Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Torkil Solheim

Extended Speciation in Open-Ended Coevolution

Master's thesis in Computer Science
Supervisor: Pauline Catriona Haddow
January 2020

NTNU
Kunnskap for en bedre verden

Torkil Solheim

Extended Speciation in Open-Ended Coevolution

Master's thesis in Computer Science
Supervisor: Pauline Catriona Haddow
January 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

Inspired by natural evolution and the dynamics between species and environments, open-ended coevolutionary algorithms are able to generate increasingly complex problems while simultaneously having a solution for each. By focusing on fulfilling simple criteria instead of objectives and fitness values commonly used in traditional evolutionary computation, the search process continues to invent without boundaries and maintains diversity. Continual innovation as in natural evolution is not easy to simulate, however, the rewards and implications of a general open-ended algorithm would be extraordinary as it could endlessly generate increasingly complex solutions and designs to any problem.

A base for open-ended coevolution was recently proposed by Brant and Stanley [2017], where a method called *Minimal Criterion Coevolution* (MCC) was used to coevolve a set of mazes and navigator agents that were able to solve them. Mazes and agents were able to increase in complexity in parallel. A method called speciation was used to split the maze- and agent populations to evolve multiple lineages in the same run. In this thesis, the speciation method in MCC is further extended with ranking and prioritization of the most performant species. Species are ranked based on their average growth in size and complexity.

Two different methods are investigated. First, the speciation is extended to support a dynamic number of maximum individuals in each species, where the capacity is transferred periodically from the least to most performant species. The other method investigates the effects of replacing the worst species with new ones periodically throughout evolution. How the mazes and agents evolve in MCC with these extensions are compared against MCC with the base speciation, where the size of all species is static and never replaced.

Sammendrag

Åpne koevolusjonære algoritmer er inspirert av naturlig evolusjon og samhandlingen mellom arter og miljøer. De kan genere problemer med økende kompleksitet samtidig som de har en løsning for hver. Ved å fokusere på simple kriterier fremfor fitness-verdier vanligvis brukt i tradisjonelle evolusjonære algoritmer, fortsetter søkeprosessen å innovere uten grenser i forskjellige retninger. Prosesser som fortsetter å innovere slik som i naturlig evolusjon er ikke enkelt å simulere, men har et stort potensial da en generell slik algoritme kan generere løsninger og design med økende kompleksitet til hvilket som helst problem.

Et utgangspunkt for åpen koevolusjon var nylig introdusert i Brant and Stanley [2017], hvor *Minimal Criterion Coevolution* (MCC) ble brukt for å parallelt utvikle labyrinter og løsningsagenter på en evoluasjonær måte. Labyrinter og agenter var i stand til å øke i kompleksitet i parallell. En metode som deler labyrint- og agent-populasjonene i forskjellige arter ble brukt for å kunne utvikle de i flere retninger samtidig. Denne oppgaven utvider denne funksjonalitet i MCC til å rangere og prioritere de beste artene, basert på hvor mye den gjennomsnittlig øker i størrelse og kompleksitet.

To forskjellige metoder er undersøkt. Den første utvider metoden til å støtte et varierende antall individer i hver art, hvor kapasitet i de dårligste artene blir periodisk overført til de beste. Den andre utvidelsen undersøker effekten av å periodisk bytte ut de verste artene med nye gjennom hele utviklingen. Hvordan labyrinter og agenter i MCC utvikler seg med disse utvidelsene vil bli sammenlignet mot MCC uten noen utvidelser, der arter alltid har like mye kapasitet og blir aldri byttet ut.

Preface

This thesis is the result of a master project conducted at the Norwegian University of Science and Technology in Trondheim, Norway. The contents in the three first chapters are partly based on a pre-project conducted in autumn 2019 [Solheim, 2019].

I would to thank my supervisor Pauline Catriona Haddow for excellent guidance through the whole project and for keeping me motivated and focused. I would also like to thank Amund Tenstad for sharing his NEAT code and letting me use it in my implementation.

Torkil Solheim
Trondheim, June 10, 2020

Contents

List of Figures

List of Tables

Abbreviations

AI	=	Artificial Intelligence
ANN	=	Artificial Neural Network
EA	=	Evolutionary Algorithm
EC	=	Evolutionary Computation
GA	=	Genetic Algorithm
MC	=	Minimal Criterion
MCC	=	Minimal Criterion Coevolution
NEAT	=	NeuroEvolution of Augmenting Topologies
NS	=	Novelty Search
NTNU	=	Norwegian University of Science and Technology
OEC	=	Open-Ended Coevolution
OEE	=	Open-Ended Evolution

Chapter 1

Introduction

This chapter presents the main background for this thesis, followed by the overall goal and research questions, the research method, the structured literature review protocol, the preliminary process, and finally, an overview of the coming chapters.

1.1 Background

Inspired by natural evolution and the dynamics between species and environments, open-ended coevolutionary algorithms are able to generate increasingly complex problems while simultaneously having a solution for each. By focusing on fulfilling simple criteria instead of objectives and fitness values commonly used in traditional evolutionary computation, the search process continues to invent without boundaries and maintains diversity. Continual innovation as in natural evolution is not easy to simulate, however, the rewards and implications of a general open-ended algorithm would be extraordinary as it could endlessly generate increasingly complex solutions and designs to any problem.

An initial base for *Open-Ended Coevolution* was recently proposed by Brant and Stanley [2017], where a method called *Minimal Criterion Coevolution* (MCC) was used to evolve a set of mazes with accompanying solutions. Mazes were used as the problem domain due to being computationally lightweight, while maze navigator agents using neural networks were used to represent the solution paths. The mazes and solutions are stored in separate populations but are constrained in a coevolutionary manner by using criteria that forces them to evolve in size and complexity in parallel.

The study was further investigated in Brant and Stanley [2019], where the maze representation was overhauled to support more ways to increase in difficulty. In the earlier study, the maze could only increase in difficulty by adding more walls within. With the new representation the mazes can also increase their overall size. However, some weaknesses were also unveiled, as the study points out that the populations can find a *"path of least resistance"* while evolving that might be exploited. Specifically, the mazes and maze navigators in some cases are able to find evolutionary paths that make it easy for them to adapt to changes, resulting in individuals that evolve minimally. This trait might not be preferable as it can lead the populations in directions that are suboptimal for further innovation, by avoiding problems that go in new directions and aim for changes in the population. Furthermore, this can lead to a bias towards trivial problems and slow further progress in innovation. Being as efficient as possible and utilizing shortcuts is important in open-ended methods to minimize the computational resources needed and time spent to evolve the solutions, which would only increase as the domain changes from two-dimensional mazes to more challenging real-life environments.

1.2 Goals and Research Questions

Goal *Find ways to increase the efficiency in generating complex mazes and solutions in Minimal Criterion Coevolution*

This thesis will investigate different speciation extensions in open-ended coevolution and analyze their impacts on the overall efficiency and behaviour. The model from Brant and Stanley [2019] is used as a base, and will be extended with different speciation methods to split the populations into different lineages.

To reach the goal, the research questions below provide a base and will lead the research.

Research question 1 *How can ranking and prioritizing species be used to increase the overall performance in Minimal Criterion Coevolution?*

Ranking and prioritizing species in MCC is inspired by the concept of 'survival of the fittest' in nature, where new species emerges while others go extinct continually. For example, changes to environments and increased competition between species can determine if they are able to survive or not. These factors can be converted to the maze domain in MCC by treating the mazes as the environments and ranking the species based on their recent performance in developing complexity. Sudden changes to an environment can for example be made by removing certain mazes from the population. As most people know, sudden changes to an environment have caused mass extinctions before [Archibald and Fastovsky, 2004].

Research question 2 *How is the diversity in the resulting solutions affected when ranking the species based on performance?*

Diversity is an important factor in open-ended processes, as the overall goal of these methods is to produce a broad range of high-quality solutions in a single run. Prioritizing the most performant species might lead the process away from this goal by focusing too much on certain species. How diversity is affected might indicate how the extensions alter the focus in MCC and if they are useful or not.

1.3 Research Method

The research process in this thesis was mainly an analytical process, where results from MCC with and without extensions were reviewed and compared. Many ideas on how to further extend the base model emerged while reading relevant studies in the open-ended evolution field. During the implementation of the model used in this thesis, these ideas were refined and adjusted as a more in-depth understanding of the process were gained from the new technical perspective. In addition, preliminary tests on the base model without any extensions aided in fine-tuning the ideas, as some issues were revealed. The research questions in section 1.2 reflects the final outcome of the ideas, and were the focus in experiments conducted.

1.4 Structured Literature Review Protocol

Several strategies were used to find relevant research articles. Since the base for this thesis comes from authors with central roles in the OEE area, relevant previous work from them served as a great entrance to understand the current status and future directions. Their work also introduces many other relevant research projects to explore.

To guide a more in-depth search, a review protocol was created, described in table 1.1. Throughout the search process two central questions were kept in mind when reviewing articles. These questions were used to estimate the relevance of other research articles and if they could bring any value to this project.

- *How does the article relate to open-ended processes?*
- *Can the method presented in the article aid open-ended processes in some way?*

Keywords	evolutionary computation, open-ended, coevolution, minimal criterion, problem generation, curriculum generation, diversity, artificial intelligence, speciation, evolutionary innovation
Selection Criteria	The article must appear relevant after reading the abstract and conclusion.
Inclusion Criteria	The study is focused on open-ended evolution, coevolution or speciation.
Qualifying Criteria	The authors are critical to their own results. All major claims in the article should be supported by sources or results.
Search Engines	Google Scholar, Web of Knowledge

Table 1.1: The Structured Literature Review Protocol

1.5 Preliminary Process Overview

The work in the preliminary process leading to this master project can be divided into three phases. The chart in Figure 1.1 visualizes the main topics visited leading to open-ended coevolution and MCC.

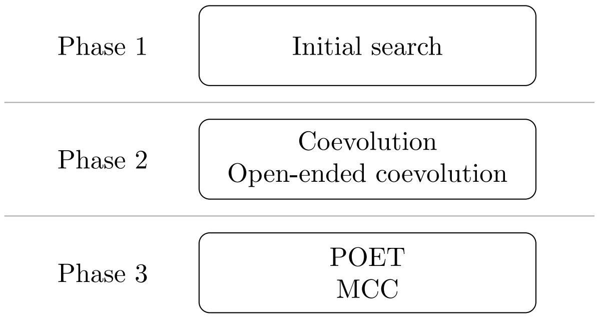


Figure 1.1: Overview of the main theme in each phase.

Phase 1: Finding an interesting area

The research process initially started with a literature search to find an interesting theme in Bio-Inspired AI, where Multi-Objective Optimization and Swarm

Intelligence (SI) were used as anchors. The latest publications from conferences such as GECCO were explored, which resulted in several branches to investigate further. Some of these areas were coevolution, quantum genetic algorithms, HyperNEAT and many different SI algorithms. Coevolution was quickly chosen as the leading and most interesting branch after reading surveys and articles in the different fields. This led to a more in-depth exploration of coevolution.

Phase 2: Coevolution

In this phase the goal was to find something to research in coevolution that could push the field further. A coevolutionary SI algorithm was found in the early stages in a research article by Sun et al. [2012], which seemed like an interesting approach to coevolution. However, after some considerations the algorithm was discarded after some weaknesses in the algorithm were revealed, in addition to some degree of vagueness in the article. At the same time, another interesting article regarding MCC was found, which sparked a new search for material in the direction of Open-Ended Coevolution.

As this field is very recent and has not been approached by many researchers, there were few articles to read for further investigation, other than a handful of studies from the authors in the original MCC paper. On a positive side, this meant that there might be many new directions that had not been explored yet in the area. After looking at studies citing the original MCC paper, an algorithm called Paired Open-Ended Trailblazer (POET) was found. As the main idea in the algorithm is to combine open-ended coevolution with reinforcement learning, it appeared very interesting and was chosen as the next direction to focus on.

Phase 3: POET and MCC

The article on POET was studied in-depth and ways to extend it was investigated. However, some flaws regarding efficiency were uncovered as the computational resources needed to run the experiments are very high. The experiments in the article used 256 CPUs for 10 days, which would be difficult to acquire for this thesis. Consequently, the focus shifted away from POET and back to MCC.

In Brant and Stanley [2019], the "path of least resistance" problem is discussed, and had not been taken into consideration in prior phases. Subsequently, it was decided that investigating methods to overcome this flaw could turn into something useful. After more in-depth studies on MCC and relevant articles, ideas on how to adjust the process were reviewed. A central focus was how the speciation method could be altered to increase the overall efficiency, and was used as a base to create a draft of the research goal and research questions. These were tweaked in many rounds and improved gradually.

1.6 Thesis Structure

The remainder of this thesis is structured in 5 chapters. Chapter 2 presents the background theory, where fundamental concepts and established methods are explained, while chapter 3 elaborates on the state of the art in open-ended evolution. Chapter 4 presents the model used in this thesis, followed by the experiments conducted and results obtained in chapter 5. Finally, chapter 6 discuss the results, concludes, and presents a future work section.

Chapter 2

Background Theory

This chapter presents the required background theory for this thesis, beginning with an introduction to evolutionary computation, genetic algorithms and speciation in section 2.1. Open-ended evolution and Novelty Search are elaborated in section 2.3 and 2.3.1, while section 2.2 presents theory on coevolution. Section 2.4 and 2.5 elaborates on artificial neural networks and an evolutionary method to evolve them called NeuroEvolution of Augmenting Topologies.

2.1 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field in *Artificial Intelligence* (AI) dealing with algorithms inspired by natural processes, such as biological evolution in nature. Evolution is the process that has produced all life on Earth and is adapted by *Evolutionary Algorithms* (EA) to optimize designs and solutions for complex problems. Processes from biology, such as survival of the fittest and heredity are central in these algorithms. In most cases they simulate populations of solutions and rank them by how fit they are. In addition, solutions are evolved further and improved on by combining other solutions and slightly adjusting their offspring. [Floreano and Mattiussi, 2008]

There are many directions and types of algorithms in EC, and the essence is most is to maintain a set of possible solutions by iteratively selecting those fittest to an objective and using them as a base to generate new ones. Approaching complex optimization problems with EAs are often more effective than a brute-force search through all solution possibilities. Typical problems often have very large search spaces and tend to be NP-hard. Consequently brute-force methods would quickly have very long run times for such problems as they increase in complexity. In some cases EAs might get stuck in local optima and fail in finding the global best solution. However, since a population of solutions is maintained, many good approximations for the global best can be found in a single run. Having multiple good options to choose from can in some cases be preferable to only having one, for example in design or multi-objective problems where the weighting of different objectives might be ambiguous.

2.1.1 Genetic Algorithms

A common EA is the *Genetic Algorithm* (GA), and have commonly been used to solve optimization problems in various fields. GAs are very flexible and can be customized to fit many domains. For instance, how the solutions are represented, how their fitness is measured, and how they evolve further are all components that can be customised to suit a specific problem.

As stated, EAs solves or approximates optimization problems by evolving a set of solutions. This set of solutions is called a population, and each solution is referred to as an individual. Each round in the evolutionary cycle is called a generation.

In GAs, all individuals in a population share a common representational pattern called the *genotype*, which can be decoded into a *phenotype* that describes the physical characteristics of an individual. Each individual consists of data following this pattern, similar to a set of genes in biological organisms. How the individuals are represented is important to consider, as the chosen genotype is the base for how they can be used and modified. Some representations can be better than others since they might organize the genes in a way that is more effective in calculations [Floreano and Mattiussi, 2008].

