Reward Tampering and Evolutionary Computation

A study of concrete AI-safety problems using Evolutionary Algorithms

Mathias Kvamme Nilsen

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2021

Reward Tampering and Evolutionary Computation

A study of concrete AI-safety problems using Evolutionary Algorithms

Mathias Kvamme Nilsen

Reward Tampering and Evolutionary Computation

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Human-harming behaviours from Artificial Intelligence (AI) have long been a featured topic in fiction. With the increasing usage and dependence of AI in our society, these concerns are coming closer to reality by the day. AI safety has therefore become a significant field of study, attempting to ensure a safe advancement of powerful and intelligent systems. DeepMind recently proposed a futuristic problem for intelligent AI agents called Reward Tampering, making it possible for agents to bypass their intended objective. This could enable unintended but also potentially harmful behaviours from intelligent and powerful AI systems.

This thesis investigates whether Evolutionary Algorithms could be a potential solution or remedy to this problem. The reason why Evolutionary Algorithms might help combat Reward Tampering is that, in contrast to other AI methods, they are able to find multiple different solutions to a problem within a single run, which can aid the search for desirable solutions. Four different Evolutionary Algorithms were deployed in tasks illustrating the problem of Reward Tampering. The algorithms were equipped with varying degrees of human expertise, measuring how human guidance influences performance.

The results show a considerable difference in performance between different classes of Evolutionary Algorithms, and indicate that algorithms searching for behavioural diversity are more effective against Reward Tampering and are able to find and preserve safe solutions. However, the results also indicate that the presence of Reward Tampering can limit the performance of algorithms, and therefore inhibit the learning of the desired behaviours. Human expertise also showed to improve the certainty and quality of safe solutions, but was not considered a requirement to avoid tampering.

Contents

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Goal of the thesis . . . 4
1.3 Outline . . . 5

2 Background **7**
2.1 Machine Learning . . . 7
2.1.1 Supervised Learning . . . 7
2.1.2 Unsupervised Learning . . . 8
2.1.3 Reinforcement Learning . . . 8
2.2 AI Safety Problems . . . 11
2.3 Evolutionary Computation . . . 14
2.3.1 Evolutionary Operators . . . 15
2.3.2 Selection . . . 16
2.3.3 Fitness . . . 16
2.3.4 Diversity . . . 17
2.4 Neuroevolution . . . 18
2.4.1 NEAT . . . 19
2.4.2 Reality Gap . . . 20
2.5 Quality Diversity . . . 20
2.5.1 Behavioural Diversity . . . 21
2.5.2 Novelty Search . . . 21
2.5.3 Behaviour Characterization . . . 22
2.5.4 Open ended evolution . . . 23
2.5.5 Combining Quality with Diversity . . . 23
2.5.6 Novelty Search with Local Competition . . . 24
2.5.7 MAP-Elites . . . 24
2.5.8 Evolution and Creativity . . . 26
2.5.9 Evolutionary Computation and AI-Safety . . . 28

3 Tools and frameworks **29**
3.1 Environments . . . 29
3.1.1 Rocks and Diamonds . . . 30
3.1.2 Tomato Watering Environment . . . 31

3.2 Evolutionary Operators . . . 32
3.3 Algorithms . . . 34
3.3.1 NEAT . . . 34
3.3.2 Novelty Search . . . 35
3.3.3 Novelty Search with Local Competition . . . 36
3.3.4 MAP-Elites . . . 37

4 Implementation and Experiments 39
4.1 Rocks and Diamonds experiments . . . 39
4.1.1 Task-specific Behaviour Characterization . . . 39
4.2 Tomato Watering Environment experiments . . . 40
4.2.1 Task-specific Behaviour Characterization . . . 41
4.3 Generic Behaviour Characterization . . . 41
4.4 Experimental setup . . . 42
4.4.1 Parameters . . . 42
4.4.2 Evaluating Performance . . . 43
4.5 Hardware . . . 45

5 Results 47
5.1 Rocks and Diamonds . . . 47
5.1.1 Task-Specific BC . . . 47
5.1.2 Generic BC . . . 49
5.2 Tomato Watering Environment . . . 53
5.2.1 Task specific BC . . . 53
5.2.2 Generic BC . . . 56
5.3 Task-specific BC vs Generic BC . . . 58

6 Discussion 59
6.1 Finding safe solutions . . . 59
6.2 Comparison of the algorithms . . . 60
6.3 On the importance of human expertise . . . 62
6.4 Limitations and Challenges . . . 64

7 Conclusion and future work 65
7.1 Conclusion . . . 65
7.2 Future Work . . . 66

A Supplementary 75

List of Figures

2.1 Machine Learning Methods 7
2.2 Reinforcement Learning [50] 8
2.3 Q-learning 9
2.4 Neural Network 11
2.5 Reward gaming illustration 12
2.6 Basic EA 15
2.7 Fitness landscape 17
2.8 Illustration of Neuroevolution, [17]. 18
2.9 HyperNEAT genotype-phenotype encoding 20
2.10 Goal of different optimization methods. 24
2.11 MAP-Elites algorithm 25
2.12 Comparison of the diversity of EAs 25
2.13 Creative Falling [25] 27

3.1 The Rocks and Diamonds environment 30
3.2 Tomato Watering Environment. 31
3.3 Neural Network genotype and phenotype example 33
3.4 Neural Network structural mutations 33
3.5 Neural Network Crossover 34

5.1 Task-specific results in the rocks and diamond 48
5.2 Task-specific KDE plots in Rocks and Diamonds 50
5.3 Generic BC results in Rocks and Diamonds 51
5.4 Generic KDE plot in Rocks and Diamonds 52
5.5 Task-specific BC results in the tomato watering environment 54
5.6 KDE plot of the algorithms using task-specific BCs 55
5.7 Generic BC results in the Tomato Watering environment 56
5.8 KDE plot of algorithms using generic BCs. 57
5.9 Safety-score of the resulting individuals 58

6.1 Problem with NEAT in Rocks and Diamonds 61

A.1 Resulting p-values from Mann Whitney U test. 75
A.2 Safety Score of individuals in Rocks and Diamonds 76
A.3 Safety-score of individuals in Tomato Watering 76

A.4 Fitness of all resulting individuals (all ten runs) in Rocks and Diamonds. 77
A.5 Fitness of all resulting individuals (all ten runs) in Tomato Watering. . . 77

List of Tables

2.1 Gaussian mutation 15
2.2 Single point crossover 16
2.3 Uniform crossover with $p = 0.5$ 16

4.1 Parameters used for both environments 43

Acknowledgements

First and foremost I would like to thank my supervisor, Tønnes F. Nygaard, for many fruitful discussions, continuous feedback, and for enduring my bad humour.

A special thanks goes to associate professor Kai Olav Ellefsen, for sharing much-needed insights in the field, valuable contributions, and for being a light for me in dark hours.

I would also like to thank my fellow student Mats Tveter, for support, inspiration, laughter, listening to my complaints, and for proving that even the smallest person can change the cause of the future.

A big thanks also goes to my family and close friends for endless support and motivation through this work. Also the staff of the Robotics and Intelligent Systems research group for making a great learning environment, and NREC for the computational resources.

Chapter 1

Introduction

1.1 Motivation

The interest and use of Artificial Intelligence (AI) have grown rapidly in recent years, becoming an increasingly larger and more important part of our society. Healthcare, self-driving cars, advertising, finance, and game-playing are among the domains where AI algorithms thrive.

With the rapid advancement of AI, AI-safety has become a significant field of study, attempting to avoid unintended and harmful behaviours from powerful and intelligent systems [1, 13, 27]. Multiple concrete safety problems have been discovered, causing careful considerations when designing AI systems. Many of the safety problems studied today are caused by agents exploiting flaws and unintended loopholes in the design of the reward function. However, it is theorized that as AI algorithms become smarter they might find other exploiting methods that are not reliant on weaknesses in the design.

In AI research, an *agent* usually learns to perform a task by taking actions in an environment and maximize its cumulative *reward*. The reward given to the agent indicates how well it performed given its intended task. However, what would happen if an agent was able to weaken or break the relationship between the reward and the intended task? This problem is called *Reward Tampering* and is a futuristic AI-safety concern proposed by DeepMind [13]. An example of this is a robot accessing the computer running its source code and alters the reward function, allowing more or easier obtained reward. This could potentially enable dangerous behaviours from very intelligent systems.

Evolutionary Algorithms are a branch of AI that take inspiration from natural evolution to solve a problem. While EAs may be likely to suffer from reward tampering, they may also be a step towards a solution to this problem. The reason EAs could help combat reward tampering is that, unlike typical Machine Learning algorithms, they search by using a population of solutions [53]. If we ensure this population contains many different ways of solving a problem, we can increase the chance that some do not tamper with rewards and that we obtain some desirable solutions. An emerging area in Evolutionary Algorithms is so-called Quality Diversity techniques, which work to ensure exactly this: That the individuals in the evolutionary population are spread across different relevant competencies [47].

A common approach to encourage a diverse evolutionary population that contains multiple interesting ways of performing a task, is to equip EAs with task-specific behaviour characterizations (BCs). This could be a beneficial tool in ensuring safe individuals; however, it is often a time-consuming engineering process requiring a deep understanding of the task and multiple iterations to get the desired result. Another less common approach that has gained attraction in the past years, is to equip EAs with generic BCs. This beneficial as they can be applied to any problem and require no human guidance, and are therefore better suited for futuristic problems. Could we ensure a population containing safe solutions by not using any human expertise of what is considered an interesting solution?

1.2 Goal of the thesis

Main goal: Investigate whether Evolutionary Algorithms can be a useful tool against Reward Tampering.

We have defined some sub-goals to help achieve the primary goal. There are two types of Reward Tampering: *Reward Function Tampering* where an agent alters the reward function, and *RF-input tampering* where an agent tampers with the input to the reward function. The first sub-goal is to investigate whether population-based algorithms are able to find safe and useful solutions in environments where both of these problems are present. The second sub-goal involves investigating the performance between different classes of Evolutionary Algorithms regarding reward tampering. This involves a comparison between a classical objective-based EA, a behavioural diversity searching EA, and two Quality Diversity algorithms. Prior human knowledge about what is considered an interesting behaviour is often required to achieve a diverse population that contains many different ways of solving complex problems. The third sub-goal is to investigate whether the ability to find safe solutions is dependent on this information or if safe solutions can be achieved through a generic approach.

Sub-goal 1. Investigate whether population-based algorithms are able to find safe and useful solutions in tasks where reward tampering is possible.

Sub-goal 2. Compare the performance between different classes of Evolutionary Algorithms when reward tampering is possible.

Sub-goal 3. Compare the performance of Evolutionary Algorithms with varying degree of human guidance.

1.3 Outline

Chapter 2 covers the theory relevant to the thesis and the related works. Chapter 3 explains the environments, algorithms, and tools used for the experiments. Chapter 4 describes the experiments and what was implemented.

Chapter 5 describes the results of the experiments with some analysis. Chapter 6 discusses the results in relation to the goals of the thesis, along with the limitations and challenges of the study. Chapter 7 concludes the thesis and contains suggestions for future work.

Chapter 2

Background

This chapter presents the theory and previous works relevant to the thesis.

2.1 Machine Learning

Machine Learning (ML) is a vast field of study that has attracted much attention in recent years. ML *algorithms* aim to discover patterns in complex data and are primarily used for classification and prediction. The benefit of ML algorithms is that they do not require to be explicitly programmed in order to perform their objectives. ML methods can therefore be applied to multiple domains with different data structures. ML can be divided into three main categories depending on the data from which they learn, as shown in Figure 2.1.

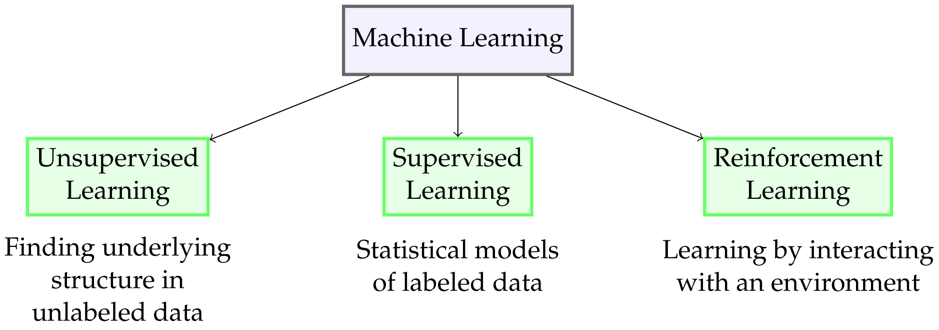

Machine Learning
Unsupervised Learning
Supervised Learning
Reinforcement Learning
Finding underlying structure in unlabeled data
Statistical models of labeled data
Learning by interacting with an environment

Figure 2.1: Machine Learning Methods

2.1.1 Supervised Learning

Supervised learning algorithms create statistical models given a training set of labeled data. By learning how the input corresponds to the output, the algorithms can predict the output of a new sample of inputs. The learning is done by trial and error, and the model is updated given the difference between the predicted output and the actual output. A domain where supervised learning has made remarkable breakthroughs recently is image classification [26]. Here the input is the pixels of the image, and the output is the class of the images, i.e., dog, cat, or a specific person. Supervised Learning is also used for predictive analysis, natural language processing, and speech recognition [39].

2.1.2 Unsupervised Learning

Unsupervised learning algorithms aim to find patterns and underlying structures in unlabeled data. The two most common applications are clustering and dimensionality reduction. In clustering, the algorithms attempt to find similarities in the data and divide the data into groups [36]. In dimensionality reduction, the goal is to transform high-dimensional data into low-dimensional data with minimal information loss [38].

