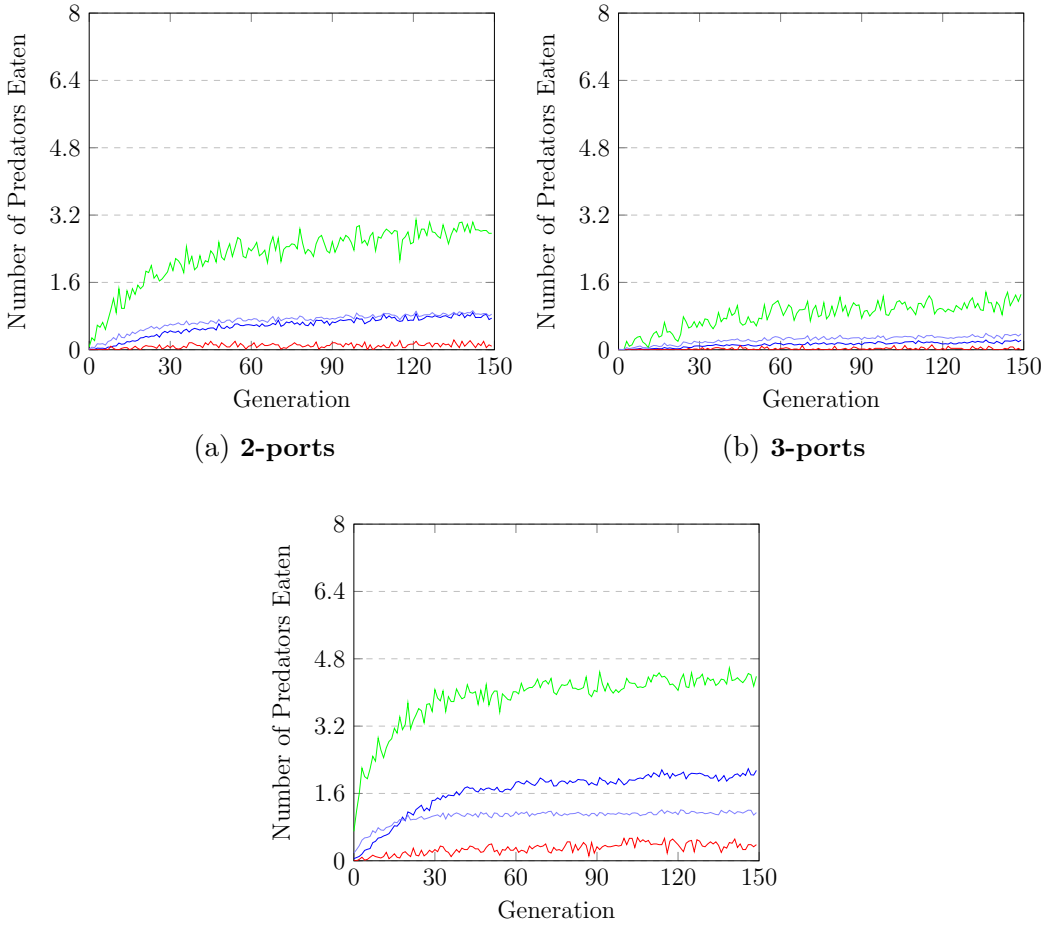
(a) **2-ports**

(b) **3-ports**

(c) **4-ports**

Figure 4.8: **Energy consumed by robot** from connection port simulations

Figure 4.8 shows the total amount of energy which is consumed by robots which are not self-assembled. All of the graphs have similar results and slopes, with the exception that the three connection port simulation performs worse. The reason the average results of figure 4.8a and 4.8c flattens out and does not increase after around generation 30 is that more of the robots are self-assembling and hence is not tracked as a part of these results. As the energy consumed is not decreasing because more robots are a part of groups, it can be deduced that more energy is consumed on a per robot basis.



(a) **2-ports**

(b) **3-ports**

(c) **4-ports**

Figure 4.9: **Number of predators eaten** from connection port simulations

Figure 4.9 tracks the number of predators that have been eaten by robot groups. The four connection port simulation performs best and is correlated with having larger group sizes than the other port configuration shown in figure 4.3c. As the robot groups must be of at least size three to consume a predator, the results shown in this figure conform with the other results shown earlier.

4.1.2 Environment Difficulty

The motivation for this experiment is to observe how changing the difficulty of the environment affects the evolved behaviour. For the experiment two environment difficulties were constructed, one easier and one more difficult. These environment difficulties were then used in the simulations so that the impact of the evolutionary pressure could be recorded and discussed. More specifically the experiments were constructed to investigate how differing environment difficulties affect the following properties of the evolved behaviour:

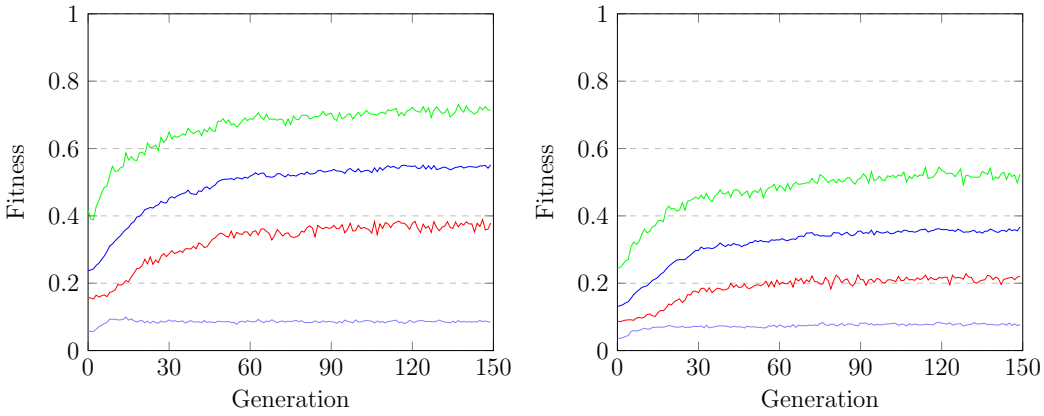
- The amount of robot groups formed through self-assembly, and the number of robots in each group.
- How the energy gathering behaviour is affected. Whether the robots prefer gathering energy individually, or as robot groups.
- How the environment difficulty affects the fitness of the experiment, regarding convergence and the final result.

The environment difficulty is modified by varying the following simulation parameters: the initial energy level the robots, the maximum amount of energy each robot can be charged with, the numer of energy items in the environment, and the number of predators present in the simulation. Table 4.2 presents the simulation parameters that are varied for the environments.

Table 4.2: The simulation parameters for the environments.

