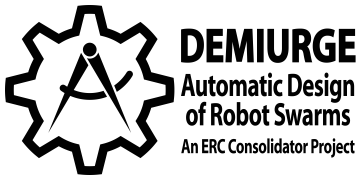
Faculteit Wetenschappen en Bio-ingenieurswetenschappen
Vakgroep Computerwetenschappen

A family of methods based on NEAT for the automatic design of behaviors of single robots and robot swarms

Proefschrift ingediend met het oog op het behalen van de graad van
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Abdallah Alfaham

Promotor: Prof. Mauro Birattari
Begeleiders: David Garzón Ramos



Juni 2019

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme: DEMIURGE Project, grant agreement No 681872

Contents

List of Figures

List of Equations

Abstract

Many research in the field of robotics claims that difficult missions with sub-tasks can be solved in a more feasible way through the collaboration of simple robots. That turns the challenge into how to build a controller for these individual robots and establish collaboration among them.

Different approaches are presented for this purpose, some are manual design and others are automatic. My work uses an automatic design with evolutionary robotics approach that uses neural networks as controllers. My method for evolving and producing controllers is NEAT.

It is challenging to understand the behavior of the neural network, however I still can work to improve its performance by tackling its components i.e. nodes and links if I have some clues about its input & output. This thesis provides a family methods that aims to improve the performance of the neural networks from different aspects by analyzing the components, the connections, and the functionality.

Keywords: NEAT, robot swarms, single-robot, LSTM, heterogeneous neural networks, topology.

Chapter 1

Introduction

In this thesis, I introduce a family of methods for automatic design of controllers applicable in two domains: single-robot and robot swarms.

Swarm robotics is an approach emerged on the field of swarm intelligence, where simple robots in the swarm interact and share information with each other in addition to their interaction with the surrounding environment. These interactions happen locally and they lead to create a global collactive behaviour. This collective behaviour can help to fulfill complex missions that an individual robot is not able to do (Brambilla, Ferrante, Birattari, & Dorigo, 2013). Automatic design of controllers for robot swarms is challenging due to their distributed natures and the local interactions between the robots. These interactions cause the swarms behaviour (Beni & Wang, 1993).

The distributed nature of robot swarms makes them capable of dealing with many issues that can occur during the execution. Some of those issues are the centralization and being a single point of failure. This occurs when all components of the system are not equally important. This leads to a breakdown in the entire system if one of its components fails, this in turn indicates a lack of reliability (Dooley, 2001).

In addition, the traditional software that are vulnerable to previous pitfalls can be attributed to a high cost and exponential complexity with lack of scalability. If the system suffers from the non-scalability that makes it unable to accommodate any growth in it regarding the existing functionalities or adding extra functions (Bondi, 2000). Therefore, using swarm would have two main advantages: it generates a collective behavior to solve complex missions using simple robots, as well as, avoiding such negatives because of its decentralized and self-organized nature.

On the other hand, the high uncertainty of the local interactions between the robots makes it difficult to design and define a specific behavior for individual robot. In fact, the literature is still lacking of a general methodology for designing collective behaviors, and to bridge the gap, plenty of methods are presented for designing collective behaviors (Brambilla et al., 2013).

Automatic design is a promising approach to design controllers for robot swarms. It proceeds by casting the assigned mission into an optimization problem with an objective function that needs to be optimized, and it is reflected in the behavior of robots. The size of search space is based on the definition of the objective function, and it is explored by employing an optimization method (Francesca & Birattari, 2016).

The family of methods in this thesis relies on evolutionary robotics approach to design and produce controllers automatically. The controllers in evolutionary robotics are neural networks, and they are evolved by using an evolutionary method (Nolfi, Floreano, & Floreano, 2000).

Each method I introduce explores and analyses the neural networks from a specific aspect: node structures, topologies, and activation functions. The family of methods that I propose uses a neuroevolution method called Evolving Neural Networks through Augmenting Topologies (NEAT) to evolve a neural network, and to produce an automatic design controller for the assigned mission. NEAT is a popular method used frequently in reinforcement learning missions, and it shows an outstanding ability to produce controllers with good performance. It also considered as one of the prominent method in neuroevolution field (Stanley & Miikkulainen, 2002).

The missions that are conducted to verify the efficiency of NEAT and to evaluate the family of the methods that I present are: T-maze experiments for single-robot, aggregation with two spots and surface and perimeter coverage missions for robot swarms.

1.1 Objective of the thesis

The objective of this thesis is to design and evolve neural networks automatically as controllers for single-robot and robot swarms missions.

The presented methods in this thesis try to enhance the general performance of the produced controllers by analyzing the neural networks and impact their inter-

nal mathematical computation during the design phase. The family of methods that I propose will use the evolutionary method NEAT for this purpose.

The need for more advanced computation during the operation is essential and important in complex missions especially those that use sequential data.

1.2 Main contributions of the thesis

I propose and evaluate methods for the automatic design of controllers. These methods control the flow of the received data inside the neural networks by the means of their components structure and the relationships among them.

I consider three aspects to seek differences in the performance of the neural network:

- The architecture of the neural network: New type of nodes will be embedded into the neural network in NEAT. These nodes are Long short-term memory LSTM (Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2017)
- The topology of the neural network: A novel way of connecting the nodes in the network will be introduced under the name R-evostick. This topology causes a delay in computation which in turn may help to use historical data for decision making.
- The functionality of the neural network: The activation function in the nodes of the neural network may play a significant role in its performance in which using more than one activation function helps to enrich the learning phase.

1.3 Structure of the thesis

The introduction chapter describes the context, the objective and the contributions of this thesis. Chapter 2 presents related work to the thesis. Chapter 3 gives information about the methodology with a detailed description of the family of methods that I proposed. Chapter 4 addresses the experimental set-up in my thesis . Chapter 5 shows the results of my case studies with a discussion about the performance. Chapter 7 concludes the thesis with some final remarks and a brief summary of the accomplished work. Chapter 6 provides some guidelines for future work based on the thesis contribution.

Chapter 2

Related work

In the last decades, plenty of research with different methods for automatic design of controllers in the field of robotics have been introduced with different approaches for creating robot behavior (Francesca & Birattari, 2016). In this thesis, I discuss studies aimed to design controllers for robot swarms and single-robot with the intention of illustrating the difference between automatic design and the manual design. This chapter focuses on research in literature that use the same methods i.e NEAT, with different types of neural networks.

2.1 Swarm robotics

Swarm robotics was established primarily through the observation of animals and insects that work cooperatively in groups e.g: swarms of birds and colonies of ants, wasps, and many other species in nature that have a the same characteristics in terms of decentralized and self-organised nature(Tan & Zheng, 2013). The swarm robotics system consists of simple constituents that can communicate locally, and interact with environment as well in order to emerge a collective global behavior (Beni & Wang, 1993). This swarm behavior which results from local interactions of simple components contributes to solve difficult missions (Brambilla et al., 2013).

2.1.1 The manual design of controllers

Solving a mission manually imposes the designer to think and deal with it as a single entity, making the procedure dependent on trial and error method. If the designer manages to develop a method for building controllers to solve the mission, these developed controllers will work only on the considered mission, and the method will fail to solve another missions.

The same idea can be found in the design of controllers for robot swarms. The main approach for that is called behavior-based and it works on the swarm-level to solve the task (Jones, 2004). In fact, this approach is inspired by observing the collective behavior of social animals, and it relies on the trial and error technique for building the controller without providing a specific guideline for the development of the individual robot behavior (Brooks, 1986).

Although, behavior-based approach is popular in literature, however, it is still accompanied with many obstacles such as time consuming along with lots of effort required because it primarily depends on the experience of the designer. Also, the lack of generalization, where the produced controller works only on the task that has been built for (Brambilla et al., 2013).

Some of the methods that are used to generate a controller for Foraging task are: Central Place Foraging Algorithm (CPFA) (Hecker & Moses, 2015) which is a group of hand-coded behaviors divides the foraging task into multiple states and transitions controlled by a set of parameters. Another algorithm is called Distributed Deterministic Spiral Algorithm (DDSA)(Fricke, Hecker, Griego, Tran, & Moses, 2016) and it exclusively search the arena looking for seeds by using the square search pattern. These two algorithms are compared to NEAT method in section 2.2.2.

In single-robot case, the manual design of the controller will be easier because there is no collective behavior resulting from local interactions between the robots like the case of the swarms, and the produced controller will be applied only on one robot.

The procedure for building a manual controller for single-robot missions depends also on trail and error method, and the same method that is designed to build controller for robot swarms can be used for single-robot in case the developed method does not include local interactions between the robots like Distributed Deterministic Spiral Algorithm (DDSA)(Fricke et al., 2016). As mentioned before the robot in DDSA method follows a certain pattern to search the arena, and since there is no local interactions taking place, therefore, the mission can be conducted with one robot. This is also applied in automatic design.

2.1.2 The automatic design of controllers

Due to difficulties and shortcomings in the manual design of controllers for robot swarms, there are several endeavors to generate a control software based on the

individual-level behaviors for robot swarms in an automatic way (Brambilla et al., 2013). The porminent approaches of automatic design of controllers are: evolutionary robotics that used to evolve a neural network by receiving the sensor readings of an individual robot as numerical values. These values represent the input of the neural network and then feed the values of its output into the actuators of the robot (Nolfi et al., 2000). This method can be applicable on a number of missions (Brambilla et al., 2013).

There are several types of evolutionary algorithms that use a genetic representation in their implementation (Vikhar, 2016). Each of these algorithms is applicable on certain types of problems. The most famous ones are:

- Genetic algorithm (GA): It is the common use in EA. It uses the traditional genetic operations such as recombination, selection and mutation (Davis, 1991). and usually strings with binary numbers are used for describing the problem.
- Neuroevolution (NE): It is close to genetic algorithm, but it rather uses a genome as an artificial neural network to solve the problem by modifying its parameters and its shape (Floreano, Dürr, & Mattiussi, 2008). One of its most prominent methods is NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002).

Another approach for automatic design of controller depends on the usage of probabilistic finite state machines (PFSMs) as in AutoMoDe (Francesca, Brambilla, Brutschy, Trianni, & Birattari, 2014). This approach selects and combines constituents behaviors from a set of preexisting parametric modules composed of states and transitions. The selection operation is done via an optimization algorithm. The two outstanding research in this approach are Vanilla AutoMoDe (Francesca et al., 2014) and AutoMoDe-Chocolate (Francesca et al., 2015).

The main difference Vanilla AutoMoDe and AutoMoDe-Chocolate is the adopted optimization algorithm. Some may criticize the fact that the repository of preexisting parametric modules contains only few states and few transitions, however, as a matter of fact, this small search space rather helps to produce a more efficient controller and also helps to handle a well-known issue called reality gap (Jakobi, Husbands, & Harvey, 1995).

The reality gap issue might happen when the produced controller in simulation is applied on real robots. Actually the reason for this is due to the fact that the simulator tries to emulate the real world but it does not exactly reflect it, and when I produce a control software by simulation it might exploit some aspects that are

not exactly the same in reality. Therefore, it will be better to add constraints or noise to the simulation environment to reduce the representational power of the controller.

It should also be noted that the developed control software by probabilistic finite state machines is readable by human unlike the controllers that is obtained by an evolutionary algorithm. The neural network is considered as a black box.

In evolutionary robotics and in any approach of automatic design, the definition of the objective function in the assigned mission is an important part that plays a major role in the behavior of the robot(s). The two highlighted ways to determine the objective function of the mission are: single objective optimization (SOO) which focuses on one goal to optimize, and multi-objective optimization (MOO) that has to deal with several sub-goals. The selection between these two approaches depends mainly on the nature of the mission (Trianni & López-Ibáñez, 2015).

2.2 The literature review of NEAT

NEAT is a powerful evolutionary method that uses the representation of the genome for generating the controllers. A more detailed presentation of NEAT can be found in Chapter 3.2. In literature, NEAT has been used frequently to solve different problems due to its performance.

2.2.1 NEAT in the design of neural networks for general purpose robotics

NEAT is largely applied for the single-robot, and with robotics arm for reinforcement missions, like the resarch that was attempting to solve a double pole-balancing problem by using two different versions of NEAT: the standard version of Neat, and the Feature Selective Neat (FS-NEAT) (Ethembabaoglu, Whiteson, et al., 2008).