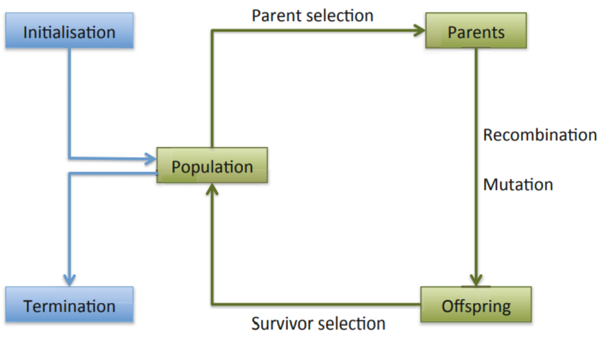


Figure 2.1: The GA main cycle [Eiben and Smith, 2015].

Evolving the population is a cyclic process consisting of several steps. Figure 2.1 illustrates the steps and general flow. The first step in a GA is to initialize the population, where the individuals are commonly generated with random values for their genes. Optimally, the initial individuals should cover a large portion of the search space to be able to explore as much as possible before converging to an optimum. As the search space might be very rugged, there are many local optima that solutions can get stuck in and would have a better base with a diverse set of initial solutions.

The evolutionary cycle begins by selecting pairs of parents that the offspring in the next generation will be based on. Commonly in GAs, a bias towards selecting the fittest individuals is used. This means that individuals with better fitness-value have a higher chance to be selected. After the parents are selected, they are recombined and mutated to form the offspring population. Recombining two parents genotypes is also called crossover. A uniform crossover where each gene is chosen randomly from either of the parents is commonly used. As seen in figure 2.1, another operator called mutation can also be used in generating the offspring, either in combination with crossover or alone. In the mutation operator there is a chance that an offspring genotype is slightly modified, by for example inverting a gene. Mutation increases the explorative capabilities in the population and leads to genetic diversity from one generation to the next.

Selecting which offspring to survive to the next generation is the last step in the cycle. Generally, survivors are selected until a new population with the same size is filled, to keep the population size constant. Selecting offspring based on their fitness value is common to do if an objective is in focus. A method called elitism can also be used, where the upper top of the fittest parents are always selected into the next generation. This is useful to avoid throwing away the best solutions found. Replacing part of the offspring population with randomly generated solutions is also useful to explore new areas and ensure new genetic diversity. Besides, this functions as a restart mechanism that can be used to avoid stagnation [Floreano and Mattiussi, 2008].

Deciding how much the solutions should converge to a location and how much they should diverge in the solution space are important properties to consider. These properties are dependent on the problem to optimize, and it is hard to balance them optimally. Converging too fast can result in a premature convergence, which means that the solutions have converged to a local optimum. Focusing more on diversity and exploration can prevent the population from getting stuck in local optima. However, this might also lead to longer run times and spending more resources on evaluating poor solutions.

2.1.2 Speciation

The search process in GAs can be extended and tweaked in many ways, for example by dividing the population into different niches, also called subpopulations. Each subpopulation might operate independently from others, like species do in nature. This can aid in exploration and to maintain diversity in the population as a whole. Dividing the population into different species is often based on finding similarities in their genotype- or phenotype-representation. Each of these species can for example represent an area in the search space near a local optima. By maintaining multiple such species, the search process is forced to explore multiple local optima simultaneously. This can aid in finding multiple good solutions in the same run. Further, speciation can be extended further with interaction between the species. For example, migration methods that transfers individuals between species can explore new perspectives and create solutions with strengths from multiple sides.

2.2 Coevolution

Coevolution is a process in nature where two or more species or individuals influence each other and their path in evolution. Coevolution is commonly split into competitive coevolution and cooperative coevolution. In competitive coevolution, the two parts compete against each other in a predator-prey pattern, where the prey will always seek new solutions to escape the predator, while the predator must adapt to how the prey evolves to keep feeding on it. On the other hand, in cooperative coevolution the two parts work together in a way that both parts benefit from. These concepts has been transferred to EC and optimization methods by using multiple populations that interact with each other in many different ways. [Floreano and Mattiussi, 2008].

2.3 Open-Ended Evolution

Open-Ended Evolution (OEE) is a branch in EC dealing with algorithms and methods inspired by the endless inventiveness and creativity found in natural evolution. These methods continue to invent and evolve new individuals without boundaries, in addition to often not being guided by objectives. Instead of ranking and selecting individuals based on fitness-values, individuals are rewarded for exploring new directions and bringing new diversity into the population. Such methods do not converge to certain areas but gradually floats through the search space by continuing to invent in the same way as evolution in nature.

An important point about search processes in OEE is that they in general focus on finding valuable *stepping stones* for further evolution. Valuable stepping stones lead the search in new meaningful directions and can avoid stagnation in the population. Another way to look at stepping stones can be found in the creativity humans have displayed in designing and creating new technology. On many occasions, humans have used older technologies to invent something completely different from what the original purpose of them was. For example, the computer would never be invented without electricity and vacuum tubes. When electricity and vacuum tubes were invented, no one thought of using them to create the first computers. In the same way, OEE tries to solve problems by finding solutions that ultimately might lead to the top [Stanley and Lehman, 2015].

2.3.1 Novelty Search

Novelty Search (NS) is a search method in EC where in contrast to traditional GAs, individuals are rewarded for being new in some way instead of how well they perform with regards to an objective. NS commonly looks at how the individuals behave in a domain instead of measuring their fitness. By using an archive that keeps track of all behaviors encountered, the novelty of new individuals and their behaviors can be measured. This metric is used likewise as a fitness metric, the individuals with a higher novelty have a higher chance to reproduce. A strength with NS is that it does not fall into local optima as traditional GAs might. NS can avoid deceptive traps that some domains have towards fitness-based methods, meaning that there are large gaps between local optima and the global optima. For example, in mazes there can be certain paths that lead close to the exit but ends up being a dead-end, and possibly fooling an objective-based method. In such domains fitness-based methods would have to walk far out of local optima to get to the global optima [Lehman and Stanley, 2008].

A typical domain that NS is well-suited for is mazes and finding navigation strategies. Mazes are useful since they are performant in addition to being deceptive in a natural way, as there are many local optima that objective-based method can get stuck in. If for example, the goal is to navigate to a target location in a maze within a time interval, individuals could be ranked by how close to the target they end up. As mentioned, certain paths can lead the navigator towards the target point but end up being the wrong path. Further, since objective-based methods reward solutions that are closest to the target point, the search might not be able to get out of this path.NS could overcome such barriers by always looking for new solutions with new end positions, effectively abandoning visited paths that did not lead to the target. By discarding solutions with already found end locations, the search process would in most cases find a path through all mazes.

Since NS does not use any objectives related to a goal to guide the individuals, a general notion in the search process is to try to reach a goal by not searching for it. NS drifts through the search space by always looking for new behaviors until it eventually reaches a goal behavior. By discarding behaviors already found, NS forces the search to not revisit areas already covered [Lehman and Stanley, 2008].

2.4 Artificial Neural Networks

An *Artificial Neural Network* (ANN) is a computational model used in AI and is typically used to approximate a function by connecting *perceptrons* together [Rumelhart et al., 1986]. Perceptrons are simple processing units that map a set of input values to a single output value. A single perceptron is a linear classifier, meaning that it can divide a set of input data into two classes with a linear function. When multiple perceptrons are connected, they can correctly classify data that is not linearly separable.

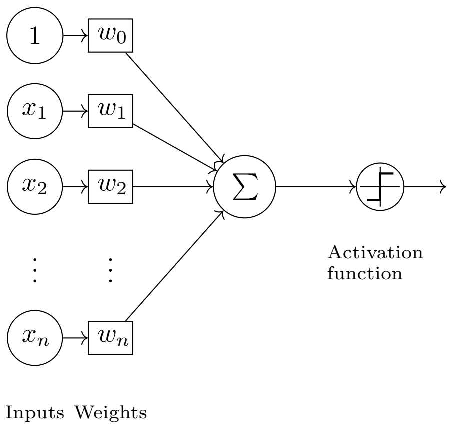


Figure 2.2: The perceptron. There are three steps in a perceptron. First, each input value is multiplied by a weight. Second, the weighted input values are summarized. Finally, the sum is run through an activation function to normalize the output value. [m0nhawk, 2013]

Perceptrons are typically arranged in layers, connected by links with weights. Generally, an ANN has an input layer, an output layer, and hidden layers in between, visualized in figure Figure 2.3. Input data are sent to the input layer and are processed through the weights and hidden layers to the output layer. The weights are commonly tuned by a training process, where a dataset with example inputs and their correct classification output is used to train them. After sufficiently training samples have been used, the weights are tuned to map any set of input values to an approximated classification. This process can also be seen as a process where the regions in the input space are tweaked and adjusted to fall into different categories.

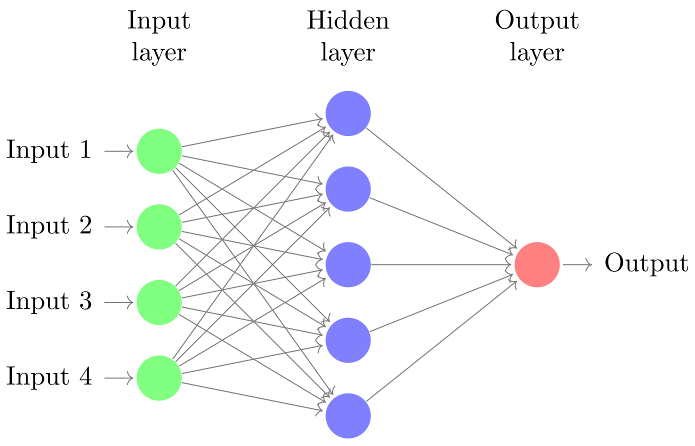


Figure 2.3: An ANN with four perceptrons in the input layer, 5 in the hidden layer and a single in the output layer (adapted from Fauske [2006]).

ANNs are commonly manually constructed and trained with large datasets and backpropagation. However, there are also methods that find suitable neural networks for classification problems by keeping a population of different neural networks and evolve their weights, connections, and neurons in an evolutionary manner. One such method is presented in the following section.

2.5 NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies (NEAT) is an algorithm used to construct ANNs proposed by Stanley and Miikkulainen [2002], and is inspired by how brains evolve over time. NEAT treats ANNs as individuals in a genetic algorithm and uses genetic operators such as crossover and mutation to modify them. They initially start with a minimal structure and are incrementally optimized and extended into models with appropriate weights and topologies. The genotype of individuals in NEAT consists of node genes and connection genes, as shown in figure 2.4.

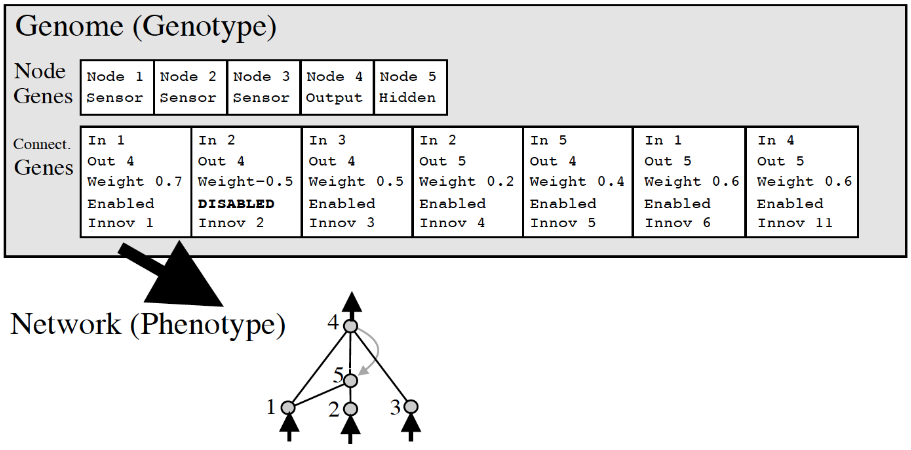

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 2.4: The genotype and phenotype used in NEAT. Each node gene in the genotype represents a node in the phenotype. The node type (sensor, hidden, output) is also described by the genotype. The connection genes represents all connections between nodes in the phenotype. Each connection gene contains which genes it goes from and to, a weight, a status of whether it is enabled or not, and an innovation number. [Stanley and Miikkulainen, 2002]

A problem with ANNs and evolution is that a random combination of two networks through crossover can result in non-functional solutions, as they might have different sizes and not be aligned optimally. NEAT overcomes this by adding a historical marking to all new genes so that the crossover process can tell which genes match up between any individuals. Two genes that have the same historical marking represents the same structure and can be used to guide the crossover process.

NEAT also uses speciation to split the population in niches based on the structure of nodes. By doing this, the individuals primarily compete with other individuals in the same species. As a result, all species optimize their structure before competing with others.

Chapter 3

State of the Art

This section presents the state of the art in Open-Ended Evolution and relevant methods used within the field. The first section presents important background roots, in addition to central techniques and algorithms. The subsequent sections go deeper on methods such as minimal criteria, open-ended coevolution, and adversarial learning.

3.1 Open-Ended Evolution

Open-Ended Evolution (OEE) has been researched for decades with a focus on how natural evolution works and how it can be used to solve problems. Progress has been quite slow and a general theory for open-ended processes such as natural evolution has not been unveiled yet. However, researchers have been able to solve many complex problems by using methods with open-ended features that move away from the objective-based approach, in contrast to most evolutionary algorithms. For example, Novelty Search (NS) was used to find navigators powered by neural networks to solve mazes, and were able in many cases able to find solutions where an objective-based approach failed [Lehman and Stanley, 2008]. In Wang et al. [2019] they used a method called open-ended coevolution to evolve walking agents that could get through a set of diverse obstacle courses. The evolved walking agents from this method was able to complete more difficult courses than agents trained with *Reinforcement Learning* (RL), which has been the most popular method to use in such domains. Being able to beat methods like reinforcement learning is promising and there is an indication that open-ended processes might be able to find solutions that gradient-based methods cannot find as easily.

The base roots for open-ended processes come from research projects in the

50s and 60s, beginning with the cellular automata [Von Neumann et al., 1951]. This lead to the creation of the field known as *Artificial Life* (Alife) in the 80s. Projects in Alife tries to simulate natural evolution in artificial worlds, where new entities are invented in a continual procedure. These worlds are commonly digital, for example in Tierra [Ray, 1993], where computer programs compete for resources while evolving and self-replicating. A more recent example from Soros and Stanley [2014], Chromaria, has been used to investigate how simple conditions can induce open-ended behavior. However, with a goal to better understand life through artificial worlds, many questions in Alife remains to be answered as the state of current knowledge is limited [Aguilar et al., 2014].

3.1.1 Open-Ended Search

Solving and optimizing problems by using open-ended processes focus on drifting through a solution space without convergence. For instance, in NS [Lehman and Stanley, 2008], the evolutionary process continues to innovate and explore new directions until halted. NS was one of the first of its kind and has been used in many research projects, in addition to sparking new directions for further research in OEE. Finding navigation strategies in mazes with NS has been a repeating success and researched by many [Urbano et al., 2014; Velez and Clune, 2014]. Methenitis et al. [2015] used NS to evolve soft-robot walking strategies, and in some cases outperformed objective-based methods. A study combining cooperative coevolution and NS obtained better results than a fitness-based cooperative coevolutionary method in various multi-robot tasks [Gomes et al., 2017]. Even though NS has had a lot of success, it requires an archive of past solutions or behaviors to find out if new individuals are novel or not. Maintaining this archive adds a layer of complexity on the whole process and steps away from the open-ended processes in nature.

Quality Diversity (QD) [Pugh et al., 2016] algorithms extends NS by in addition to searching for novelty, also focus on finding quality in individuals. A *Behaviour Characterization* (BC) function is used to define a set of interesting behavioral features to search for in individuals phenotype behaviour. For example, NS has been extended to focus on individuals with high performance in a method called *Novelty Search with Local Competition* (NSLC) in Lehman and Stanley [2011a]. NSLC enforces a local competition in new areas of the search space, turning NS into a multi-objective algorithm balancing novelty and performance.