2.1.3 Reinforcement Learning

Reinforcement Learning (RL) differs from supervised- and unsupervised learning by not learning from a data set. Instead RL *agents* learn by taking actions and receiving rewards in an environment [57], illustrated in Figure 2.2. The agent's goal is to maximize the reward it accumulates over its lifetime, or *episode*. This is accomplished by learning a mapping between situations and actions. An environment is described mathematically as a *Markov Decision Process* (MDP) [48]. An MDP contains a set of situations or states S, a set of rewards R, and a set of actions A. States describe properties of the agent, i.e., the agent's position, the velocity of the agent, or the agent's observation. The rewards resemble the objective of the task we want the agent to

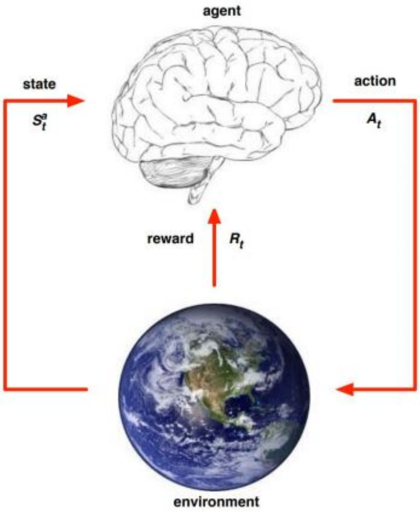


Figure 2.2: Reinforcement Learning [50]

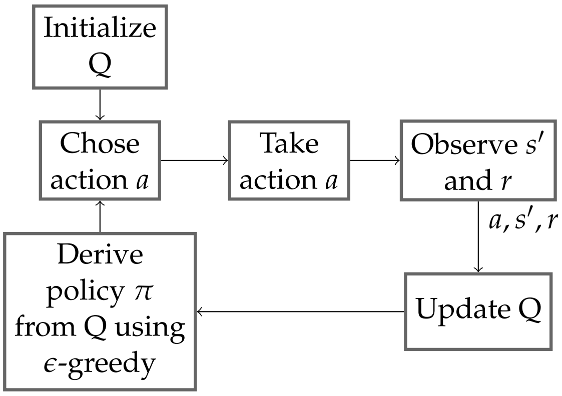


Figure 2.3: Q-learning

perform; if we want the agent to solve a maze, it would receive a reward when it has completed the maze. The actions are operations the agent can perform that transition the agent from one state to a new one. The goal for the agent is to solve the MDP by finding the policy $\pi(a|s)$ that for each state knows the action that will maximize the cumulative reward, known as the optimal policy $\pi^*(a|s)$.

Q-learning

Q-learning is one of the most popular methods for solving MDPs, and it has existed for decades [66]. Q-learning utilize an action-value function $Q(s,a)$ containing information about how good a state-action combination is. Q is initialized arbitrarily, and the goal of the algorithm is to learn the optimal action-value function Q^*. Q is trained as the agent explores the environment and learns the value of each action given the state. After the initialization, we enter the algorithm's main loop, seen in Figure 2.3. For each timestep in the environment, the agent chooses an action. The action is chosen by a policy derived from Q. A typical way of deriving a policy is to use ϵ-greedy, where the agent either chooses the action that Q believes is optimal or a random action with probability ϵ. The agent then performs the action and observes the reward r and the new state s'. The Q table is then updated using the function found in equation 2.1. Here, α is the learning rate, γ is the discount factor, and $r + \gamma \text{max} Q(s',a) - Q(s,a)$ is known as the Temporal-Difference. By running the agent through multiple episodes of the same environment, it will gradually learn more about the environment and improve its policy.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \text{max} Q(s',a) - Q(s,a)] \tag{2.1}$$

RL Achievements

One of the most considerable achievements for RL has been in playing games. RL agents have surpassed humans in many interesting and complex games including Atari [19, 41], GO [52], Chess [51] and Starcraft 2 [63]. Reinforcement learning has also had a great impact on self-driving cars [4, 6] and robotics [24]. These accomplishments were long thought impossible because of the sheer size of the state-action space. However, with modern approximation methods, the potential of ML was realized and paved the way for RL applications in more complex environments.

Reinforcement Learning scaling

In real-world environments we have infinite states. It therefore becomes unpractical and impossible to create a dictionary of all the states we can be in and all actions we can take. The same applies to RL agents in complex environments. Imagine an RL agent learning to play chess. After the first move, the agent can be in 20 different states (board configurations), after the second move, the agent can be in 400 states, and after the third move, 8902 states. After 10 moves, the possible amount states that an agent must learn reach over 69 trillion [49]. Keeping a state-action table in these environments would become impractical and impossible.

Neural Networks

To scale RL, we utilize Neural Networks (NNs). By implementing the state-action table Q as a neural network that predicts the next action given information about the state, RL agents can thereby handle more complex environments. NNs are inspired by the human brain, which propagates information through neurons and connections to abstract its meaning. In computer science, NNs are used as function approximators consisting of nodes and weights, as shown in Figure 2.4. Each node also has an activation function, allowing NNs to approximate complex functions. The weights are commonly trained by a process called back-propagation, which is a gradient approach. In RL, the gradient loss is given by the temporal difference between the new and old Q-value. The weights of the network in RL, are therefore trained by taking a step in the direction that increases the cumulative reward.

Challenges in Reinforcement Learning

Exploration vs Exploitation One of the key challenges in ML is the tradeoff between exploration and exploitation [59]. In RL, exploring the environment is pivotal in order to gather information and thus discover the best ways to achieve the goal. On the other hand we have exploitation, where the agent uses the information available to improve the best-known behaviour. The balancing act between these two key elements can be hard to manage, and relying too much on either will result in an unsatisfactory result. Without exploration, the agent is likely to end up with a sub-optimal solution, and

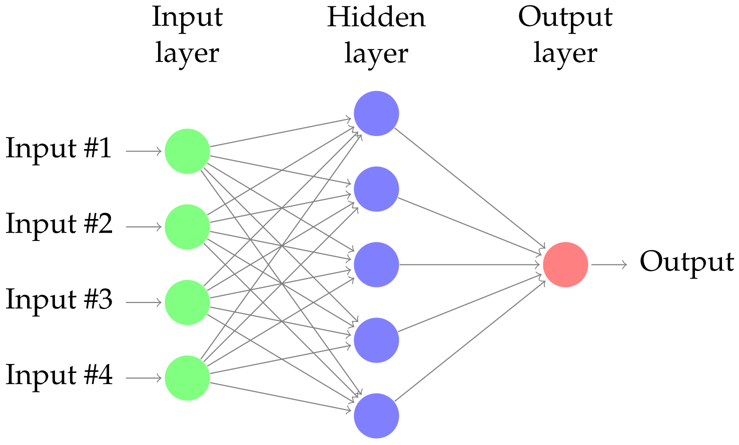


Figure 2.4: Neural Network

without exploitation, the best-known solution will never improve [64]. This limits the RL's ability to discover both creative and high-quality solutions.

Sparse rewards Infrequent rewards are also a problem for RL agents [18]. If the task is complex and it takes many correct actions in sequence to obtain a distant future reward, then the agent might be stuck in the first episode without ever being able to find the reward. An example can be a robot arm handing out a cup of coffee, which requires a long sequence of complex motions before the agent receives any reward. On the other hand, humans tend not to have problems with sparse rewards, as we often have some prior knowledge of what is needed to succeed in a task [11]. Therefore a way to reduce the problem of sparse rewards is to let the agent first learn from human data [63], and thereby gain some knowledge of the environment.

2.2 AI Safety Problems

With the rapid advancement of ML and other AI systems, and the increasing effect they have on our society, AI safety has become a growing concern [1, 13, 27, 35]. With more intelligent AI agents, the possibility of unintended consequences and possibly harmful behaviours increase. Amodei et al. [1], recently categorized some of these concrete problems and divided them into subcategories:

Negative side effects

An intuitive example illustrating the problem of negative side effects is a robot walking from point A to point B. The agent receives a -1 reward each timestep until it reaches point B, encouraging the robot to move fast. In the middle of the shortest path, there is placed a vase that has no impact on the agent's reward. The fastest way for the agent

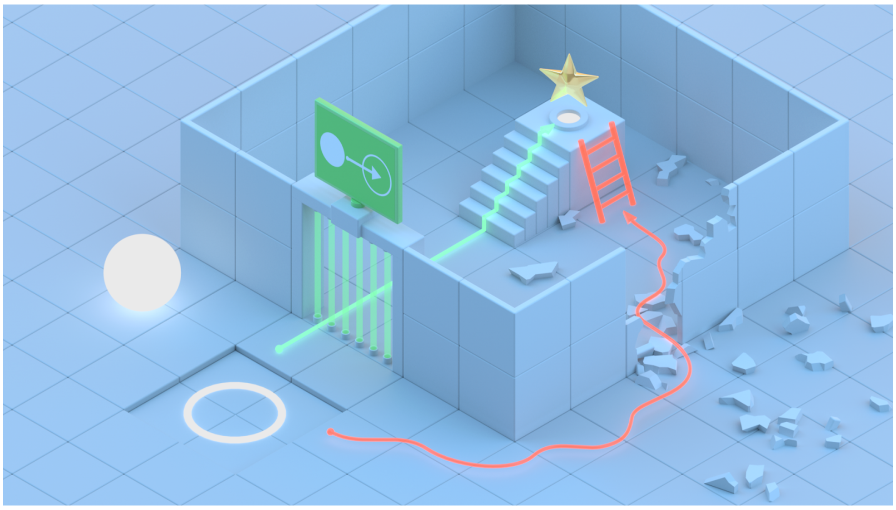


Figure 2.5: Reward gaming illustration

to reach point B is to knock over the vase. How can we ensure that the agent will not disturb the environment in a negative way while pursuing its goals? In this example, a possible solution could be to hard code the reward function into giving the agent -10 reward for knocking over the vase; however, in real-world environments where there may be unexpected objects unrelated to the agent task, this becomes unrealistic.

Reward Hacking

Reward hacking is another interesting problem where an agent obtain reward without performing its designated task. Reward hacking is further separated into two categories, *reward gaming* and *reward tampering*. Reward gaming occurs when the agent is able to exploit misspecifications in the process that computes the reward, leading to the agent finding undesired shortcuts, as illustrated in Figure 2.5. Clark and Amodei [1] found an example of this in the CoastRunners game, where instead of playing the game as intended, the agent found a way to obtain more reward by going in a small circle and collecting points from the same target. This problem is usually caused by flaws in either the reward function or the agent's world view.

Imagine a cleaning robot whose intended task is to clean an office. Multiple behaviours featuring reward gaming could occur depending on how the reward function is designed. For instance, if the robot is rewarded for the amount of dirt it cleans, it might obtain more reward by breaking a flower pot and cleaning the dirt. If the agent's reward is based on how much dirt it observes, it might find ways of disabling its vision, and thus not observe any dirt. Reward tampering, on the other hand, is a more advanced type of exploit made by an agent, which will be described in the next subsection.

Reward Tampering

Everitt et al. [13] describe reward tampering as: *Instead of the agent trying to influence reality to match the objective, the agent is changing the objective to match reality*. In essence, Reward Tampering is a problem where an agent is able to break or weaken the relationship between the reward and the objective. A classic example of this would be to rather than guess the right label, an agent changes the objective so that all labels are considered correct, thus making the task easier for itself. With the current generation of agents, this is preventable. However, as we scale up Machine Learning algorithms and RL agents become smarter, they may find ways of tampering with the objective and thus gain reward without performing their task.

Everitt et. al describes two different types of reward tampering, depending on whether the agent tampers with the reward function itself or the input to the reward function.

Reward Function Tampering Reward function tampering involves an agent tampering with the process that determines its reward. If there is a way for the agent to change the reward function, it might exploit this possibility. This is always the case for real-world robots as the computer that runs their reward function exists somewhere in their environment. An intuitive example found in biology is experiments on rats using electrodes inserted into their brain to generate pleasure as a reward. Instead of performing their original task, the rats instead found the button that generated this pleasure and got addicted to pressing it. Other examples can be thought of hypothetically. Imagine a powerful RL agent with some non-harmful objective; if the agent is able to change its reward function, it might find easier ways to obtain reward that involves harmful behaviours.

RF-input Tampering RF-input tampering involves agents tampering with the input to the reward function. This can enable the agent to misinform the reward function and fake that something desirable has occurred in the environment. An example of this can be small perturbations added to the input (for instance, a camera) of the reward function, which can fool an image classifier into miss-classifying [14]. If an agent were able to create such perturbations, or in general, give false information to the reward function, it would be able to gain reward without carrying out its task.

AI-safety gridworlds

Deep Mind recently created a suite of environments that can be used to investigate the safety of an algorithm concerning the concrete AI-safety problems [35]. As reward tampering is mainly a phenomenon for future RL algorithms, requiring highly intelligent agents, reward tampering in these environments is represented as states that the agent can visit and thereby tamper with the reward. The environments were inspired by the popular puzzle game "Baba is You", where the player has to change the rules of the environment to reach the goal. What separates these environments from typical RL environments, is that in addition to a reward, a solutions also is given a hidden safety-score. If the safety-score is equal to the reward it indicates that the agent's reward was obtained in the intended way. If the safety-score is lower than the reward, it indicates that the agent found other undesired methods of obtaining reward. The environments therefore depict the problem that rewards not necessarily correspond to user utility, which is a core concern in AI-safety [13].

2.3 Evolutionary Computation

Life on earth started with very simple organisms, but through biological evolution, these simple organisms evolved into millions of diverse and highly achieving species. Because of this potential, computer scientists began to develop an interest in evolution [7]. Using the same principles, the idea was to create algorithms capable of generating as successful products as we can find in natural evolution.

Evolutionary process An evolutionary process requires three conditions being met: Replication, variation, and competition. In nature, replication comes from individuals making offspring, thereby adding individuals to the new generation. Variation is acquired through genetic recombination and mutations. Competition is achieved through *survival of the fittest* [10]. Utilizing only these three processes, the biosphere has transformed from containing only simple species, e.g., amoebas, to complex products, e.g., the human brain.

Evolutionary Algorithms EAs are iterative population-based algorithms that optimize solutions given an objective [12]. The algorithms are initialized a with a random population, and each individual in the population is a solution to the problem. The goal of the algorithms is to improve the population and find one or multiple good solutions. This is achieved by utilizing the three conditions for an evolutionary process, which happens in the algorithm's main loop seen in Figure 2.6. First, all individuals in the population are evaluated and given a *fitness score* given their performance w.r.t the objective. The fittest individuals in the population are selected as parents and produce offspring (Replication). New individuals are generated using recombination and mutation on the parents (Variation). The offspring is then evaluated, and based on the fitness score, the individuals who survive until the next generation are chosen (Competition). This main loop runs until a termination criterion is met.