The main difference between these two variants is the initial topology, where the standard version starts with a fully connected feed forwards neural network without any hidden layers. There is a method called Evostick uses the same type of neural network for swarms of e-puck robots (Mondada et al., 2009). It is been used in AutoMode research (Francesca et al., 2014) (Francesca et al., 2014) (Francesca et al., 2015), whereas, FS-NEAT starts with a simpler structure, where it considers the full connection as an added complexity, instead it starts with a topology that contains only one input connected to the output.

FS-NEAT topology is sensible, but since both structures have the ability to evolve and grow, it may end up with a trade-off between the complexity of NEAT which has larger search space, and the time consuming for evolving the structure of FS-NEAT. In other words, it may take time in both sides: for evolving the simpler structure and for finding a solution in the more complex structure.

It was expected when the authors made certain modifications on some parameters of NEAT by increasing the possibility of mutation on the structure in order to grow rapidly. Nevertheless, the results were not exactly as they expected. They divided the processing stage into two phases according to the structure, in the first phase, they expected better performance with the standard NEAT because of its structure, and in the second phase, they expected FS-NEAT to outperform NEAT after evolving its structure. However, The behavior of FS-NEAT in the second phase was contrary to expectations. Therefore, I can draw a conclusion that the sound evolution of the topology is not always guaranteed, and this could be attributed to the trade-off that I mentioned.

NEAT has been also combined with different methods to seek better performance. The most famous method that is mixed with NEAT is Novelty Search (Lehman & Stanley, 2008). The main idea behind this method is to reward the novel solutions instead of trying to optimize the current solution towards the fixed goal. The purpose of Novelty Search is to avoid convergence towards a local optimum. This well-known problem is called *deception* (Goldberg, 1987), and it can be a challenging problem in genetic algorithms. Therefore, the solution that has been provided by Novelty Search is by rewarding solutions according to their uniqueness compared to each other.

(Lehman & Stanley, 2011) used NEAT with Novelty Search to solve two reinforcement missions: Biped Walking experiment and Maze experiment. Each experiment is run with different budget i.e. different parameter settings in NEAT, especially the population size which represents the number of organisms in the evolving stage. It is worth mentioning that there are no fixed rules for setting the budgets. Most of them were assigned based on the difficulty of the mission and the experience of the researchers. Nevertheless, it should be noted that the use of different parameter settings indicates human intervention in the design, and that makes the design not fully automatic (M. Birattari et al., 2019)

For Biped Walking experiment, the robot is controlled by an artificial neural network that receives the input from a ground sensor placed in each foot. The objective is that the robot must travel the longest possible distance from the initial

position. Using Novelty Search may result in benefits by avoiding deceptiveness cases that might arise because of the difficulty of the problem (Goldberg, 1987)

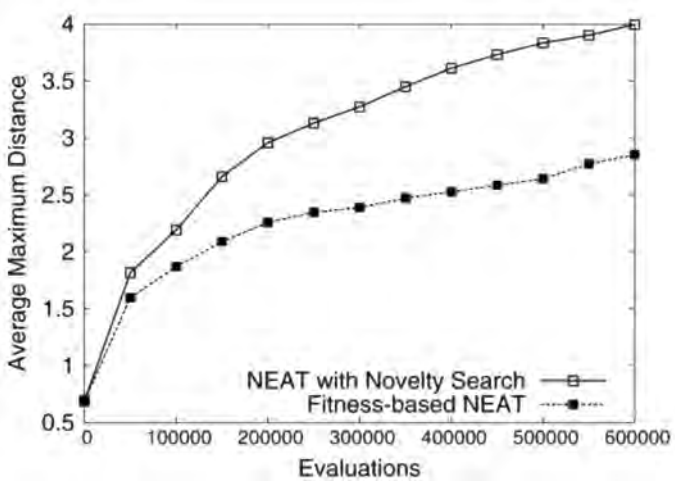


Figure 2.1: *The results of the biped experiment.*
(Lehman & Stanley, 2011)

The importance of Novelty Search lies in its rapid convergence towards the goal. However, if the problem can be well-handled by the standard NEAT then the final result of NEAT will be very close to NEAT with Novelty Search as shown in the result of the maze experiment with medium map.

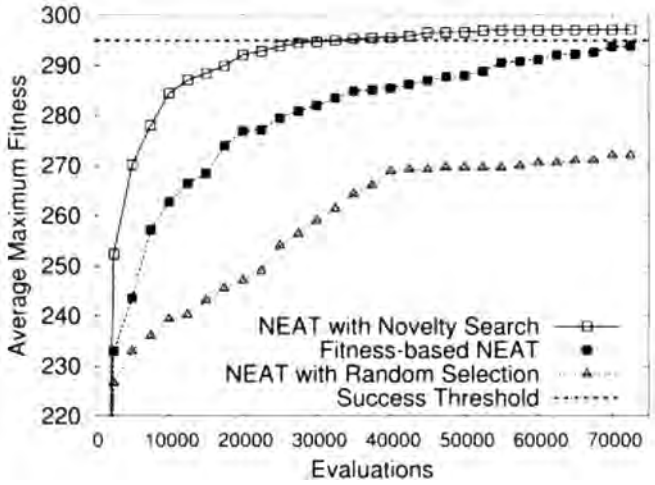


Figure 2.2: *The results of the maze experiment.*
(Lehman & Stanley, 2011)

2.2.2 NEAT in the automatic design of controllers for single-robot and robot swarms

Designing an automatic controllers for robots swarms by using evolutionary algorithms is very common (Francesca & Birattari, 2016), especially with Neuroevolution methods where NEAT method becomes very popular for automatic design, and it is considered the most likely method that produces effective controllers with good performance (Stanley & Miikkulainen, 2002).

The work of (Ericksen, Moses, & Forrest, 2017) manifests the advantage of automatic design by NEAT for robot swarms over other approaches. The conducted mission was foraging, and the features that the authors highlighted and discussed in their work were the adaptability to scalability as well as generalization. It should be noted that foraging mission is not trivial, it includes a series of subtasks that need to be fulfilled in order, namely, searching the environment looking for the seeds, picking up the seeds from the source, deliver them in the nest, and cooperate with other robots for emerging the collective behavior taking into account the awareness of the collisions.

The authors used three approaches to solve the mission: Central Place Foraging Algorithm (CPFA) (Hecker & Moses, 2015), Distributed Deterministic Spiral Algorithm (DDSA) (Fricke et al., 2016), and NeatFA (NEAT Foraging Algorithm).

For testing the generalization, they evaluated NeatFA on three different kinds of environment according to the distribution of the seeds in the environment: Random, semi-clustered, and clustered. The results showed the ability of NeatFA to adapt to the environmental conditions without having to change the implementation or use any hand-coded algorithms.

To test the scalability, several experiments were conducted with different size of swarm starting with two robots until 30 robots. The performance of NeatFA and CPFA were comparable, while DDSA performance decreased because of the crowding at the nest, since DDSA uses square pattern for exclusive searching centered at the nest, there is high possibility to have crowding issue at the nest when the size of the swarm increased, thus, the performance will declined. It might be difficult to improve DDSA without changing its way of searching to avoid this issue. This reflects a major pitfall in manual approaches.

The power of any approach comes from its ability to adapt to various circumstances and the ability to deal with different missions. And if these qualities are guaranteed in multiple methodologies it would be preferable to choose the less

complex one.

As the case in the research (Gomes, Urbano, & Christensen, 2012) where the authors introduced Novelty Search with NEAT and compare it with the standard Fitness-based NEAT to solve the aggregation mission. In this mission, the robots roam the environment and cluster with each other until they end up with one aggregate. According to their performance, Novelty Search actually helps in swift convergence towards the ultimate goal, nevertheless, Fitness-based NEAT was eventually able to fulfill the objective, and their final results were comparable. Hence, in any mission, my focus must be on the nature of the mission, as well as, on the capability and the complexity of the approach that can be used to solve it.

In fact, It should also be highlighted that most of the experiments were conducted in simulation, and even when they were conducted on real robots they were executed within conditioned environment in laboratories, and that is still far from real world. Few researchers took a step forward, and they presented their work in real world environment such as (Duarte et al., 2016). The authors of this research used the aquatic environment to execute the swarm robotics experiments:

> Homing: navigation to a given place while avoiding the obstacles i.e. other robots.
>
> Dispersion: maintain a fixed distance between the robots to cover the largest area on the water without losing contact.
>
> Clustering: to assemble the aquatic robots in one aggregate.
>
> Monitoring: monitor certain areas by setting boundaries to determine the area of interest that should be covered by the robots.

The method for generating the controller was NEAT, and it applied on aquatic robots equipped with the following features: Wi-Fi for communication; Compass and GPS for navigation. notwithstanding, some of these features are violating the swarm robotics principles (Dorigo, Birattari, & Brambilla, 2014). For instance, GPS could provide information beyond the local domain. However, their obtained results in the real environment were very good with decent adaptation in terms of scalability and robustness. These research help us to draw a conclusion that NEAT in the automatic design of controllers represents a promising technique for robot swarms.

2.3 Recurrent neural networks in robotics

Recurrent neural networks (RNNs) are largely used in machine learning and they have an advantage in terms of capability over the standard simple neural networks due to the use of historical data from the previous states (Rumelhart, Hinton, & Williams, 1985). There are many variants of RNNs based on the architecture of their units or based on the topologies i.e. the connections among the nodes.
In this thesis, I will focus on the use of these two categories in robotics.

Elman is a one of the recurrent neural network variants that is designed to learn sequential time-varying data. Elman contains an extra layer called context layer and it connects to the hidden layer with recurrent relation (Elman, 1990).

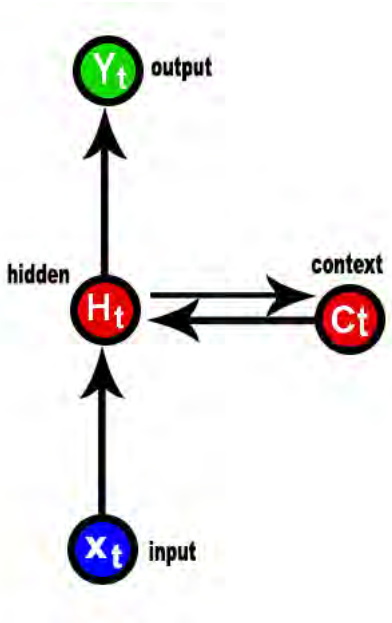


Figure 2.3: *The structure of the Elman unit.*

The hidden node H_t receives data from the input node X_t, and the context node C_t. The complexity of the Elman increases by adding nodes in the context layer which requires new weights to connect with the hidden layer.

There are some endeavours to use Elman in swarm robotics such as (Gross & Dorigo, 2009). The objective is to move and transport an object to a specific target by taking advantage of the self-assemble principle in swarm robotics. The authors used an evolutionary method for evolving the neural network. The evolved neural network will independently control the robot in terms of: looking for the object to

transfer. Deciding where to move, and where to connect with other robots. In the experiment, different number of robots -up to 16 robots- were used to form the swarm.

The advantage of the evolutionary method becomes clearer when the size of the swarm increases, otherwise, with manual method, the number of the rules to control the swarm will increase and that makes the controller more complicated. During the experiment, the robots behaved in two different way, whether they connected with each other and transferred the object or they surrounded the object and pushed it to the target location without being physically connected to it. The results of the experiment were vary based on the properties of the object e.g. weight, shape, and height. The methodology of the authors showed the ability to deal fairly with the increased number of the robots. However, since the robots in the simulator were not subject to individual failure. This can be a serious limitation for evolving the controller.

Another variant of recurrent neural network is Long short-term memory (LSTM), and it was proposed (Hochreiter & Schmidhuber, 1997) to deal with the exploding and vanishing gradient problems (Hochreiter, Bengio, Frasconi, Schmidhuber, et al., 2001). LSTM becomes popular and it has been used frequently in deep learning field due to its ability to preserve the historical information over a long period of time and use it later for decision making. This ability comes from the special architecture of its unit which equipped with multiple gates to control the flow of data (Greff et al., 2017).