Another QD-method called *MAP-Elites*, short for Multi-dimensional Archive of Phenotypic Elites, is similar to NSLC but does not require the same type of archive of past solutions. Instead, the behavior space in MAP-Elites is discretized by a grid, where the most performant (or elite) individual found in each cell based on a BC is remembered. Both NSLC and MAP-Elites have been extended further in many practical ways, explored in [Pugh et al., 2016]. For example, Chatzilygeroudis et al. [2018] used MAP-Elites to find walking strategies for multi-legged robots. In addition, MAP-Elites were extended in Colas et al. [2020] to use deep neural networks to control parameters. This approach was able to find robot walking strategies that could function even when the robots were damaged.

However, methods discussed in this section steps away from the open-ended processes in nature, as they require archives of past solutions and BCs that describe what a good behaviour is. Nature does not aim for certain behaviors, individuals just need to survive long enough to reproduce to continue their lineages.

3.1.2 Minimal Criteria

Constraining populations in an evolutionary algorithm with *Minimal Criteria* (MC) is a method used to eliminate unviable individuals and maintain quality in a population. MCs can for example be used to only let individuals that fulfill a specific criteria to reproduce, like being able to complete a task or expressing a behaviour. In the Alife world of Chromaria, MCs were also proved to be a vital requirement for OEE [Soros and Stanley, 2014].

For example, Lehman and Stanley [2010] extended NS by constraining individuals evolved with an MC that eliminates unviable behaviors. Compared to regular NS, this extension was able to evolve solutions in a maze domain more consistently. In addition, a higher efficiency in finding solutions to the mazes was achieved compared to regular NS.

An early version this concept were introduced in Mattiussi and Floreano [2003], where a method called *Viability Elimination* used viability boundaries to constrain solutions to be within an area in the solution space to survive.Initially, the boundaries allow all solutions but gradually becomes more strict and narrow down the viable solution space, resulting in the solutions converging to an area. Viability boundaries are different from MCs in that they are based on objectives and do not measure the behaviour of individuals. They do not possess the same open-ended traits as found in MCs.

3.2 Open-Ended Coevolution

Open-Ended Coevolution (OEC) is a recently formed branch in OEE consisting of algorithms that combines coevolutionary and open-ended methods. Using this as an open-ended search method was first presented in Brant and Stanley [2017], where a method called *Minimal Criterion Coevolution* (MCC) was introduced. MCC further builds on OEE and QD methods by showing that two interlocked populations of problems and solutions can display a continual inventiveness of complexity under certain circumstances. More specifically, a population of mazes and a population of maze navigators were used similar to the maze domain problem described in 2.3.1 to induce a never-ending increase in complexity through evolution by constraining each population with a simple MC: A maze must be solved by at least one maze navigator and each maze navigator must be able to solve at least one maze to be deemed viable. Both the mazes and maze navigators mutates between generations to increase in size and complexity. This lead to a process where the MCs gradually becomes more challenging on their own, as the populations ultimately control the difficulty of the mazes.

Similar to QD-methods, MCC displays creativity and finds complex behavior without any being guided by objectives. However, MCC does not use any novelty archive or behavior characterization and serves as an alternative way to open-ended search by only relying on the MC. In Soros [2018], tests from the Chromaria world indicated that a non-trivial MC is required to induce open-endedness, such as the MCs used in MCC by . A part of MCC is also based on some of the functionality in Chromaria, where a queue system that ensured all viable individuals to get at least one chance to reproduce was used. With this queue, MCC can continually explore multiple lineages of problems and solutions in a single run, initially started by a set of seed individuals. These seed individuals are randomly generated and lay the foundation for all further evolution.

MCC was extended further in Brant and Stanley [2019], where an updated maze domain that supported increase in size as well as complexity without boundaries was presented. However, as mentioned in 1.1 a characteristic known as "path of least resistance" was unveiled. This trait might not be desirable, as it hints that the mazes and navigators find paths in evolution that creates an illusion of increasing complexity when they just follow the paths that are easiest to adapt to.

POET

A very recent approach OEC called *Paired Open-Ended Trailblazer* (POET) [Wang et al., 2019] combines MCC with *Evolution Strategies* (ES) [Rechenberg, 1973] to evolve a population of walking agents that can solve a diverse set of coevolving obstacle courses. MCs were used to maintain quality in individuals in the same way as in the original study. Further, POET uses a pairing strategy that locks walking agents with specific obstacle courses for some generations. During this pairing duration, the agents are fine-tuned for their paired domain by reinforcing the strategies used to get through it. Agents are then transferred to other obstacle courses to use experience gained to solve other courses. As a result, the process were able to evolve agents that could complete obstacle courses that agents trained with RL were not able to complete. While this proves that combining open-ended processes with coevolutionary methods has a great potential, POET required a cluster of CPUs for ten days to train the agents. Further, this computational requirement would only increase as the complexity in the domain increases.

3.3 Adversarial Learning

A branch in RL called *Adversarial Learning* (AL) resembles competitive coevolution in that two opposing sides are trained simultaneously and try to overcome each other. Methods such as self-play, where a single agent plays against itself, have displayed impressive results in learning game strategies and have achieved superhuman performance in a game called Go [Silver et al., 2016]. Simultaneous training of environments and agents has also been explored, for example in Gabor et al. [2019] that uses RL to evolve agents in a factory environment. The factory environments were generated by a Markov decision process that tries to find those that maximize the learning rate in the agents. *Generative Adversarial networks* (GAN) is another approach that uses two neural networks in a competitive setup, and has had much success in generating realistic images of human faces [Goodfellow et al., 2014; Karras et al., 2019].

While many of these and similar methods have received much attention, they are mainly based on extensive training with gradients and in some cases need a large amount of training data. These methods do not approach a search space in the same way as open-ended methods do, as gradients are based on objectives. Open-ended methods do not require training data and explore search spaces in all kinds of directions. In Wang et al. [2019], POET was able to complete problems that objective-based RL could not. Perhaps open-ended evolutionary methods are able to reach levels of complexity that gradient-based methods simply cannot reach.

3.4 Summary

An important goal in open-ended processes is to produce both diverse and functional solutions in a single run. Methods in QD and OEC have explored many different approaches to open-endedness, but have also unveiled issues to overcome. These methods typically use natural evolution as inspiration for how to simulate such behaviour. However, this has proven to be very challenging to accomplish as the complexity and resources needed rapidly rises when the domain increases in difficulty.

Speciation was used in OEC to increase the diversity discovered [Brant and Stanley, 2017]. Multiple lineages of solutions could be evolved simultaneously by dividing the populations into many smaller subpopulations. While the effectivity of speciation was considerable, further extensions that utilize the dynamics between species as in nature has not been explored in many ways yet. This thesis will investigate the effects of extending speciation with prioritization and replacement strategies, that rewards the most performant species. Focusing more resources on high functioning solutions and discarding stagnant species might speed up the overall process. By replacing species in MCC the seeds evolved in the initial phase won't have as much control of the process, and scenarios where they stagnate can be avoided.

Chapter 4

Model

This chapter describes the model proposed in this thesis. An overview of the base model and extensions made is presented in 4.1. Section 4.2 describes the representation used for the mazes and agents, while section 4.3 describes all the steps in the model algorithm in detail. The main contributions in this thesis are described as part of the model algorithm in section 4.3.2. Finally, section 4.4 presents the parameters used in the model.

4.1 Overview

The base model is adapted from Brant and Stanley [2019] and uses the same main components as well as agent and maze representation. The main differences in the original model compared to the model proposed in this thesis lies in the speciation functionality and the maze genotype-to-phenotype converter. Extensions were made to the original MCC algorithm to support a more dynamic and flexible queue system used in speciation and selection.

4.1.1 Base Model

The base model without speciation consists of four main steps: seeding, selection, reproduction and evaluation. Figure 4.1 gives an overview of how these are connected. A seeding phase bootstraps the process by randomly generating mazes and finding viable agents with the NEAT algorithm combined with Novelty Search (NEATNS), as described in 2.3.1. These agents and mazes represent the first generation of the maze and agent populations. A queue system is used in each population, where a pointer points to the next individual scheduled for selection. In each generation agents and mazes are selected and reproduced

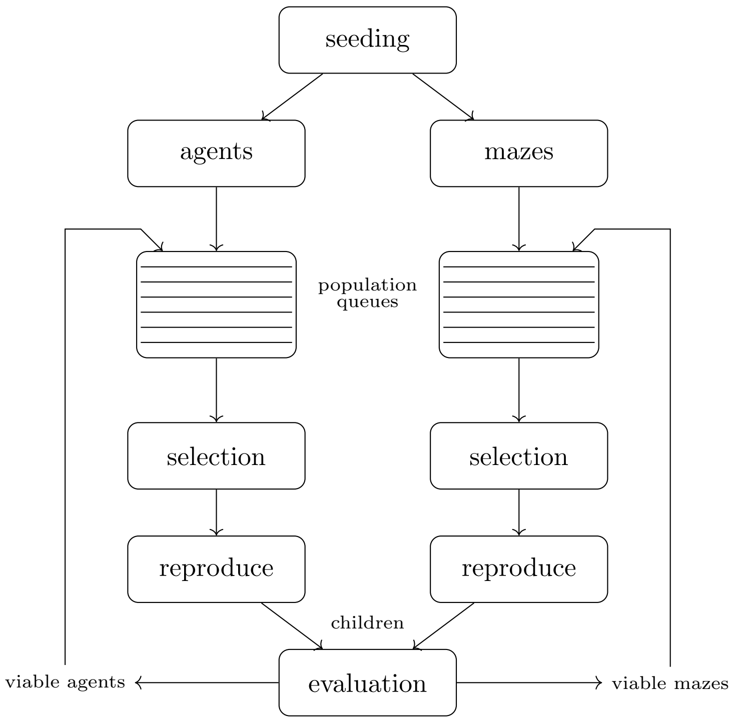


Figure 4.1: Overview of the original MCC model from Brant and Stanley [2017].

through mutation, before being sent to evaluation. The mutations are biased towards increasing the overall complexity of the individuals resulting in an increase in average maze size and the amount of turning points in the solution paths. All mutations are described in section 4.3.3. All agent children are evaluated in all maze children (see section 4.3.4). An agent is considered viable if it can reach the target position in at least one maze, while a maze is considered viable if at least one agent can reach its target position. All new viable agents and mazes are inserted at the top of their respective population queues. If a queue reaches its maximum capacity, the oldest individuals are removed from the queue. This process results in a continual evolutionary drift where the populations contain gradually more complex individuals.

4.1.2 Speciation and Extensions

The queue system used in each population in the original model can be extended to support different species, where the population is split into different species based on genetic similarity. Each of these species has a queue with a pointer and maximum capacity. In the model from Brant and Stanley [2019], each species queue has the same maximum capacity and remains static throughout the whole process.

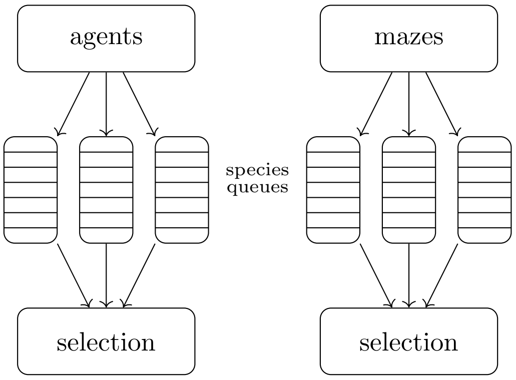


Figure 4.2: A snippet of the original model with speciation activated presenting multiple queues in the agent and maze populations.

The main contributions in this thesis further extend the original model with different speciation methods. There are two variants investigated in this thesis, described in detail in 4.3.2. In the first variant, the maximum capacity for species is adjusted periodically to prioritize those with highest performance, measured in how much they increase in complexity. The most performant species are rewarded with more capacity, while the capacity in the least performant is decreased with the same amount. The second variant extends the speciation with a restart-mechanism, where the worst species are replaced with new ones. How these extensions work is described in detail in 4.3.2.

4.2 Representations

4.2.1 Maze

A maze is built as a two-dimensional grid of square cells, where walls are placed between the cells. This grid uses a conventional coordinate system to place the cells, where the bottom left cell has position 0 in both x- and y-direction. The x- and y- values for each cell increases by moving in the right and north direction. At all times the mazes are formed as squares, with the same amount of cells in width and height. However, this amount can vary from maze to maze. All mazes only have one solution path and do not contain any loop paths, meaning that a path never leads back to its entrance point. A path either lead to a dead-end or to the target point. The solution path is represented by a list of coordinates in the maze genotype, and always begins in the upper left corner cell and ends in the lower right corner cell.

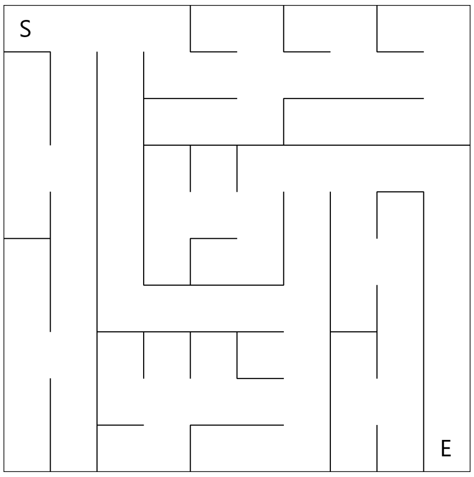


Figure 4.3: Example of a maze. The "S" and "E" marks the start and end positions.

Genotype

The maze genotype contains four components as listed in the overview in table 4.1. Table 4.2 presents an overview of the wall gene contents.

Maze Genotype	
Width and height	Numerical value, width and height always has the same value.
Initial orientation	Can either be horizontal or vertical. This value decides if each pathgene should begin with a path in the horizontal or vertical direction.
Path genes	List of path genes containing an x- and y-coordinate used to map the solution path.
Wall genes	List of wall genes used to construct the areas surrounding the solution path.

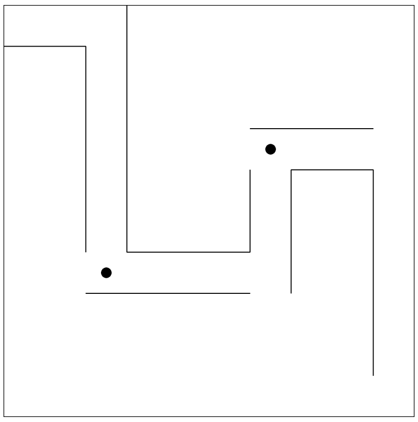
Table 4.1: Overview of the maze genotype.

Wall gene values	
Wall position	Position of wall within a subdivision.
Passage position	Position of passage on walls within a subdivision
Orientation	Can be horizontal or vertical.
Opening location	Defines which side the opening location to the solution path the subdivision should prioritize.

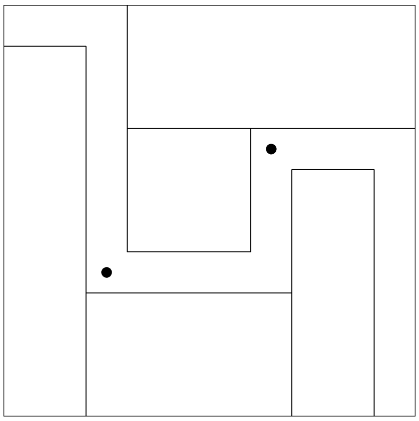
Table 4.2: Overview of the wall gene.