Environment	Predators	Initial energy	Food items	Maximum energy
Easy environment	4	8000	25	10000
Hard environment	7	6000	20	8000

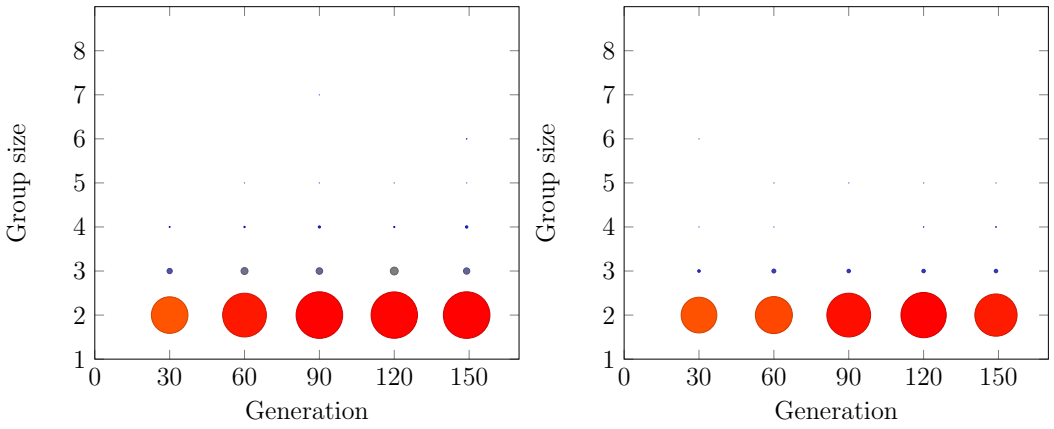
The results were obtained by running 20 simulation trials for each difficulty, where each of the trails run for 150 generations.



(a) **easy** (b) **hard**

Figure 4.10: The **fitness** from environment simulations

Figure 4.10 shows the results for achieved fitness for the environment difficulty simulations. The results for the easy environment are noticeably better than the results from the hard environment, starting with an average fitness of 0.24 at generation 1 and rising to 0.55 at generation 150.

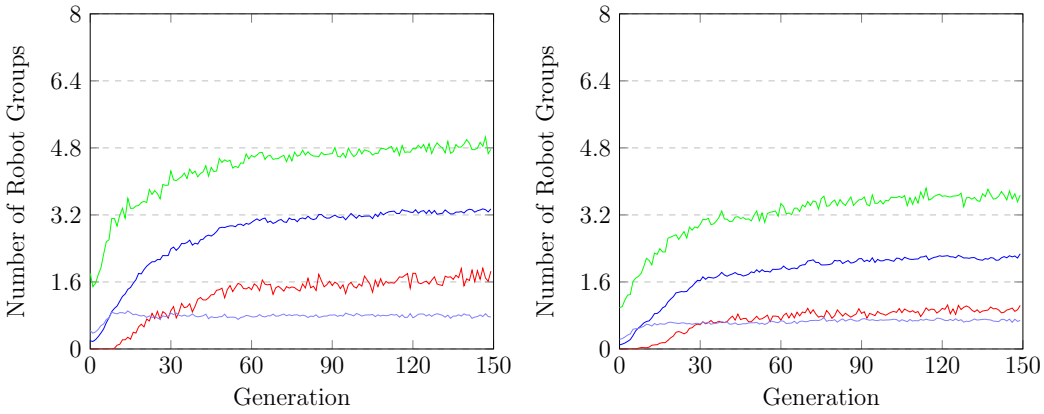


(a) **easy** (b) **hard**

Figure 4.11: The **group distribution** from environment simulations

Figure 4.11 presents the distribution of group sizes formed during simulation. The distribution for the easy and hard environments are very similar, but one can see that the easy environment simulation has slightly more groups of two and three robots.

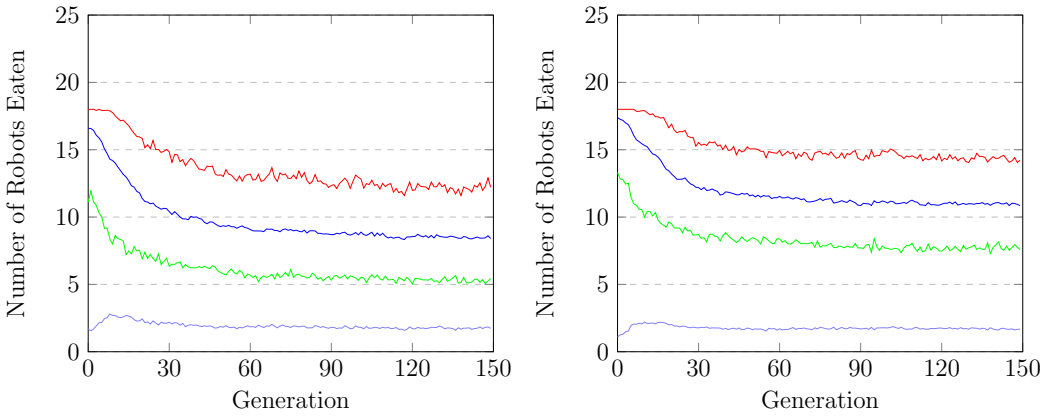
Best case
Average case
Worst case
Standard Deviation



(a) **easy** (b) **hard**

Figure 4.12: **Number of groups** from environment simulations

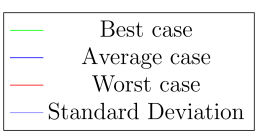
Figure 4.12 shows the average number of groups formed at a given timestamp in the simulation. One can see that the curves for both simulations are quite similar, but the number of groups formed in the easy environment is around one more at any given generation.

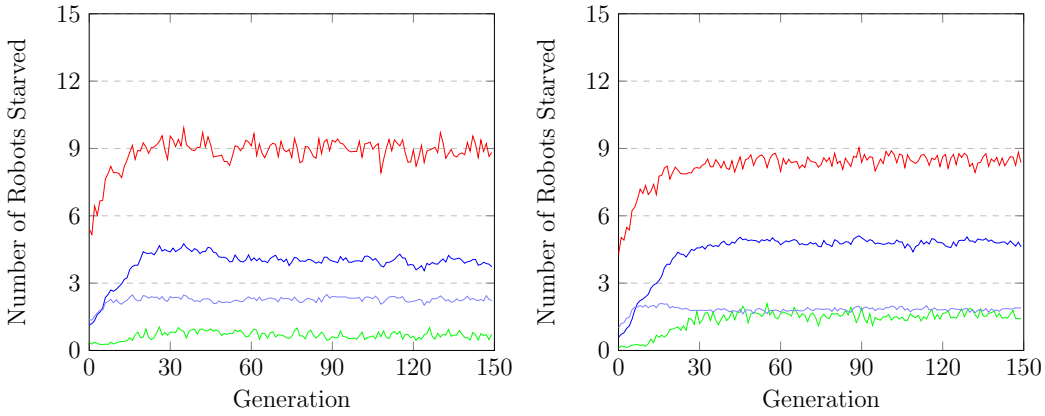


(a) **easy** (b) **hard**

Figure 4.13: **Number of robots eaten** from environment simulations

Figure 4.13 shows the number of robots which were eaten by predators during the simulations. The easy environment simulation performs a bit better than the hard simulation. In the easy environment simulation, an average of around 16 robots are eaten in the first generation and decreases to around eight robots in generation 150. In the hard environment simulation, an average of around 17 robots are eaten in the first generation and decreases to around 11 robots in generation 150.