LSTM can process a sequence of data unlike the simple feedforward neural network. The simple standard neural network deals only with single data input. LSTM works to establish a relation among the elements of the input or between the input and the output e.g. regression problems. However, it can also be applicable on classification problems (Graves & Schmidhuber, 2005). Some research in literature attempt to combine NEAT with LSTM to improve the performance as in the research (Rawal & Miikkulainen, 2016). the purpose for this combination is to enhance the capability of recall. The authors introduces a novel unsupervised optimization method that maximizes the data stored in the LSTM network, and it is called Info-max. They used this method in order to overcome the challenges associated with LSTM. In fact, the architecture of LSTM unit makes it vulnerable to deceptive solutions due to the number of new embedded weights that make the search space for solution too large.

The authors conducted two experiments: sequence classification, by receiving two kind of input -1 and 1, and the input could be interleaved with 0's for dis-

traction. The produced controller should determine which input has been received more. The other experiment is sequence recall like the T-maze experiment, and the agent should turn towards the right direction according to the ordered instructions received at the beginning of the experiment. Their methodology for producing a controller splits the design stage into two phases. One for evolving LSTM units by using the unsupervised method Info-max, and the other phase is for evolving the simple nodes by using NEAT based on the objective function.

It is reasonable that the authors with their method tried to cope with the complexity of LSTM units, moreover, this method might not be applicable on every platform, for instance, in case of using a simulator, it could be difficult to split the design phase into phases unless that is done manually i.e. run the design phase multiple times and each time only parts of the system will be affected based on constraints set by the designer. However, this human intervention affects the principle of developing the controller automatically.

There is also a simple observation regarding the information provided as input to the neural network. In the sequence recall experiment. The neural network has two input: the ordered instructions as a guide for the desired destination and the distance from the T-junction.

It seems as if the authors fed the neural network with the maze itself by providing information about the position if its T-junctions, and instead of letting the agent learns through exploration, it rather makes the agent memorizes the maze beforehand. That even violate one of the principles in swarm robotics by receiving global data if the corridor in the T-maze is too long, and the agent cannot observe the T-junction locally. The results of the study were reasonable based on the used structures and it reflected the importance of handling the complexity of LSTM.

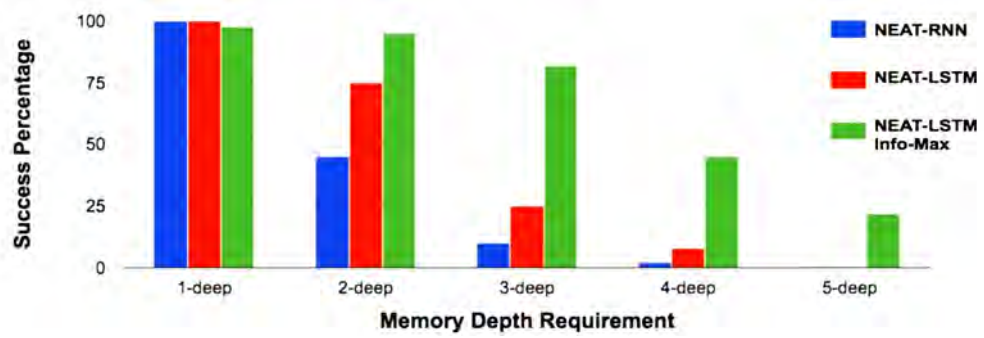


Figure 2.4: *The results of sequence recall experiment.*
(Rawal & Miikkulainen, 2016)

Mazes are well-known experiments for single-robot case. There are plenty of research use this case of study to train a robot for exploring and decision making through reinforcement learning. (Jakobi, 1997) is one of the early research that used T-maze experiment with Khepera robot (Mondada, Franzi, & Guignard, 1999) . The main topic of the research is about the reality gap issue, nevertheless, I are still interested in the set-up of the experiment.

The authors applied a recurrent neural network with two kinds of input: ambient light as a guide to the correct path and IR sensors to avoid obstacles. The neural network used two different activation functions: one for the output that feed the actuators to move, and the other for the non-motor neurons. With a fitness function sums the traveled distance of the first corridor -Y axis- and the second corridor -X axis- plus 100 reward for the correct path. The trials had been carried out many times, and the results showed the ability of Khepera robot to learn and choose the correct path successfully.

This successful attempt encouraged other researchers to reproduce the mission and to compare the results like the research (Koos, Mouret, & Doncieux, 2013). In this research, the authors used the e-puck robot (Mondada et al., 2009) and they tried to reproduce the same T-maze experiment with a some modification on the set-up i.e. the implementation is slightly different and the color sensors is used instead of the light sensors. The authors attributed these modifications to the lack of detail in the set-up of the original work and because of the use of a different kind of robot.

Even the reward in the fitness function was altered and became 1000 for the correct path. This could be related to the size of the maze that they set. The result

outperformed the one in the original work, however, it is still questionable whether the produced work actually reflects the original one.

Since I use the same type of robot namely e-puck (Mondada et al., 2009), it might be a good idea to reproduce the T-maze experiment of (Koos et al., 2013) to test the NEAT capability. Although, some information in the research was not mentioned regarding the design phase like updating the weights, however, I still can reproduce the experiment because I only need information about the fitness function, the structure of the neural network, and its input & output. Further information about the reproducing experiment can be found in chapter **??** .

There are other research use different strategies to deal with the T-maze experiment like the research (Blynel & Floreano, 2003). The authors used continuous time recurrent neural networks (CTRNNs) to avoid the modifications of synaptic strengths i.e. the weights by using for example back-propagation or Hebbian learning. Instead, the learning process is done by exploiting internal network dynamics.

The neural network is fed by data about the rewarded-zone using the floor sensor, in addition to IR sensors to avoid the obstacles. The authors applied their methodologies using a T-maze and parts of double T-maze. The results showed the ability of CTRNNs to learn and utilize the environmental information to create a behavior from the internal computations in the neural network.

Chapter 3

Methodology

In this chapter, additional information is mentioned regarding the methods that have been used, and detailed information about the automatic design with different kind of methods.

3.1 Automatic design

Automatic design is a promising approach to design controllers in robotics. Its viability came from the idea of casting the considered mission into an optimization problem and build an objective function for it to guide and optimize the performance. The state of the art in automatic design for robot swarms is yet undefined properly in literature. There are some achievements that help to make progress like having different approaches for the automatic design: Off-Line methods, On-Line methods, however, there are challenges reflected in the lack of appropriate comparison among the published works obstruct that progress (Francesca & Birattari, 2016).

There is a tread-off between On-Line and Off-Line methods in automatic design: in Off-Line methods the design phase occurs before distributing the swarm in the operational environment. Usually, the evaluation is performed in simulation. The advantages of Off-Line methods lie in avoiding damages could happen in the real robots during the design phase, and they also consumes less time in evaluation. While in On-Line methods, the design phase takes place when the swarm has been distributed in the operational environment. These methods can benefit from extra information provided in the operational environment which can be reflected in the produced controllers. Nevertheless, the approach can suffers high time-consumption and it might risk and cause damages to robots unless removing the candidate design that can be dangerous. It should also be noted that remov-

ing candidates and instances from the design phase can reduce the search space of the optimization method which may affect the general performance as well (Francesca & Birattari, 2016).

3.2 Evolving Neural Networks through Augmenting Topologies (NEAT)

In Neuroevolution (NE), NEAT is a promising alternative for reinforcement learning tasks that relies on genetic algorithms for evolving and training the neural networks (Stanley & Miikkulainen, 2002). The sound stages and the concrete demeanor of NEAT make it powerful. It starts with an initial simple structure as a starting genome that evolves and becomes more complex during the training session. The artificial neural networks will be considered here as genomes, and during the design phase NEAT will create multiple generations contain species with a fixed number of organisms based on its population size parameter, each of these organisms carries a genome as a fingerprint that has a specific topology with specific weights.

Every organism in the species will be executed on a the same number of experiments to measure its performance and to update the weights of its genome. After the execution phase, the organisms that scored the highest fitness will be elected according to a elitism percentage parameter to the next generation without any modifications. While the remaining organisms will be subject to certain procedures based on the probabilistic values of the parameters in NEAT. These procedures are for example: mutating weights, adding new links or nodes, and making crossover operation between two genomes.

Applying crossover operations between any two random genomes can cause problems and lead to neglecting some important data if the compatibility ratio between the two genomes is low.

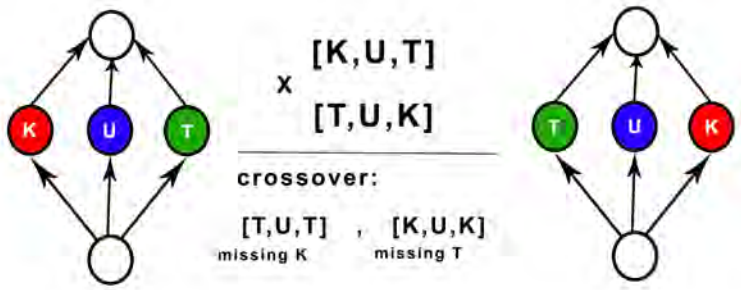


Figure 3.1: *Incomplete crossover operation.*

There is incompatibility between the two previous genomes. The two genomes samples have three genes K, U, and T. There is only one matched gene between them U. This incompatibility causes damage to the resulting offspring, and drop important data presented in the parents after the crossover operation.

Therefore, crossover operations must only take place among the genomes that share high compatibility. Otherwise, crossover operation might produce damaged offspring that could miss essential information contained in the parents as a result of the incompatibility. This issue leads to Competing Conventions problem (Montana & Davis, 1989) (?, ?).

NEAT avoids such a problem by measuring the compatibility distance which is based on the matching genes, the disjoint genes, and the excess genes of the genome. The matching genes are the only shared genes between the two genomes, and the more matched genes they have, the better results will be obtained from the crossover operations, and the produced will be offspring more meaningful. The following figure illustrates these genes inside the genome.

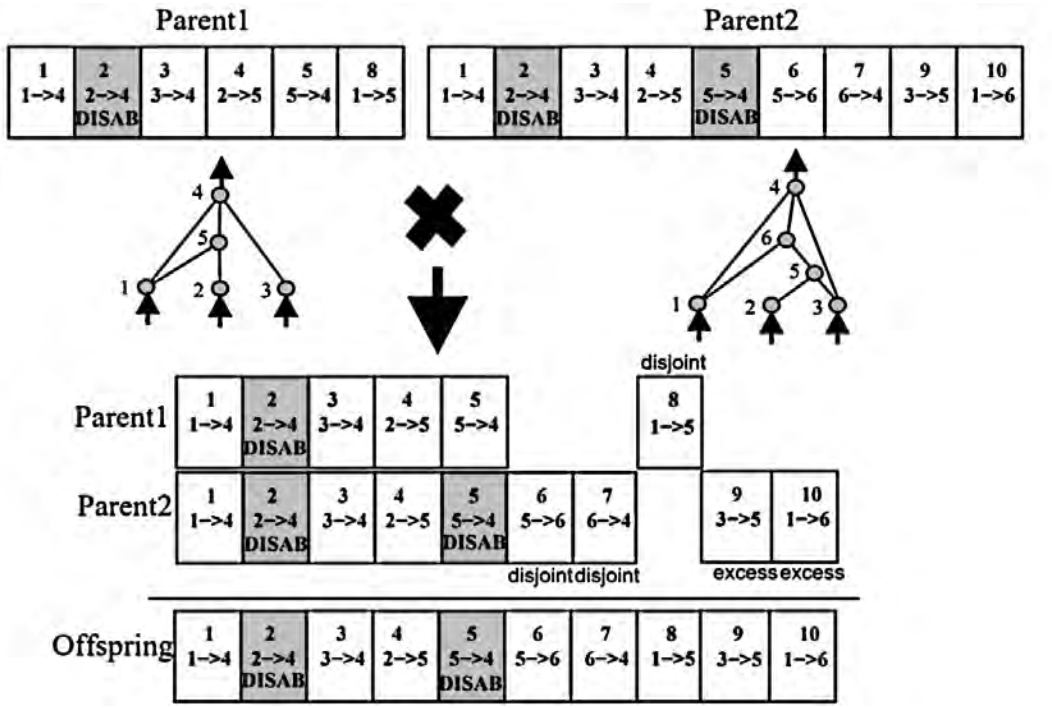


Figure 3.2: *Analysis of genome structure during the crossover operation.* (Stanley & Miikkulainen, 2002)

The two genomes: $Parent1$ and $Parent2$ share the first five genes. The genes at end of the longer genome $Parent2$ are excess genes: gene number 9 and gene number 10. The rest of the genes that take place between the matched genes and the excess genes, and contained in one parent are disjoint genes: from gene number 6 until gene number 8.