Genotype to Phenotype

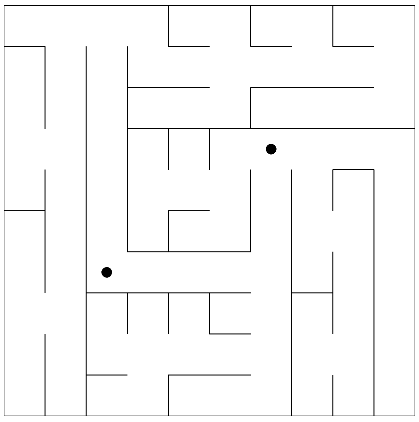
When decoding a maze genotype to the maze phenotype the maze initially contains no walls or paths. All cells contain four boolean values that describe which directions the surrounding walls are located, which are all set to false initially.



(a) The solution path consist of walls and turns described by the path genes. The black circles indicate where path genes place the turning points.



(b) Subdivisions are the rectangular shapes surrounding the solution path.



(c) The walls within the subdivisions are determined by values in the wall genes.

Figure 4.4: There are three main steps in this process. (a) add solution path, (b) add subdivisions, (c) add subdivision walls and create passages to the solution path from the subdivisions.

The path genes in the genotype are used to map the solution path from the starting point to the target point. Each path gene adds a turning point and a horizontal or vertical set of walls on each side of the cells leading to the turning point. The initial orientation value in the genotype decides if the solution path should begin in a horizontal or vertical direction. All maze cells in the solution path are marked with a direction describing the way to the target location.

The next step is to find subdivisions in the areas surrounding the solution path. Subdivisions are found by a search through all maze cells for those unused by the solution path. This search goes through all cells in all rows from left to right beginning with the top row. It continues downwards until all cells in all rows have been checked. When a cell not used by the solution path or any other subdivision is found, a subdivision start point is set and the search begins a sub-search to add cells to the new subdivision. The sub-search continues to add cells to the subdivision in the same row until the right edge of the maze is reached or a cell in the solution path is found. Further, the search continues to add cells to the subdivision in the rows below until the bottom is reached or a cell that breaks the rectangular shape is found. Only cells that are in the same columns as the cells in the initial subdivision row are added. A subdivision endpoint is marked in the last row where the rectangular shape is still kept.

The final step is to open a passage from the subdivisions to the solution path and add walls within them. All subdivisions border the solution path in at least one cell, and the passage position value from a random wall gene is chosen to select which side to place the passage opening. If the chosen side does not share borders with the solution path in any location, the other sides are attempted, beginning with the opposite side. To add walls within a subdivision, a recursive process that splits a subdivision into two smaller ones is used. Wall genes are iteratively to decide how a subdivision should be split. Their wall value is used to place a wall that divides a subdivision into two, while the passage value opens a location in the created wall. This process is repeated in the child subdivisions until their width or height is one cell.

4.2.2 Agent

The agents are maze navigators consisting of sensors and a neural network. In total, an agent has ten sensors, six of which are rangefinder sensors while the other four are radar sensors. These sensors are placed at certain locations on an agent body, visualized by figure 4.5. Each rangefinder measures a distance to the surrounding walls. How these values are calculated are described in section 4.3.4. The radar sensors together act as a compass towards the target location, by changing which is active when the agent is rotating. A radar sensor has an area of sight of 90 degrees and activates when a straight line from the target location to the agent is within this area. Only one radar sensor is active at any time.

Agents have a positional velocity and an angular velocity that controls their movement in a maze. A neural network is used to find new values for these

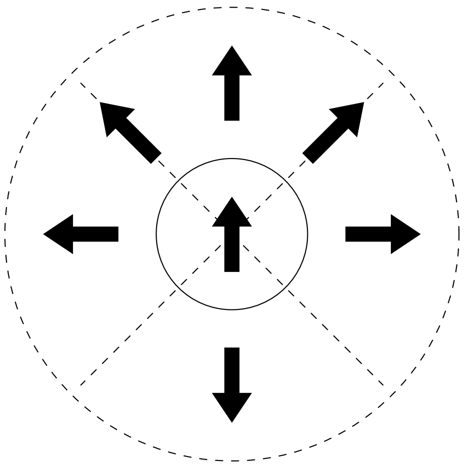


Figure 4.5: Image of agent sensor positions. The arrow in the center points in the forward direction for the agent, while the surrounding arrows indicates the directions the rangefinder sensors are measuring in. The dashed areas around the agent represents the radar sensor angles. This is the same agent body as used in Lehman and Stanley [2011b].

velocities in each simulator time step, by using the sensor values as input. The neural network is encoded in the agent genotype, in the same way as in the NEAT-algorithm (see section 2.5). All of the agent genotypes contain ten input nodes and two output nodes, one input for each sensor and one output for each value to update.

4.3 Algorithm

An overview of the model algorithm is presented in figure 4.6. The main key events in the algorithm is found in the figure, while more details are presented in the following subsections.

4.3.1 Seeding

The first step in the model algorithm is to find viable mazes and agents to use as seeds in the MCC process. All mazes are generated randomly with the same size and contains up to a number of path genes, specified by model parameters.

Agent seeds are found by searching for solutions in the maze with the NEAT algorithm combined with Novelty Search, in the same way as described in section 2.3.1. The navigation agents in this search have the same representation as used in the MCC model, described in 4.2.2. Each maze seed requires a separate seed search process to find an agent that can complete it.

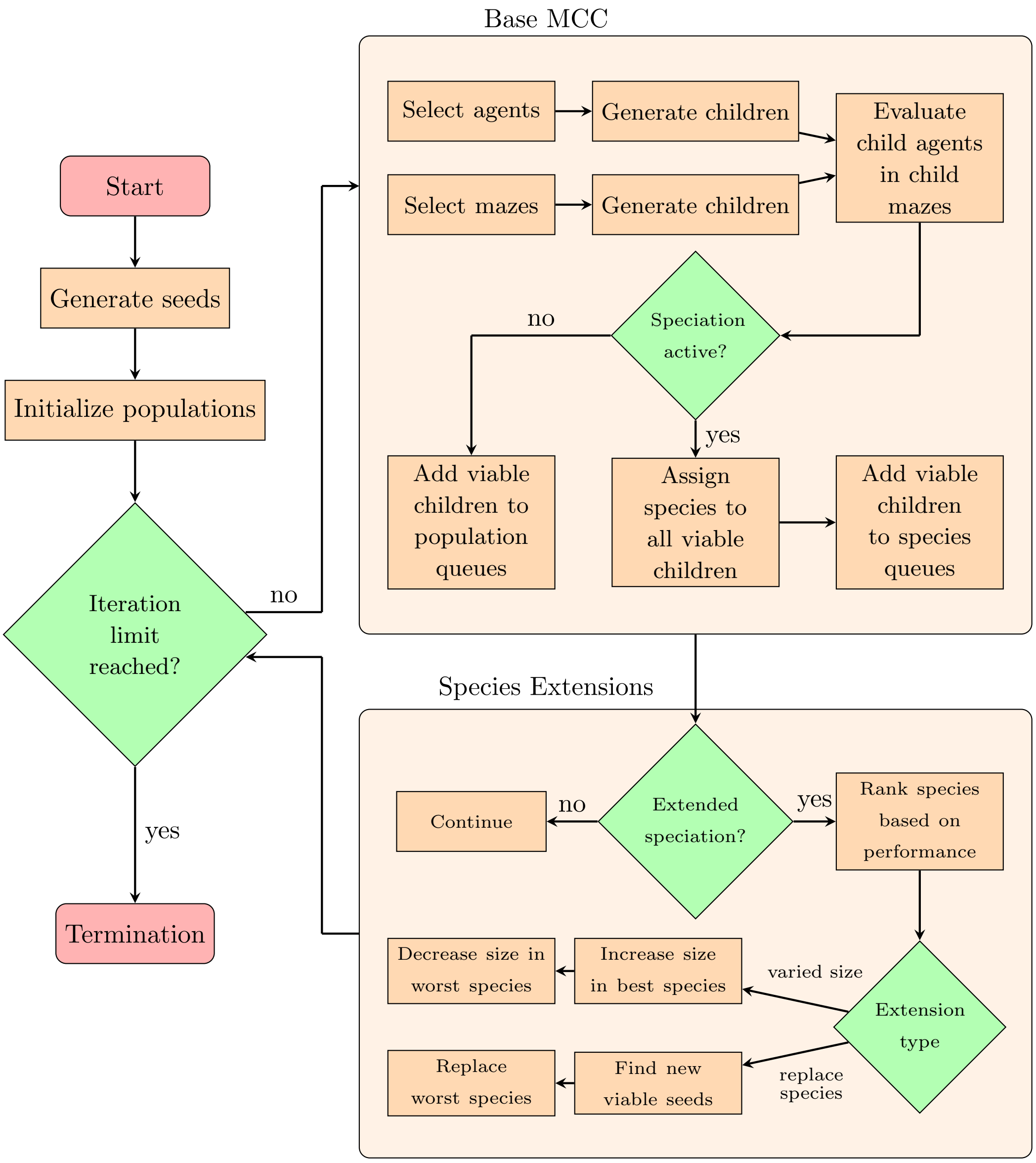


Figure 4.6: Overview of the model algorithm flow. The upper large container box contains the main functionality in the MCC cycle, where agents and mazes are selected, reproduced and evaluated. The flow in the base speciation without extensions is also included. The lower container box contains the flow in the extended speciation methods, and is the main contribution in this thesis. They are elaborated in more detail in section 4.3.2

4.3.2 Queue System and Speciation

A queue system is used to store all mazes and agents in the evolutionary cycle in MCC (the queue system is not visualised by the algorithm figure, 4.6). In the initial round, all agent- and maze seeds are placed in two queues, one for agents and one for mazes. Both queues have the same functionality and represent the populations in MCC. A pointer points to the next individual for selection. When the last individual in the queue is reached, the pointer goes back to the first index. If the queue reaches a maximum capacity set by model parameters, the oldest individual is removed.

Speciation functionality can be added to the queue system to split the two population queues into many smaller ones, each representing a species in the population. When speciation is active, all seed agents and mazes are placed in separate queues in the first cycle. This means that the number of queues is the same as number of seeds. Each seed is regarded as the centroid for a species and is used to determine how similar new individuals are to the species. The maximum capacity for the whole population, determined by a model parameter, is spread uniformly to all species queues. This speciation method will be referred to as the base speciation and is the same as used in Brant and Stanley [2019].

Agent Speciation

An adaption of the speciation method in NEAT is used as speciation method for the agents in the MCC model [Stanley and Miikkulainen, 2002]. Individuals in NEAT use the same genotype as the agents in the MCC model in this thesis (see section 2.5). New individuals are added to the species that they are most similar to, by calculating the genotypic distance between the new individual and all species centroids. A smaller genotypic distance means that they are more similar. Links between nodes in the neural network described by agent genotypes are used when comparing two agents. Table 4.3 gives an overview of the variables used in the comparison.

Description	**Abbreviation**
Links present in only one agent genotype	LP
Total sum of difference between links present in both genotypes	DL
Total number of unique links in both genotypes	TU

Table 4.3: Overview of variables used to calculate similarity in agents.

The difference between links present in both genotypes is calculated with their weight values. Equation 4.1 is used to calculate the difference between a link in the same position in genotype A and B. If the link is active in one and not the other, an additional value of 0.5 is added to the difference value.

$$Difference(A, B) = 0.5 * |tanh(A.weight - B.weight)| \tag{4.1}$$

If no links are present in any of the agent genotypes, a value of zero is set as the similarity. Equation 4.2 is used otherwise.

$$Similarity = (LP + DL)/TU \tag{4.2}$$

Maze Speciation

Mazes are also speciated based on genotype, where path genes and wall genes in two mazes are compared to find a similarity metric. This metric is used to determine which species queues new mazes should be added to by finding the most similar one.

The difference between a maze A and a maze B consists of two steps. First, all path genes in A and B with the same indexes in the mazes path gene lists are compared by calculating the euclidean distance between their coordinates, described by equation 4.3. If a path gene at a certain index is only present in one of the mazes due to different amounts of path genes, a substitute point with coordinates (0, 0) is used for the missing path gene. The differences between the path genes are then summed to obtain a total difference.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.3}$$

The second step compares all wall genes in two mazes. A scalar is constructed for both mazes by using the values within the wall genes. Equation 4.4 describes how each value in the scalars are calculated. To find the distance between the scalars, the euclidean distance function is used, specified in equation 4.6.

$$Scalar(w, p) = 0.5 * (w + p) * (w + p + 1) + p \tag{4.4}$$

w and p are the wall and passage values in a wall gene.

$$distance\,(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{4.5}$$

p and q are the wall genes scalar for two mazes.

Finally, the total difference is calculated by summing the total path gene difference and the wall gene distance.

$$(w, p) = 0.5 * (w + p) * (w + p + 1) + p \tag{4.6}$$

Speciation Extensions

Two extensions have been made to the original MCC model in this thesis. Both utilize a prioritization strategy, where both maze- and agent species are ranked by how performant they are. The performance of all species, both agent and mazes, is saved in every iteration in the evolutionary cycle.

Agent species are ranked based on their average increase in genotype size across all individuals, where a high increase is positive to avoid stagnant species. The maze species are ranked based on their average increase in size and complexity, where complexity is regarded as the number of turning points in the solution path in a maze.

Another problem occurring in species, both agents and mazes, is that some species are not able to evolve from their initial seed, which leads to very low performances. Since they are never able to evolve and reproduce into new viable individuals, their queue capacity is never filled. This means that the global population will not reach the maximum limit and not utilize all agent- and maze slots..

Dynamic Size in Species

The first extension is to use a dynamic maximum size in species. Base on the ranking strategy explained above, the top-ranking species are rewarded with an increase in queue capacity, while the capacities of the bottom-ranking species are reduced by the same amount. The total capacity in the maze and agent populations are static. Initially, all species are equally prioritized and have the same capacity in their queues. Changes to the capacities happen every 100 generations, to give species sufficient time to evolve after changes have been applied. Further, adjusting queue capacities to favor the performant species gradually overcomes the issue of empty slots in the species queues.

Species Replacement

The other speciation extension to the model in this thesis is to replace the least performant species with new ones during evolution. This effectively removes all stagnant species and those that are not able to evolve from their initial seed. Like the other experiment, the replacement process is run every 100 generations.

Three rules determine which species are replaced and how. First, if both stagnant maze- and agent-species are found, they are both replaced with a maze and agent pair as in the seeding with NEATNS. This means that the seed in the new agent species can solve the seed in the new maze species.

The second rule replaces stagnant agent species that are not able to evolve from their initial seed with species containing a seed that can solve a random maze in the maze population. To find the seed in the new species, NEATNS is used as in the seeding phase.

The third rule replaces the least performing maze species every time the replacement scheme is run. A random maze is generated and used as the seed in the new species. There are no guarantees that this new species will be able to evolve from its initial seed, as there are no agent individuals paired up with it when added.

However, adding new maze species increases the diversity in the maze population and encourages new agent lineages to emerge. This is also useful to see if agents can adapt to new environments.

4.3.3 Selection and Reproduction

The queues pointers are used to select parents to reproduce from. How many agents and mazes selected for each generation are set by parameters. Without speciation, all agents are selected from a single agent queue and all mazes are selected from a single maze queue. With speciation, individuals are selected uniformly from all agent- and maze-species. For example, if there are ten maze species and the amount of mazes to select is twenty, two mazes from each species are selected. All parents are copied and mutated to create the produce the children, and placed in at the last position of their queues by moving the queue pointers to the next index. Both agent- and maze children can be mutated in several different ways and are controlled by mutation chance model parameters, described in table 4.5.

Agent Mutations

Table 4.4 gives an overview of the possible agent mutations.

Add neuron	Adds a single hidden neuron.
Delete neuron	Marks a single hidden neuron as inactive.
Add connection	Adds a connection between two neurons.
Update connection	Updates the weight of a connection.

Table 4.4: Overview of possible agent mutations.