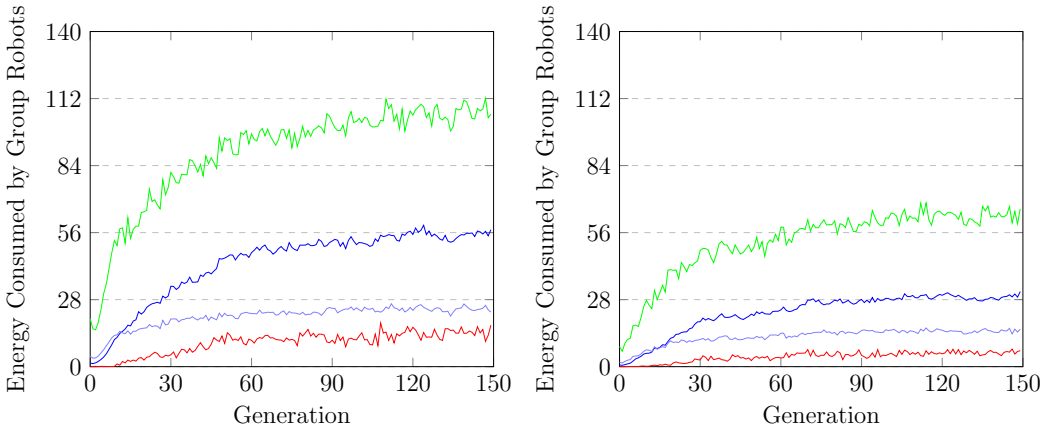




(a) **easy** (b) **hard**

Figure 4.14: **Number of robots starved** results from environment simulations

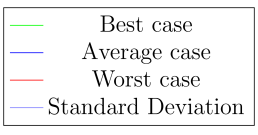
Figure 4.14 shows the number of robots dying from starvation. The graphs are relatively similar, with the worst case being almost the same. In the easy environment, compared to the hard environment, there is around one robot less dying from starvation for the average and best case. Surprisingly, these graphs show that during the first 30 generations the results are getting worse. The increase in robots starving can be explained by that many robots are getting eaten by predators before they have time to starve in the early generations.

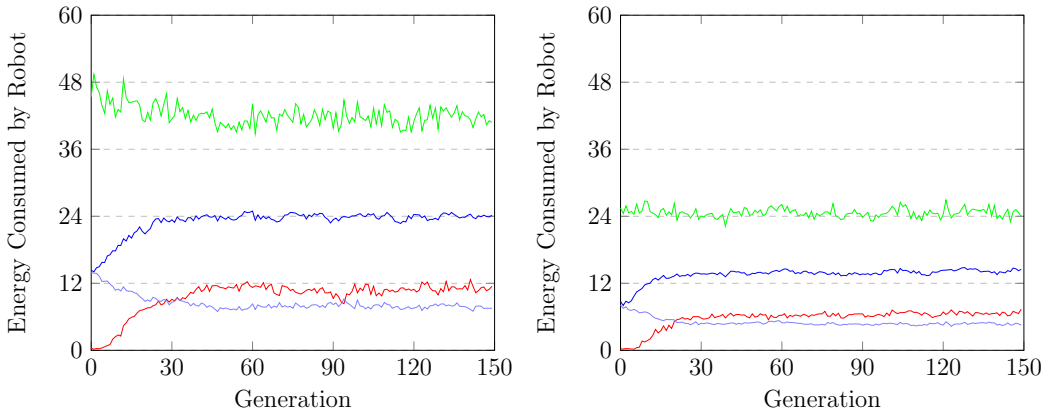


(a) **easy** (b) **hard**

Figure 4.15: **Energy consumed by group** from environment simulations

Figure 4.15 shows the total amount of energy consumed by robot groups during simulation. In the easy environment simulation, the robot groups gather far more energy than in the hard environment simulation. Gathering an average of 56 pieces of energy at generation 150 for the easy environment compared to just 31 pieces in the hard environment simulation.

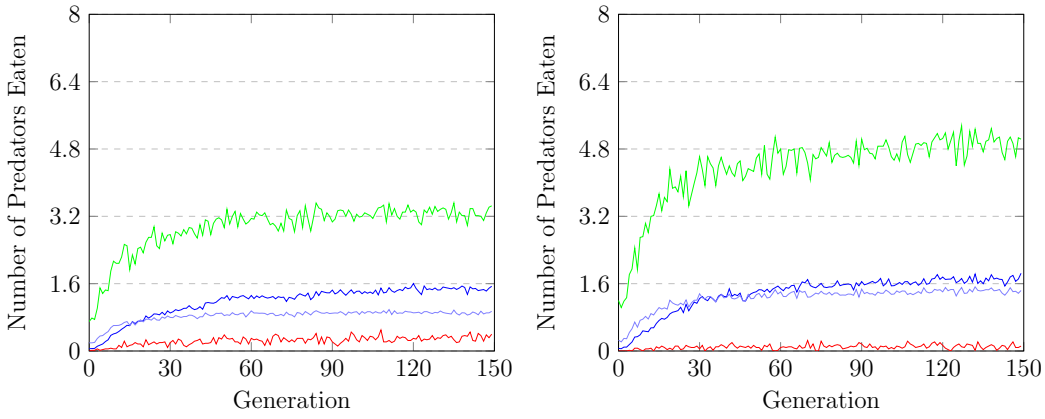




(a) **easy** (b) **hard**

Figure 4.16: **Energy consumed by robot** from environment simulations

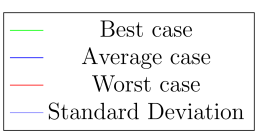
Figure 4.16 shows the total amount of energy eaten by individual robots during simulation. The easy environment performs better, collecting an average of 14 energy in the first generation and an average of 24 in the final generation. The amount of energy gathered in both environments flattens out at around 30 generations. The reason for this is that more robots are self-assembling and is hence not tracked as part of these results.



(a) **easy** (b) **hard**

Figure 4.17: **Number of Predators eaten** from environment simulations

Figure 4.17 shows the number of predators eaten by robot groups. On average, the amount of consumed predators is about the same for both environment difficulties, with the robots in the hard environment being consumed at a slightly higher rate. However, in the best case, the hard environment performs significantly better. The results may be correlated with it being more predators present in the hard environment that the robots can eat.



4.1.3 Local Communication

The goal of this experiment is to investigate the impact local communication has on the behaviour of the robots. The experiment is performed by selecting the fittest genomes found during a simulation, removing the communication module, and then observing the change in behaviour, if any.

The observed robot behaviour can be split into two phases, the individual behaviour, and the group behaviour.

Individual behaviour

The individual robots have two observed movement strategies. The strategy which is employed depends on if a wall is within the range of the robot sensors. The first strategy involves moving in a wide circular path. This strategy occurs when no walls are detected by the sensors. This strategy allows the robots to collect more energy, and will attempt connections with other robots if they collide. However, they make no attempt to avoid predators in their path. This behaviour is usually observed at the beginning of the simulation since most of the robots are initialised away from the walls.

The circular motion of the robots is wide enough to make them crash into the walls of the environment. The behaviour of the robots changes when the sensors detect a wall. Instead of moving in circles the robot changes its movement pattern to follow the wall of the environment. Figure 4.18 shows how a robot follows the wall while keeping it within sensor range.

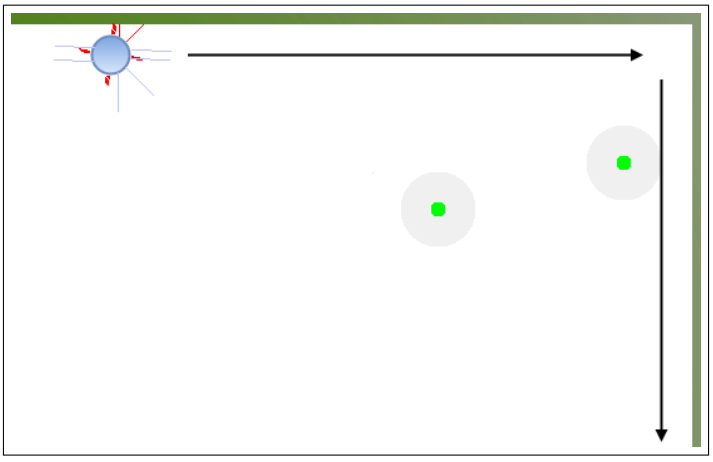


Figure 4.18: A robot using its sensors to follow a wall.