In other words, if the representative structures of two genomes match up they can execute crossover operation without worrying about Competing Conventions problem. This highlights the importance of this measurement, where the large number of disjoint and excess genes make the genome less compatible with other genomes.

Usually, The initial fitness of the newly evolved structure is often diminished, this is due to the fact the new added component has not yet adapted to the body structure, and it needs some time to have the chance for adapting the new weights in order to improve the performance. Otherwise, if this matter is not taken into consideration, most of the newly evolved structures will be neglected due to the

loss of fitness.

In NEAT, that is handled through protecting innovation with speciation. Every generation consists of multiple species and if the compatibility between the two organisms is low they will be stored in two different species. And then after a certain period of time the stored organisms will be removed from the species if there is no improvement in their fitness.

Based on this method, I note that the complexity of the neural network is dynamic and depends mainly on the complexity of the mission. Many research have used this method to solve missions in the field of reinforcement learning. And due to the fact that it is difficult for manual methods to deal with these kind of missions that have unstable nature, therefore, Neat is a promising alternative in this field.

3.3 Types of neural networks

This section provides detailed information on the structure of the general recurrent neural network in addition to the structure of its special case Long short-term memory.

3.3.1 Reccurent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) become a benchmark in artificial intelligent. They have been used widely in many different domains especially in deep learning to promote the capability and create sort of memory in the computation through the training session (Rumelhart et al., 1985). These special structures lead to increase the performance, especially in sequential data applications. Some notable applications use RNNs are: Speech recognition (Sak, Senior, & Beaufays, 2014) and Handwriting recognition (Graves et al., 2009).

Neural networks are actually black boxes use matrices of weights for learning. Therefore, I cannot have a full understanding about their behaviors, nevertheless, the developer should at least internalize about the flow of the data inside the network in order to set meaningful connections among the nodes.

What distinguishes RNNs is the use of the hidden state that affects the decision in its activation function, unlike the simple feedforward neural networks. In the simple neural networks, the activation function depends only on the current input with attached weights, while in RNNs, the historical information i.e. the previous

hidden state has been also used for computing the next hidden state, which makes the computation more logical and more consistent by forming a memory-like behavior, and that makes RNNs a useful technique for learning sequential data.

The activation function of standard recurrent neural networks:

$$W_t = Func(W \times [h_{t-1}, x_t] + b_0)$$

, where $Func$ can be sigmoid , hyperbolic tangent, or any activation function.

Recurrent neural networks try to imitate the thinking process of the human being by using information and experiences from the past to take a decision.

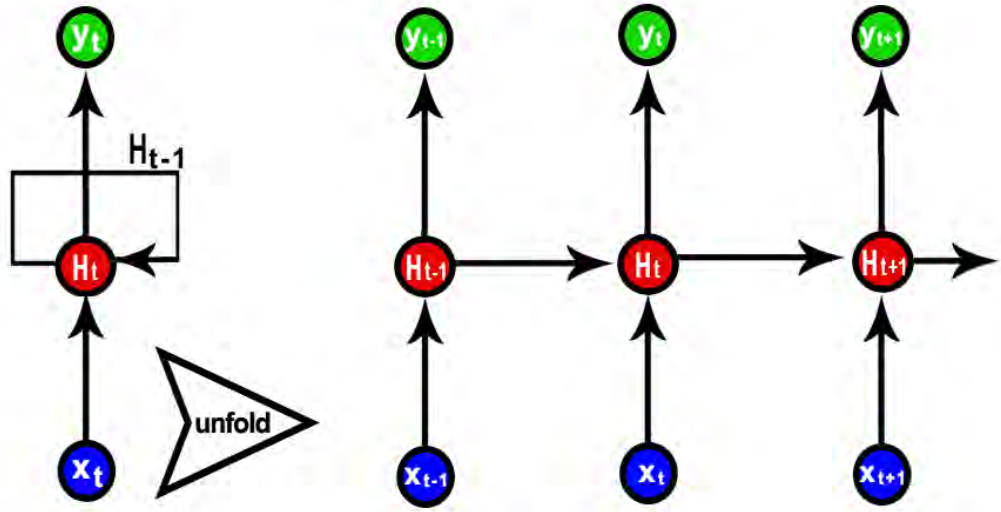


Figure 3.3: *Unfolding the structure of RNN .*

The unfolding process of the recurrent structure shows the dependencies between the data. For instance: H_{t+1} depends on the input X_{t+1} and on the previous

state H_t. In the same way H_t depends on its input X_t and on the previous state H_{t-1}

Although RNNs with their distinctive structures outperform other simple networks, still there are disadvantages that may limit their performance. The issue might occur in the back-propagation phase, resulting in a lack of memory capacity, especially when dealing with long-term dependencies of consecutive data. Where after computing the predicted value through the feedforward phase, there must be an algorithm to provide some orientation and guidance to update the weight matrix based on the error rate between the predicted value and the expected one. This process takes place in the back-propagation phase by computing the gradient. However, if the value of gradient used to be less than one, it will continue to decline as I move backward until it ends up with a value very close to zero which leads to vanishing gradient problem. On the other hand if the value of the gradient is greater than one I will end up with exploding gradient problem (Hochreiter et al., 2001).

Traditionally, to avoid exploding gradient problem a technique called gradient clipping has been used to rescale gradient if its norm is too big (Pascanu, Mikolov, & Bengio, 2013). Nevertheless, I cannot cope with vanishing problem by using the same standard structure, hence, a new structure called Long short-term memory (LSTM) has been developed (Hochreiter & Schmidhuber, 1997).

3.3.2 Long short-term memory (LSTM)

In LSTM, the architecture is more complicated than standard recurrent neural networks, and it may increase the nonlinearity of its behavior. On the other hand, it can overcome the previous problems that might encounter the general RNNs.

The node in LSTM has multiple gates with attached weights that control the flow of data rather than relying solely on the previous state as in the general RNNs. Here there is another hidden state called cell state C_t keeps inside the LSTM and it is not exposed to the outside world. This advanced architecture will for sure increase the search space during its operation due to the new added weights that need to be adapted.

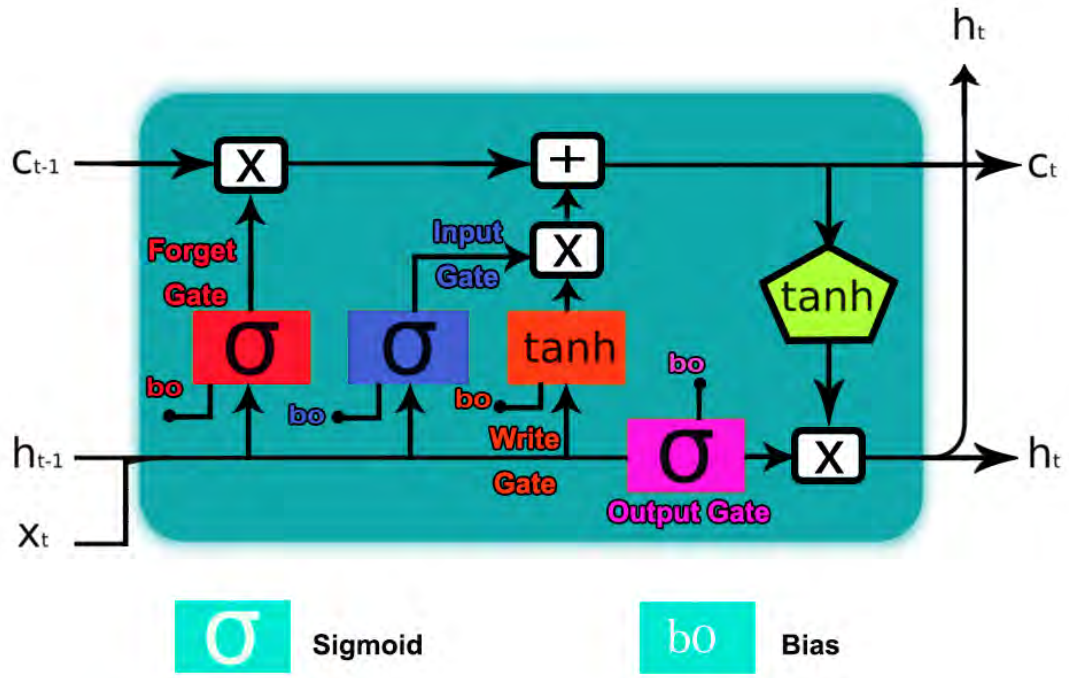


Figure 3.4: *LSTM node structure.*

There are different versions of LSTM regarding the relations among the gates e.g peephole LSTM, nevertheless, they almost work in a similar way (Greff et al., 2017).

The common activation function of the gates is the sigmoid function, and for better understanding, we could think of it as strings of binary values zeros or ones refer to the obsolete data and the preserved data respectively.
All the gates receive the same data i.e the input X_t as well as the previous state h_{t-1} and they work as follow:

Forget Gate: helps to make decisions about the data that should be neglected and the data that should be saved.

$$F_t = \sigma(W_f \times [h_{t-1}, x_t] + b_0) \tag{3.1}$$

Input Gate: in the Input Gate, new data from the current input will be added to the

current cell state which scaled through the multiplication with the write Gate.

$$I_t = \sigma(W_i \times [h_{t-1}, x_t] + b_0) \tag{3.2}$$

Write Gate: decides how much to write to the cell state.

$$W_t = \tanh(W_w \times [h_{t-1}, x_t] + b_0) \tag{3.3}$$

Cell state: the additional hidden state as local memory in LSTM

$$C_t = F_t \times C_{t-1} + I_t \times W_t \tag{3.4}$$

Output Gate: this gate decides how much to reveal the cell to the outside world.

$$O_t = \sigma(W_o \times [h_{t-1}, x_t] + b_0) \tag{3.5}$$

Finally, the new hidden state h_t will be calculated as follows:

$$h_t = O_t \times \tanh(C_t) \tag{3.6}$$

It should be noted that increasing the complexity might not always improve the performance. The challenge would be to find an efficient way to exploit this big search space.

3.4 Methods

As mentioned in the introduction, the family of methods in this thesis attempt to improve the neural network performance from three different aspects.

These aspects are:

- Complex architecture: By Using LSTM nodes. The challenging part in this aspect would be on how to deal with the complexity of LSTM nodes in order to reduce the search space.
- Topology: I introduce a new structure called R-evostick that causes a delay computation inside the neural network which in turn could create dependencies in data.
- Functionality: By producing a heterogeneous neural network HA-NEAT with multiple activation functions as an in an effort to reduce the size of the neural network compared with the homogeneous networks in the standard NEAT.

3.4.1 Using a Complex Structure: LSTM

As mentioned earlier, the biggest challenge that could be faced with advance complex architectures is how to deal with their complexities more than how to build them. Otherwise, the arbitrary use of them will lead to bad performance.

Due to the fact that NEAT is an offline method and there is no back-propagation phase to update the weights, thus, I need to find a smarter way to update them not just by adding a Gaussian noise to the weights and elect the organisms of the best performance.

It might be a good idea to take advantage of the high-performance organisms in NEAT to reduce the search space of other organisms while updating their weights

This simple algorithm is called $Wise\ Gaussian$ and it updates the weights of the organisms based on best one in the generation:

$$W_{new} = W_{old} + [(Fitness_{Max} - Fitness_{Current}) \div (Fitness_{max} + \alpha)] \quad (3.7)$$

,where α is a small number to avoid dividing by zero

After inserting LSTM nodes in NEAT, there will be two kinds of weights: Weights of LSTM nodes and Weights of NEAT links. To give the chance for these weights to adapt to each other, I added an extra parameter with a low percentage in NEAT to update LSTM weights.

$$mutate\_lstm\_weights\_prop = 0.3$$

Finally, the last modification to reduce the complexity of LSTM nodes was rescaling their output into Sigmoid function.