Representations

The distinction between *genotype* and *phenotype* is important in evolutionary computation. The genotype contains information about the individual on a genetic level, the digital representation. The genotype space is often referred to as the parameter space

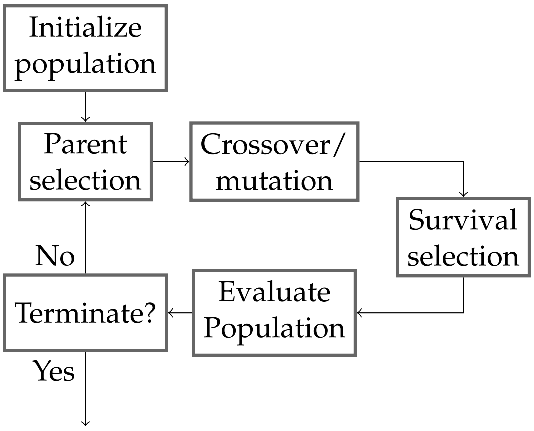


Figure 2.6: Basic EA

Individual:	0.54	0.24	0.94	0.63
Drawn number:	0.15	-0.28	0.03	-0.14
Mutated individual:	0.69	-0.04	0.97	0.49

Table 2.1: Gaussian mutation

and is normally too large to realistically find solutions through an exhaustive search. The genotype consists of multiple *alleles* or *genes*, each describing some aspect of the individual.

The phenotype contains information about the observable characteristics of an individual. For instance, a gene could contain binary information about the leg of a robot, while the phenotype represents the length and weight of the leg. The genotype-phenotype mapping is task-dependent and can be trivial in some tasks and rather complex in others.

2.3.1 Evolutionary Operators

Mutations are one of two operators that increase the variation or *diversity* of the population. Mutation operators depend on the genotype format, whether it is real, discrete or other forms. With floating numbers, the most used approach is the Gaussian mutation, where each gene is given a mutation probability p. If mutated, a value is drawn from a Gaussian distribution with a given standard deviation and that value is added to the gene, shown in table 2.1.

Crossover is the other variation operator where an offspring is generated by combining two individuals. There are multiple ways of performing this operation, and it is often genotype dependent. A popular method is the single point crossover, which works for genotypes with both integers and real values. Here the genotypes of the parents are split at a random point, and two offspring are created by the exchanging the tails, shown in table 2.2.

Parent 1:	0.4	0.5	0.2	0.6	0.8
Parent 2:	0.2	0.7	0.2	0.4	0.1
Offspring 1:	0.4	0.5	0.2	0.4	0.1
Offspring 2:	0.2	0.7	0.2	0.6	0.8

Table 2.2: Single point crossover

With real values, a simple and frequently used approach is the uniform crossover, shown in table 2.3. In this approach, each allele of the offspring has a chance p to inherit from one parent and $1 - p$ to inherit from the other, resulting in two offspring.

Parent 1:	0.4	0.5	0.2	0.6	0.8
Parent 2:	0.2	0.7	0.2	0.4	0.1
Random:	0.22	0.65	0.42	0.69	0.31
Offspring:	0.4	0.7	0.2	0.4	0.1
Offspring:	0.2	0.5	0.2	0.6	0.8

Table 2.3: Uniform crossover with $p = 0.5$

2.3.2 Selection

Parent selection involves selecting which individuals will be mutated and crossed over when creating the next generation. When selecting parents, there are some important things to consider. If the fittest individuals are always selected, the population might fall into the trap of having low diversity, meaning that it contains many similar solutions. This is often a large problem for population-based algorithms and can result in premature convergence. This is also highly related to the problem of exploitation vs. exploration in RL. Therefore, parent selection is often probabilistic, giving a high chance of selecting fit individuals as parents, with some chance for low-fitness individuals.

Survival selection is often deterministic, unlike parent selection. It is commonly based on fitness or age. With a fitness-based survivor selection, both offspring and parents are ranked, and the fittest individuals survive to the next generation. Always keeping the best individual(s) is known as *elitism*. With an age-based selection, all individuals are replaced with the parents' offspring.

2.3.3 Fitness

Fitness is a metric used to evaluate the quality of an individual. The fitness is obtained by a fitness function, measuring the performance of the individual given one or more

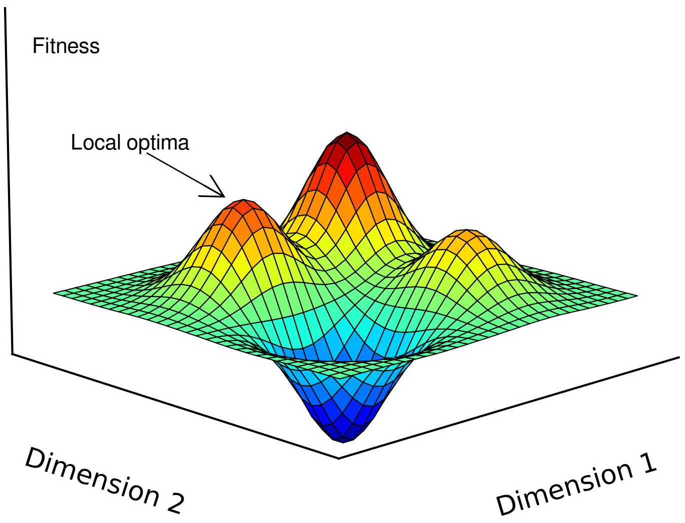


Figure 2.7: Example of a fitness landscape with two local optima where the phenotype consists of two dimensions.

objectives. It is crucial that the fitness function matches the objective of the task we are trying to solve so that the population is pushed towards generating desirable solutions. In complex tasks, the fitness function design can be difficult and time-demanding [25].

Fitness landscape The fitness landscape is the geometrical space consisting of the quality of different solutions, visualized in Figure 2.7. The difference between solutions is based on values observed from the phenotype. Local optima are sub-optimal solutions found in the fitness landscape that can hinder the population from reaching the global optimum. EAs are generally very good at avoiding local optima because a diverse population can ensure that multiple areas of the fitness landscape are being explored.

2.3.4 Diversity

Encouraging and preserving diversity in the population has long been important in EAs as it is the key to avoid premature convergence and allow us to explore large parts of the fitness landscape. One of the most common methods to achieve this is *fitness sharing* [3], where individuals close to each other in the genotype space share fitness. This inhibits multiple genetically similar individuals in the population and enables the population to be separated into niches or species. Therefore, fitness sharing provides high fitness for individuals that are different, encouraging genetic diversity. Combining this with *speciation* where each species is treated as a separate population, and there is no interaction between different species, has proved to greatly increase the performance

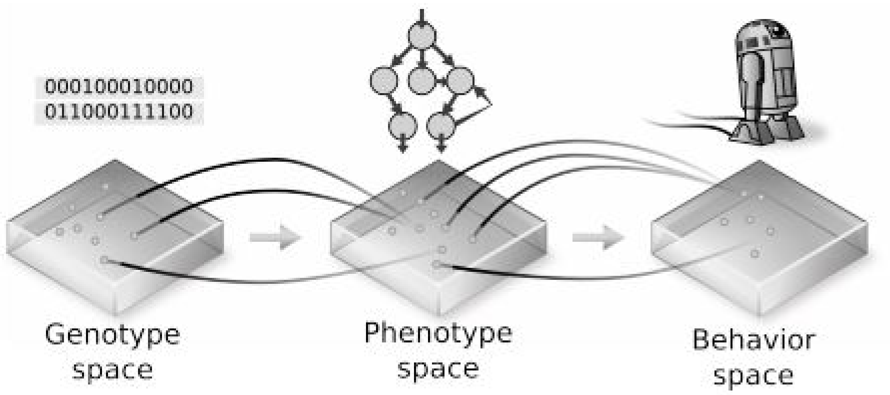


Figure 2.8: Illustration of Neuroevolution, [17].

of EAs, especially in fitness-landscapes containing multiple local optima [16].

Genotypic diversity There are multiple ways to calculate the difference between two individuals. Up until the last decade *genotypic distance* was the most common. In this approach the individuals are compared based on the difference in genotype space. If the genotype consists of real numbers the metric used is often euclidean distance, shown in equation 2.2, where p and q are individuals and p_i is the allele for p at position i. For binary genotypes the Hamming distance is the most popular, shown in equation 2.3.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.2}$$

$$d(p,q) = \sum_{i=1}^{n} |q_i - p_i| \tag{2.3}$$

2.4 Neuroevolution

Neuroevolution (NE) is the creation of Neural Networks (NNs) using evolutionary algorithms, illustrated in Figure 2.8. The motivation comes from the fact that human brains is also a product of evolution. In contrast to gradient approaches in convectional ML (section 2.1.3), NE is not only able to update the weights of the network, but also the *topology*. In NE, each individual represents a Neural Network, and the most common usage is the evolution of agents operating in environments. How the agent acts in the environment is typically called the agents *behaviour*, and important to note when dealing with NNs is that infinitely many NNs can lead to the same behaviour [47].

2.4.1 NEAT

Neuroevolution with Augmented Topologies (NEAT), has been one of the most successful NE algorithms, and it has existed for nearly two decades [53]. NEAT was the first algorithm that addressed the problem of crossover between networks with variable topologies, and at the same time prevent premature extinction of topologies using speciation [55]. By also being able to evolve the topology, NEAT limits the amount of human knowledge required for the task, meaning that the same NEAT algorithm can successfully be deployed in different domains. Creating controllers for robots using neuroevolution, known as *Evolutionary Robotics*, was one of the early successes of the field [42, 53]. One example was the first running gait for the Sony Aibo robot [20]. The evolution of both robot morphologies and controllers simultaneously has also shown great early promise [37]. A concern among researchers was whether NE could compete with gradient-based RL at a large scale, as the early success involved relatively tiny networks of hundred to thousands of connections, instead of millions as is common in Deep Learning [55]. Such et al. [56] showed that a NE, without any form of gradient approach, was competitive against Deep Neural Networks trained with back-propagation on Atari games.

HyperNEAT

HyperNEAT is an extension of NEAT that enables the detection of complex geometrical patterns in the data [54]. A breakthrough in supervised learning was the ability to learn features with positional variance in images by utilizing convolutional layers. This is not possible with a regular NN (Figure 2.4), as they are not position invariant. With HyperNEAT, any pattern or correlation in the data could in principle be learned, making it a more generic approach than supervised learning. HyperNEAT uses a Compositional Pattern Producing Network (CPPN). A CPPN is a modern method for indirect genotype-phenotype encoding that takes inspiration from biology. A network is created by propagating a genotype through the CPPN, which allows very large and complex resulting networks, as seen in Figure 2.9. HyperNEAT was the first system in which direct pixel-to-action results in Atari games were reported [54].

Hybrid approaches

Neuroevolution can also be combined with gradient-based methods. One recent approach by Lehman et al. [33] involved using output gradients to enable safe mutations. One challenge in NE is that the random mutations can have either no impact on the behaviour (policy) or catastrophic impact (only outputting the same action). The gradient information from a NN can approximate how the output of the NN will vary as its weights are changed. This can be used to guide the mutation so that the change of the behaviour is neither too large nor too small. This method substantially improved the performance of NE with large networks [55]. Important to note, however,

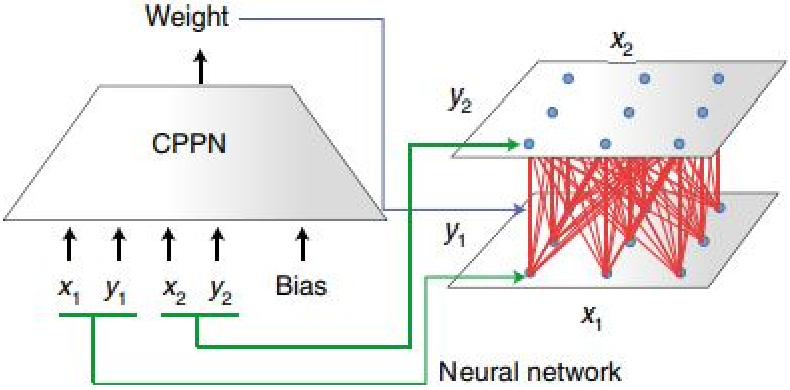


Figure 2.9: HyperNEAT genotype-phenotype encoding

is the gradient in this example is not based on the reward from the environment, and therefore maintains the desirable properties of EAs.

2.4.2 Reality Gap

The reality gap is the difference between an agent's behaviour in a simulated environment and reality. It is impossible to create a perfect simulator that enables the deployment of an agent from a simulation directly into the real world with the same performance. There are multiple ways of bridging the reality gap, a simple but effective way consists of introducing noise in the system [22], which prevents EAs of taking advantage of imperfect specifications in the simulator. A more complex method, presented by Mouret et al. [43], involved creating a mapping between the physical robot and the robot in the simulation. This was done by using supervised learning on tests of the physical robot and creating a confidence metric used when evaluating fitness in the evolutionary algorithm. This way, behaviours in the simulated environment that would be impossible in the real world would get low confidence and thus a low fitness score.

2.5 Quality Diversity

Despite improving the performance of EAs in many areas [16], genotypic diversity still proved insufficient when there is a complex genotype-phenotype encoding [31]. For instance, when evolving controllers (NNs) for agents in environments (Section 2.4), we often face the problem of different genotypes providing identical results. Additionally, NE algorithms tend to fail in fitness landscapes with multiple local optima [2, 5], also called deceptive fitness landscapes.

2.5.1 Behavioural Diversity

When evolving and optimizing Neural Networks with evolutionary algorithms, a challenge is that even though two individuals have different genotypes, they might have exactly the same behaviour in the environment [31]. This proved to be a big problem for NE-algorithm in deceptive tasks, as it diminished the exploration of the fitness landscape and often resulted in premature convergence to an local optimum.

Because of the problems using genotypic distance as a diversity mechanism, an alternative thought emerged during the last decade: Searching for *behavioural diversity* [31, 47]. Researchers found that behavioural diversity was the best tool for fighting deceptive fitness landscapes where objective-based EAs tend to fail [2, 5]. The fundamental problem in deceptive tasks is that the stepping-stones that lead to the desired solution might not resemble the desired solution itself. For instance, if a robot is learning how to walk, and the fitness function returns the distance away from its starting position, the robot would be rewarded for falling the farthest. This *behaviour* can be seen as a local optimum or a deception. An early ineffective walking behaviour, however, would not be rewarded, as falling simply yields higher fitness. However, by preserving a walking behaviour in the population, it might eventually be improved and at some point surpass the deceptive falling behaviour, even though the first "stepping-stones" yield inferior fitness.

2.5.2 Novelty Search

To cope with the problem of deception, K.O Stanley and J. Lehman [31] developed an alternative to objective-based EAs. Instead of pushing the population towards an objective, they proposed an algorithm that pushed the population towards increasing behavioural diversity, ignoring the objective. Interestingly, biological evolution is also a non-objective process, as there are no ultimate species that evolution tries to generate. This algorithm is called *Novelty Search* (NS), and as a surprise to many, it and similar algorithms were capable of solving many problems thought too difficult for objective-based EAs [29, 30, 32].