The robot will continue moving along the wall until it meets another robot, and can form a group, or if the sensors detect a bypassing robot group. If a robot group comes within sensor range while the robot is moving along the wall, the robot will abandon the wall and attempt to follow the group instead.

Group behaviour

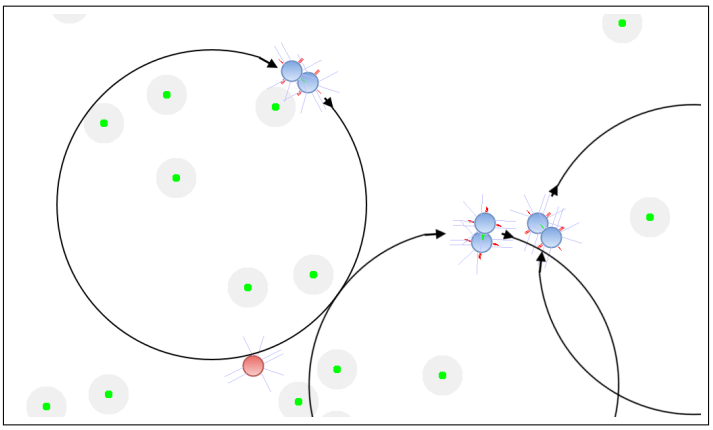


Figure 4.19: Robot groups moving with circular motion.

The behaviour of the robot groups is similar to the first strategy of the individual robots. Robot groups also move in wide circular motions, displayed in figure 4.19, but if the group crashes into a wall, it will simply turn around and continue. The robot groups consume predators in their path, but they make no attempt to follow detected predators. The robot groups will continue to move in circular motions, consuming energy, predators, and making connections, until the end of the simulation, or until the members of the groups starve.

Communication module

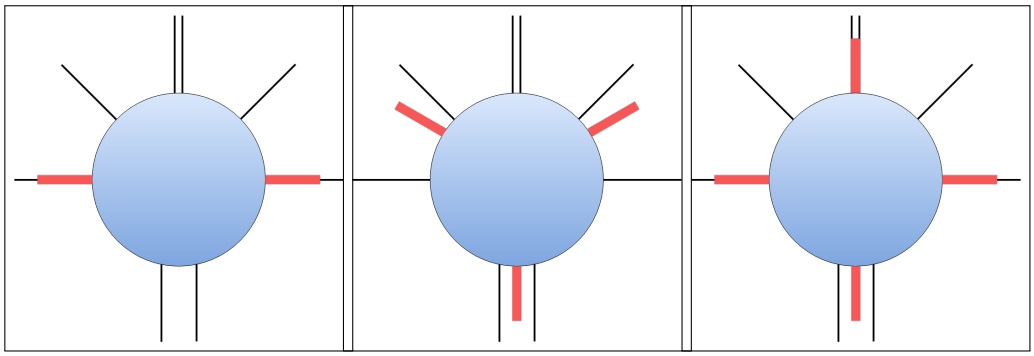
Disabling the local communication module has a significant effect on the behaviour. With the communication disabled the robots will no longer switch to the group behaviour once they are connected. Instead, the groups will continue to perform the individual robot behaviour regardless of the number of connected robots.

4.2 Discussion

This section covers the analysis of the obtained results. The section is split into three parts. The first part covers the connection port simulations and reviews the impact these results have on achieving self-assembly. The second part covers the different environmental difficulty simulations. The third part covers the local communication module and explains the effect it has on the robot system.

4.2.1 Port configuration Analysis

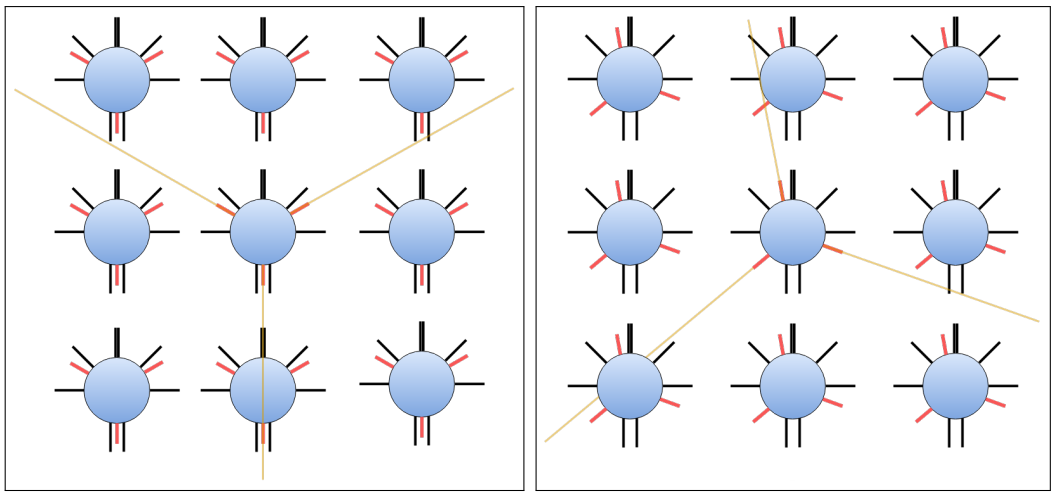
Regarding the results fetched from the port configuration simulations the first obvious remarks stem from the simulation running a three port configuration. The results of this simulation are much weaker concerning performance and promotion of self-assembly than the simulations running two and four connection ports. The reason for this can not be deduced completely from the empirical results, but as the only difference in these simulations are the number of connection ports and the alignment; it is clear that the connection port configuration can significantly impact the performance of the simulation. It can also be deduced that it is not the number of connection ports that has the primary impact of the solution, but rather the placement. The reason one can make this claim is that the configuration using two connection ports and four connection ports perform quite similar in terms of performance. If the number of connection ports had a significant impact on the results, one would expect the simulation using either two or four connection ports to yield even poorer results than the three connection port simulation. This effect narrows the port configuration problem down to the alignment of the connection ports.