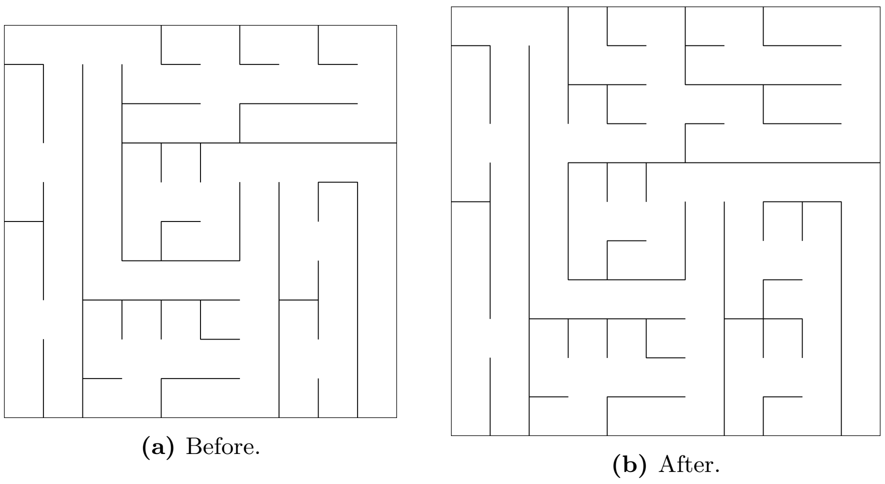
Maze Mutations

There are three main branches in the maze mutations. Mutations can increase the dimensionality of a maze, update the path genes, and update the wall genes. The following sections present all possible maze mutation variants.

Some mutations might lead to invalid maze phenotypes. For example, certain updates to the path genes might result in a solution path that crosses itself and could end up having multiple paths to the target location. A maze validation is therefore used to ensure that the mutation on a maze genome leads to a valid phenotype. If the resulting mutated maze is invalid, the mutation is discarded.

Mutate Structure

A maze before and after increasing the dimensions is seen in figure 4.7. Mutating the structure of a maze increases the width and height of the maze by one cell in the south-east direction. To accommodate, the last juncture in the solution path is moved to the right to follow the expanding right edge. This way it is still aligned with the target location.

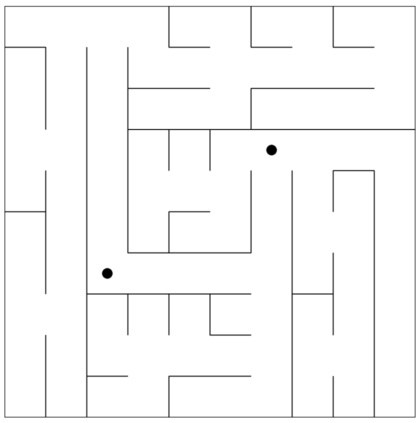

(a) Before.

(b) After.

Figure 4.7: A maze before and after the mutate structure mutation

Add Path Gene

Adding a path gene inserts a new path gene with random values at the end of the path gene list, visualized by figure 4.8. This means that this path gene represents the last turning point in the maze.


(a) Before.

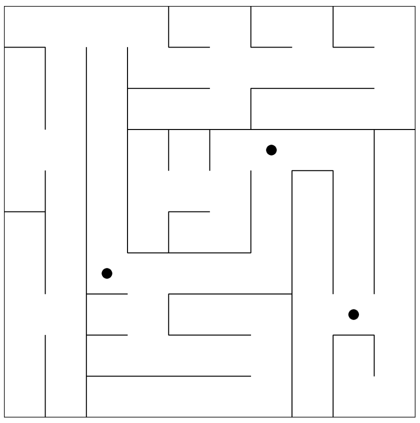
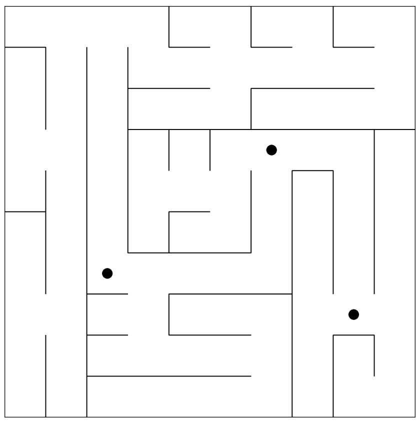

(b) After.

Figure 4.8: A maze before and after the 'add path gene' mutation.

Update Path Gene

Updating a path gene moves the turning point specified in the path gene one cell in a random direction, visualized by figure 4.9. Path genes cannot be updated to go outside of the maze boundaries.


(a) Before.

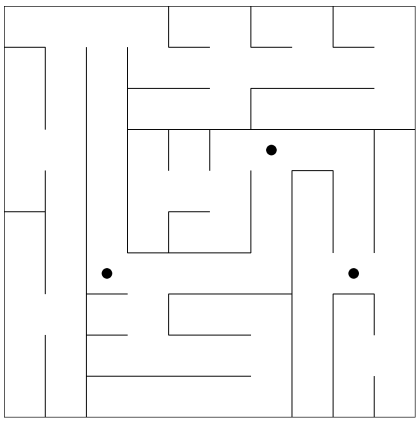

(b) After.

Figure 4.9: A maze before and after the 'update path gene' mutation.

Wall Gene Mutations

Wall genes can be added, updated and deleted. Adding a wall gene inserts a new wall gene at the end of the wall gene list in the maze genotype. There are two update-variants used, one that updates the wall value and one that updates the passage value in a single wall gene. When a value in the wall genes are updated it is assigned a new random value between 0 and 1. Deleting a wall gene removes a random wall gene from the wall gene list.

All wall gene mutations might change the structure of the walls in the subdivision quite drastically. However, they do not update the solution path in any way.

4.3.4 Evaluation

To determine whether the agent and maze children are viable or not, a maze navigation simulator is used to simulate and evaluate all agent children in all maze children.

Simulator Overview

The simulator begins by generating the maze and agent phenotype and places the agent body in the upper left corner, with its front pointing towards the bottom right corner. An agent uses its neural network to map sensor values to velocity updates. This process has multiple steps and is repeated until the agent has reached the target position or the maximum number of timesteps is used. Figure 4.10 presents the general flow in this process.

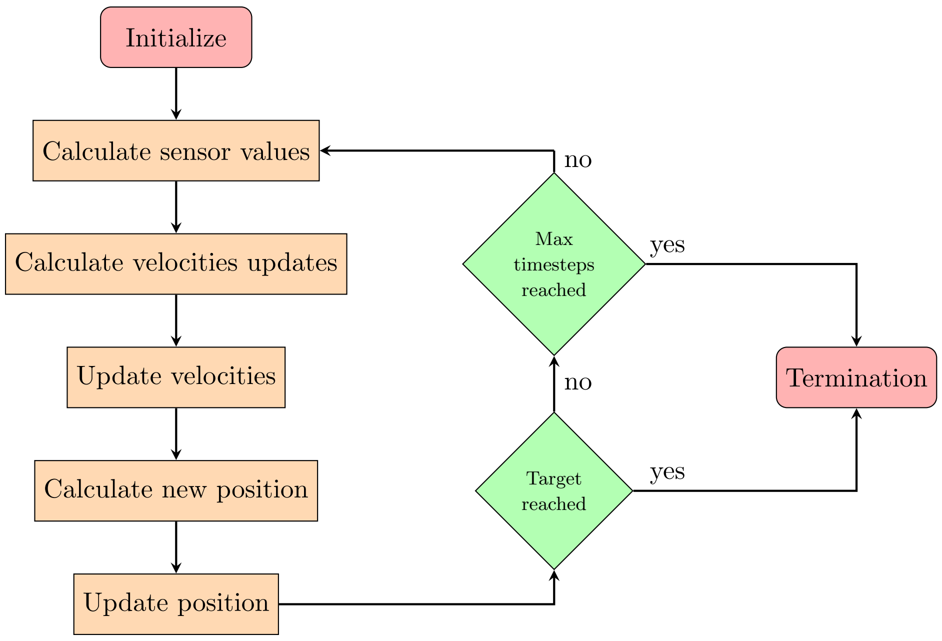


Figure 4.10: The simulator loop for an agent in a maze.

The maximum number of timesteps is determined by the length of the solution path in the maze. This assures that agents must find the correct path in a reasonable time and discards agents that go amiss for too long. The agent is successful when it has reached the boundaries of the maze cell in the rightmost lower corner.

A unit system is used to keep track of an agent's position and how fast it is currently moving. Each maze cell is measured as 32 units, and an agent can have a maximum speed of 3 units per timestep, both in positional and angular velocity. Further, an agent has a physical radius of 3 units and is used to detect wall collisions, explained in more detail in 4.3.4.

Sensor Values

The first step in the simulator cycle is to calculate the sensor values. These are calculated with trigonometric functions and positional values, where the Pythagorean theorem is a central component. Figure 4.11 gives an overview of how the rangefinder sensors are used in a maze to find distances to the surrounding walls. The distances are measured from the center of the agent body.

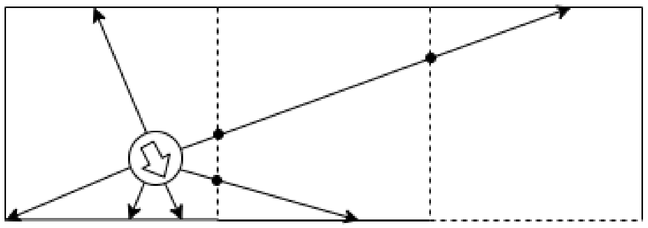


Figure 4.11: Overview of an agent and its rangefinder sensors in a maze.

In figure 4.11 there are three maze cells. The solid lines indicate where walls are placed, while the dashed lines represent the boundaries of the maze cells without walls. When a rangefinder sensor value is calculated, the distance traveled in each maze cell is found in a recursive process and summed. For each maze cell in the rangefinder line of sight, the process checks if the line has hit an actual wall in the collision point at the cell boundaries. If a wall is found, the total distance is returned. Otherwise, the search continues into the cell on the other side of the boundary. The black dots in the image represents the collision points between the rangefinder sensors and cell boundaries.

Rangefinder Calculation

There is a general pattern in calculating the rangefinder sensor values. The position of an agent and the angle between the rangefinders direction and the x-axis is used to determine the length of two catheti in a perpendicular triangle, illustrated in figure 4.12 and 4.13. The Pythagorean theorem is then used to calculate the length of the hypotenuse.

The following figures only cover rangefinder sensors that are pointing towards the north east. Sensors that points in other directions follow the same general pattern but use different angles to calculate the sensor value.

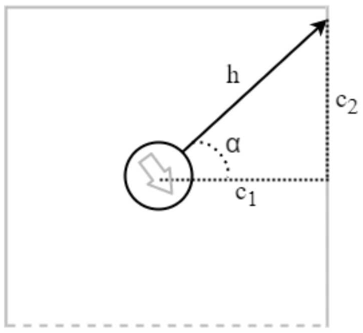


Figure 4.12: Catheti setup.

In figure 4.12, the c1 and c2 catheti can be found by using the position of the agent and the location of the maze cell in the maze, described by equation 4.7. The angle α can be found by using the heading of the agent and the direction of the rangefinder sensor.

$$c_2 = (x_c - x_a) * tan(\alpha) = c_1 * tan(\alpha) \tag{4.7}$$

where x_c is the x-coordinate at the right edge of the cell and x_a is the agents current x-coordinate.

$$h = \sqrt{c_1^2 + c_2^2} \tag{4.8}$$

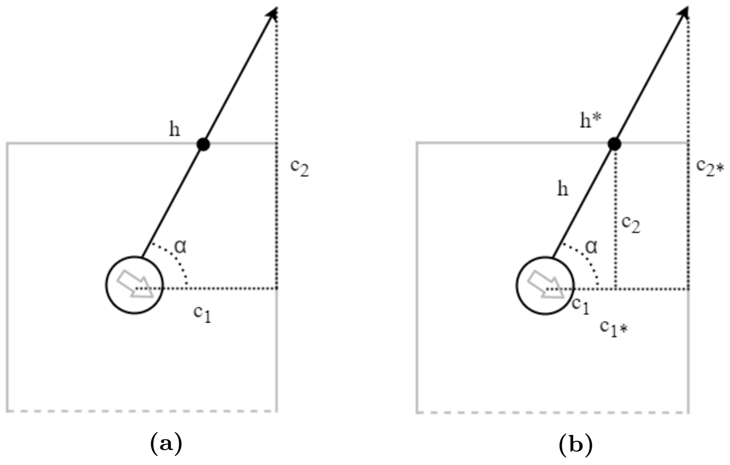


Figure 4.13: Catheti setup with extra calculations.

Figure 4.13 visualizes a case where the rangefinder line points towards the upper border of the maze cell. Some extra calculations are required in such situations. First, the catheti c_1* and c_2* are calculated in the same way as in figure 4.12. Second, if c_2* extends beyond the upper border in the maze cell, the hypotenuse has to be shortened down. Another set of catheti is therefore found by using the vertical position of the agent and y-coordinate of the upper maze cell border. Equation 4.9 describes how to find these. Finally, the hypotenuse is calculated in the same way with the Pythagorean theorem with equation 4.8.

$$c_1 = (y_c - y_a) * tan(90 - \alpha) = c_2 * tan(90 - \alpha) \tag{4.9}$$

where y_c is the y-coordinate at the upper edge of the cell and y_a is the agents current y-coordinate.

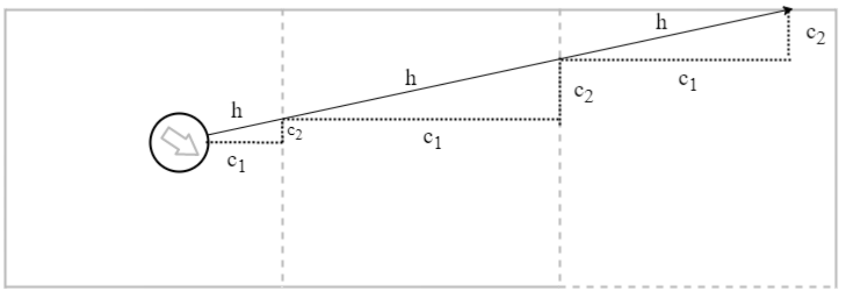


Figure 4.14: Rangefinder continuing to other maze cells.

If the rangefinder line does not meet a wall in a maze cell, it continues into the cell on the other side of the cell border, visualized in figure 4.14. The same process is repeated here where with the border collision point as the base point, and continues recursively until a wall is met. The sum of all hypotenuses found is the final value of the rangefinder sensor.

Radar Sensors

To find which radar sensor is active, a line from the target location is drawn to the agent's current position. The radar sensor with this line within its area of sight is the active one, visualized by 4.15.

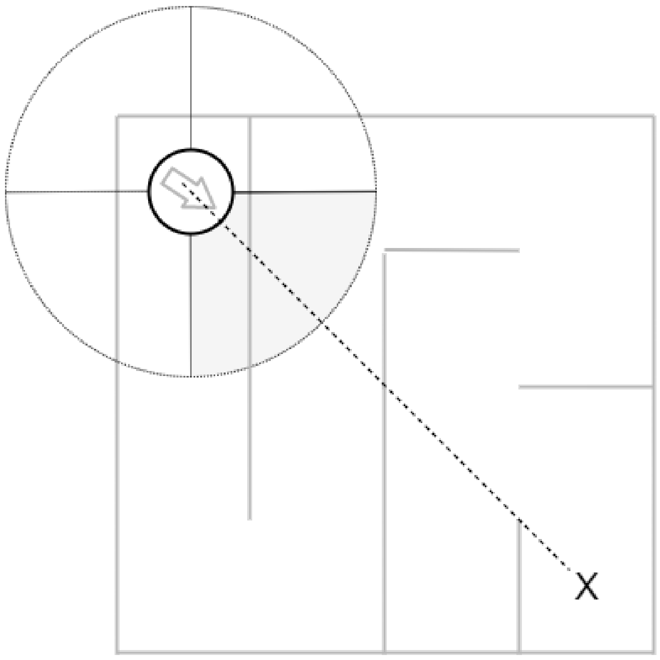


Figure 4.15: The agent has four radar sensors, where the grey one is currently active.