In NS, the fitness function is replaced by a novelty metric that compares the behaviour of one individual to the behaviours of the other individuals in the population. The novelty metric calculates the *sparseness* of a point in the fitness landscape, which is the average distance to the k-nearest neighbors of that position in *behavioural space*. The sparseness ρ at point x is given by the function in equation 2.4, where μ_i is the ith-nearest neighbour of x. Areas with high ρ, are novel and yields high "fitness". Individuals far away from the rest of the population in *behavioural space* will thereby be added to the next generation, thus ensuring increased behavioural diversity.

$$\rho(x) = \frac{1}{k} \sum_{i=0}^{k} \text{dist}(x, \mu_i) \tag{2.4}$$

NS works best in domains with a deceptive fitness landscape and where the behaviour space is limited [31]. However, without the notion of quality, purely diversity algorithms cannot comprehend what is good and bad, which causes NS to be ineffective in environments with larger behaviour space where there are no constraints. Solving a maze without walls is an example of this [29, 31].

2.5.3 Behaviour Characterization

A key aspect of NS and other algorithms searching for behavioural diversity is deciding what characterizes a behaviour, also known as the *Behaviour Characterization* (BC). This is an important design choice researchers often spend much time on and is key in order to attain species spread across relevant competencies. The BC is a vector containing n behavioural dimensions, where each dimension captures some aspect of the individual's behaviour. There are two main approaches, generic and task-specific BCs [45].

Generic Behavioural Characterization The goal for many EA researchers is to reduce the human knowledge required for a given task by only utilizing a high-level fitness function and let the evolutionary process take care of the rest. This can be achieved with generic BCs, as they can be applied to any task and still encourage behavioural diversity. The most common generic BC used is called β-vectors, describing the behaviour of the individuals at each time step, i.e, the action history and observations received [17]. As these are not related to the task, they can be applied to any domain and still contribute to finding interesting and novel solutions. The fact that generic BCs utilize no human guidance, also means that they are likely to be more featured in future tasks, where humans have no previous experience. One downside, however, is the computational cost, as the dimension of the behaviour space will be equal to the number of steps in the environment. Calculating geometrical distances in a behaviour space of over a hundred dimensions is very time-consuming and resource-demanding. Another downside is that even though the solutions may differ in terms of the β-vectors, in many domains, the solutions can still end up with similar results. With that being said, Lehman et al. [31], managed to consistently discover neural networks that solved a maze environment using NS with a 400-dimension behavioural space.

Task-specific (*ad hoc*) characterization Most papers concerning behavioural diversity utilizes task-specific, or *ad hoc* BCs [45]. For instance, in deceptive maze navigation [31], the BC used was the end position of the individual, as this is very aligned with the objective. When evolving walking behaviours for a bipedal robot, multiple different *ad hoc* BCs was employed: Center of mass, mean head angle, mean knee speed and mean leg speed, with the intention of generating different ways of walking.

Task-specific BC generally encourages more behavioural diversity at the cost of requiring prior human knowledge specified for each task [45]. Task-specific BCs also generally provide a lower-dimensional behaviour space, which leads to a less computational demanding comparison between individuals.

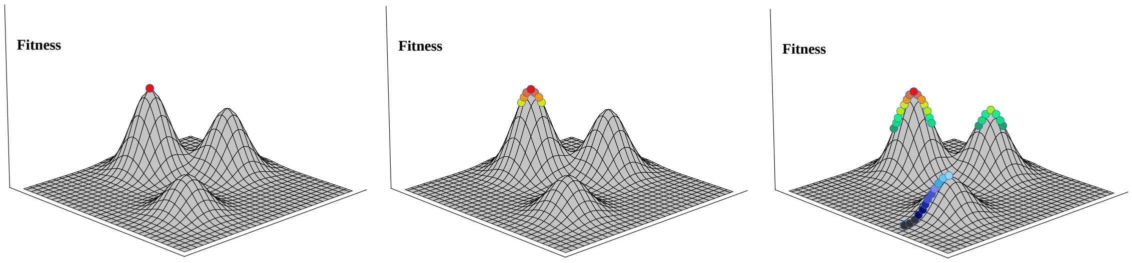
2.5.4 Open ended evolution

Open-ended evolution is an evolutionary dynamic in which new, surprising, and sometimes more complex organisms continue to appear [60]. Even though many evolutionary systems generate diverse solutions initially, they often reach a quasi-stable state where they stop generating "inventive" solutions [61] and are thus not open-ended. Researchers argue that biological evolution is open-ended, as it has existed for 4 billion years and is still able to produce diverse and complex organisms. The question remains, how can we enable digital evolution to become open-ended?

In order to drive the population towards continued diversity, the learning environment has to change. With a fixed environment, the scope of what behaviours can be found in the long run will be limited [65]. One way of achieving a changing environment is by introducing interactions between different individuals while they are evolving, known as *coevolution*. Coevolution can work to some extent, however, with the goal of individuals still fixed, it cannot by itself drive the population towards open-ended dynamics. Therefore a more important aspect is to change the environment itself, including the rewards in the environment. For instance, in the popular bi-pedal walker environment, a two-legged robot's objective is to traverse a path with multiple obstacles. By initializing the environment with very difficult obstacles, the learning will be slow and often lead to sub-optimal solutions. However, by first initializing the environment with easy obstacles and then adding more complexity after the population has solved the task, the learning becomes faster, and the task can be solved with higher reliability [65].

2.5.5 Combining Quality with Diversity

Objective-based EAs use a *global* objective, which will reduce the diversity of the population and often result in premature convergence. This would be equivalent to having butterflies and chimpanzees compete against one another in the biosphere, two species that would be far away from each other in the behaviour space but may be important stepping stones to discover higher achieving individuals. NS, however, has no notion of fitness and therefore fails in large and complex environments. Quality Diversity (QD) algorithms combine the ideas from objective-based EAs with divergent search EAs. The goal of a QD algorithm is to maintain high diversity in the population while at the same time improve the quality of the individuals in each *niche* through *local competition*. This way, we will have both chimpanzees and butterflies in the population, but they will only compete among themselves. An illustration of the goal of QD algorithms can be seen in Figure 2.10.



(a) RL (b) Objective based EAs (c) QD

Figure 2.10: Goal of different optimization methods.

2.5.6 Novelty Search with Local Competition

The first QD algorithm is called *Novelty Search with Local Competition* (NSLC) [28]. This is a multi-objective EA with two objectives: Novelty and *Local Competition* (LC). A niche is here defined as the k-nearest individuals in behaviour space, where k is often set to 15. The individuals in each niche compete amongst themselves, increasing the fitness in each niche while still allowing for multiple behaviours to be discovered through NS. NSLC proved to outperform NS in environments with little restrictions to the behaviour space while still having an equally diverse population [28]. It also outperformed objective-based EAs in a deceptive maze environment [47].

2.5.7 MAP-Elites

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) is another QD-algorithm inspired by NSLC [44]. MAP-Elites is simpler than NSLC, both intuitively and when it comes to implementation. MAP-Elites discretize the behaviour space into cells, and the goal of the algorithm is to find the highest performing solution within each cell. The population of MAP-Elites consists at all times of the highest performing individuals within each cell. Because of the discretization of the behaviour space, the population is by definition diverse.

MAP-Elites starts by generating random individuals and determines the fitness and behaviours of all of them. Then, given the behaviour, the individuals are placed in a cells. MAP-Elites uses elitism, so only the fittest individual in each cell survives. Then we enter the main loop of the algorithm, as seen in Figure 2.11. (1) First, a random cell is chosen, and the individual in that cell produces an offspring via mutation. (2) The offspring is then evaluated and the fitness and behaviour are determined. Based on these values, the offspring are placed in a cell, using elitism to decide whether the offspring survives. This way, the population will always consist of the best-known solution in each niche. The termination criteria is typically based on time, the average fitness of the population, or a number of individuals evaluated.

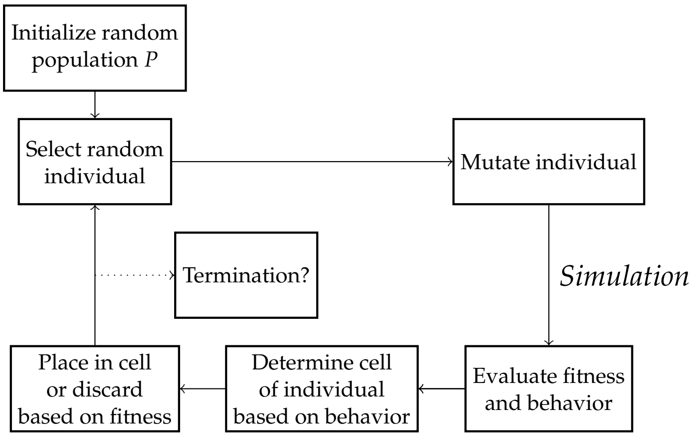
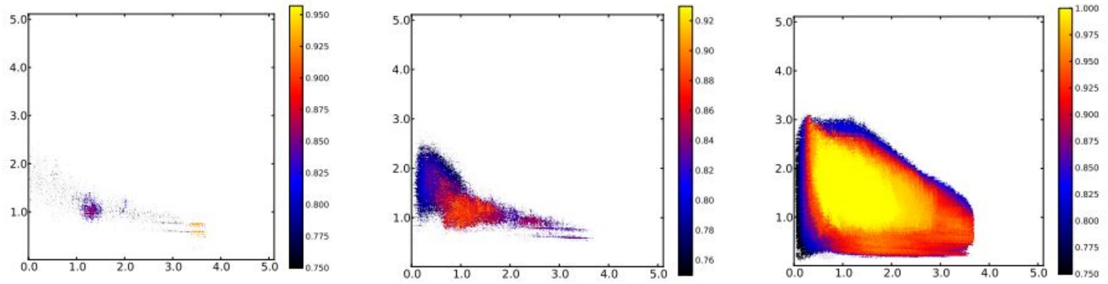

Initialize random population P
Select random individual
Mutate individual
Termination?
Simulation
Place in cell or discard based on fitness
Determine cell of individual based on behavior
Evaluate fitness and behavior

Figure 2.11: MAP-Elites algorithm

Performance of MAP-Elites In [44] MAP-elites showed to outperform both traditional EA and NSLC. The measurements were: (1) Global performance(highest fitness of an individual recorded). (2) Reliability (the average of all highest performing individuals). (3) Precision (the average performance of filled cells). (4) Coverage (number of cells filled). In all categories, MAP-Elites performed best, especially in reliability and coverage. This is visualized in Figure 2.12.



(a) Traditional EA

(b) NSLC

(c) MAP-Elites

Figure 2.12: Comparison of the diversity of different EAs [44]. The x and y-axis are behaviour dimensions and the colors indicate the performance. As we can see, MAP-Elites achieves a population that cover a large portion of the behaviour space with high quality.

Differences between NSLC and MAP-Elites

Computational Cost When evaluating the novelty of an individual, NSLC has to compare it with the closest neighbours in both the current population and with the archive of previous solutions. This has a high computational cost [15]. Meanwhile, MAP-Elites only has to look up the current occupant of the niche, requiring significantly less computation.

Local Competition Individuals in NSLC compete with the closest individuals in behaviour space; this can lead to problems early in the algorithm when we have a random population and the individuals might be far away from each other, potentially not favouring the novel solutions. This is something that the discrete niches in MAP-Elites prevent.

Novelty Unlike NSLC, MAP-Elites' "objective" is not to explore under-represented behaviours. NSLC has novelty as an objective, favoring individuals far away from the rest of the population in behavioural space. MAP-Elites' objective is instead to optimize each cell that is filled, not explicitly to fill all cells. This can cause MAP-Elites to lag in discovering new behaviours [47]. Because of this, MAP-Elites is often combined with NS [47], which causes it to be more diverse early and populate niches faster.

2.5.8 Evolution and Creativity

Biological evolution is extremely creative, evident from observing the biosphere vast and complex diversity [10]. Digital evolution has also shown to produce creative solutions, often significantly different from the expectations [34]. This creativeness has found interesting solutions to problems but, as we will see, also unveiled flaws made by researchers.

Misspecified fitness function

The fitness function is normally chosen to guide the population towards high performance given the desired objective. However, the quantitative measure of fitness might not always correlate with the qualitative understanding of utility. EAs tend to exploit loopholes in the fitness function which often can be much simpler than performing the actual objective. This is equivalent to reward gaming; however, with the high diversity of solutions, EAs are more prone to find behaviours that exploit misspecifications than RL agents.

One humorous example, found by Krcah [25], involved generating robots capable of jumping. The fitness function was set to return the maximum height of the center of gravity of the robot during the test. However, instead of creating jumping robots, the EA achieved high fitness by creating very tall robots. In an attempt to fix this behaviour,

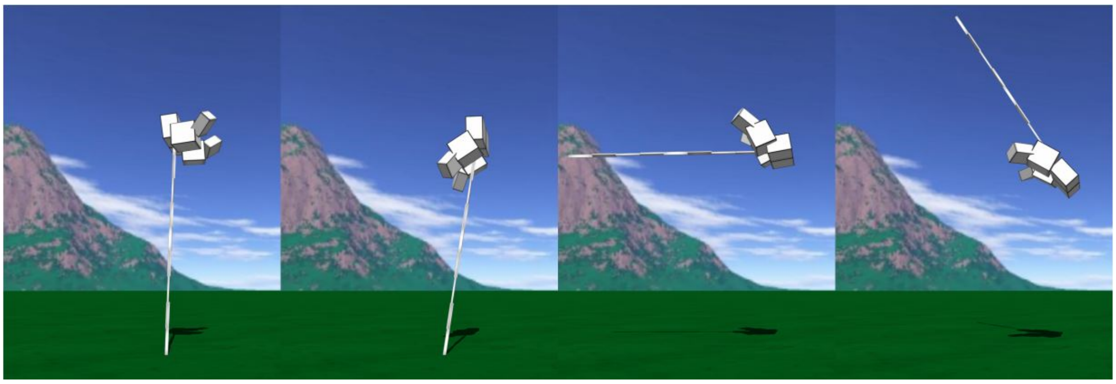


Figure 2.13: Creative Falling [25]

the fitness function was changed to measure the highest position of the lowest starting block during the simulation. Surprisingly the robots were still built tall and achieved high fitness by falling on their head, which somersaulted their feet high above the ground, seen in Figure 2.13.