(a) 2-ports (b) 3-ports (c) 4-ports

Figure 4.20: The three different port configurations used in simulation

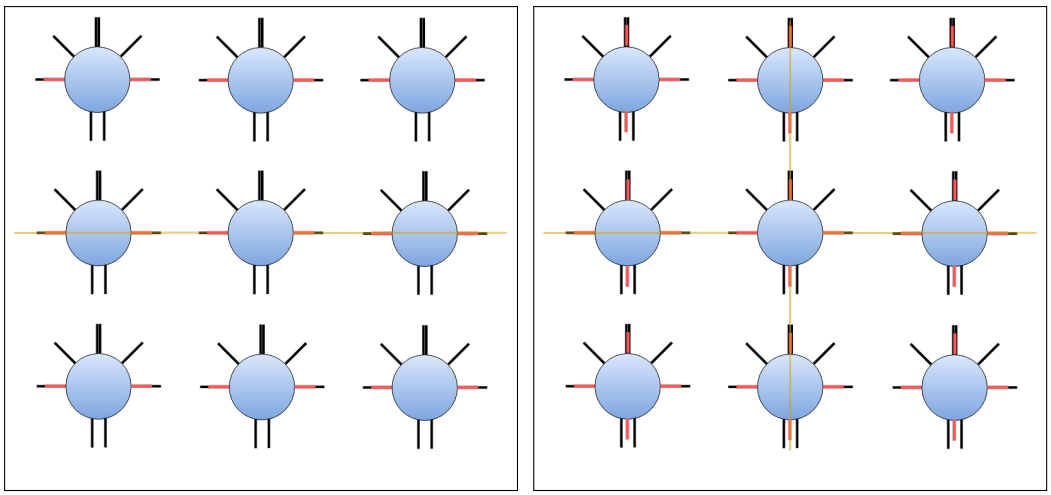
Figure 4.20 shows a closer view of the alignment that the robots initially have when spawned into the environment. As explained in 3.5, the robots can rotate their connection ports as a group. A standard strategy which is usually evolved is to either constantly rotate the connection ports in hopes of lining up the ports to another robot, or, the robots start rotating their ports when the sensors see another robot in an effort to self-assemble. There are two main problems that the three connection port robots have compared to the other port configurations. First, the initial port location does not align to any other robot.



(a) Initial alignment (b) After 50° port rotation.

Figure 4.21: This figure shows how the three connection port robots align

As seen in figure 4.21a, there is not a trivial alignment for the robots to connect. One might initially presume that this is not a problem as the robots have a mechanism for rotating their ports to solve this exact issue. However, as all the robots are running the same genome, as per off-line evolution and hence the same behaviour, it becomes increasingly difficult for the robots to solve this problem. As explained earlier, the robots in this simulation tend to evolve a strategy which involves constantly spinning the connection ports in one direction. However, as seen in figure 4.21b, in the case where all robots at some time step have rotated their connection ports 50°, the same issue of port alignment would still hold.

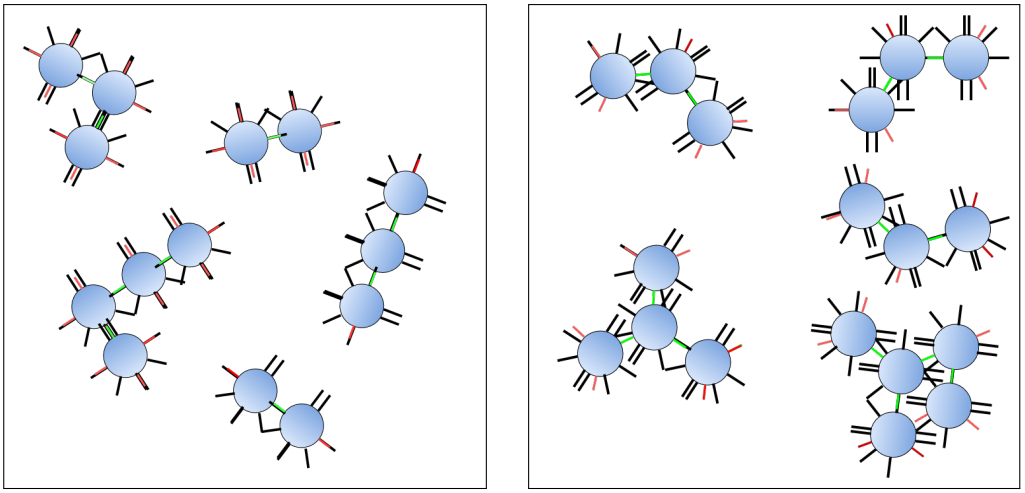


(a) Two connection ports alignment (b) Four connection ports alignment

Figure 4.22: This figure shows how the two and four connection ports robots align from initial configuration

Consider figure 4.22. In this example, there are two and four port configurations. It can be observed that with an initial rotation of the ports, there exist possibilities for the robots to self-assemble without having the robots behave differently in terms of rotating their connection ports. This differentiation seems to be the main reason that the three connection port configuration is being outperformed.

The second problem with robots having three connection ports, in this alignment, is the possible group formations the robots can form. Chapter 2 covers the chain and lattice architectures that the robots can form when self-assembling. The simulator is developed to support the lattice architecture because if its simplistic method of coordinated movement.



(a) Robot groups with two and four connection ports

(b) Robot groups with three connection ports

Figure 4.23: This figure contains self-assembled robot groups with different assembly combinations

It can be seen from figure 4.23 that the different connection port configurations create various types of groups. With two and four connection ports (figure 4.23a), the robot groups either take the form of a line or some square grid formation. Possible formations of the three connection ports robot groups(figure 4.23b) breaks the pattern of a square grid configuration which makes it harder for other robots trying to connect to the group. The main reason for this connection problem is the relative position a connecting robot needs, is harder to attain because of the larger distance between the connection ports.

There are not significant discrepancies between the results from the port configuration simulation containing two and four connection ports. The only result which differs significantly is "predators eaten"(figure 4.9a and 4.9c). The reason for this can be deduced from figure 4.3a and 4.3c which shows that robots with four connection ports tend to form larger groups. It can however be viewed from figure 4.4a and 4.4c that two and four connection ports have roughly the same number of groups. The occurrence of a greater amount of larger groups naturally agrees with eating more predators as groups need to be of at least size three to consume a predator. The reason for four connection ports robots to attain larger groups is simply that more connection ports allow more points of entry for other robots trying to connect, which increases the probability of succeeding self-assembly to the group.

From these results, it can be deduced that larger groups do not give rise to better fitness in this experiment, but rather the number of groups (a group is of minimum size 2) correlates with the fitness. The reason for this is that the robots in the port configuration simulations are not in a great need of energy.

The robots are able to naturally attain what they need in the environment and hence do not have to rely on a strategy involving predator consumption.

4.2.2 Environmental difficulty analysis

The analysis and discussion on the impact of environmental difficulty can be divided into three main categories: the impact of environmental threats when the difficulty is modified, how the promotion of self-assembly is effected by the environmental difficulty and finally, how the evolved energy collection strategy is influenced by the environment difficulty.

Environmental threats

The robots have two threats in the environments presented, starvation and getting killed by predators. The robots in the easy environment receive more energy from each energy item, and there are more energy items available. From figure 4.14 one can see that this reduces the amount of robots dying from starvation in the easy environment, but the improvement is minuscule.

Increasing the number of predators in the environment seems to have a higher impact on the difficulty presented by an environment. Figure 4.13 shows that increasing the amount of predators present in the environment has a greater impact on the difficulty of the environment than limiting the energy available. The reason for why increasing the number of predators has a much higher impact on difficulty is not completely clear from the results. However, the observed behaviour described in section 4.1.3 can help explain the results. The sensors are used by the robot to detect walls and other robots, but not predators or food. This behaviour means that the robot may miss some food, but there is enough food in the environment so the robot will eventually find more food. On the other hand, failing at predator avoidance has much more severe consequences as the predator will instantly kill the robot.