Updating Position and Rotation

Values from sensors are sent to the neural network of an agent, which returns two values used to update the positional and angular velocities. If an updated velocity reaches higher than the limit, it is decreased down to the limit.

A new position for the agent is calculated based on the current position, rotation, and new velocities. Equation 4.10 and 4.11 describes the equation used to find new x- and y-coordinates.

$$x = x_0 + cos(\frac{d}{180 * \pi}) * v \tag{4.10}$$

$$y = y_0 + sin(\frac{d}{180 * \pi}) * v \tag{4.11}$$

where x_0 and y_0 are the current coordinate values, d is the current direction angle, v is the updated positional velocity.

In many cases, the new position would lead the agent to collide with a wall. To detect collisions, the simulator checks if the new position is closer to a wall than the radius of the agent body. If this is true, the position update is skipped. Otherwise, the agent is moved to the new position and begins a new loop of the simulator cycle.

4.4 Parameters

4.4.1 MCC

Table 4.5 presents the parameters used in the MCC cycle of the model.

MCC parameters	
Generations	The number of generations to evolve the populations.
Maze seed amount	Amount of viable mazes to find in initial seeding phase.
Maze queue capacity	Capacity of the queue storing all viable mazes found.
Maze selection limit	Maximum amount of mazes to select in a round. This is the amount of maze children generated and evaluated in each round.
Initial maze size	The initial size of a maze in cells when randomly generated .
Initial maze waypoint amount	The maximal amount of waypoints a maze can have when randomly generated .
Agent seed amount	Amount of viable agents to find in initial seeding phase.
Agent queue capacity	Capacity of the queue storing all viable agents found.
Agent selection limit	Maximum amount of agents to select in a round. This is the amount of agent children generated and evaluated in each round.
Maze Mutation Parameters	
Increase size	Chance of expanding the maze by one cell in width and height.
Add wall	Chance of adding a wall gene.
Update passage value	Chance of updating the passage position value in a wall gene.
Update wall value	Chance of updating the wall position value in a wall gene.
Delete wall	Chance of deleting a wall gene.
Add waypoint	Chance of adding a waypoint gene.
Update waypoint	Chance of updating a waypoint gene.
Agent Mutation Parameters	
Update connection weight	Chance of updating a random weight value.
Add connection	Chance of adding a connection between two neurons.
Disable connection	Chance of disabling a connection.
Add neuron	Chance of adding a neuron.

Table 4.5: Overview of parameters.

4.4.2 Seeding Phase: NEATNS

The seeding use a set of parameters separate from the parameters used in the MCC cycle, and is presented in table 4.6.

NEAT Parameters	
Population size	Amount of agent individuals.
Speciation threshold	Threshold for adding a new individual to an existing species.
Interspecies reproduction chance	Chance to reproduce new individuals with parents from different species.
Interspecies tournament size	Size of selection tournament with multiple species.
Dropoff age	Amount of generations before an individual is marked as stagnant.
Young species fitness multiplier	A constant used to adjust the fitness of young individuals.
Young age limit	Amount of generations where the individual is marked as young.
Stagnant species fitness multiplier	A number used to adjust the fitness of stagnant individuals.
Survival ratio	Ratio between surviving and dying individuals used when truncating the population.
Elitism amount	The number of individuals to keep from the last generation.
Activation variants	Specifies which activation functions that can be used.
NEAT Mutation Parameters	
Add node	Chance of adding a node.
Add connection	Chance of adding a connection.
Disable connection	Chance of disabling a connection.
Update connection weight	Chance of updating a connection weight.
Hidden bias	Chance of updating a hidden node bias.
Hidden activation	Chance of updating a hidden node activation.
Output bias	Chance of updating a hidden node bias.
Output activation	Chance of updating a output node activation.
NS Parameters	
Neighbors amount	How many of the closest neighbors to use when calculating the novelty of a new end position .

Table 4.6: Overview of parameters used in NEATNS.

Chapter 5

Experiments and Results

This chapter presents the experimental plan and results in this thesis. First, the preliminary tests during implementation will be discussed, followed by an overview of the main experiments is presented in section 5.2. The experimental parameters used are described in section 5.3, while results from the experiments are presented and elaborated in 5.4. A comparison to the results in Brant and Stanley [2019] is presented in 5.5. Finally, a summary of the results is presented in section 5.6.

5.1 Preliminary Tests During Implementation

During implementation, the model was gradually tested as more base functionality was added. The main focuses were to achieve similar results as Brant and Stanley [2019] with and without speciation. Note that this was before the extended speciation methods were implemented. The base speciation method described in 4.3.2 was tested in this phase.

5.1.1 MCC Without Speciation

The implementation was tested as a whole to validate the base functionality and expected behavior before any speciation functionality were made, by verifying that the initial seeds were able to increase in size and complexity. Also, the performance of the implementation was tested to find possible bottlenecks and errors.

5.1.2 MCC With Speciation

Testing the model with the base speciation was conducted many times to find and explore patterns in how the species evolved. Results from these test-runs indicated that in many cases species were not able to evolve out from their initial seed. Since each species have the same amount of slots in the population reserved, this meant that many slots were never utilized. This issue guided the development of the extended speciation methods, by aiming for functionality that minimizes the number of unused slots and prioritizing the species with the highest increase in complexity.

These results were also compared to the results in Brant and Stanley [2019] to see if the implementation could reach the same maze sizes and complexity in the same number of generations. Surprisingly, the mazes grew much faster than expected and surpassed the sizes reached in Brant and Stanley [2019]. However, after further inspection, the rapid expansion did not balance well with how the complexity in the solution paths through the mazes grew, which resulted in many uninteresting mazes. The solution paths were fairly simple in the majority of the mazes, with few turning points and almost no passage openings to subdivisions that could set the agents off course. This led to an analysis of how the mazes were mutated, which revealed that the mazes were in few cases able to add more junctures to the solution path. On the other hand, the size expansion mutation was always successful. A rebalance of this was therefore implemented, where the "add path gene" mutation tries to add a juncture for up to ten times.

5.2 Overview of Experiments

This thesis investigated two experiments, where the MCC algorithm with different speciation extensions is tested. Also, the same MCC implementation was run without any speciation extensions to have comparable results. Table 5.1 gives an overview of the experiments naming and abbreviations. Each experiment is run twenty times in separate batches.

Dynamic Size in Species Experiment	DSE
Species Replacement Experiment	SRE
Base Speciation Experiment	BSE

Table 5.1: Overview of experiments and abbreviations.

5.2.1 Dynamic Size in Species Experiment

This experiment runs MCC with the dynamic size speciation described in section 4.3.2 turned on.

Hypothesis

In this experiment, the number of queue slots in the most performant species increases, while the queue sizes of the least performant decreases. The most performant species are given more room to explore their local optima in additional directions. It is expected that the overall complexity in mazes and agents will increase with a faster rate than BSE. Further, the average population size is expected to be higher as stagnant species give queue capacity to species that can evolve further.

The diversity in the resulting solutions is expected to be quite similar to BSE, as the base in the process is still the initial seeds. However, the diversity might be lower due to disregarding the potential lineages in the least performant species, resulting in fewer being found and evolved.

5.2.2 Species Replacement Experiment

This experiment runs MCC with the species replacement functionality described in section 4.3.2 turned on.

Hypothesis

Some of the behavior from the BSE is also expected to occur in this experiment. By ranking species and replacing the worst, a faster growth rate in size and complexity is expected, as well as higher utilization of empty population slots. The diversity in the resulting solutions is expected to increase as new species are gradually introduced to the population, resulting in new lineages discovered continually.

5.3 Experimental Setup

This section describes the setup and configurations used for each run. All experiments use the same values in the general MCC cycle but with different speciation variations. All parameters are presented in table 5.2.

General parameters	
Batches	20
MCC parameters	
Generations	2000
Maze seed amount	10
Maze queue capacity	250
Maze selection limit	10
Initial maze size	10
Agent seed amount	20
Agent queue capacity	250
Agent selection limit	40
Maze Mutation Parameters	
Increase size	0.1
Add wall	0.1
Update passage value	0.05
Update wall value	0.05
Delete wall	0.005
Add waypoint	0.1
Update waypoint	0.05
Agent Mutation Parameters	
Update connection weight	0.6
Add connection	0.1
Disable connection	0.005
Add neuron	0.01

Table 5.2: Overview of parameters.

5.3.1 NEATNS Parameter Values

The values used for the parameters used in NEATNS in the seeding phase is presented in 5.3. These values were used in all experiments.

NEAT Parameters	
Population size	100
Speciation threshold	0.85
Interspecies reproduction chance	0.15
Interspecies tournament size	2
Dropoff age	30
Young species fitness multiplier	1.05
Young age limit	20
Stagnant species fitness multiplier	0.2
Survival ratio	0.4
Elitism amount	1
Activation variants	None, ReLU, Sigmoid, Linear, Sine, Square, Exponential
NEAT Mutation Parameters	
Add node	0.05
Add connection	0.08
Disable connection	0.05
Update connection weight	0.8
Hidden bias	0.8
Hidden activation	0.1
Output bias	0.8
Output activation	0.05
NS Parameters	
Neighbors amount	15

Table 5.3: Overview of parameter values used in NEATNS.

5.4 Experimental Results

This section presents and evaluates the results from the experiments. First, figure 5.1 and figure 5.2 presents mazes that were generated from a single experiment run.

There are five different metrical types of results presented. The first three concerns how the agents and mazes evolve throughout the run, where maze dimensionalities, maze complexities, and the size of agent genotypes are measured and analyzed from the beginning to end in each run. Results from all runs in an experiment are averaged in terms of these metrics and are presented in the following sections. The standard deviation for all values is also presented in the graphs, indicated by the faded color surrounding the lines. Additionally, how much these values increase on average is presented at the end of each section. Further, the average number of individuals in the maze- and agent populations throughout the experiment runs are presented in section 5.4.4. Finally, the diversity metric in all experiment runs as well as the average diversity metric in each experiment is presented 5.4.5.

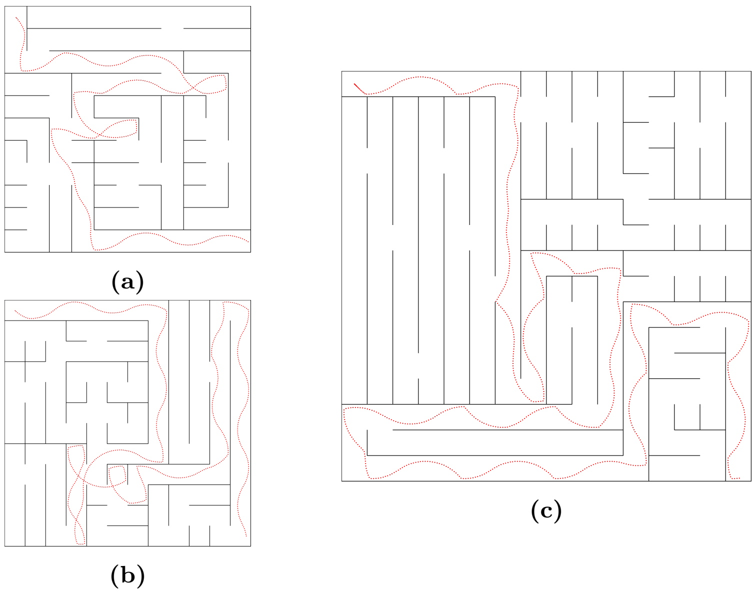


Figure 5.1: All trajectories are generated from a single SRE run, and displays that diverse maze and large mazes can be found. The red dots inside the mazes indicates the paths the agents found to get through the mazes. **(a)** has a size of 11x11 cells, **(b)** has a size of 12x12 cells and **(c)** has a size of 16x16 cells.

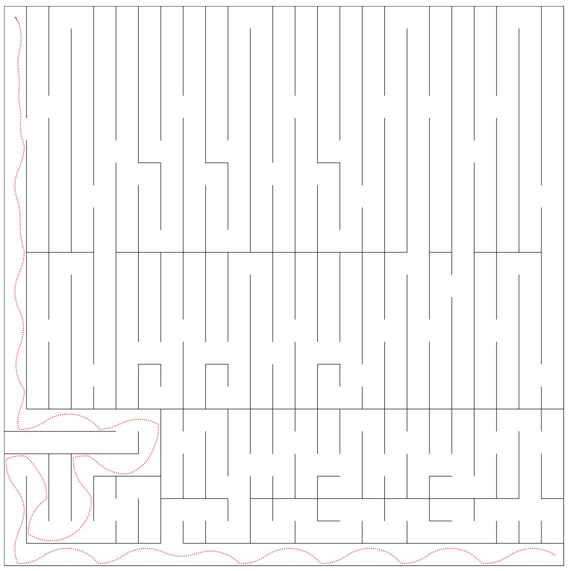


Figure 5.2: Successful agent trajectory in a 25x25 maze. Note that the solution path is fairly simple, consisting of mostly straight lines. This is an example of a maze that most likely has followed the "path of least resistance" mentioned in 3.2.

5.4.1 Dimensionality in Mazes

The following graphs present how the size of all mazes evolves in each experiment. Results from all runs in an experiment are averaged to the maximum, average, and minimum size in mazes.

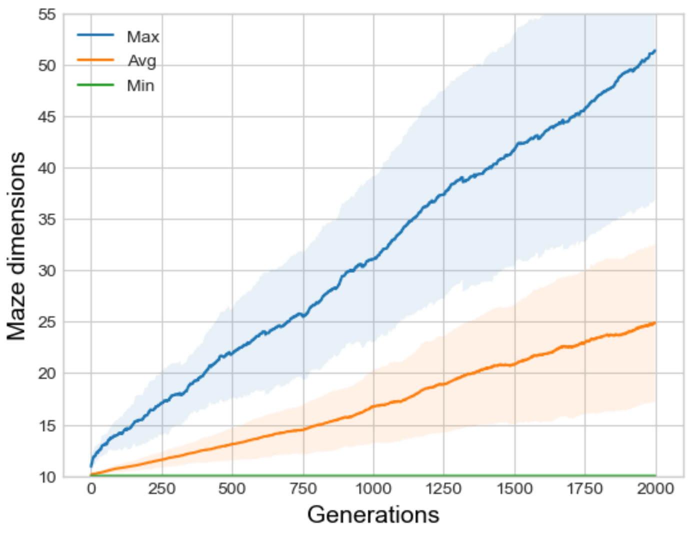


Figure 5.3: Maze sizes in BSE

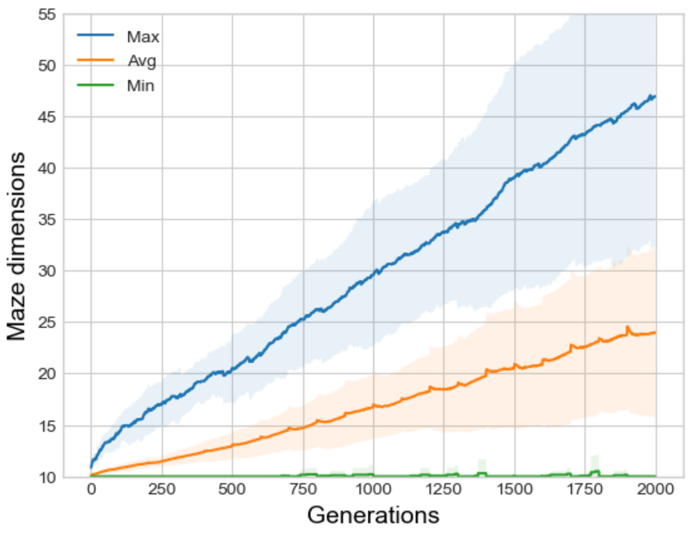


Figure 5.4: Maze sizes in SRE

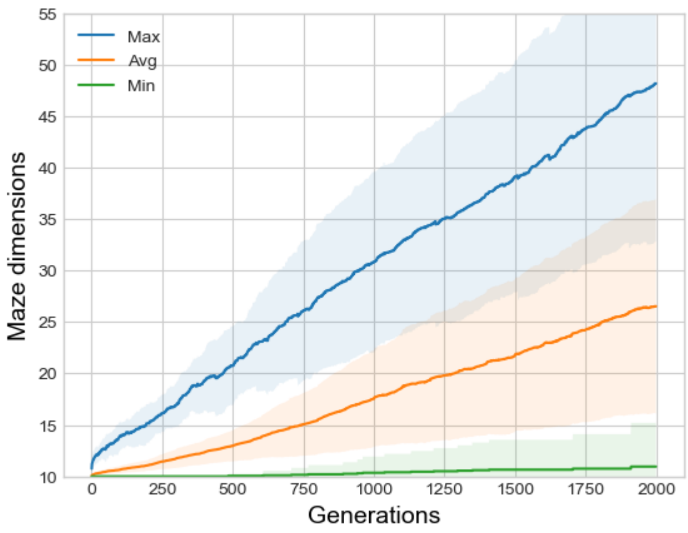


Figure 5.5: Maze sizes in DSE

All three experiments are quite similar when compared in terms of maze dimensions throughout the runs. There are some small differences. Runs from DSE, seen in figure 5.5, were able to achieve the highest average maze dimensions, while runs in BSE were never able to increase the dimensions of the smallest mazes, visualized by figure 5.3. However, BSE achieved the highest maximum dimensions. In the SRE graph (figure 5.4) there are spikes located periodically, which most likely are due to the points in time where species are replaced. Another trait in the DSE graph is that the standard deviation is higher and grows faster, indicating that the gap between the lowest and highest average maze dimensions in these runs is high.