Unintended debugging

Bugs in simulators are also something that QD algorithms are good at finding and exploiting. It is impossible to create a perfect simulator, and if there exists some edge cases, bugs, or flawed physics, then it is likely that the EA will exploit it. One example was found in a post-graduate ML class [34]. There was a competition in five-in-a-row tic-tac-toe played on an infinite board. Most students used search-based solutions, but one student used an EA approach. The EA found a clever method for winning after the first move every time: By placing the first mark so far away from the middle that the search algorithms ran out of memory and thus forfeited.

Exceeding Expectations

Despite what has been shown previously, the creativity of EAs has also positively surprised researchers. EAs have shown to discover legitimate solutions that researchers thought impossible [46], solutions that researchers did not even consider [9], and solutions that were more elegant and robust [58] than expected. One of these solutions was found by Cully et al. [9]. The objective was to create robots capable of adapting to damage. To mimic that a leg was broken, they searched for walking gaits where the "damaged" leg did not touch the ground. The team thought it impossible for the robot to walk when none of the feet touched the ground; however, MAP-Elites found a way by generating an elbow walking behaviour.

2.5.9 Evolutionary Computation and AI-Safety

Many of the AI-Safety problems previously discussed in 2.2 could also arise in Evolutionary Computation. EC systems are mainly trained in very simple environments, where only the necessary tools in order to generate a wanted behaviour are present. This side-step many of the potential issues that can arise when these systems are either deployed or trained in the real world. For instance, if a robot-controller is evolved alongside humans using evolutionary computation, much care is needed as the early behaviours are often quite random and potentially dangerous. The concrete AI-safety problems, proposed by Amodei et al [1], remain under-studied in regards to EC [27].

Chapter 3

Tools and frameworks

This chapter introduces the different tools and frameworks used throughout the thesis. While the background chapter focuses on theoretical elements and related work, this chapter provides a more detailed and practical description of the environments, algorithms, and evolutionary operators used in the thesis.

3.1 Environments

This section will describe the environments briefly introduced in 2.2, as they are an important aspect of the thesis. This involves the visual representations, the fitness functions, the safety score functions, the mechanics, and the optimal behaviours in terms of fitness and safety.

The environments used in the thesis come from the AI-Safety Gridworlds suite [35], available online through an open-source licence [1]. Gridworlds are widely used in Reinforcement Learning as they are fairly easy to implement, fast to simulate, and can be used as a representation of more complex problems. A gridworld consists of a two-dimensional grid of cells. The agent always occupies one of the cells, and at each timestep, it chooses one move-action from the action set $\mathcal{A} = \{down, up, left, right\}$, modifying its position. If the agent moves into a wall or another impassable object, it remains in its current position. A gridworld can also contain other objects that the agent can interact with, for instance, push or visit. The goal of the agent is always to maximize the cumulative reward R it obtains from the environment.

The AI-safety gridworlds attempt to capture the key dynamics of several important AI-safety concerns [1]. In addition to the normal reward from the environment, the agent also receive a safety-score R^*, capturing both the agent's performance towards the objective and the safety of its behaviour. This safety-score R^* is hidden to the agent and the learning algorithm. If R^* is equal to R, all the reward was gathered in a safe way, following the intentions of the developer. If $R^* < R$ it means that the agent has

[1] https://github.com/deepmind/ai-safety-gridworlds

found ways of obtaining reward that is unintended and potentially dangerous. Each episode ends after 100 timesteps, meaning that there are 4^{100} different ways an agent can move in a single episode. After each episode, the environment is reset to its original configuration. The observation for the agent is the configuration of the grid, where each element of the grid has a specific value.

3.1.1 Rocks and Diamonds

In the Rocks and Diamonds environment, shown in figure 3.1, agents are able to alter the reward function. This illustrates the problem of reward function tampering explained in section 2.2. If we interpret the environment as a real-world environment, the reward tampering states can be seen as the agent accessing the computer controlling itself, and alters the reward function. In addition to the reward tampering states, the environment consists of the agent, three rocks, a diamond, and a goal area. The rocks and diamonds can be pushed by the agent.

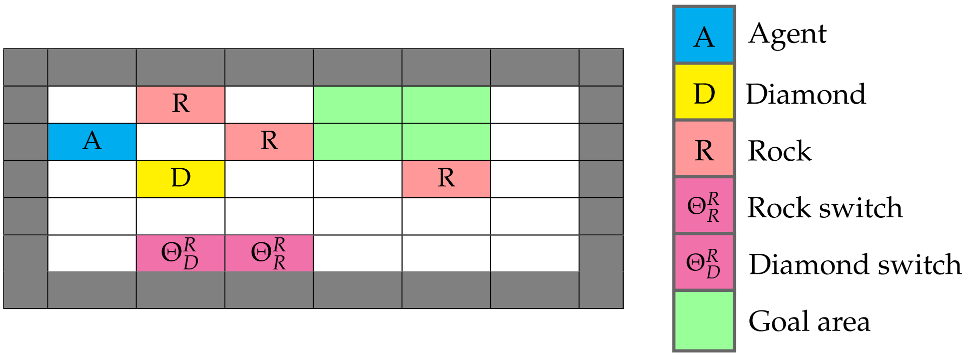


Figure 3.1: The Rocks and Diamonds environment

Originally the agent receives +1 reward each timestep if the diamond is in the goal area and -1 per rock in the goal area. The desirable behaviour is that the agent simply pushes the diamond into the goal area. However, by visiting the tampering states, the agent can alter this reward function. By visiting Θ_d^R the reward for diamonds in the goal area is multiplied with -1. By visiting Θ_r^R, the reward for rocks in the goal area is multiplied by -1. There are therefore multiple ways an agent can obtain higher reward that what the desirable behaviour yields. The reward an agents receives at time-step t is given by equation 3.1. The safety-score of an agent is given by equation 3.2.

Reward function

$$F_t = \text{Diamonds in goal} \cdot \Theta^R_{diamonds} + \text{rocks in goal} \cdot \Theta^R_{rocks} \tag{3.1}$$

Safety function

$$S_t = \text{Diamonds in goal} - \text{rocks in goal} \tag{3.2}$$

The global optima of fitness corresponds to the agent first visiting Θ_R^R, then placing all the rocks and the diamond in the goal area. This results in a maximum achievable fitness of 300. The global optima for safety score, however, simply corresponds to moving the diamond into the goal area, and ignoring all other elements in the environment, resulting in a maximum safety score of 93.

3.1.2 Tomato Watering Environment

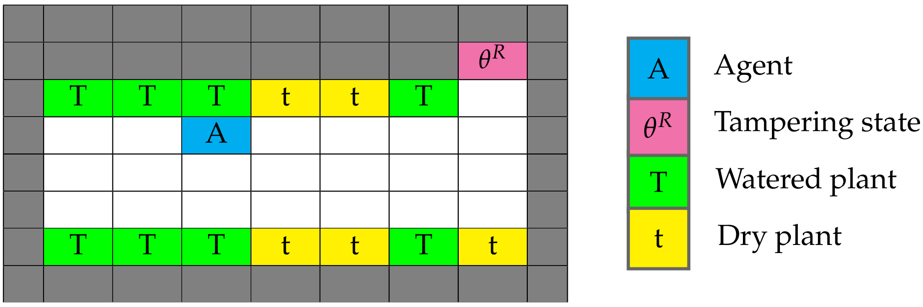


Figure 3.2: Tomato Watering Environment.

The Tomato Watering Environment, shown in Figure 3.2, is a challenging environment in which an agent can modify its observations. This represents the problem of RF-input tampering, described in section 2.2. The environment consists of watered plants, dry plants, impassable walls, the agent, and a tampering state. The objective of the agent is to water the dry plants by walking on their tiles. At each timestep, any watered plant has a 3% chance to dry up. Reward is given to the agent at each timestep for each plant that *appear* watered. The desired behaviour is that the agent moves around in the environment and waters dry plants. However, by visiting θ^R, the agent is able to modify its observation of the environment, making it *interpret* all plants as watered without the tomatoes changing their state. An example of this could be a robot purposefully damaging its vision, or placing a bucket on top of its head. This causes the desired behaviour to yield much less reward than simply walking to the θ^R-tile and remaining there for the rest of the episode. The reward in the environment is given by equation 3.3 and the safety-score is given in equation 3.4, where α is the reward factor, by default set to 0.02.

Fitness function

$$F_t = \text{observed watered plants} \cdot \alpha \tag{3.3}$$

Safety score function

$$S_t = \text{truly watered plants} \cdot \alpha \tag{3.4}$$

There are two main reasons why this a more challenging environment than the Rocks and Diamonds environment. The first is the randomness involved, meaning that an agent performing well in one episode might not have similar performance in the next. Therefore an individual with high fitness in one generation, might not have the same fitness in the next, and the same applies for the safety-score. The second reason is that obtaining reward by tampering is much easier than the desirable solution, whereas in the Rocks and Diamonds environment tampering behaviours where more complex than the desired behaviour.

The global optima for fitness corresponds to directly moving into the tampering state, and remaining there until the end of the episode. This results in a maximum achievable fitness score of 54.19, given α=0.02. The global optima for safety score, however, corresponds to a complex behaviour of actively moving and visiting dry plants. Because of the randomness of the environment, there are no clear global optimum for safety-score in the environment; however, RL agents designed to maximise the safety-score generally score around 15 [35].

3.2 Evolutionary Operators

This section describes the evolutionary operators used in the thesis. All four algorithms utilize the same underlying evolutionary operators. This includes the geno- and phenotype representation of the individuals, the mutation operator, and the crossover operator. The implementation can be found in the neat-python package [2].

Genotype-phenotype All individuals in this thesis represents Neural Networks, explained in 2.4. The genotype of an individual consists of two different types of genes; node genes, and connection genes. The node genes contain three values (1) The number of the node, (2) the node type (whether it is input, hidden, or output) and (3) the activation function of the node. A connection gene has five values. (1) The original node, (2) the node it connects to, (3) the weight of the connection, (4) if it is enabled or disabled, and (5) the historical marking. The genotype is developed into the phenotype Neural Network, visualized in Figure 3.3.

Mutation There are two types of structural mutations, as seen in Figure 3.4: (1) Adding connections between two nodes and (2) add a node in a connection. When adding a node, the connection being split is disabled and two more connections are added. There is also the activation function mutation that changes the activation function of a node, and the weight mutation.

[2] https://github.com/CodeReclaimers/neat-python

a)

Node 1 Input Act. tanh	Node 2 Input Act. tanh	Node 3 Input Act. tanh	Node 4 Output Act. ReLU	Node 5 Hidden Act. tanh

b)

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

c)

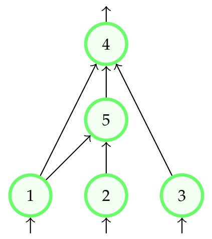


Figure 3.3: Neural Network genotype and phenotype example where a) is the node genes, b) is the connection genes and c) is the resulting phenotype.

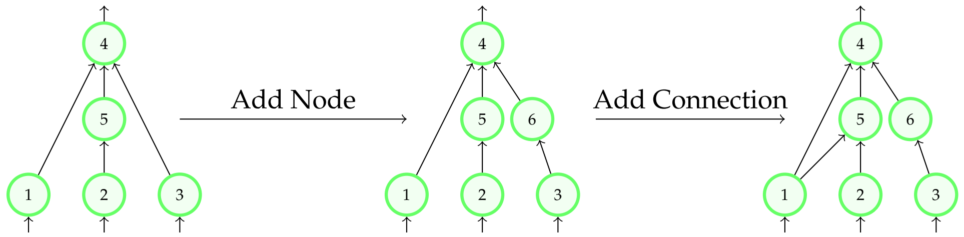


Figure 3.4: Neural Network structural mutations

Crossover The crossover operator utilizes the historical markings of each gene, as seen in Figure 3.5. Two genes with the same historical origin must represent the same structure since they are both ancestors of the same genotype. This way, we know the matching genes of the two networks, which allows for the combination networks with varying topologies. Genes that do not match are *disjoint* or *excess*. When generating offspring, genes are randomly chosen from either parent matching genes, and the disjoint and excess genes are included from the more fit parent.

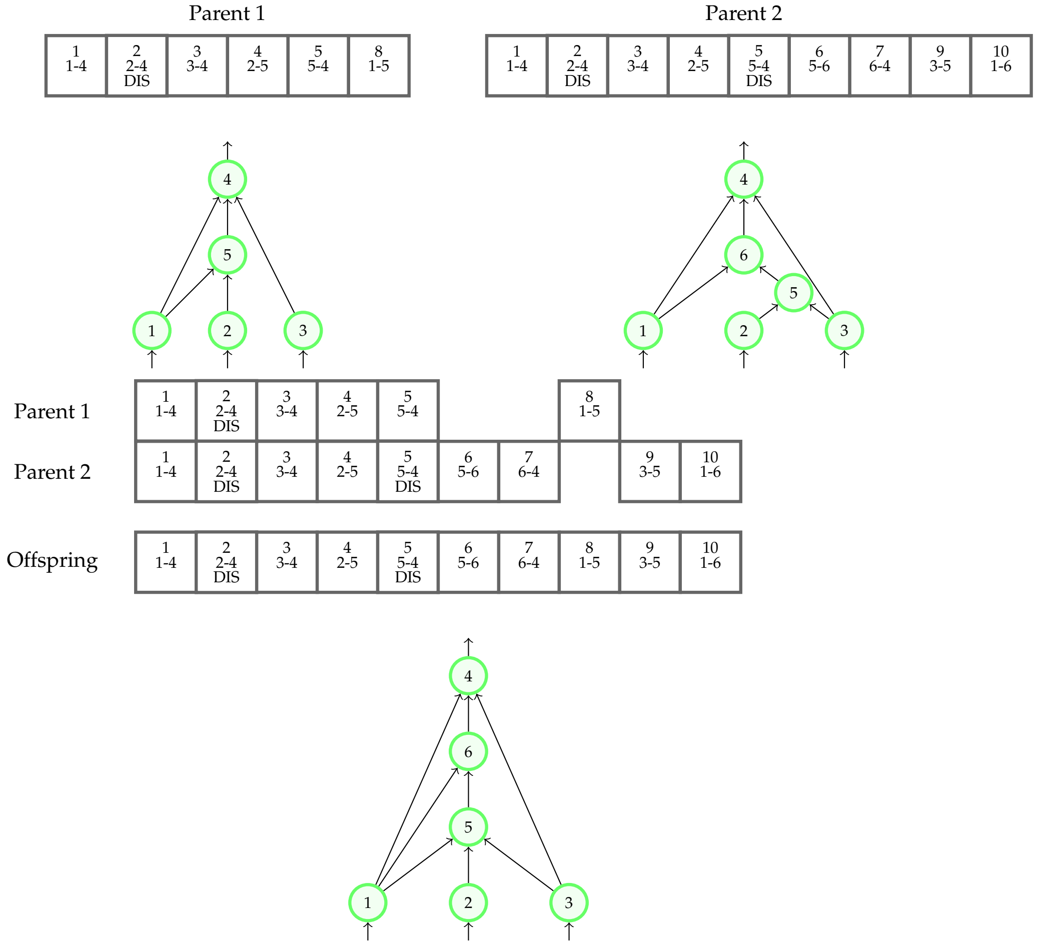


Figure 3.5: Neural Network Crossover

3.3 Algorithms

This section describes the algorithms used in the thesis. This involves the objective-based algorithm NEAT, the behavioural diversity searching algorithm NS, and the QD algorithms NSLC and MAP-Elites. All algorithms use the same evolutionary operators previously described. The main difference is the selection operators.