Promotion of self-assembly

One of the motivations for this experiment was to see how modifying the evolutionary pressure affects the promotion of self-assembly. Figure 4.12 show that the robots form more groups in the easy environment. At first glance, this seems to indicate that the easy environment is more successful at promoting self-assembly. This difference in the number of groups may be explained by examining the lifetime of the robots. As explained in section 3.7.6, the fitness of a genome is determined by the average lifetime of a robot. The fitness achieved in the easy environment, figure 4.10a, is higher than the fitness achieved in the hard environment, figure 4.10b. The fitness means that the robots in the easy environment live longer, and as a consequence have more time to form groups.

However, figure 4.11 shows that the size of the robot groups formed is not affected by modifying the difficulty of the environment. In both environments, the distribution of group sizes is heavily weighted towards groups of two. The reason for this may be that the environments give a high reward for being in a group. That is protection from predators, the additional reward for forming larger groups(being able to eat predators) is diminished as there is an abundance of energy items in the environment.

Energy collection strategy

One can see from the figures 4.15 and 4.16that the robots in the easy environment collect far more energy than the robots in the difficult. This result can likely also be attributed to the fact that the robots in the easy environment live longer, and that there is more energy available, instead of a more optimal energy gathering strategy. One can look at the ratio of energy collected by individual robots versus energy collected by groups of robots for the environments. This relationship is presented in table 4.3.

Table 4.3: The percentage of energy collected by groups of robots for the environments.

Generation	Easy	Hard
10	54%	51%
50	71%	67%
100	71%	71%
150	72%	73%

Table 4.3 shows that in both environments the ratio of energy collected by robot groups is approximately the same. The ratio means that although the robots in the easy environment collect more energy in total, the strategies evolved in the different environments are similar. This also coincides with that the observed behaviour described in section 4.1.3 is very similar for the different environments.

4.2.3 Local communication analysis

As described in section 4.1.3, the evolved behaviour makes use of the communication module. When the communication module was disabled, the robots did not change their behaviour when they formed groups. Therefore, it is reasonable to assume that local communication is at least involved in modifying the robot behaviour once connected to a group. Exactly how the evolved neu-

ral network interprets the messages received is challenging to infer, but one can observe the messages sent to get a conceptual understanding.

[0.992 0.999 0.423 0.002]

Figure 4.24: The message passed between the robots.

Without any other inputs, all robots send the message displayed in 4.24 by default. Receiving other inputs, such as sensors, changes the message by a negligible amount. The surprising thing about the message is that the components in the communication messages have wildly different values. It turns out that the values in the messages have an interesting interaction with the port connection status that is also propagated to the neural network. The port connection status contains the robot's connection status of each port.

Table 4.4: The resulting desired rotations for different port combinations with the evolved message and a test message for comparison. A port status value of 1 means the particular port is connected, 0 means it is not connected.

Message:[0.992 0.999 0.423 0.002]		Message: [1.0 1.0 1.0 1.0]	
Port status	Desired rotation$_{deg/step}$	Port status	Desired rotation$_{deg/step}$
1 1 0 0	0.9482	1 1 0 0	0.719
1 0 1 0	0.999	1 0 1 0	0.976
1 0 0 1	0.517	1 0 0 1	0.658
0 1 1 0	0.997	0 1 1 0	0.866
0 1 0 1	0.705	0 1 0 1	0.674
0 0 1 1	0.997	0 0 1 1	0.822

Table 4.4 shows how the desired rotation for the robots varies with the local topology of the connected robots. The table shows this variation with the evolved message and a dummy message for comparison. It can be observed that the resulting desired rotations for the robots have different values for the two messages. The desired rotation for the robots determines the radius of the circular motion that dictate the robot group movements. These results show that the local communication module is used for two purposes. The first purpose is to act as a switch to change from the individual robot behaviour to the group behaviour. Additionally, the communicated message decides the robot group's behaviour depending on the different connection topologies.

Chapter 5

Conclusion

The main focus and goal of this thesis have been to discover and test the elements present in a self-assembly system when robots are given basic learning capabilities. The experiment has been conducted using a heavily modified version of the roborobo framework. The main factors that have been researched in accordance with self-assembly have been: connection port configuration, environmental influence and local communication.

The results obtained from the connection port simulations, show that configuration of the connection ports can significantly impact the emergence of self-assembly using an evolutionary algorithm. The port configuration consists of the number of connection ports each robot has available and the relative positioning of the connection ports on the robot. Both elements influence the size and frequency of self-assembling robot groups. It can, however, be narrowed down to a single influential self-assembly mechanism: the assembly protocol. It is evident from the results that providing the robots with tools that allow an efficient and simple assembly protocol to be evolved, is essential to achieve successful results.

The main focus of the learning algorithm should be to solve the task at hand and not deriving a complex strategy for achieving self-assembly. Having the ability to form a complex assembly protocol can be appropriate in a particular situation as it may significantly improve performance. However, it should not be a minimum requirement for the ability to self-assemble. An evolutionary algorithm performs better when an incremental solution to some desired behaviour is possible. If the least complex achievable self-assembly protocol requires sufficiently advanced cognition, then the robots may only have a few occurrences of self-assembly or none at all.

The results gained from the environment difficulty simulations implies that the difficulty of the environment is not directly correlated with promoting self-assembly. The results show that the robots perform better in an easy environment, but this is rather due to restrictions in the environment and not

due to the ability to self-assemble. The only effect that difficult environments impose on self-assembly is making robots die earlier, giving them fewer opportunities to self-assemble. Promoting self-assembly may rather see a larger impact if the rules of the environment change(examples include translation speed of predators and the physical size of the environment).

One of the problem statements this study aimed to examine was the introduction of a local communication module. From the results discussed in chapter 4, it was seen that using the local communication module drastically changed the behaviour of self-assembled groups. However, it cannot be made any conclusive remarks as to the local communication module promoting the robots to self-assemble as deciphering the evolved values of the neural network is very difficult. From a logical point of view, one would not expect there to be a difference, because the local communication module only transmits information between the robots in a self-assembled group. Hence, there is seemingly no reason why this would help two singular robots to self-assemble. However, when using an evolutionary algorithm, the use of certain modules may be utilised differently than the developer predicts. The evolved genomes may use the module as a state machine instead of a message passing module.

In concluding remarks, it is shown that there should be a larger focus on the connection mechanism which the robots have equipped. To promote self-assembly, the ideal hardware mechanisms would be one which makes it easy for the robots to evolve an efficient assembly protocol, but also yields an interface to evolve complex assembly behaviour. In the case where static connection points are used, using many, initially aligned connection ports increase the frequency and size of the self-assembled groups. The difficulty of the environment does not seem to impact the frequency of self-assembly, and one may consider altering the static rules of the environment which may yield a noticeable impact. A local communication module is an advantageous asset to provide for self-assembly robots as it may be used to communicate a transition to group behaviour as well as assisting in the type of behaviour which emerges from the group.

5.1 Future work

From the observed results of self-assembly mechanisms and environment, it is seen that improvements and further experimentation can be implemented. This forms the basis for exploring other factors of self-assembly mechanisms.

The port configuration has the possibility to be explored further as the results of this study determined that it has a significant impact on the emergence of self-assembly. A possible exploratory field would be to challenge the static nature of the ports presented in this study. Hardware mechanisms which do not depend on fixed positions on a robot where the robots are able to self-

assemble at any point on a connecting robot should, according to the results obtained in this study, perform at a higher rate. An obvious end goal of studies like this one is to be able to realise these robots into the real world. In these scenarios, the reality gap will probably inflict even stricter conditions on performing a simple assembly strategy which could detriment the ability to self-assemble.