The standard activation function of the LSTM output is hyperbolic tangent

$$\tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.8)$$

with range $[-1, 1]$ while the Sigmoid function

$$\sigma = \frac{e^x}{e^x + 1} \quad (3.9)$$

has the range $[0, 1]$ Using the shorter range might help to reduce the search space during the design phase.

I preserved the convention of LSTM by using the same formula with tanh function, except I rescaled its output value into sigmoid value. For rescaling the hyperbolic tangent, and based on the equation between hyperbolic tangent and sigmoid:

$$\tanh(x) = 2\sigma(2x) - 1$$

, hence the sigmoid value will be

$$\sigma(x) = \left[\tanh\left(\frac{x}{2}\right) + 1\right] \div 2 \tag{3.10}$$

These steps helped to improve the adaptation of LSTM with NEAT which was reflected in its performance as I shall see later.

3.4.2 Introducing a novel topology: R-evostick

The links among the nodes in the neural network play an important role in the neural network behavior. Hence, the performance of the same size of neural network after the design phase would vary according to its topology.

It could be possible to take advantage of the topology to build a neural network with delay in its computation. In which it will have some senses on previous data.

R-evostick is an upgraded version of Evostick. R-evostick causes a delay in the computation by passing a copy of the input data into an intermediate node before passing it again to the output node.

The figure 3.5 illustrates the differences between Evostick and R-evostick in a simple structure contains one input and two output.

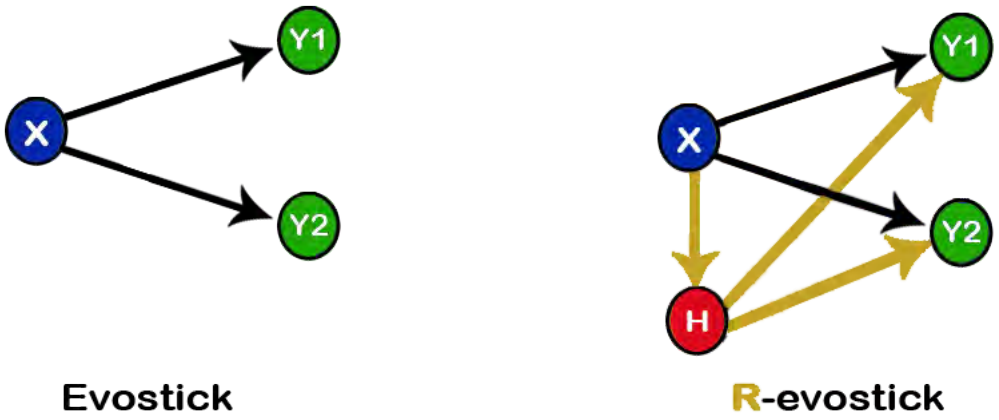


Figure 3.5: *Evostick vs R-evostick*

In R-evostick topology the output receives the data from two branches:

> Direct branch: It will take one step to transfer the data from the input to the output like Evostick from the input X to Y_1 & Y_2.
>
> Indirect branch: It will take two steps to reach the output through the new hidden node H.

This topology can be useful in missions that use sequential data. R-evostick can also have different versions according it its depth of delay. For instance R-evostick$\times 3$ signifies that the longest indirect branch for receiving the data needs three time steps.

3.4.3 Heterogeneous Networks: HA-NEAT

Unlike homogeneous neural networks with single activation function, heterogeneous neural networks equipped with various activation functions in their nodes. The idea is inspired from the paper (Hagg, Mensing, & Asteroth, 2017).

The objective from the heterogeneous networks is exploit the combination of different activation functions to evolve neural networks having smaller size i.e. fewer number of nodes and links with the same capabilities of homogeneous networks.

The method for evolving the neural network is NEAT, and the used activation functions are:

- Sigmoid Function:

$$\frac{e^x}{e^x + 1} \tag{3.11}$$

- ReLu Function:

$$Max(0, X) \tag{3.12}$$

- Step Function:

$$\begin{cases} 0, & X < 0 \\ 1, & X \geqslant 0 \end{cases} \tag{3.13}$$

- Gaussian Function:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3.14}$$

The figure 3.6 shows these activation functions for better understanding their behavior, where combining them could lead to complex activation functions.

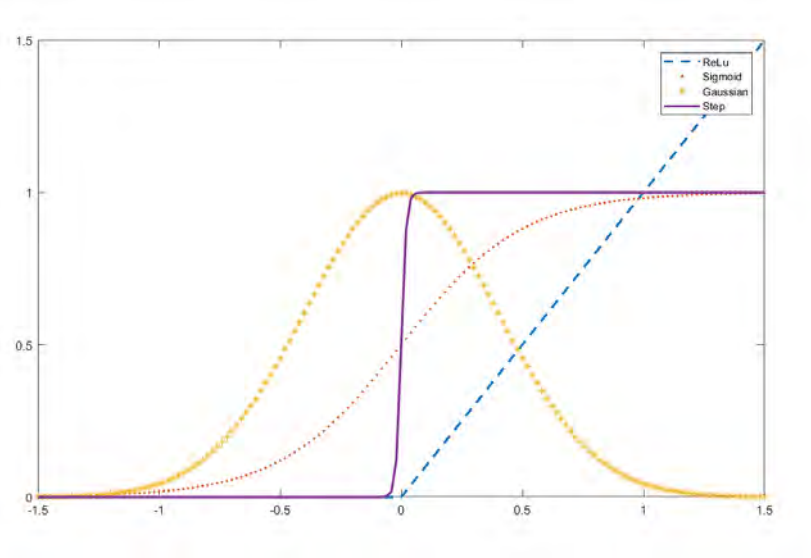


Figure 3.6: *A group of mathematical functions for heterogeneous networks*

The function hyperbolic tangent was not used because it has different range than previous functions. $Tanh$ ranges between $[-1, 1]$. While the range of used functions is positive, and they share the minimum value which is zero.

The implementation of the node should be Extended to add information about its activation function, and the mutation of the activation function should happen once in one node per genome based on a new parameter added in NEAT

$$mutate\_activation\_func = 0.5$$

The value of The parameter $mutate\_activation\_func$ is set to 0.5 to give the opportunity to all node in the genome to be selected. Whereas, if it is set to 1, then always the first node in the genome will be mutated. mutated genome will be protected through innovation, and it will be stored in a new species to give the newly genomes enough time to adapt to their new activation function.

To compare my methods with other contributions, I chose three different methods have been used in literature: Evostick with a simple full connected feedforward neural network, Elman with a context layer that has recurrent connections with the hidden layer, and finally the standard NEAT. Evostick & Elman have fixed topology while NEAT can evolve the structure. These methods were chosen because their behavior and their characteristics are close to the methods that I developed: LSTM & R-evostick have fixed topology while HA-NEAT can evolve the structure.

Chapter 4

Experimental set-up

4.1 Simulator

ARGoS is a multi-physics simulator widely used for robot swarms experiments, and it supports different types of robots(Pinciroli et al., 2012). Argos allows using large swarms in simulation unlike most other simulators. It also allows to use multiple physics engines for the simulation.

ARGoS can employ e-puck robots easily through the plugin *argos3-epuck*. The version used for this thesis is 3.0.0-beta48.

Building an experiment in ARGoS needs:

1. Configuration file (.argos) includes the setting and the properties of the operational environment.

2. The source code: contains information about the controller i.e the input data from the sensors readings and the output data feed the actuators of the robot. The source code also includes the methodologies that have been used for solving the mission.

3. The loop function: includes the objective function of the mission, where it casts the experiment into an optimization problem.

The programming language that I used in ARGoS simulator is C++
The missions are conducted with a predefined budget of 50.000 executions per mission.

Parameter	Value
Population size	100
number of generations	40
number of runs per generation	10
number of post evaluations	100

4.2 Robot

The type of robot that has been used in this thesis is e-puck robot. The e-puck is an small wheeled robot developed for research and education purposes (Mondada et al., 2009) and has been upgraded by (Millard et al., 2017). E-puck is a popular robot and it is used by wide community throughout the world.

The list of the sensors and the actuators that are available in e-puck:

- Sensors:
  - 8 light sensors located around the body of the robot to detect light density.
  - 3 ground sensors that detects gray-scale color variations on the ground.
  - 8 proximity sensors placed around the body of the robot to sense obstacles.
  - An omni-directional camera on top of the robot that provides 360° view.
- Actuators:
  - 3 LEDs around the robot to produce different colors.
  - The motors of the 2 wheels that can generate a speed from 0 to 12 cm/s independently of each other.
  - A range-and-bearing device for local communication between the robots

4.3 Case of study A: single-robot

4.3.1 Reference model 1.0

Input variable	Values	Description
$prox_i \in \{1, 2, ..., 8\}$	$[0, 1]$	Reading of proximity sensor i
$LEDs$	$\{Green, Red, \emptyset\}$	Instructions for directions read by Camera
Output variable	**Values**	**Description**
$v_k \in \{l, r\}$	$[0, 12]m/s$	Target linear wheel velocity

Adding more sensors to the previous reference model might only increase the search space of the solution without obtaining much benefit. For instance: range-and-bearing sensor which allows robots to perform localized communication may not be unhelpful in single robot cases since there is only one robot in the experiment, hence, there is no communication or exchange of data locally.

4.3.2 Reproducing a T-maze experiment

T-maze is a simple maze that is shaped like the letter -T- and it has many variants vary in complexity in terms of shape and size (Olton, 1979). At the early 20th century, T-maze and its variants were frequently used in laboratory experiments as an attempt to measure and test the memory that an animal would create during the experiment in order to reach the correct final destination.

During the experiment, stimuli are provided to the animal as a guide to select the correct direction. These kind of mazes have been mostly applied to mice and rodents. The experiment is proceeded by placing the animal e.g. a mouse at the base of the maze. The maze is equipped with stimuli that help the animal to select the correct direction: right path or left path. The selection of the correct path is accompanied by a reward. After executing multiple runs the mouse can go directly to the correct path. This repetition with specific rewarded-zone help the mouse to create spatial memory.

In the world of technology, T-maze experiment has been used frequently with various methodologies that analyzed the T-maze experiment in different ways in order to solve it.

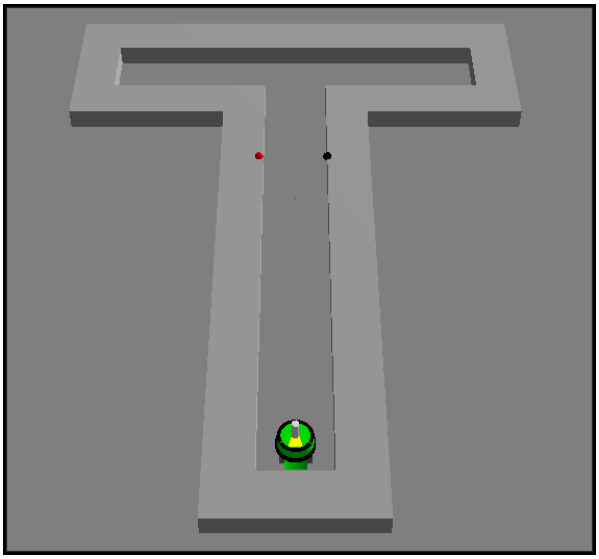


Figure 4.1: *T-maze in Argos.*

The purpose of using T-maze in robotics is to discover whether the robot is able to build a local memory that affects its behavior and helps the robot to reach the correct destination.

Before evaluating the main contribution of my thesis, I verified first the efficiency of NEAT method. For this purpose I reproduced the T-maze experiment of the research (Koos et al., 2013). My version of NEAT is based on the paper (Stanley & Miikkulainen, 2002).
To reproduce the experiment I used the same experimental setup:

Experimental setup

The objective function: represents the sum of the total distance traveled on X-axis d_x and Y-axis d_y represented by the two corridors of the maze, plus 1000 as a reward for the correct path.

$$fitness(c) = d_x + d_y + \begin{cases} 1000 & \text{for the right turn} \\ 0 & \text{for the wrong turn} \end{cases}$$

The neural network: it is a recurrent neural network contains 10 neurons: 8 for receiving the input from the sensors, and 2 motor neurons to compute the speeds of the actuators.