Average Increase

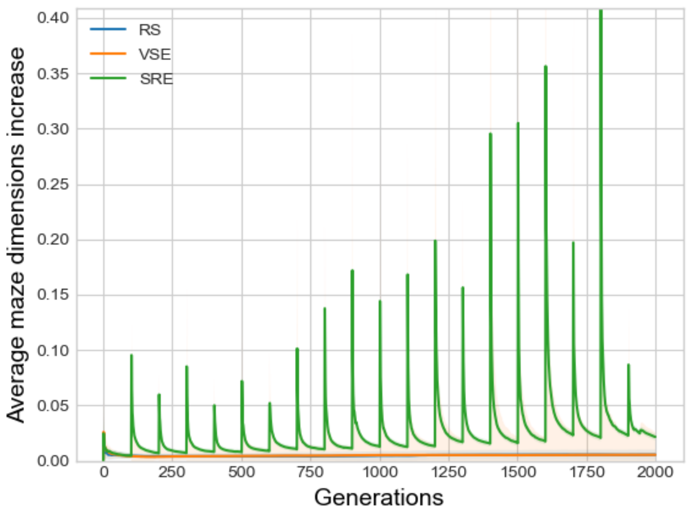


Figure 5.6: Average increase in maze dimensions in each generation.

Figure 5.6 visualises that the average increase in BSE and DSE are very similar and converges quite fast. In the graph, the BSE line is hidden by the DSE line. The SRE line has a very different behavior than the other two, as the value is always higher and seems to restart every 100 generations when species are replaced.

5.4.2 Complexity in Mazes

The following graphs present how the complexity of all mazes evolves in each experiment. Results from all runs in an experiment are averaged to the maximum, average, and minimum complexities in mazes. The complexity of a maze is measured by counting the number of junctures in the solution path where the agent must make a turn.

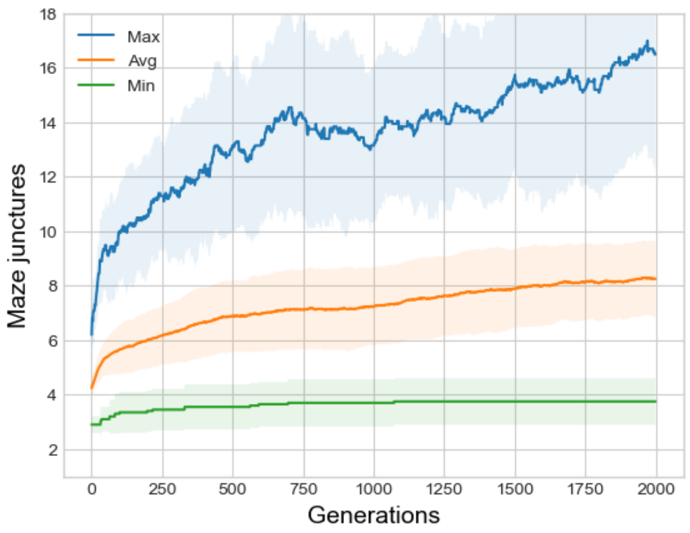


Figure 5.7: Maze complexities in BSE

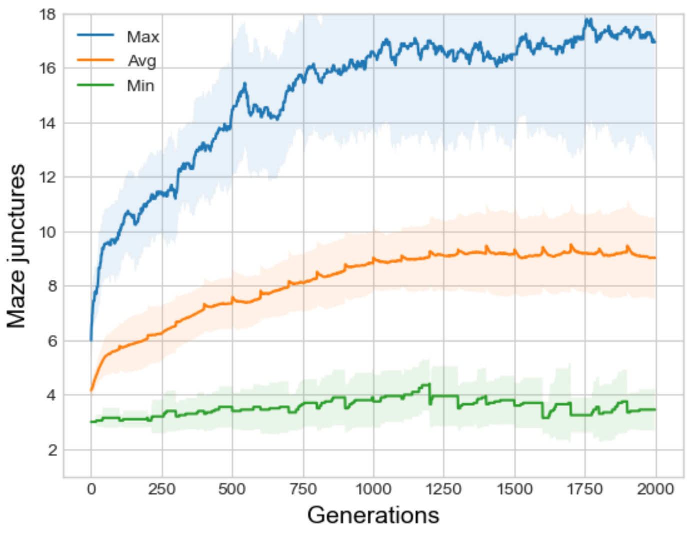


Figure 5.8: Maze complexities in SRE

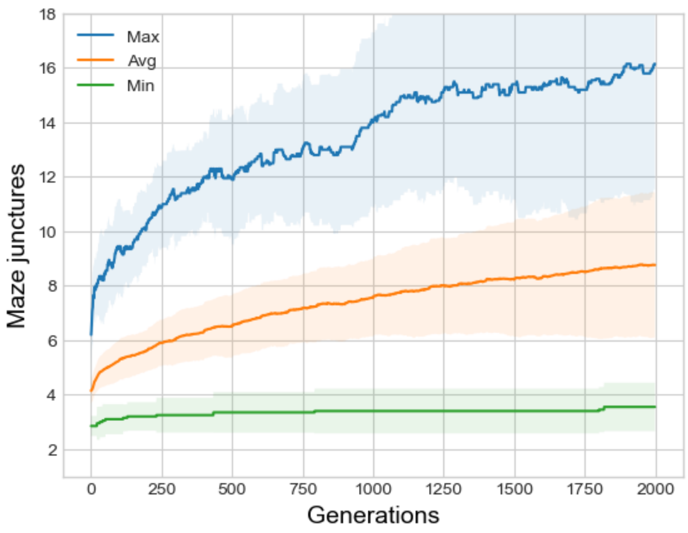


Figure 5.9: Maze complexities in DSE

The complexities in mazes are able to increase throughout the runs in all experiments. The lines representing the maximum complexities in mazes are a lot more turbulent than the lines representing average and minimum complexities, as they have a lot of ups and downs. Some lines also have long periods with almost only decrease, for example in figure 5.8, where the maximum complexity right after generation 500 decreases for about 100 generations. This indicates that some maze individuals might have increased too rapidly in complexity without being able to reproduce into viable children, resulting in being discarded from their queue.

Both SRE and DSE were able to achieve a higher average complexity than BSE. However, the average complexity in SRE seems to have stagnated from around generation 1000 and is decreasing towards the end. Similar to the maze size graphs, the SRE has spikes in the average line, while lines in DSE have higher standard deviations. This further indicates the replacement functionality in SRE and high variations in DSE.

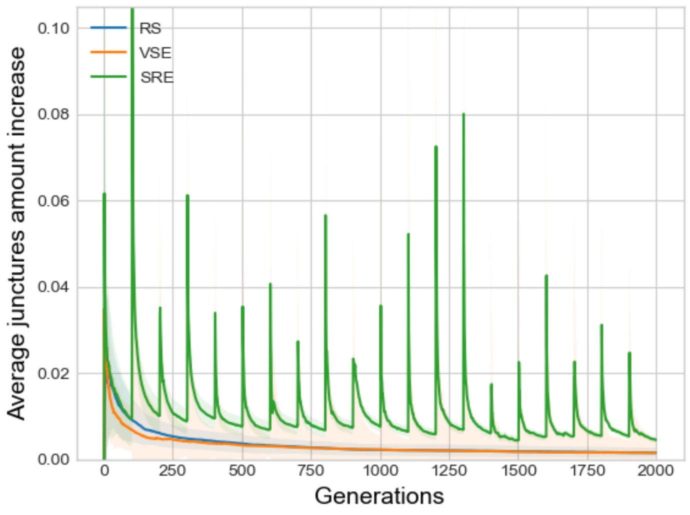


Figure 5.10: Average increase in maze complexities in all experiments.

How the mazes increase in complexity in figure 5.10 have very similar traits to how they increase in size, seen in figure 5.6. Both BSE and DSE are overlapping and converges to the same value. The green line representing SRE have spikes like all prior SRE graphs. A periodically high increase in complexity for short periods matches the line representing the average maze complexities in figure

5.8. However, even though SRE always has a higher average increase in maze complexities, it has similar performance as the other experiments with regards to average and highest maze complexities achieved. A possible reason for this is that SRE continually adds new species that might not have high complexity. The line representing minimum complexity in figure 5.8 further indicates that SRE replaces maze species with a worse maze complexity. Subsequently, the increase in maze complexity in SRE spikes as the 'add path gene' mutation has a higher chance of being successful for mazes with few path genes.

5.4.3 Size of Agent Genome

The following graphs present how the size in all agent genotypes evolves in each experiment. Results from all runs in an experiment are averaged to the maximum, average, and minimum agent size in mazes. The size of an agent genotype is measured by counting the number of links between nodes in the neural network.

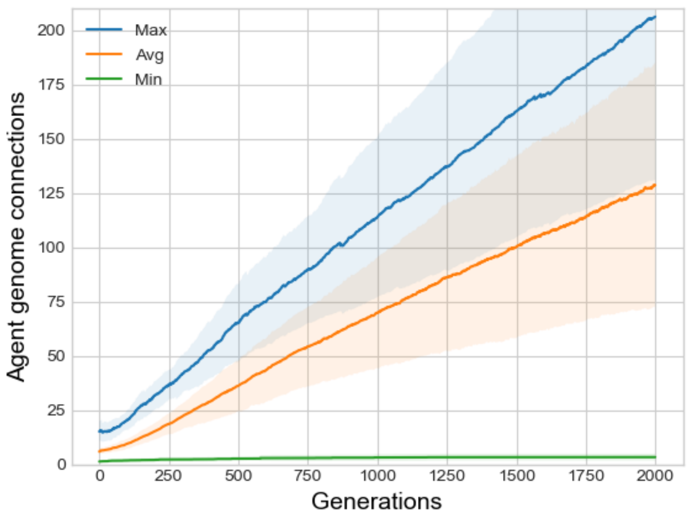


Figure 5.11: BSE

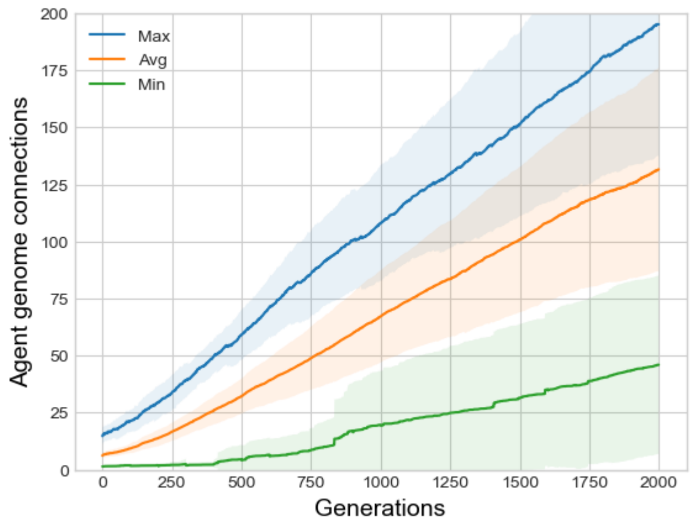


Figure 5.12: SRE

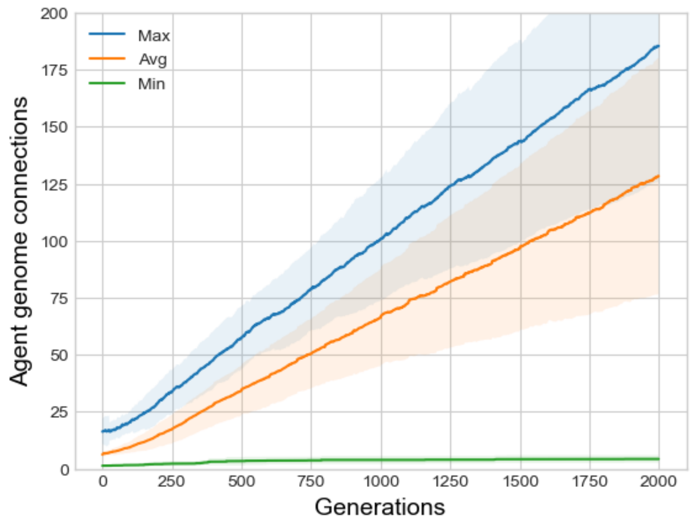


Figure 5.13: DSE

The most significant difference between the experiments in terms of agent genotype sizes is the minimum size obtained by SRE, as seen in figure 5.12. Both DSE and BSE are never able to evolve all agent individuals, resulting in the minimum agent size being stagnant.

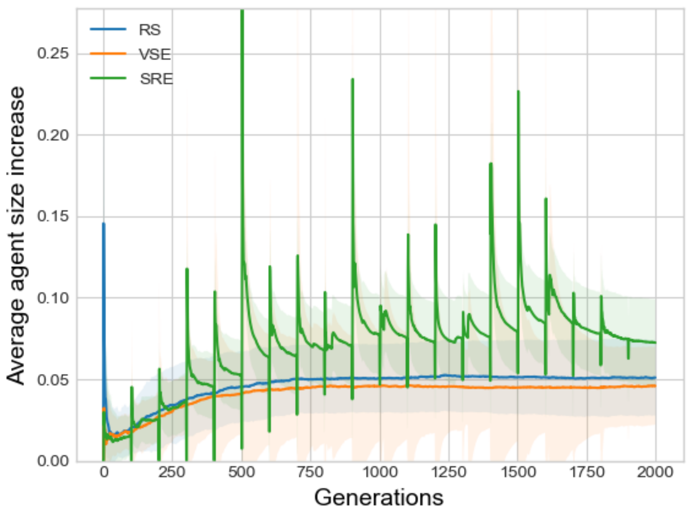


Figure 5.14: Average increase in agent genotype size in all experiments.