Since the impact of changing the environment did not suggest a change in assembly protocol or strategy, a reasonable continuation would be to categorize which environmental scenarios self-assembly, through evolution, is most appropriate. Changing the atomic rules of the environment and robot problem tasks may yield results indicating scenarios where evolutionary self-assembly is more appropriate.

Local communication is a mechanism which has not been significantly researched in this field. According to the results obtained in this study, further exploration into communication modules between the robots can give rise to increasingly complex behaviour. In this study, a very simple protocol of passing floating point numbers from one robot to another was implemented. Perhaps there are better communication protocols available which could further the performance. It is also possible to look at the possibility for robots to communicate on a local spectrum where they are not necessarily self-assembled. Expanded local communication may improve their ability to form an effective assembly strategy through evolution.

The evolutionary algorithm used in this study is simple and standard. There exists additionally advanced evolutionary algorithms and other bio-inspired algorithms where the mechanisms presented in this study should be additionally explored.

Bibliography

[1] S. G. Bailey, S. Hubbard, and R. P. Raffaelle. Chapter 18 - Nanostructured Solar Cells. In M. Henini, editor, *Handbook of Self Assembled Semiconductor Nanostructures for Novel Devices in Photonics and Electronics*, pages 552 – 564. Elsevier, Amsterdam, 2008.

[2] T. Balch. Behavioral diversity in learning robot teams. 1998.

[3] S. Bauer. Termites_rush_to_damaged_portion_of_mound.jpg, July 2007.

[4] W. Baxter. flocking.jpg, 2008.

[5] R. D. Beer. The dynamics of adaptive behavior: A research program. *Robotics and Autonomous Systems*, 20(2):257–289, 1997.

[6] S. Binitha and S. S. Sathya. A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.

[7] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Programmable parts: a demonstration of the grammatical approach to self-organization. pages 3684–3691. IEEE, 2005.

[8] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, Mar. 2013.

[9] D. Brandt, D. J. Christensen, and H. H. Lund. ATRON robots: versatility from self-reconfigurable modules. In *Mechatronics and Automation, 2007. ICMA 2007. International Conference on*, pages 26–32. IEEE, 2007.

[10] N. Bredeche, J.-M. Montanier, B. Weel, and E. Haasdijk. Roborobo! a fast robot simulator for swarm and collective robotics. *arXiv preprint arXiv:1304.2888*, 2013.

[11] A. Castano, W.-M. Shen, and P. Will. CONRO: Towards deployable robots with inter-robots metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, 2000.

[12] D. D. L. Chung. Use of polymers for cement-based structural materials. *Journal of materials science*, 39(9):2973–2978, 2004.

[13] S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois. Evolutionary robotics: Exploring new horizons. In *New horizons in evolutionary robotics*, pages 3–25. Springer, 2011.

[14] K. L. Downing. *Introduction to evolutionary algorithms*. Citeseer, 2009.

[15] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[16] A. E. Eiben, E. Haasdijk, and N. Bredeche. Embodied, on-line, on-board evolution for autonomous robotics. *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution.*, 7:361–382, 2010.

[17] S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, Oct. 2007.

[18] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[19] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu. Miche: Modular Shape Formation by Self-Disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, Mar. 2008.

[20] K. S. Goh, A. Lim, and B. Rodrigues. Sexual selection for genetic algorithms. *Artificial Intelligence Review*, 19(2):123–152, 2003.

[21] R. Gro, M. Bonani, F. Mondada, and M. Dorigo. Autonomous Self-Assembly in Swarm-Bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, Dec. 2006.

[22] R. Gross and M. Dorigo. Evolution of Solitary and Group Transport Behaviors for Autonomous Robots Capable of Self-Assembling. *Adaptive Behavior*, 16(5):285–305, 2008.

[23] R. Gross and M. Dorigo. Self-Assembly at the Macroscopic Scale. *Proceedings of the IEEE*, 96(9):1490–1508, Sept. 2008.

[24] R. Gross, E. Tuci, M. Dorigo, M. Bonani, and F. Mondada. Object transport by modular robots that self-assemble. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*, pages 2558–2564, May 2006.

[25] S. Hettiarachchi, W. M. Spears, D. Green, and W. Kerr. Distributed agent evolution with dynamic adaptation to local unexpected scenarios. In *Innovative Concepts for Autonomic and Agent-Based Systems*, pages 245–256. Springer, 2006.

[26] F. Heylighen and others. The science of self-organization and adaptivity. *The encyclopedia of life support systems*, 5(3):253–280, 2001.

[27] F. Heylighen and others. The science of self-organization and adaptivity. *The encyclopedia of life support systems*, 5(3):253–280, 2001.

[28] B. Kirby, J. Campbell, B. Aksak, P. Pillai, J. Hoburg, T. C. Mowry, and S. C. Goldstein. Catoms: Moving Robots Without Moving Parts. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INLIGENCE*, volume 20, page 1730. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[29] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126. ACM, 2010.

[30] J. B. Lee and R. C. Arkin. Adaptive multi-robot behavior via learning momentum. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 2029–2036. IEEE, 2003.

[31] H. Li, H. Wei, J. Xiao, and T. Wang. Co-evolution framework of swarm self-assembly robots. *Neurocomputing*, 148:112–121, Jan. 2015.

[32] L. Li, A. Martinoli, and Y. S. Abu-Mostafa. Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4):199–212, 2004.

[33] L. Li, A. Martinoli, and Y. S. Abu-Mostafa. Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4):199–212, 2004.

[34] M. J. Mataric. Interaction and Intelligent Behavior. Technical report, DTIC Document, 1994.

[35] M. Mitchell. Life and evolution in computers. *History and philosophy of the life sciences*, pages 361–383, 2001.

[36] J.-M. Montanier and P. C. Haddow. Adaptive self-assembly in swarm robotics through environmental bias. In *Evolvable Systems (ICES), 2014 IEEE International Conference on*, pages 187–194. IEEE, 2014.

[37] S. Murata and H. Kurokawa. Self-reconfigurable robots. *Robotics & Automation Magazine, IEEE*, 14(1):71–78, 2007.

[38] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2210–2217. IEEE, 2000.

[39] R. Mckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. Ijspeert. YaMoR and Bluemovean autonomous modular robot with bluetooth interface for exploring adaptive locomotion. In *Climbing and Walking Robots*, pages 685–692. Springer, 2006.

[40] K. Nahar. Artificial Neural Network. *COMPUSOFT, An international journal of advanced computer technology*, 1:1, 2012.

[41] N. Noskov, E. Haasdijk, B. Weel, and A. E. Eiben. *Monee: using parental investment to combine open-ended and task-driven evolution*. Springer, 2013.

[42] M. Park, S. Chitta, A. Teichman, and M. Yim. Automatic Configuration Recognition Methods in Modular Robots. *The International Journal of Robotics Research*, 27(3-4):403–421, Mar. 2008.

[43] L. E. Parker. L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, 11(4):305–322, Jan. 1996.

[44] M. Polyglottus. mockingbird-tales-readings-in-animal-behavior-5.1.pdf, Jan. 2011.

[45] J. H. Powers. Further Studies in Volvox, with Descriptions of Three New Species. *Transactions of the American Microscopical Society*, 28:141, Sept. 1908.

[46] J. Pugh and A. Martinoli. Parallel learning in heterogeneous multi-robot swarms. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3839–3846. IEEE, 2007.