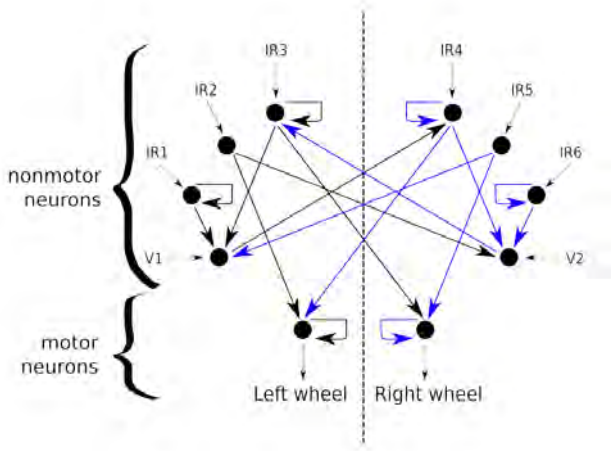


Figure 4.2: *The controller of the T-maze* (Koos et al., 2013).

Guidance: instructions were given by the use of the color sensors to determine the correct directions (left & right), and it was provided near the T-junction to facilitate the experiment.

Result

As long as, the authors in the original work used bar-chart to show their results, hence, for a fair comparison, I also used bar-chart to display the obtained results of the reproduced work with NEAT along with the original one. However, It could be better to use box-plot to give detailed information about the outliers, the medium and the quartiles.

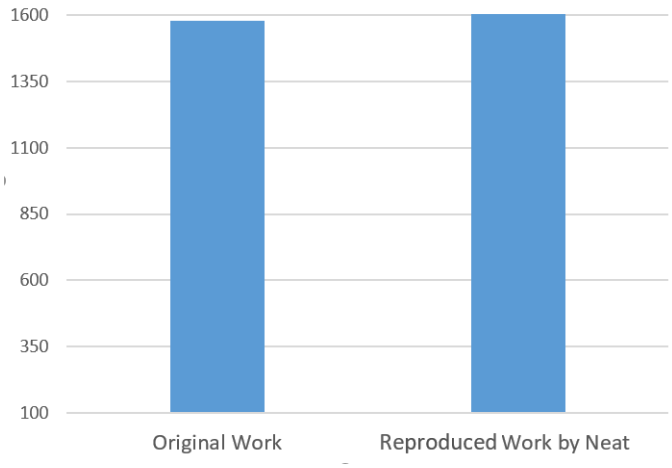


Figure 4.3: *The results of the reproduced vs the original work*

The performance of the reproduced work with NEAT is close to the original one. Nevertheless, there is something interesting about the behavior of the robot in the testing session. Unlike the animal behavior, the robot took decision as soon as it received the instruction about the correct path, and it did not need to create a local memory for that, even if it received the instructions at the very beginning of the experiment.

The robot actually heads towards the correct direction when it receives the instruction. This action will eventually leads the robot to the correct final destination. This behavior will enable the robot to accomplish the experiment no matter how large the maze is. The following figure illustrates the behavior of the robot in the T-maze experiment.

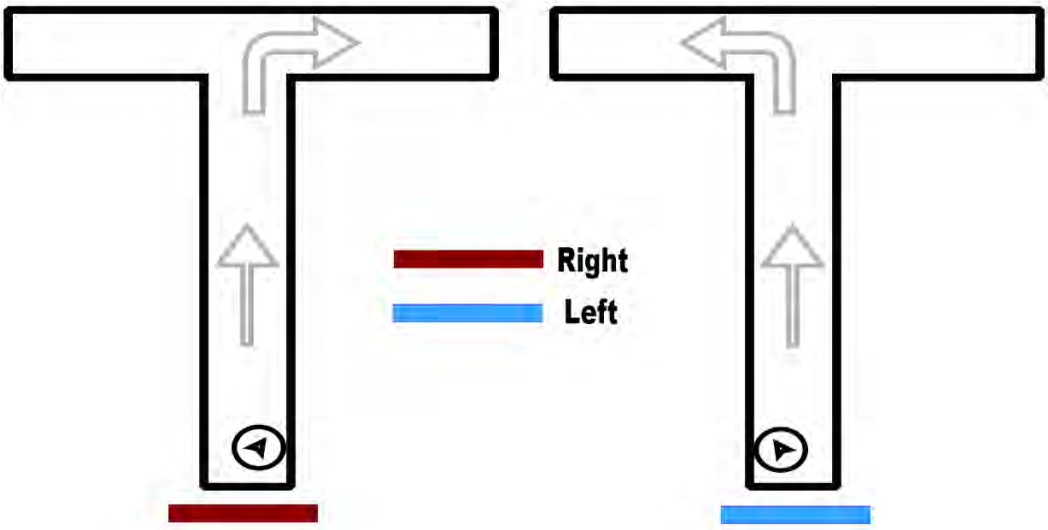


Figure 4.4: *The behavior of the robot during the experiment*

From the figure 4.4 I can conclude that even simpler structure of neural network like in Evostick method can solve this mission. The simpler structures might even outperform the complex ones due to the fact that they have smaller search space.

To test the performance of Evostick on the work (Koos et al., 2013) I only changed the structure of the controller into a simple neural network with two layers: input and output, with full connection among them without having any recurrent links. The weights had been mutated by using NEAT.

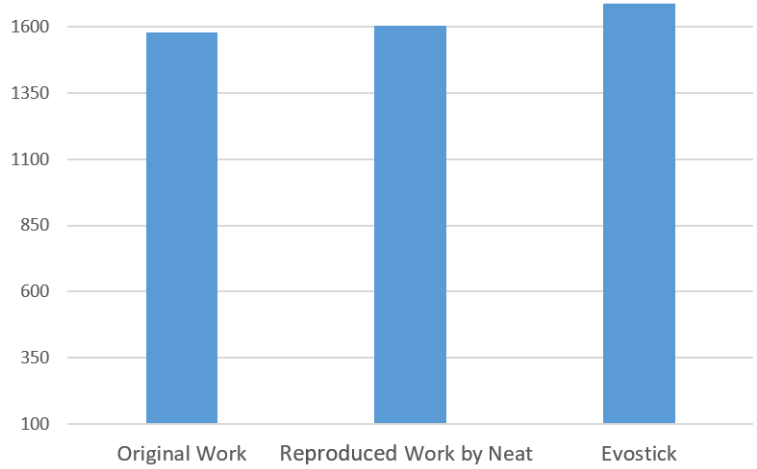


Figure 4.5: *Results of three different methods*

The high performance of Evostick can be attributed to the fact that the mission can be solved without having to create a local memory on one hand, and the fact that the search space of Evostick for finding the solution is small on the other.

4.3.3 Extended T-maze experiment

Since the T-maze can be easily solved, I created a more complicated T-maze for the single-robot experiment. My maze is an Extended T-maze with four final destinations. Using this complex maze provides better evaluation on my methods, and it will give the robot wider scope for exploration, leading to generate different behaviors.

To cast this new maze experiment into optimization problem properly, a new objective function should be defined taking into account the fair access to the new four destinations.

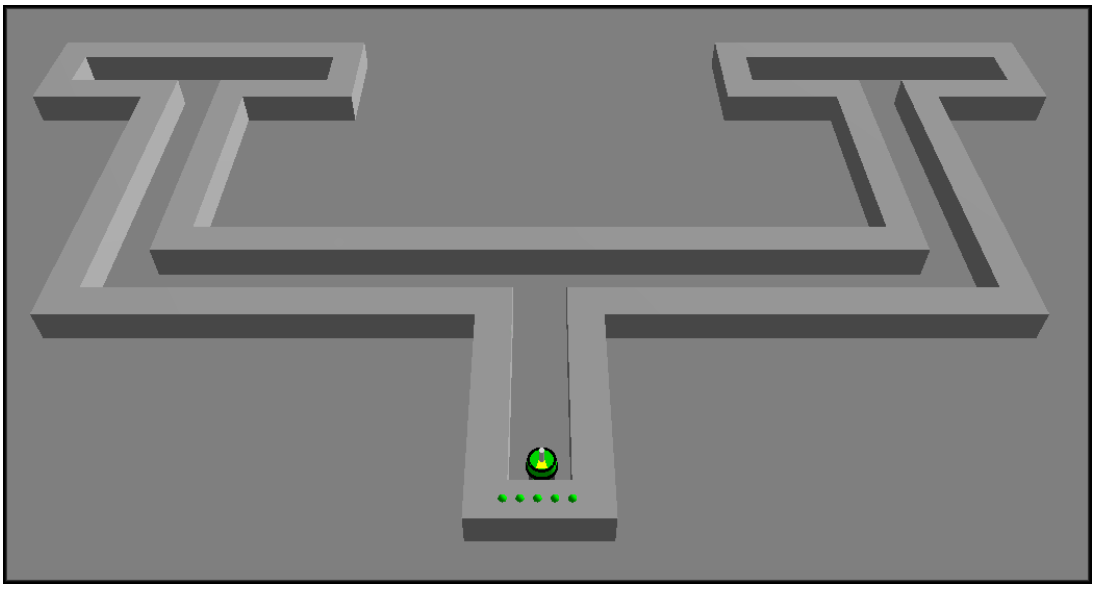


Figure 4.6: *Extended T-maze in Argos.*

In the design phase, the abstract dependence on the traveled distance of the robot to optimize the performance does not always give accurate results. For example if the generated experiment sets the destination in the right side and the robot goes to the left side, in this case, the objective function must return bad performance. Therefore, the new objective function should rely on the distance between the location of the robot and the correct destination. I used for that the distance formula between two points. The objective is minimize the distance between the last location of the robot and the correct destination by using the distance formula between two points.

$$\sqrt{(x_{robot} - x_{place})^2 + (y_{robot} - y_{place})^2}$$

4.4 Case of study B: robot swarms

4.4.1 Reference model 2.0

For the two robot swarms missions, I used the same reference model to support the automatic design in different missions without human intervention.

Input variable	Values	Description
$prox_i \in \{1, 2, ..., 8\}$	$[0, 1]$	Reading of proximity sensor i
$gnd_j \in \{1, 2, 3\}$	$\{Black, Gray, White\}$	Reading of ground sensor j
n	$\{0, ..., 20\}$	Number of neighboring e-pucks
$r_m \in \{1, 2, ..., n\}$	$[0, 0.70]m$	Distance of neighbor m
$\angle b_m \in \{1, 2, ..., n\}$	$[0, 2\pi]$rad	Angle of neighbor m
$r_C \in BlobList$	$> 6cm$	Distance of color c
$\angle b_c \in BlobList$	$[0, 2\pi]$rad	Angle of color c
Output variable	Values	Description
$v_k \in \{l, r\}$	$[0, 12]m/s$	Target linear wheel velocity

This reference model is general and holds features might not be required by my missions. However, the purpose of building a general model for robot swarms is in order to use it with other missions without the need for any modification.

Because any per-mission human intervention on the robots would be too expensive and time consuming, in addition, using the same reference model supports the automatic design (M. Birattari et al., 2019).

4.4.2 Aggregation with two spots

Aggregation is one of the common missions in robot swarms where the robots must aggregate on a specific spot (Kuckling, Ligot, Bozhinoski, & Birattari, 2018).

There are different versions of aggregation missions based on the size and the number of the spots. My experiment is the aggregation with two spots. The swarm should use the global collective behavior to aggregate in one of the two black areas in order to maximize the performance.