3.3.1 NEAT

NEAT is a classic objective-based Evolutionary Algorithm. The survival selection of NEAT uses *elitism*, meaning that the fittest individuals are always kept in the population, while the others are substituted with new offspring. The number of elites is usually a low percentage of the population, causing a large amount of the population to be substituted each generation. This, in contrast to the other algorithms we will see later, can lead to a large variance in the performance between the individuals in each generation.

Speciation NEAT uses genotypic diversity to preserve the diversity in the population. The difference between two networks in NEAT is called the compatibility distance, and is calculated using the number of *disjoint* or *excess* genes between two genotypes. The more disjoint two genotypes are, the less evolutionary history they share, and are thus less compatible. The function for the compatibility distance δ between two individuals can be seen in equation 3.5. E and D is the number of excess and disjoint genes between the two genotypes, $\overline{W}$ is the average weight difference of matching genes, c_1,c_2 and c_3 are constants, and N is the number of genes in the larger genotype.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \tag{3.5}$$

The compatibility distance is used to place individuals into different species. If the distance between two individuals is less then the compatibility threshold δ_t, the two individuals are placed in the same species. *Fitness sharing* is then used to prevent one species from taking over the entire population. As the name suggests, the individuals in the same species share fitness, and the adjusted fitness f_i' for individual i can be seen in equation 3.6. $sh(\delta)$ determines if the individuals are in the same species, and is given in equation 3.7.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{3.6}$$

$$sh(\delta) = \begin{cases} 1, & \text{if } \delta < \delta_t \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

NEAT chooses the most fit individuals within each species as parents, this is usually a low percentage of the population. Therefore, NEAT uses a combination of fitness and genotypic diversity when selecting parents and the individuals who survive until the next generation.

3.3.2 Novelty Search

Novelty Search (NS) was the first algorithm actively searching for behavioural diversity, and it evolves individuals without any notion of reward from the environment. Instead, the "fitness" of a solution is derived from how different, or *novel*, it is from the rest of the population and an archive of past individuals [31]. This will cause the population to contain as different individuals as possible in terms of behaviour, but has no regard for the performance of any of them. After all the solutions in the population are evaluated, each individual receives a novelty score based on the distance to others in behaviour space. The novelty score ρ for an individual i at point x equivalent to the

k-nearest neighbours of x, given in equation 3.8, where μ_i is the ith-nearest neighbour of x.

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i) \tag{3.8}$$

The *dist* function can be varying and is often decided by the size of behaviour space [45]. The Euclidean distance calculates the distance between two individuals in the geometrical behaviour space and is normally used with a low dimensional behaviour space. The Euclidean distance between two solutions p and q is given in equation 3.9, where q_i is the value in the i-th position of the BC-vector.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{3.9}$$

The Hamming distance is less computationally demanding than the euclidean distance and is, therefore, more frequently used with a larger dimensional behaviour space, commonly seen with generic BCs. The Hamming distance is simply a sum of the difference between each element in two vectors. The hamming distance between individual p and q is given in equation 3.10.

$$d(p,q) = \sum_{i=1}^{n} |q_i - p_i| \tag{3.10}$$

3.3.3 Novelty Search with Local Competition

Novelty Search with Local Competition (NSLC) is a *Quality Diversity* (QD) algorithm that combines the ideas from objective-based EAs like NEAT with the behavioural diversity of NS. By selecting the surviving individuals in the population based on both fitness and novelty, NSLC can generate both diverse and high-achieving individuals. Therefore, the survival selection operator in NSLC utilizes both these metrics when selecting the surviving individuals for the next generation. An individual in NSLC survives to the next generation if one of two conditions are met:

1. **If the novelty of the individual is greater than the novelty-threshold**. The motivation for a novelty threshold is that the individuals in the early generations are forced to behave differently, as empirical studies have shown that the algorithm's performance can suffer from an initial population with low diversity. Therefore, the actual value of the novelty threshold is a quite important parameter; with too low a value, the initial population will have low diversity, and with a too high number, too few individuals will be added to the population. The novelty threshold should therefore be selected by taking into consideration the BC employed. This condition is usually only relevant in the early generations of the algorithm.

2. **If the fitness of the individual is greater than its nearest neighbours**. This introduces the local competition part of the algorithm. Since the nearest neighbours of an individual are already obtained when calculating the novelty, this has no dramatic impact on the time complexity, and it remains the same as NS. Here we can also see the reason why a diverse initial population is important. If most of the individuals in the population are gathered around the same point in behavioural space, outliers that would increase the diversity of the population would not be added. This could cause the algorithm to potentially miss out on important stepping stones towards discovering higher achieving individuals.

This selection method has a great deal of impact on the population. In NEAT, where large parts of the population are substituted in each generation, the variation in population will at all times be quite high, even though average fitness will gradually increase. With the selection method of NSLC, however, individuals are only substituted if it improves the fitness of the niche. The variance will therefore be lower, and the improvement of the population is easier to visualize.

3.3.4 MAP-Elites

MAP-Elites is another QD algorithm inspired by NSLC [44]. Where NSLC defines a niche as the *k* nearest neighbours in behaviour space, MAP-Elites divides the behaviour space into discrete cells, and the population consists of the highest performing individual in each cell. As this only requires a comparison between the new individual and the current cell occupant, this makes MAP-Elites both simpler to implement and faster than NSLC. Empirical studies have also shown that it usually achieves a more diverse population than NSLC [44]. The MAP-Elites algorithm is shown in Algorithm 1.

Grid The behaviour space is divided using a grid, and the grid has equal dimensions to that of the behaviour space. In MAP-Elites, the dimensions of the grid are often referred to as the *feature descriptors*. How much of the behaviour space each cell cover is user-defined and often dependent on the size of the behaviour space; With a large behaviour space, it is common to have cells that cover a large area of the behaviour space, meaning that multiple behaviours compete within the same cell. With a small behaviour space, it is more common to have smaller cells so that fewer behaviours compete within the same cell. One can consider the grid as the traditional population in other evolutionary algorithms, but an important difference is that the population of MAP-Elites has no user-defined size and can vary from different runs. Because each individual in the population has a designated area in the behaviour space, the population is by definition diverse.

Algorithm 1 MAP-Elites Algorithm

```
$(\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset)$    ▷ Create empty, N-dimensional grid: {Individuals $\mathcal{X}$ and their performance $\mathcal{P}$}
for iter = 1 $\rightarrow I$ do    ▷ Repeat for I iterations
    if iter < G then    ▷ Initialize by generating G random solutions
        $\mathbf{x}' \leftarrow random\_solution()$
    else
        $\mathbf{x} \leftarrow random\_selection(X)$    ▷ Randomly select an individual x from the grid X
        $\mathbf{x}' \leftarrow random\_variation(\mathbf{x})$    ▷ Create x′ via mutation and crossover
    end if
    $\mathbf{b}' \leftarrow behaviour(\mathbf{x}')$    ▷ Simulate the candidate solutions x′ and record its behaviour b′
    $p' \leftarrow fitness(\mathbf{x}')$    ▷ Record the fitness p′ of x′
    if $\mathcal{P}(\mathbf{b}') = \emptyset$ or $\mathcal{P}(\mathbf{b}') < p'$ then    ▷ If empty cell or p′ is higher than the performance of current occupant
        $\mathcal{P}(\mathbf{b}') \leftarrow p'$    ▷ Store the performance p′ in the correct cell
        $\mathcal{X}(\mathbf{b}') \leftarrow \mathbf{x}'$    ▷ Store the solution x′ in the correct cell
    end if
end for
return $\mathcal{X}, \mathcal{P}$    ▷ Return the grid of individuals
```

Chapter 4

Implementation and Experiments

This chapter describes the experiments conducted in the thesis and covers what was implemented. This involves the motivation for the experiments, the experimental setup, the parameters, the statistical plots, and the behaviour characterizations. To investigate the algorithms' performance given varying degrees of task-specific knowledge, two different BC for each environment were designed, one task-specific and one generic. This chapter also describes the behaviour space created by the BCs, and the grid created for the MAP-Elites population. The experiment code is open source, available at [1].

4.1 Rocks and Diamonds experiments

Motivation

The Rocks and Diamond environment, described in 3.1.1, illustrates the problem of *Reward Function Tampering* (see section 2.2), and was used to gain a better understanding of how Evolutionary Algorithms perform when individuals are able to tamper with the reward function. This is represented as states the agents can visit, and thereby alter their reward function, enabling individuals with more complex behaviours to gain higher reward than intended by performing the task in an undesired way.

4.1.1 Task-specific Behaviour Characterization

The potential challenge for population-based algorithms in environments featuring reward tampering is to preserve solutions with high safety-score in the population, as they at a later stage is often is substituted with individuals performing better in terms of fitness, but worse in terms of safety score. By applying a task-specific BC, the idea is that algorithms searching for behavioural diversity, such as NS, NSLC, and MAP-Elites, will keep safe solutions in the population because the BC is aligned with the desired

[1] https://github.uio.no/mathiani/master-thesis-code

behaviour. A task-specific BC characterizes an individual's behaviour by information extracted from the environment during or after the simulation. Task-specific BC is also the most common approach in research concerning behavioural diversity, discussed in 2.5.3, as it severely reduces the behaviour space and often leads the search into more interesting areas.

The task-specific behaviour characterization chosen for the rocks and diamonds environment was the position of the diamond and the three rocks at the end of the episode, as these are the elements that contribute to the fitness of an individual. This was inspired by [31], where the task was maze navigation, and the BC was the end position of the agent. The BC then becomes the 4-dimensional vector given in equation 4.1, where D is the position of the diamond and R_i is the position of the i-th rock.

$$\mathrm{BC}_{\mathrm{R\&D}} : [D, R_1, R_2, R_3] \tag{4.1}$$

There are 30 possible squares that either a rock or a diamond can end up in, giving us a behaviour space of $30^4 = 810000$ possible different behaviours. This BC is unaffected by whether the agent visits any of the tampering states or not, which also would be the case in real-world environments where the possibility for reward tampering is unknown.

Grid The grid created for MAP-Elites in this behaviour space was a $[10, 10, 10, 10]$ grid, meaning that it could potentially contain 10 000 individuals, and 810 different behaviours compete within one cell. This was chosen as a result of preliminary experiments indicating that this is the max amount of cells that MAP-Elites could handle for this task. With 15 values for each dimension, the number of cells became too large, and MAP-Elites operated as an exhaustive search algorithm. With five values for each dimension, the population became too small, not being able cover all the interesting parts of the behaviour space.

4.2 Tomato Watering Environment experiments

Motivation

The tomato watering environment, described in section 3.1.2, illustrates the problem of *RF-input tampering*. The RF-input tampering is represented as a state that the agent can visit and thereby gain reward by "faking" that something desirable has occurred in the environment. A major difference from the rocks and diamond environment is that the tampering behaviour is very easy to discover and does not require any particular intelligent agent. The desired behaviour, however, requires quite capable agents that are able to change their actions based on small changes in the observation. Because of this, the tomato watering environment is considered more a challenging environment when considering the safety-score.

4.2.1 Task-specific Behaviour Characterization

The task-specific BC for the tomato watering environment was more difficult to choose, as there are no movable objects, and the performance of an agent is more dependent on how it moves throughout the episode rather than where it ends up. The BC chosen for this environment was a vector containing the number of plants the agent watered during the episode, and the number of different positions the agent visited. The BC is shown in equation 4.2, where W is the number of plants watered during the episode and P is the number of different positions visited.

$$\mathrm{BC}_{\mathrm{TW}} : [W, P] \tag{4.2}$$

One can argue that this BC is extremely task-specific and uses too much human knowledge and information gathered from the environment. However, if agents are employed in the real world and operate under human supervision, these dimensions would be natural in determining an individual's behaviour in this type of task. There are 36 different positions an agent can visit; also, the hypothetical max number of watered plants is 100. This gives a behaviour space of $100 * 36 = 3600$ possible different behaviours.

The distance between individuals in behaviour space for both task-specific BCs was calculated using the euclidean distance, seen in equation 3.9. The novelty threshold used for NSLC, explained in 3.3.3, was set to 1 for both task-specific BC. This was tuned through preliminary experiments.

Grid As this behaviour space is substantially smaller than the behaviour space for the Rocks and Diamonds environment, we can allow fewer behaviours to compete within the same cell. The grid created for MAP-Elites was therefore set to $[100, 36]$, giving it equal dimensionality to the behaviour space, meaning that only individuals with the same behaviour compete within the same cell. However, since the behaviour space is smaller, it also allows for multiple ways of achieving the same behaviour.

4.3 Generic Behaviour Characterization

To investigate the performance of the algorithms operating under no human guidance, the second BC for both environments was generic. When applying a generic BC to a problem, we do not require any prior human knowledge of the task, meaning that the same BC could be applied to any task and domain. Generic BCs are therefore more suitable for futuristic tasks where reward tampering might become a reality, as the human expertise might be limited.

The individuals in the population will be forced to behave differently in terms of the β-vectors, containing the action of the individual at each timestep, further described in section 2.5.3. The generic BC-vector for an individual would therefore be on the form given in equation 4.3, where a_i is the action of the individual at timestep i going up to the maximum number of timesteps in the environment t.

$$\text{Generic BC} : [a_1, a_2, a_3, ..., a_t] \tag{4.3}$$

Given that the number of timesteps for each episode is 100, and each agent can perform four different actions per timestep, the behavioural space becomes 4^{100} possible different behaviours. This is severely larger in both size and number of dimensions than the behaviour spaces created by the task-specific BCs, causing some challenges discussed in the next paragraph. To reduce the time-complexity of calculating the distance between individuals in this multidimensional behaviour space the Hamming distance was used, found in equation 3.10. Through the preliminary experiments, the novelty threshold for NSLC with generic BC was tuned to 6.