[47] A. Rosenfeld, G. A. Kaminka, S. Kraus, and O. Shehory. A study of mechanisms for improving robotic group performance. *Artificial Intelligence*, 172(6-7):633–655, Apr. 2008.

[48] D. Rus, Z. Butler, K. Kotay, and M. Vona. Self-reconfiguring robots. *Communications of the ACM*, 45(3):39–45, 2002.

[49] S. Russell and P. Norvig. Artificial intelligence: a modern approach. 1995.

[50] J. Schmidhuber. Evolutionary computation versus reinforcement learning. In *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE*, volume 4, pages 2992–2997. IEEE, 2000.

[51] B. Siciliano and O. Khatib, editors. *Springer handbook of robotics*. Springer, Berlin, 2008.

[52] V. Siracusa, P. Rocculi, S. Romani, and M. D. Rosa. Biodegradable polymers for food packaging: a review. *Trends in Food Science & Technology*, 19(12):634–643, Dec. 2008.

[53] G. C. Sirakoulis and A. Adamatzky, editors. *Robots and Lattice Automata*, volume 13 of *Emergence, Complexity and Computation*. Springer International Publishing, Cham, 2015.

[54] C. Skjetne, P. C. Haddow, A. Rye, H. Schei, and J.-M. Montanier. The ChIRP Robot: A Versatile Swarm Robot Platform. In J.-H. Kim, E. T. . Matson, H. Myung, P. Xu, and F. Karray, editors, *Robot Intelligence Technology and Applications 2*, volume 274, pages 71–82. Springer International Publishing, Cham, 2014.

[55] G. Studer and H. Lipson. Spontaneous emergence of self-replicating structures in molecube automata. In *Proc. of the 10th Int. Conf. on the Simulation and Synthesis of Living Systems (Artificial Life X)*, pages 227–233. Citeseer, 2006.

[56] V. Trianni, E. Tuci, and M. Dorigo. Evolving functional self-assembling in a swarm of autonomous robots. *From Animals to Animats*, 8(July):405–414, 2004.

[57] P. Trueba, A. Prieto, P. Caamao, F. Bellas, and R. J. Duro. Task-Driven Species in Evolutionary Robotic Teams. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. M. Ferrndez, J. R. lvarez Snchez, F. de la Paz, and F. J. Toledo, editors, *Foundations on Natural and Artificial Computation*, volume 6686, pages 138–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[58] B. Weel, E. Haasdijk, and A. E. Eiben. The Emergence of Multi-cellular Robot Organisms through On-Line On-Board Evolution. In *Applications*

of Evolutionary Computation, volume 7248 of *Lecture Notes in Computer Science*, pages 124–134. Springer Berlin Heidelberg, 2012.

[59] H. Wei, Y. Chen, J. Tan, and T. Wang. Sambot: A Self-Assembly Modular Robot System. *IEEE/ASME Transactions on Mechatronics*, 16(4):745–757, Aug. 2011.

[60] G. M. Whitesides. Self-Assembly at All Scales. *Science*, 295(5564):2418–2421, Mar. 2002.

[61] J. S. Yadav, M. Yadav, and A. Jain. Artificial neural network. *International Journal of Scientific Research and Education*, 1(06), 2014.

[62] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):43–52, 2007.

[63] M. Yim, Y. Zhang, and D. Duff. Modular robots. *Spectrum, IEEE*, 39(2):30–34, 2002.

[64] V. Zykov, A. Chan, and H. Lipson. Molecubes: An open-source modular robotics kit. In *IROS-2007 Self-Reconfigurable Robotics Workshop*, pages 3–6. Citeseer, 2007.

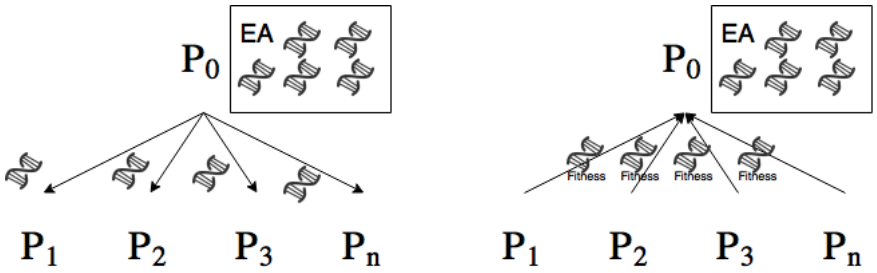
Appendices

Appendix A

Parallelization

In the experiments described in this study, the bulk of processing time is spent evaluating the genomes using roborobo. The genome evaluations are independent of each other and is therefore a good candidate for parallelization. With a great number of global variables, the roborobo framework itself is not easily parallelized. The solution was to use *Message passing interface(MPI)* to run multiple cooperating roborobo processes.

The parallelization is done by letting one root process take responsibility for running the evolutionary algorithm, with multiple slave processes for evaluating the genomes.



(a) Distributing the genomes from the root process to the slave processes for evaluation.

(b) Gathering the evaluated genomes from the slave processes.

Figure A.1: Distributing and gathering genomes.

At the beginning of each generation the root process generates the new genomes from the evolutionary algorithm. The new genomes are then distributed evenly to each process, see figure A.1a. Once all the genomes are evaluated, the root process gathers the evaluated genomes from the slave processes, see figure A.1b. The evaluated genomes are then used by the evolutionary algorithm to create the next generation. This process is repeated until

the target fitness is reached or a processing threshold is met.

Appendix B

Live graphing tool

Since the evaluation is quite time-consuming, there was a need for a tool that could give feedback during the simulation instead of having to wait for it to complete. A simple "live" graph which plots fitness statistics was therefore created, pictured in figure B.1

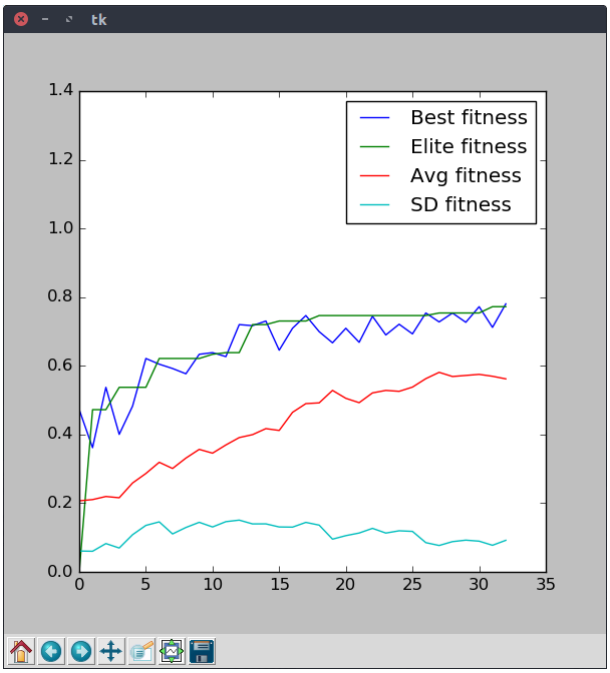


Figure B.1: The graphing tool showing fitness statistics for a trial.

Having such a tool makes it easier to evaluate how well new configurations are working, and saves time by shortening the feedback cycle.

Appendix C

Configuration & Code

The project code and system configuration can be found at https://github.com/christjt/ntnu-project-2016