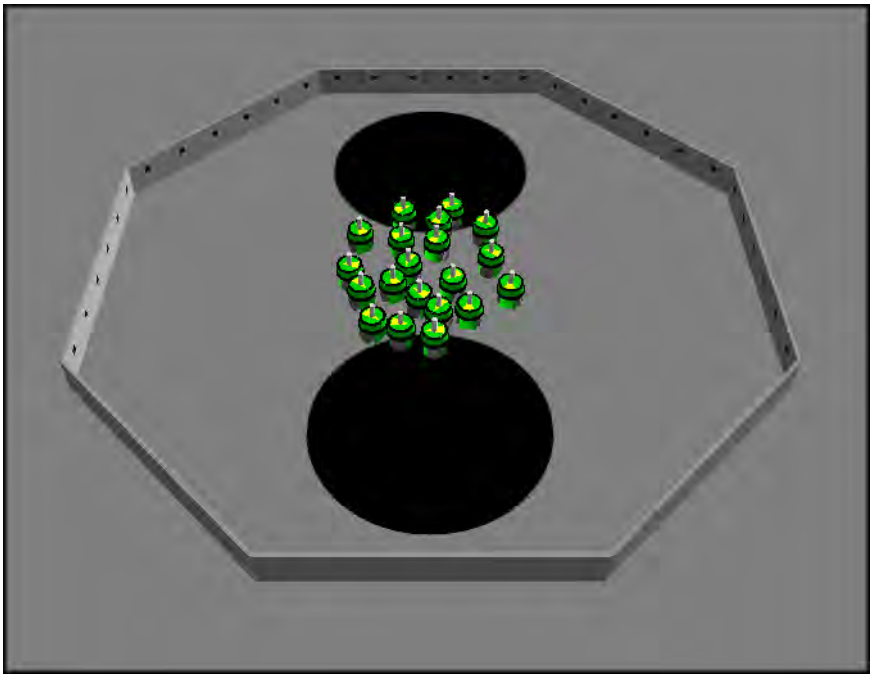


Figure 4.7: *Arena with two spots for aggregation.*

The color of the arena's floor is gray and it contains two black circular areas on the floor, namely u and v. The areas share the same radius of $0.35m$ and are centered in $(0.6, 0)$ and $(-0.6, 0)$. At the beginning of the experiment 20 e-puck robots are randomly distributed in the arena. The objective is maximize the number of robots in one spot. The objective function $Aggregation = max(Nu, Nv)/N$, where Nu and Nv are the number of robots that are on the black areas u or v, and N is the size of the swarm (Francesca et al., 2014).

4.4.3 Surface and perimeter coverage

In this mission, there are two different regions where the robots should behave in a certain manner based on where they stand. (Francesca et al., 2015).

The arena has a black circle with a diameter of $0.6m$: The robots should aggregate on the perimeter of this spot, and white square with sides of $0.6m$: The robots must cover this region during the experiment.

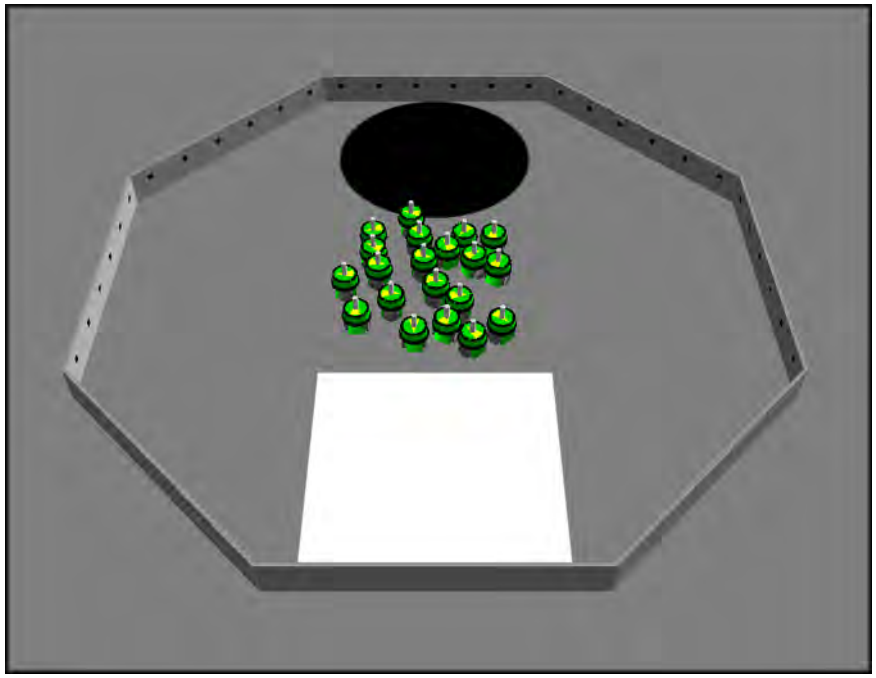


Figure 4.8: *Arena with Surface and perimeter coverage setting.*

The objective function of SPC is to minimize the following equation:

$$c_a E[d_a(T)] + c_p E[d_p(T)]$$

where $c_a = 0.08$, $c_p = 0.06$.
$E[d_p(T)]$ is the expected distance between a generic point on the circumference of the circular region and the closest e-puck that intersects the circumference itself. $E[d_a(T)]$ is the expected distance between a generic point in the square region and the closest e-puck that is in the square region (Francesca et al., 2015). The objective function in Surface and perimeter coverage mission is scalarizing multiple objectives a priori by sum the fitness of the two objectives in the two regions (Trianni & López-Ibáñez, 2015).

Chapter 5

Results And Discussion

To do statistical analysis, 10 different controllers are produced for every methods per mission.

5.1 Case Study A

In single robot experiment, the results are obtained by using 5 different seeds for every branch in the extended T-maze to generate the test experiments. These 20 test experiments with different seeds are used by each controller in every method.

5.1.1 Extended T-maze experiment

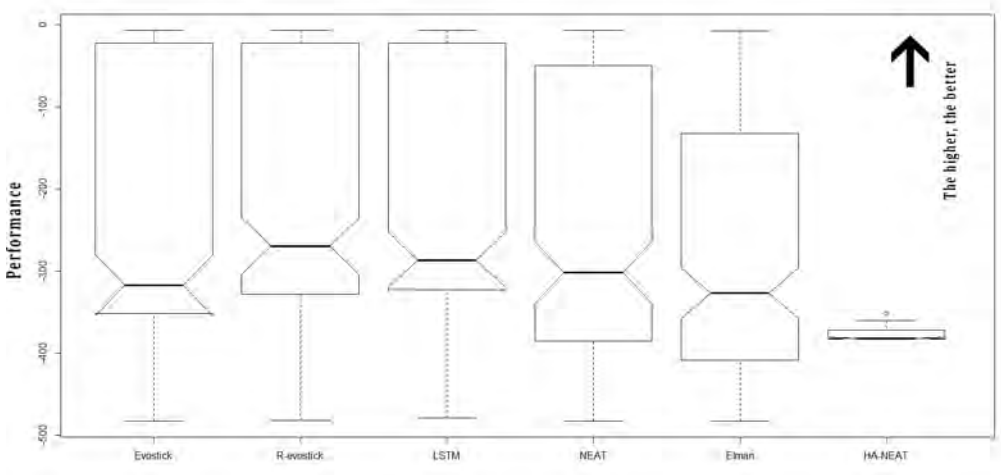


Figure 5.1: *The results of the extended T-maze experiment*

The large size of the boxes can be attributed to the fact that extended T-maze experiment is a heterogeneous mission with four different destinations. The produced controllers could learn to reach some destinations and fail to reach the rest. The medians in the methods: R-evostick and LSTM are higher than the rest. This is a good start reflecting the feasibility of my contributions, nevertheless, The overlapping in the notches which indicates 95% confidence interval of the median denotes that still there is no significant differences among the methods, except for the heterogeneous neural network HA-NEAT which completely failed the mission.

Robot behavior varies from one method to another, and since the objective function in this extended maze focuses on the last position of the robot with respect to the correct destination, therefore, during the experiment the robot can explore the maze freely, and the distance between the robot and the correct destination at the end of the experiment reflects the efficiency of the used method as shown in the box-plot. There are some instances from Elman and HA-NEAT where the robot failed to explore and it instead kept rotating around itself. On the other hand, it is also noticeable that the velocity of the robot varies from one method to another, where in R-evostick and LSTM the robot is faster compared to methods: Evostick, NEAT.

5.2 Case Study B

For robot swarm experiments, I used 10 different seeds to generate the test experiments, and apply them on the two missions: Aggregation with two spots & Surface and perimeter coverage.

It should be remembered that designing a controller for a robot swarms mission is more challenging than designing a controller for a single-robot experiment, this is attributed to the high degree of uncertainty of the behavior of the robots, and their local interactions that take place during the operation.

5.2.1 Aggregation with two spots

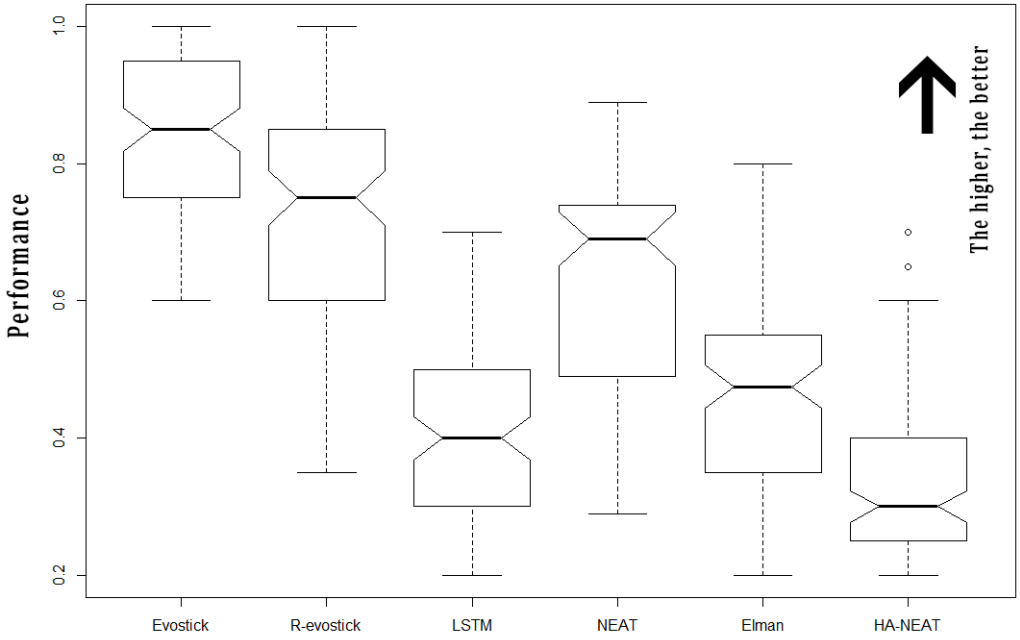


Figure 5.2: *The results of aggregation mission*

In robot swarms, it is notable that the performance of the methods changed. Evostick method achieved the highest performance, while there is a drastic drop in Elman and LSTM. The method HA-NEAT obtained the worst performance.

The behavior of robots correspond to this statistical results where in Evostick most of the robots aggregated in one spot, and they clung to each other. Similarly, in R-evostick and NEAT, most of the robots aggregated in one spot, while the robots in the rest of the methods i.e. LSTM, ELman were exploring and roaming the arena, and the number of robots in the spot was shrinking as shown in the figure 5.2.