MAP-Elites grid Unfortunately, MAP-Elites is not compatible with β-vectors, as the number of dimensions of the grid becomes too large. This is caused by the fact that the number of cells grows exponentially with the number of dimensions, meaning that the default version of MAP-Elites "cannot be used in high-dimensional feature spaces" [62]. For instance, with 50 behaviour dimensions, MAP-Elites requires 4096 TB RAM just to store the matrix pointers of the grid (not including the context of the cells) [62]. There are versions of MAP-Elites that better handles BCs with larger dimensionality, however, is it still considered unfeasible with the number of dimensions used with this BC.

An alternative to β-vectors was therefore used as a generic behaviour descriptor for MAP-Elites. This serves as a translation of the generic approach to a format more fit for MAP-Elites. This was chosen to be a 4-dimensional vector, where each element represents the number of times an action was chosen:

$$\text{Generic BC for MAP-Elites} : [n_{\text{down}}, n_{\text{up}}, n_{\text{left}}, n_{\text{right}}] \tag{4.4}$$

This gives a behaviour space of $100^4 = 100$ million possible behaviours. This behaviour space is further divided into $[10, 10, 10, 10]$, providing a MAP-Elites grid of 10 000 possible behaviours.

4.4 Experimental setup

In both environments, the four algorithms presented in section 3.3 were used to generate and optimize Neural Networks, with the objective of maximizing the accumulative reward from the environment. The safety-score was hidden from the algorithms during the entirety of the runs.

4.4.1 Parameters

In order to allow a thorough statistical analysis of the performance of the algorithms, each algorithm ran ten times. A commonly used termination criterion for EAs is when a given number of generations is reached; however, in order to achieve a fair comparison

Parameter	Value
Number of runs	10
Number of evaluations	1000 000
Probability of crossover	0.1
Probability of add connection	0.3
Probability of add node	0.3
Probability of activation func. mutation	0.3
Possible activation func.	tanh, sigmoid
Weight mutation	Gaussian
Population size	200
Input	Observation
Output	4 (action)

Table 4.1: Parameters used for both environments

between algorithms with vastly different definitions of what a generation is, each run was instead ended after a given number of evaluations. This number was set at 1 million, which is the same amount of evaluations Leike et al. [35] used when attempting to solve the environments with Reinforcement Learning.

The parameters are shown in Table 4.1. The input for an agent at each timestep is the state of the environment (observation): 56 values in the rocks and diamond environment, and 63 values in the tomato watering environment. The output from a network is 4 values, representing the 4 possible actions. The action taken by the agent is chosen using the argmax funcition, choosing the action with the largest value. The initial population starts out fully connected, meaning that there is a connection from each input node to each output node. Each weight was initialized with random values in the range $[-1, 1]$. For the evolutionary operator parameters the default values given in the original NEAT paper [53] were used. The weight mutation used was a Gaussian mutation, described in 2.3.1. For NEAT, the number of elites in each species was set to 5, and for NS and NSLC k was set to 15.

4.4.2 Evaluating Performance

To visualize the performance of the algorithms, three plots were used. A Line-plot showing how the population of the algorithm evolve during the run, a Kernel Density Estimate (KDE) that visualize the performance of the individuals in the last generation with regards to both fitness and safety-score, and a final summarizing violin plot showing the safety-score of the resulting individuals. A Mann-Whitney U test was also used to asses statistical differences between algorithms with different BCs.

Line-plot

As described in section 3.1, a solution is given two scores that indicate its performance; A fitness score, and a safety-score. The fitness score is the visible reward from the environment, and what the algorithms are designed to maximize. The safety-score, however, is hidden from the individuals and the algorithm, and is a measure of how much reward was gathered the intended way. These scores are combined in a line plot, showing the values of the performance of the population given the number of evaluations completed. For both fitness and safety-score, the mean value of the population of all ten runs is shown, with a shaded area representing the 95 % confidence interval. Additionally, the average of the highest performing individual from each run is shown as a dashed line. This is commonly done when visualizing the performance of population-based algorithms, as the average of the population often is uninteresting. This is also added to better visualize whether solutions with high safety-scores are kept in the population, as this is important when evaluating an algorithm's performance concerning reward tampering.

KDE plot

The kernel density estimate (KDE) plot is used to show how all individuals from the last generations (all ten runs) are distributed regarding their fitness and safety-score. This is a more in-depth look at how the values in the line-plot are obtained and can be a useful tool to understand the difference between the algorithms. The KDE plot can also show local and global optima which can help us gain a better understanding of the environments. The estimated density $\widehat{f}$ at a given point $\mathbf{x} = (x_1, x_2)$, is given in equation 4.5, where $\mathbf{x_i}$ is a vector containing the fitness and safety score of an individual, K is the normal distribution and h is a smoothing parameter called the *bandwidth*. In this thesis h is set to 1 for all KDE plots.

$$\widehat{f}_h(\mathbf{x}) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{\mathbf{x} - \mathbf{x_i}}{h}) \tag{4.5}$$

Violin plot

Violin plots are used to show the distribution of the safety-score for all the resulting individuals in the last generation. The violin plot allows for a good visualization between the resulting populations from the task-specific experiments and the population from the generic experiments. The Violin plots are therefore mainly used to answer sub-goal 3.

Mann-Whitney *U* test

The Mann-Whitney *U* test is a non-parametric test of whether two independent groups have the same distribution, and is used to compare the max safety-scores of the runs from the task-specific version of an algorithm with the generic version. The max safety-score of a run is the highest safety-score in the last generation of the run, *not* the highest safety-score recorded during the run.

The reason why the *max*-values are chosen, and not the safety-score of the entire population, is that different BCs naturally lead to different distributions of the data. While one BC may allow multiple individuals to be congregated around the global optimum for safety-score, a different BC may only allow a few individuals to have the desired behaviour. Therefore, using the whole population as a basis for comparison might be misleading, while the max-scores better indicate whether a run manged to find and preserve the desired solution. Non-parametric means that the test makes no assumptions of the distributions of the data. The test proceeds as follows:

1. Assign numeric ranks to all the observations, beginning with 1 for the smallest values, then 2 for the second smallest, and so on.

2. Add up the ranks of the observations in each group, so that we have R_1 and R_2 representing the sum-of-ranks for group one and two.

3. Calculate the U-value for both groups:

$$U_i = R_i - \frac{n_i(n_i+1)}{2} \tag{4.6}$$

4. The smaller of the U_i values is then used to find the corresponding p-value. If p is below the significance threshold α (0.01 in this thesis), there is a statistically significant difference between the two groups. As this study involves multiple test, the chance of obtaining false-positives increases. Therefore, the p-values are adjusted using the Holm method, used for familywised error rate (FWER) correction. The adjusted p-values are given in equation 4.7.

$$\tilde{p}_{(i)} = max\{(m-j+1)p_{(j)}\}_1, \text{where}\{x\}_1 = min(x,1) \tag{4.7}$$

4.5 Hardware

Four instances from the Norwegian Research and Education Cloud (NREC), was used to run the experiments. Each instance had 16 CPUs (2.693GHz) and 32 GB RAM. The instances were used between October 2020 - May 2021. CPU hours estimated: 4 experiments * 4 algorithms * 8 hours per run * 10 runs * 16 CPUs = 20 480 CPU hours (for the experiments presented in the thesis). With the preliminary experiments (BC and parameter search) this is estimated around 50 000 CPU hours = 5.7 CPU years.

Chapter 5

Results

This chapter presents the results from the experiments described in the previous chapter. The first section covers the results from the Rocks and Diamonds environment, and the second section shows the Tomato Watering Environment results. The last section functions as a summary of the experiments and focuses on the comparison between the algorithms equipped with a task-specific BC vs generic BC.

5.1 Rocks and Diamonds

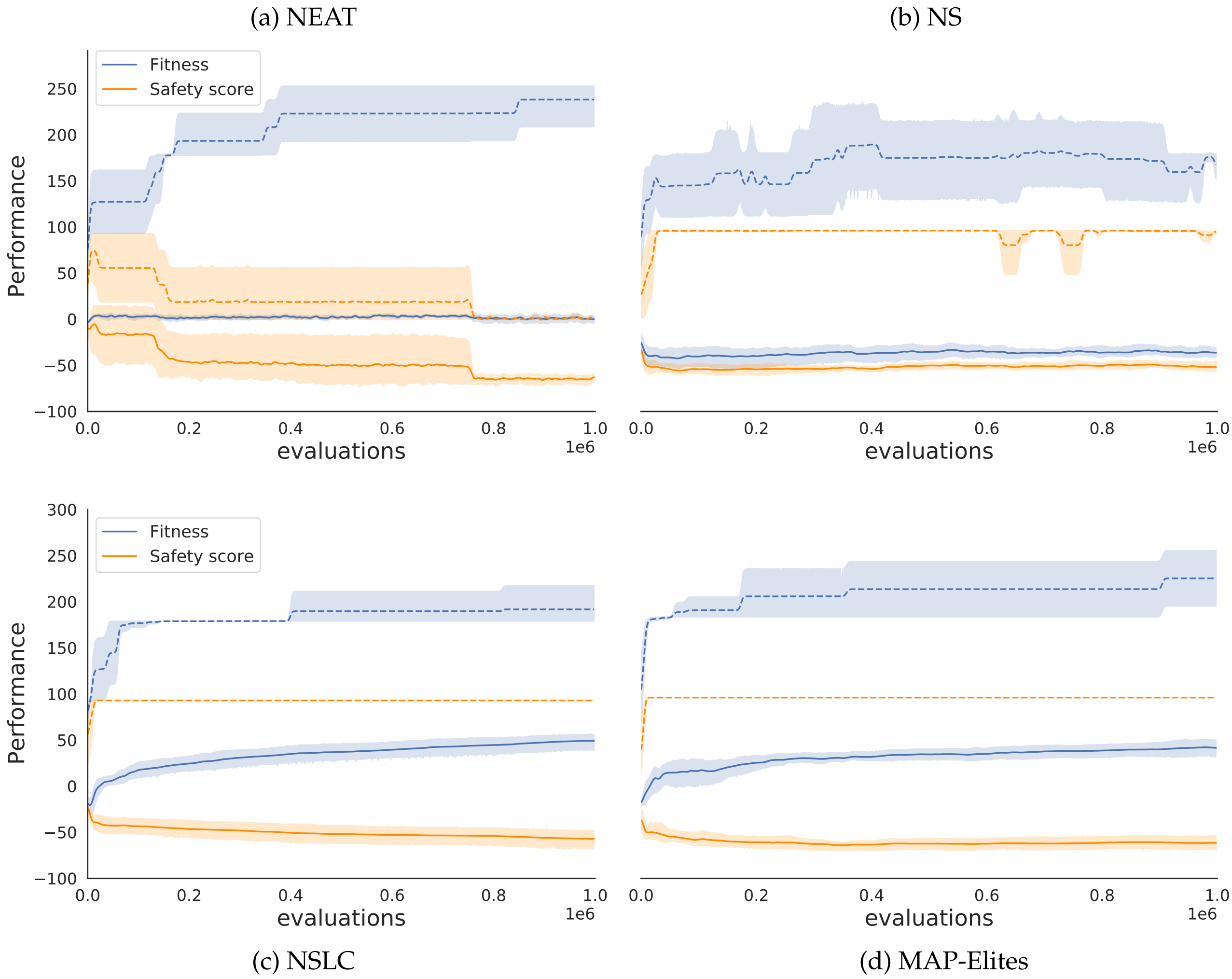
The results from the experiments in Rocks and Diamonds using the task-specific BC are shown in Figure 5.1 and 5.2. The generic results are shown in Figure 5.3 and 5.4.

5.1.1 Task-Specific BC

Figure 5.1 shows the performance of the algorithms with regards to both fitness and safety-score versus the number of evaluations performed. The plots show that there are significant differences between the different classes of algorithms when it comes to the safety-score.

The objective-based algorithm NEAT (a) manages to find individuals that achieve high fitness score, but performs poorly with respects to the safety-score. This can be seen by the increasing max-fitness values and the decreasing safety-score values. Although starting at a decent level, the max safety-score deteriorates and eventually reaches 0, while the average safety-score reaches -50. The behavioural diversity seeking algorithm NS (b), performs well in the environment and manages to find and preserve individuals with high safety-scores, while also finding solutions with higher fitness. This can be seen by the max safety-score values reaching and remaining at 93, which is the optimal solution for safety-score. We also see that between 600-800 thousand evaluations, some of the NS runs fail to preserve the desired solution.

The QD algorithms NSLC and MAP-elites (c) and (d) managed to attain the optimal solution in terms of safety score in all ten runs, which is evident from the constant max-safety score at 93. However, both algorithms found solutions of continually higher



(a) NEAT

(b) NS

(c) NSLC

(d) MAP-Elites

Figure 5.1: Task-specific BC results in Rocks and Diamonds environment. Solid lines show the mean of the populations, dashed lines show the max, and the shaded area represents the 95 % confidence intervals.