From figure 5.14, the agent genotypes increases similarly in DSE and BSE, while SRE behaves similar to figure 5.10 and 5.6 with spikes every time species are replaced. All graphs seem to increase steadily for a while before converging. A difference between DSE and the others is the standard deviation which also has spikes every 100 generations.

5.4.4 Population Capacity Used

The following graphs present the total number of individuals in the maze- and agent populations in all experiments.

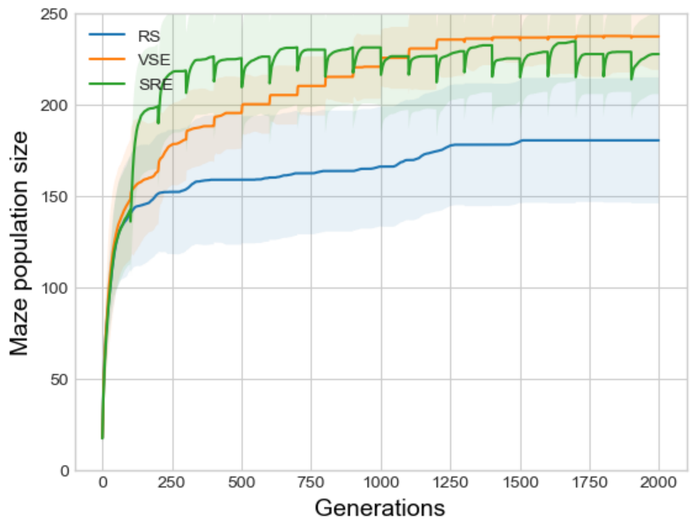


Figure 5.15: Average amount of mazes.

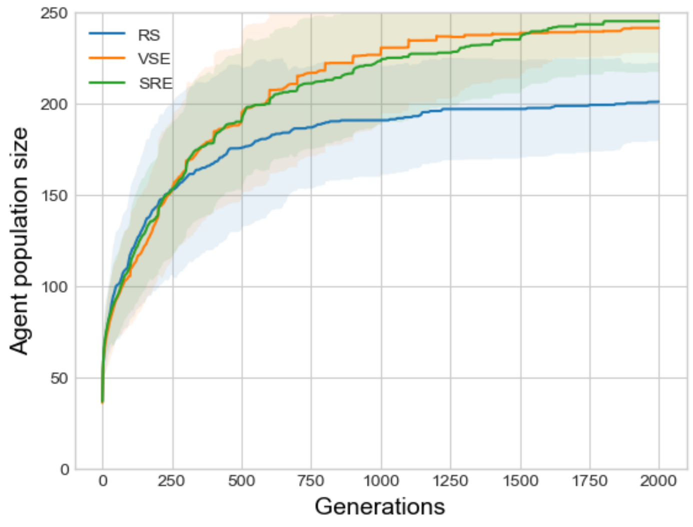


Figure 5.16: Average amount of agents.

As seen in figure 5.15 and 5.16, SRE and DSE are not surprisingly able to utilize more slots in the population than BSE, by giving more capacity to the most performant species and replacing the worst species. The functionality in the two speciation extensions is also visible in the graphs representing their population sizes. DSE follows a staircase pattern indicating that the capacity is gradually transferred from stagnant species to the most performant species. Further, the replacement functionality in SRE can be seen in both graphs. In figure 5.15 the SRE graph has these periodically small gaps representing where the worst maze species are replaced. On the other hand, in figure 5.16 the agent population in SRE continues to grow until maximum capacity is reached, as the worst species are replaced until only non-stagnant are found.

5.4.5 Diversity in Solutions

Diversity is measured by comparing trajectories of successful agents in the end populations, in the same way as in Brant and Stanley [2017]. Each point in a trajectory is compared with the euclidean distance function to the point at the same time step in all other trajectories. These distances are then averaged to find the diversity metric.

RS	**SRE**	**VSE**
16.294	9.846	8.250
15.359	6.079	13.464
5.341	16.422	5.778
22.049	9.049	6.623
15.439	8.938	11.202
11.832	10.244	15.196
14.823	10.965	14.327
12.459	11.519	24.279
15.227	6.572	10.791
15.124	16.067	12.319
7.746	12.147	17.893
16.290	9.904	14.165
7.046	9.945	10.418
11.207	8.097	5.137
7.766	17.198	11.278
11.478	12.461	12.805
14.785	7.346	9.906
5.492	8.717	16.241
18.910	6.654	17.515
14.785	6.082	4.472

Table 5.4: Overview of diversities in all runs.

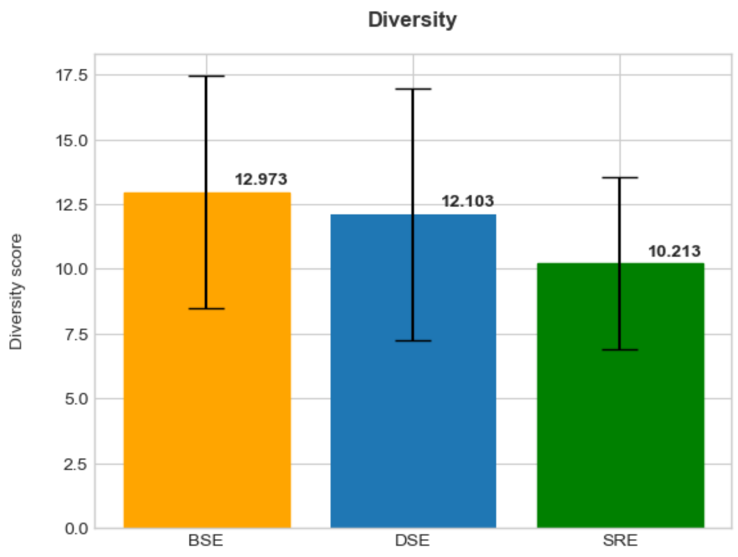


Figure 5.17: The average diversity of all runs in all experiments.

Figure 5.17 visualizes the diversity metric for all experiments, where the metric for BSE and DSE are quite similar, while the metric in SRE is surprisingly the lowest one. This opposes the SRE hypothesis, which stated that the overall diversity would increase and achieve a higher diversity than BSE. A possible reason might be that species are replaced before lineages are able to emerge from them, which forces the diversity to be based on fewer lineages. Another factor might be that the replacement process targets mazes each time it is run, while the agents are only replaced if they are completely stagnant. Perhaps the agents are not able to adapt to new mazes that easily, resulting in multiple agent lineages focusing on the same maze lineages. Based on the way diversity is calculated this would lower the overall metric as the same mazes and same solution paths are used over again.

5.5 Versus Initial Research

The work in this thesis is based on Brant and Stanley [2019] where MCC is used to generate mazes and navigator agents that can solve them. The speciation method they used is the same as used in the BSE experiment. In general, the results in this article follow the same pattern as the BSE experiment. However, as the maze representations used are not completely identical, there are some differences in how quickly the mazes are able to increase in size and complexity. The maze representation used in this thesis leads to slightly ”easier” mazes, containing fewer crossroads where the agents can make the wrong turn. Consequently, experiments in this thesis were able to increase the size of mazes faster.

5.6 Summary

Overall, the SRE and DSE are not that different from BSE. Some differences are noticeable in the evolutionary process but the end populations are still quite similar in complexity, size, and diversity. While the speciation extensions are better at utilizing all slots in the population, this does not seem to have that much impact in general and is in some metrics beaten by the BSE.

Chapter 6

Conclusions and Future Work

This section discusses the results with regards to the research goal and research questions, followed by the contributions in this thesis and an overview of future work in section 6.2 and 6.3.

6.1 Discussion

In this thesis, the overall goal was to find ways to increase the efficiency in MCC with regards to the maze domain. The research questions highlighted in this section led the overall direction in this project and are discussed below.

Research question 1 *How can ranking and prioritizing species be used to increase the overall performance in Minimal Criterion Coevolution*

This thesis has investigated two different ways to extend the speciation method in MCC. Each of them investigates how small tweaks in the speciation method can lead to considerable changes in how the population evolves from start to end.

The first method investigated how prioritizing the most performant species with additional queue capacity affects the process, while the second investigated the effects of replacing the least performant species during evolution. These methods were chosen based on initial testing without any speciation extensions. The overall performance of these extensions is quite similar to runs with regular speciation. In some metrics they perform better, in others, they perform worse. As MCC is an open-ended process that can explore in all possible directions without boundaries, it would be interesting to see if the same results could be obtained by running the experiments again.

Ranking and prioritizing species force them to compete against each other indirectly, as they do not aim for an objective or high performance in any way. Instead, they evolve and drift in an open-ended way, where the overlying process dictates how they are prioritized and if they should survive. Species are not even aware of their performance, and individuals are only constrained by fulfilling the minimum criteria.

The maze domain implemented has some issues regarding the maze representation. In some cases, the process found mazes that were able to increase in size much faster than complexity. This resulted in mazes that were huge but easy to solve, as their solution path was mostly straight lines with few obstacles. This can give a false image of how complex the mazes and accompanying solution agents are, and indicates that the ”path of least resistance” discussed in Brant and Stanley [2019] is present here as well. A balance between the size and complexity in mazes is important to keep the results interesting.

Research question 2 *How is the diversity in the resulting solutions affected when ranking the species based on performance?*

Maintaining diversity in the population throughout the process in MCC is important to keep multiple lineages alive and be able to obtain many different problems and solutions in the end population. The speciation extensions seem to have a negative impact on the overall diversity, as they both have a lower average diversity metric in their runs. While some of their runs were able to reach high diversities, they were on average beaten by the base speciation. Replacing species achieved notably worse scores than the other two, and turned out to remove more diversity than it added.

6.2 Contributions

This thesis conducted experiments on how the general process in MCC is affected when the speciation method is extended to rank and prioritize based on performance. Results reveal that a higher average maze size and complexity could be achieved by using such methods. However, the overall diversity in the end populations decreased.

An important aim in open-ended processes is to produce multiple complex solutions with high diversity in the same run. The experiments in this thesis were able to achieve complexity at the cost of decreased diversity. Fewer lineages in species were maintained, as the extended speciation methods prioritized the most performant ones.

The experiments in this thesis do not reveal a way to achieve general open-endedness but show that tweaks to the speciation method in MCC can potentially increase performance.

6.3 Future Work

Analysis of Process

A more advanced analysis method is required to better understand what the results of different MCC variations mean. Based on the results from this thesis, it is still unclear how the extended speciation methods affect species on a deeper level. More insight into how all the different species lineages interact, evolve, and reacts to different speciation variants could be interesting and might reveal patterns in a species' behavior in MCC.

Further Speciation Tweaks

Extending the speciation method is not limited to the variants investigated in this thesis. There are many other directions to further investigate, for example by combining the two methods use this thesis. In this thesis the number of generations between each time the species were ranked, prioritized and replaced was static. Finding a more optimal number of generations or using a dynamic amount of generations could potentially increase performance.

Other speciation extensions, such as not being limited by maximum population size and varying the number of children generated from each species are also interesting further studies. In addition, investigation of other extensions could lead to a better understanding of how speciation affects MCC. Other perspectives might be needed to better compare speciation methods and understand how they contribute. Exploring different methods is also useful to find ways that target an increase in diversity rather than complexity growth.

Other Domains

The maze domain is great as a baseline and benchmark domain since it is performant and fast computationally. However, the domain is limited in terms of how helpful it really is to the world. An important future research is to find ways to transfer successful open-ended methods to other domains that humans can benefit from. Finding general rules for how tweaks and extensions, such as the main contributions in this thesis, affect open-ended processes is important to understand how they can be used in other domains.

Bibliography

Aguilar, W., SantamarÃa-Bonfil, G., Froese, T., and Gershenson, C. (2014). The past, present, and future of artificial life. *Frontiers in Robotics and AI*, 1:8.

Archibald, J. D. and Fastovsky, D. E. (2004). Dinosaur extinction. *The dinosauria*, 2:672–684.

Brant, J. C. and Stanley, K. O. (2017). Minimal criterion coevolution: a new approach to open-ended search. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '17*, pages 67–74, Berlin, Germany. ACM Press.

Brant, J. C. and Stanley, K. O. (2019). Benchmarking open-endedness in minimal criterion coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 72–80, New York, NY, USA. ACM.

Chatzilygeroudis, K., Vassiliades, V., and Mouret, J.-B. (2018). Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems*, 100:236 – 250.

Colas, C., Huizinga, J., Madhavan, V., and Clune, J. (2020). Scaling map-elites to deep neuroevolution. *ArXiv*, abs/2003.01825.

Eiben, A. and Smith, J. (2015). *Introduction to Evolutionary Computing*. Natural Computing Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2nd ed. 2015 edition.

Fauske, K. M. (2006). *Example: Neural Network*. `http://www.texample.net/tikz/examples/neural-network/` [Accessed: 2019-12-10].

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press.

Gabor, T., Sedlmeier, A., Kiermeier, M., Phan, T., Henrich, M., Pichlmair, M., Kempter, B., Klein, C., Sauer, H., AG, R. S., and Wieghardt, J. (2019). Scenario Co-evolution for Reinforcement Learning on a Grid World Smart Factory Domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 898–906, New York, NY, USA. ACM. event-place: Prague, Czech Republic.

Gomes, J., Mariano, P., and Christensen, A. L. (2017). Novelty-driven cooperative coevolution. *Evolutionary Computation*, 25(2):275–307. PMID: 26652102.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lehman, J. and Stanley, K. (2011a). Evolving a diversity of creatures through novelty search and local competition. In *Evolving a diversity of creatures through novelty search and local competition*, pages 211–218.

Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*.

Lehman, J. and Stanley, K. O. (2010). Revising the evolutionary computation abstraction: Minimal criteria novelty search. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 103–110, New York, NY, USA. ACM.

Lehman, J. and Stanley, K. O. (2011b). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223. PMID: 20868264.

m0nhawk (2013). *TikZ: Diagram of a perceptron*. `https://tex.stackexchange.com/questions/104334/tikz-diagram-of-a-perceptron` [Accessed: 2019-12-10].

Mattiussi, C. and Floreano, D. (2003). Viability evolution: Elimination and extinction in evolutionary computation. In *Viability Evolution: Elimination and Extinction in Evolutionary Computation*.

Methenitis, G., Hennes, D., Izzo, D., and Visser, A. (2015). Novelty search for soft robotic space exploration. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO â15, page 193â200, New York, NY, USA. Association for Computing Machinery.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Ray, T. S. (1993). An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artif. Life*, 1(1-2):179–209.

Rechenberg, I. (1973). Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution.

Rumelhart, E., D., Mcclelland, J., and L., J. (1986). *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations.*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Solheim, T. (2019). Open-ended coevolution. Pre-project to master thesis, available from the author.

Soros, L. (2018). Necessary Conditions for Open-Ended Evolution. *Electronic Theses and Dissertations.*

Soros, L. and Stanley, K. (2014). Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. *The 2019 Conference on Artificial Life*, pages 793–800.

Stanley, K. O. and Lehman, J. (2015). *Why Greatness Cannot Be Planned: The Myth of the Objective.* Springer.

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

Sun, Y., Zhang, L., and Gu, X. (2012). A hybrid co-evolutionary cultural algorithm based on particle swarm optimization for solving global optimization problems. *Neurocomputing*, 98:76–89.

Urbano, P., Naredo, E., and Trujillo, L. (2014). Generalization in maze navigation using grammatical evolution and novelty search. In *International Conference on Theory and Practice of Natural Computing*, pages 35–46. Springer.

Velez, R. and Clune, J. (2014). Novelty search creates robots with general skills for exploration. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO â14, page 737â744, New York, NY, USA. Association for Computing Machinery.

Von Neumann, J. et al. (1951). The general and logical theory of automata. *1951*, pages 1–41.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. *arXiv:1901.01753 [cs]*. arXiv: 1901.01753.