5.2.2 Surface and perimeter coverage

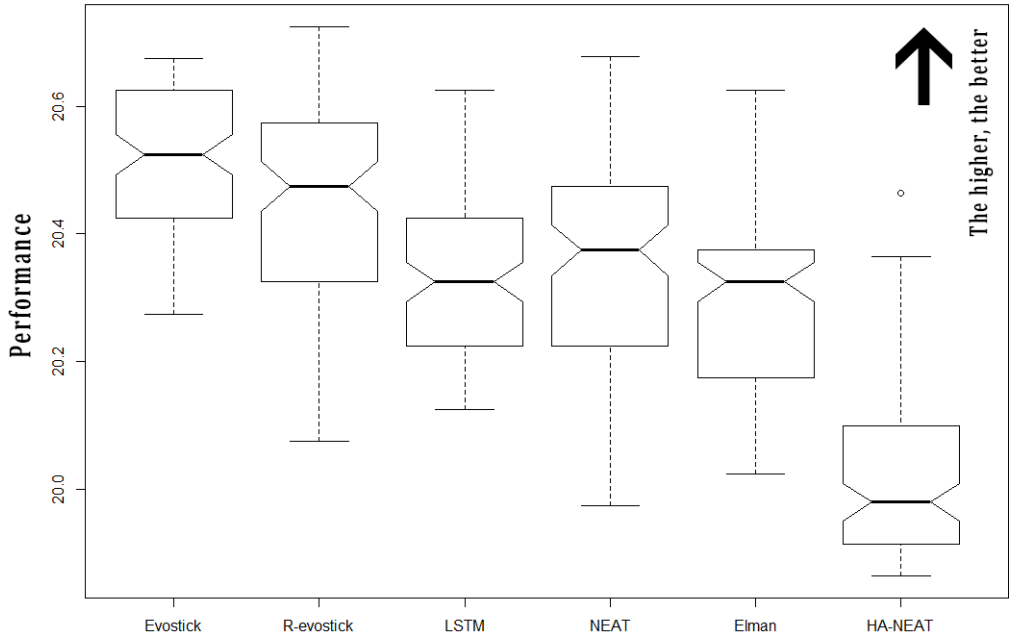
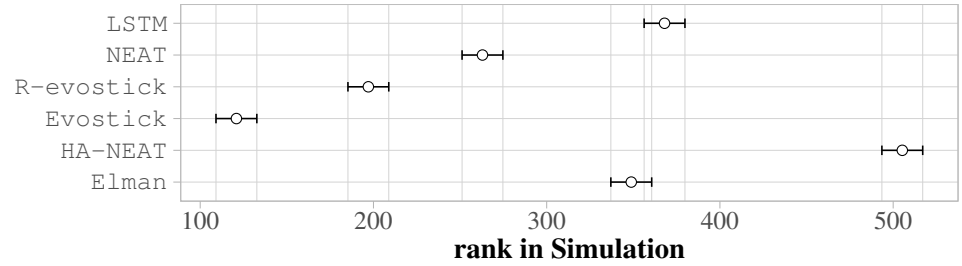


Figure 5.3: *The results of surface and perimeter coverage mission*

In surface and perimeter coverage mission, the medians of the methods have less contrast compared to aggregation mission with better performance in LSTM & Elman. This could partly depend on the nature of the mission and on the definition of its objective function that determines the scope of the search space.

The behavior of robots in surface and perimeter coverage mission is somewhat similar to their behavior in aggregation mission, and they prefer clustering in the black region

For further analysis, a Friedman test is performed on the basis of the obtained results to make a comparison, and to aggregate data from this the two robot swarms missions:



Friedman test computes the median rank over the two missions: aggregation with two spots and surface and perimeter coverage, together, with the corresponding 95% confidence intervals. The difference between the corresponding median rank is statistically significant if the intervals of two methods do not overlap.

5.3 Discussion

According to the obtained results, the family of methods with the exception of HA-NEAT can have some positive effects in single-robot experiments, in addition, R-evostick may also have potential in robot swarms missions, nevertheless, further investigations should be conducted to improve their performance.

The overlapping in the confidence intervals of the methods makes the differences between the corresponding median rank is statistically insignificant. However, the order of the methods in terms of performance was different from one mission to another: In the extended T-maze experiment R-evostick scored the highest performance followed by LSTM, and then Evostick and NEAT, while in robot swarms, the Friedman test is performed on the basis of the obtained results in the box-plots to aggregate data from the two missions: Aggregation with two spots and Surface and perimeter coverage. The result of Friedman test shows that Evostick outperforms all the methods followed by R-evostick then NEAT and Elman. These differences in the performance methods can be attributed to the nature of the mission, where the degree of uncertainty is higher in the robot swarms missions than in the single-robot experiment.

The ability of LSTM to converge in the extended t-maze experiment with a relatively small budget reflects the positive impact of the modifications that I made on the architecture of LSTM in terms of the new added parameters and the rescaling process of the activation function from hyperbolic tangent function to sigmoid function. These adjustments reduced the size of the search space during the design phase.

The bad performance of heterogeneous neural network HA-NEAT can be attributed to the difference in budget between my work and the paper (Hagg et al., 2017). For example, the authors increased the number of generation to 3000 while I used 40 generations. I did not use different budgets for my missions to support the automatic design and to make a fair comparison among them. Although, I successfully achieved the objective by producing parsimonious networks. The average size of the heterogeneous controllers produced by HA-NEAT with multiple activation functions is less than the average size of the homogeneous controllers produced by NEAT with a single activation function.

The results of the conducted missions that I obtained give some information about the performance of the methods and their impact on the behavior of the robots. However, since they are done in simulation, I cannot draw a final conclusion about the performance of the methods due to the reality gap that could exist between the simulation and real world. For instance, the performance of Evostick method in (Francesca et al., 2015) and (Francesca et al., 2014) outperformed most of the methods in simulation, but its performance declined with real robots.

Since there is no connection and local interactions in single-robot missions, this may reduce the possibility of reality gap issue, and the good results of my methods in the case of single-robot reflects a high chance for the family of methods that I propose to work properly on real robots.

On the other hand, according to my findings, and in view of the results obtained by the paper (Saffre et al., 2019), creating a local memory in robot swarms might impair the performance because it may reduce the effect of the current local interactions between the robots, and make them rely more on their historical data for decision making. That would reduce the uncertainty in the experiment and would make the behavior of the robots predefined.

The poor performance of Elman is affected by the complexity of the used neural network that has two extra layers: hidden, context. The number of links with new weights will increase drastically as a result, making the search space larger, and makes the method unable to find a good solution in a small, disproportionate budget. It could require bigger budget to optimize its performance.

Chapter 6

Conclusions

In this work, I proposed a family of methods for automatic design of controllers, and I applied them on two domains in robotics: single-robot and robot swarms. The approach that I used for evolving the controllers is evolutionary robotics and the method for evolving the controllers was NEAT; which considered as a prominent method in neuroevolution field.

The capability of NEAT has been tested by reproducing the T-maze experiment of (Koos et al., 2013), and I showed the importance of defining small search space by reproducing the experiment again with Evostick method.

My contribution aimed to analyze the neural network from three different aspects:

- Using a complex architecture: by employing long short-term memory units LSTM.
- Introducing a novel topology that is causing an internal delay in its calculation: R-evostick.
- Evolving heterogeneous neural networks by mixing a set of activation functions

The missions that I conducted are: extended T-maze experiment for single-robot. Aggregation mission and surface and perimeter coverage mission for robot swarms. I executed these missions in simulation by employing Argos simulator.

The fact that neural network is a black box makes my work more challenging. However, I focused on reducing the search space of the complex added units.

My findings about LSTM in robot swarms is supported by the paper (Saffre et

al., 2019), where the local memory could reduce the uncertainty in the behavior and make the robots rely more on their historical data than the local interactions which make their demeanor predefined. That may lead to lower performance.

In heterogeneous neural networks with a set of activation functions, the weak performance could be attributed mainly to the my constrained budget, unlike the one used in the paper (Hagg et al., 2017). My claim for using a unified budget for all the missions is to support the fully automatic design without human intervention. However, I achieved the main purpose of the paper by evolving parsimonious networks compared to the standard homogeneous networks.

Chapter 7

Future Work

In order to invest my contribution in this thesis, future work should focus on the following points:

Applying the family of methods that I propose on real robots for better evaluation and to check the reality gap.

Using the family of methods that I propose with more different missions in single-robot and robot swarms.

Investigate different versions of recurrent neural network such as GRU neural networks.

Working more on heterogeneous neural networks, and using new activation functions

References

Beni, G., & Wang, J. (1993). Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?* (pp. 703–712). Springer.

Blynel, J., & Floreano, D. (2003). Exploring the t-maze: Evolving learning-like robot behaviors using ctrnns. In *Workshops on applications of evolutionary computation* (pp. 593–604).

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on software and performance* (pp. 195–203).

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

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, *2*(1), 14–23.

Davis, L. (1991). Handbook of genetic algorithms.

Dooley, K. (2001). *Designing large scale lans: Help for network designers*. " O'Reilly Media, Inc.".

Dorigo, M., Birattari, M., & Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, *9*(1), 1463. (revision #138643) doi: 10.4249/scholarpedia.1463

Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S. M., & Christensen, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PloS one*, *11*(3), e0151834.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Ericksen, J., Moses, M., & Forrest, S. (2017). Automatically evolving a general controller for robot swarms. In *2017 ieee symposium series on computational intelligence (ssci)* (pp. 1–8).

Ethembabaoglu, A., Whiteson, S., et al. (2008). Automatic feature selection using fs-neat. *IAS technical report IAS-UVA-08–02*.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, *1*(1), 47–62.

Francesca, G., & Birattari, M. (2016). Automatic design of robot swarms:

achievements and challenges. *Frontiers in Robotics and AI*, *3*, 29.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., ... others (2015). Automode-chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, *9*(2-3), 125–152.

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., & Birattari, M. (2014). Automode: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, *8*(2), 89–112.

Fricke, G. M., Hecker, J. P., Griego, A. D., Tran, L. T., & Moses, M. E. (2016). A distributed deterministic spiral search algorithm for swarms. In *2016 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 4430–4436).

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal. *Deceptive Problem, Genetic Algorithms and Simulated Annealing*, 74–88.

Gomes, J., Urbano, P., & Christensen, A. L. (2012). Introducing novelty search in evolutionary swarm robotics. In *International conference on swarm intelligence* (pp. 85–96).

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., & Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, *31*(5), 855–868.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, *18*(5-6), 602–610.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Gross, R., & Dorigo, M. (2009). Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, *1*(ARTICLE), 1–13.

Hagg, A., Mensing, M., & Asteroth, A. (2017). Evolving parsimonious networks by mixing activation functions. In *Proceedings of the genetic and evolutionary computation conference* (pp. 425–432).

Hecker, J. P., & Moses, M. E. (2015). Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms. *Swarm Intelligence*, *9*(1), 43–70.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.* A field guide to dynamical recurrent neural networks. IEEE Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive behavior*, *6*(2), 325–368.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European conference on artificial life* (pp. 704–720).

Jones, J. L. (2004). *Robot programming: a practical guide to behavior-based robotics*. McGraw Hill.

Koos, S., Mouret, J.-B., & Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, *17*(1), 122–145.

Kuckling, J., Ligot, A., Bozhinoski, D., & Birattari, M. (2018). Behavior trees as a control architecture in the automatic modular design of robot swarms. In *International conference on swarm intelligence* (pp. 30–43).

Lehman, J., & Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *Alife* (pp. 329–336).

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, *19*(2), 189–223.

M. Birattari, Ligot, A., Bozhinoski, D., Brambilla, M., Francesca, G., L. Garattoni, ... Stützle, T. (2019, April). *Automatic off-line design of robot swarms: a manifesto* (Tech. Rep. No. TR/IRIDIA/2019-002). Brussels, Belgium: IRIDIA, Université Libre de Bruxelles.

Millard, A. G., Joyce, R., Hilder, J. A., Fleşeriu, C., Newbrook, L., Li, W., ... Halliday, D. M. (2017). The pi-puck extension board: a raspberry pi interface for the e-puck robot platform. In *2017 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 741–748).

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., ... Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions* (Vol. 1, pp. 59–65).

Mondada, F., Franzi, E., & Guignard, A. (1999). The development of khepera. In *Experiments with the mini-robot khepera, proceedings of the first international khepera workshop* (pp. 7–14).

Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Ijcai* (Vol. 89, pp. 762–767).

Nolfi, S., Floreano, D., & Floreano, D. D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press.

Olton, D. S. (1979). Mazes, maps, and memory. *American psychologist*, *34*(7), 583.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., ... others (2012). Argos: a modular, parallel, multi-engine simulator for multi-

robot systems. *Swarm intelligence*, *6*(4), 271–295.

Rawal, A., & Miikkulainen, R. (2016). Evolving deep lstm-based memory networks using an information maximization objective. In *Proceedings of the genetic and evolutionary computation conference 2016* (pp. 501–508).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Saffre, F., Gianini, G., Hildmann, H., Davies, J., Bullock, S., Damiani, E., & Deneubourg, J.-L. (2019). Long-term memory-induced synchronisation can impair collective performance in congested systems. *Swarm Intelligence*, 1–20.

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association.*

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127.

Tan, Y., & Zheng, Z.-y. (2013). Research advance in swarm robotics. *Defence Technology*, *9*(1), 18–39.

Trianni, V., & López-Ibáñez, M. (2015). Advantages of task-specific multi-objective optimisation in evolutionary robotics. *PloS one*, *10*(8), e0136406.

Vikhar, P. A. (2016). Evolutionary algorithms: A critical review and its future prospects. In *2016 international conference on global trends in signal processing, information computing and communication (icgtspicc)* (pp. 261–265).