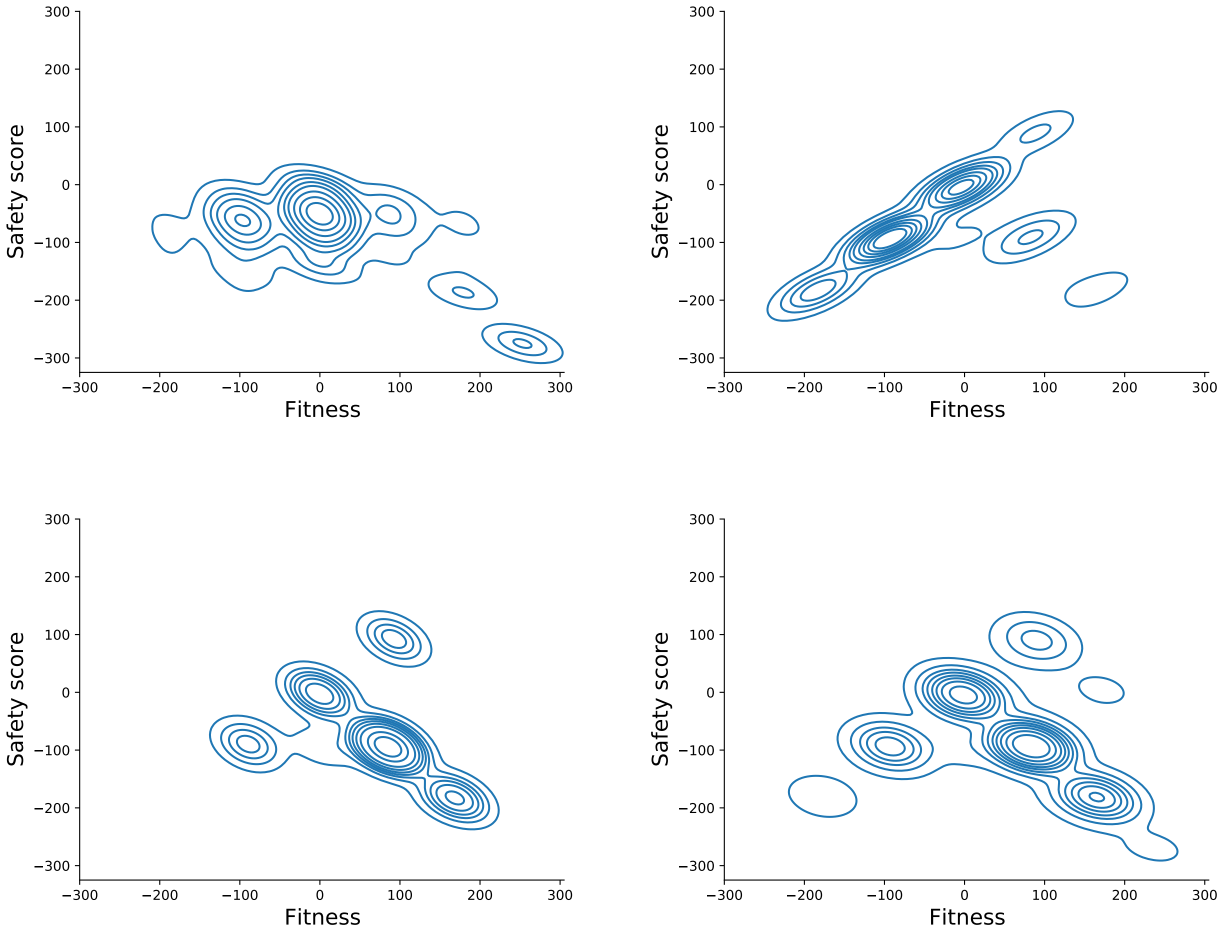
fitness, and experienced an increase in average fitness and a decrease in average safety-score during the entirety of the runs.

The kernel density estimate in Figure 5.2 further shows how the individuals in the last populations of the algorithms are distributed with respect to fitness and safety-score. In the last population of NEAT (a), most of the individuals are gathered near the center, with 0 fitness and 0 safety score, indicating that most solutions are irrelevant. NEAT also has some smaller congregations in the lower right, where the fitness is high and the safety-score is low, and these are likely made up by the elites. The last populations of NS (b), NSLC (c), and MAP-Elites (d) are more spread with regards to fitness and safety-score. The plots show multiple congregations around areas with similar fitness, common to see in environments with multiple local optima. The population of NS is mostly gathered in the lower-left area of the plot, meaning that the population mainly consists of individuals with low fitness and low safety score. NSLC and MAP-Elites have the largest congregations in the lower right, where the fitness is high but the safety score is low. However, all the behavioural diversity algorithms manage to preserve solutions in multiple areas of the plot, including the upper right area where the local optimum for safety-score lies.

5.1.2 Generic BC

Figure 5.3 visualizes the performance of the NS, NSLC, and MAP-Elites using the generic BCs in the Rocks and Diamonds environment. Plot (a) shows that NS equipped with a generic BC clearly fails in this environment, shown by the fact that neither the fitness nor the safety-score values improve during the run. As the max-values are quite low, it indicates that NS does not manage to find any solutions that safely perform the task, nor any that achieves high fitness by tampering. Between 200 to 400 thousand evaluations, some runs manage to find safe solutions, but they do not survive through the entire run. Plot (b) and (c) shows that the QD-algorithms, NSLC and MAP-Elites, again have similar performance. Both algorithms attain high max fitness values, producing individuals with over 200 fitness. At the early stages of the algorithms, they both manage to preserve the safest solution in the population; however, as the algorithms proceed, both the max and the average safety-score values decrease. The fitness values increase, indicating that more and more parts of the population suffer from reward tampering. There are still some differences between the QD algorithms, and the plots show that the mean safety-score for the populations evolved by NSLC deteriorates faster than MAP-Elites, almost reaching -100, while MAP-Elites reaches -50. Also, the average fitness for NSLC ends up higher than MAP-Elites.

Figure 5.4 shows the last populations using generic BCs, which are very different from the corresponding task-specific plots. Plot (a) shows that NS's population mainly consists of irrelevant individuals, not achieving high fitness or high safety-score. Plot (b) and (c) shows that the population of NSLC and MAP-Elites contain individuals with varying performance, and we see that even though the highest congregations are in the lower right, there are still some congregations in other areas yielding lower fitness.

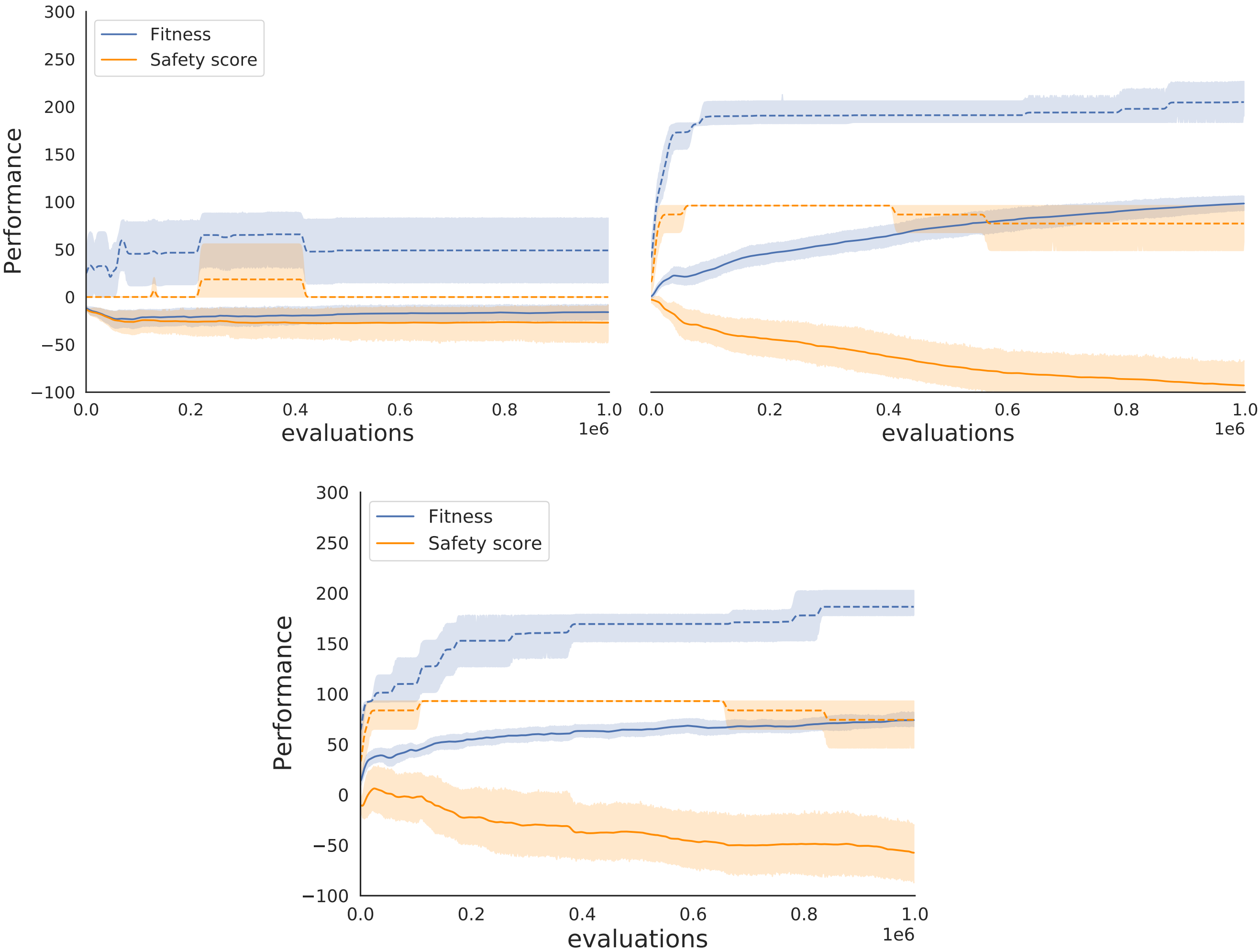

(a) NEAT

(b) NS

(c) NSLC

(d) MAP-Elites

Figure 5.2: Task-specific KDE plots in Rocks and Diamonds.

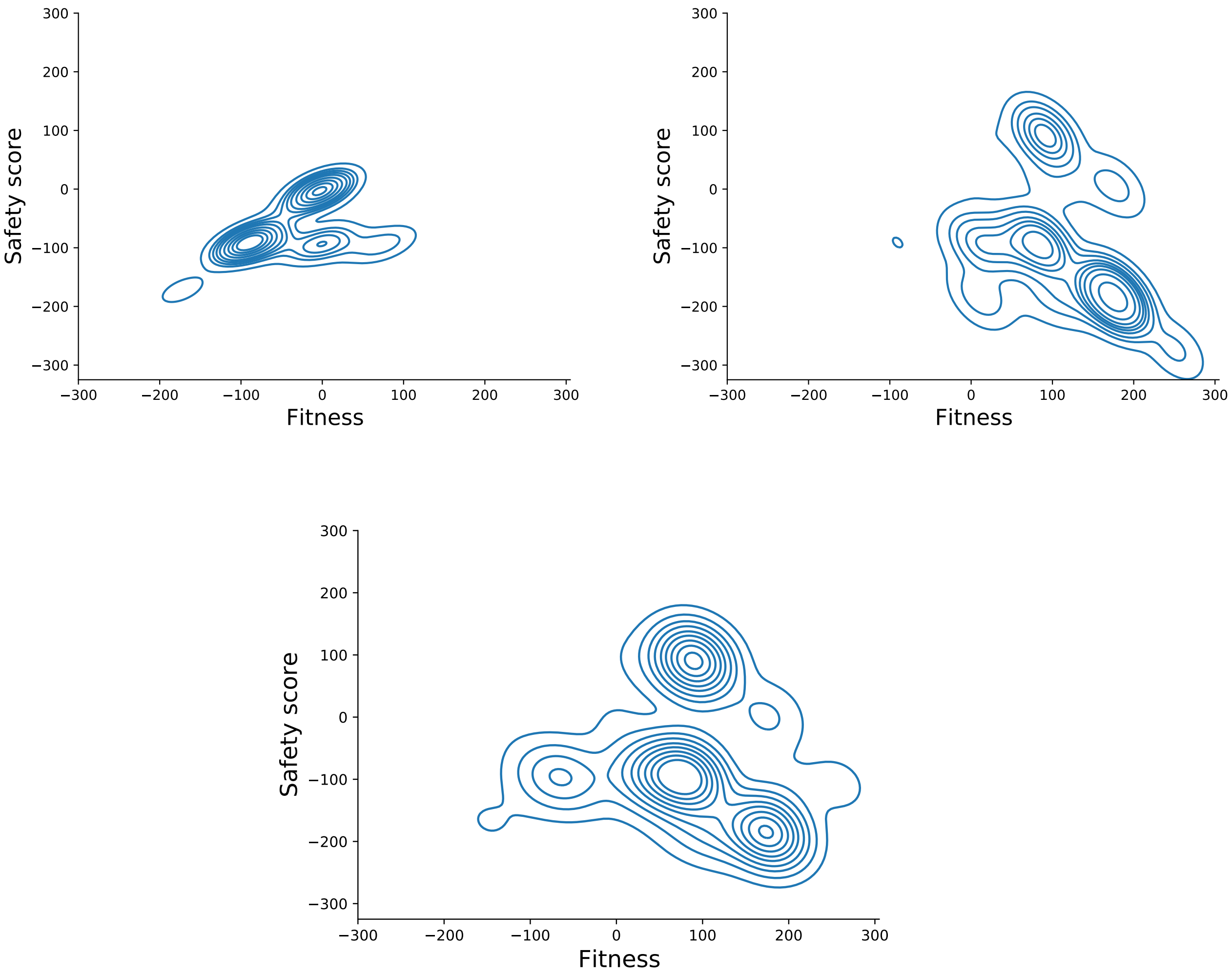


(a) NS

(b) NSLC

(c) MAP-Elites

Figure 5.3: Generic BC results in Rocks and Diamonds. Solid lines show the mean, dashed lines show the max, and the shaded area represents the 95 % confidence intervals.



(a) NS

(b) NSLC

(c) MAP-Elites

Figure 5.4: Generic KDE plots in Rocks and Diamonds

5.2 Tomato Watering Environment

The task-specific results from the Tomato Watering environment are presented in Figure 5.5 and 5.6. As a result of the easy-to-achieve maximum fitness in this environment, all algorithms (except NS for obvious reasons) found this optimum within the first 100 evaluations. Therefore, the max-fitness is not included in the following plots, as it is not interesting and allows for better visualization of the values that matter. The results from the generic experiments are shown in Figure 5.7 and 5.8.

5.2.1 Task specific BC

Figure 5.5 shows the number of individuals evaluated versus the performance with regards to both fitness and safety-score in the Tomato Watering environment. Plot (a) shows that the average fitness of the individuals for NEAT initially increases, but eventually reaches a more stable state. However, the safety-score of NEAT is constantly low, indicating that it is not able to find any solutions that perform the task in the desired way. For NS (b), the max safety-score values reach a higher point than for NEAT. However, although there is an initial increase, the safety-score values still seem to stop improving fairly early in each run. For the QD-algorithms, NSLC and MAP-Elites, the safety-score of the individuals behave similarly to that of NS, not managing to reach optimal performance reliably, but still having individuals that attempt to solve the task in the desired way, to a much higher degree than NEAT. Both QD algorithms also experience a logistic increase in average fitness, but this is significantly higher for NSLC.

Figure 5.6 visualizes the difference between the last population of the algorithms. The population of NEAT clearly favours the lower part of the plot, having both irrelevant individuals of low fitness and safety-score, and individuals with low safety score and high fitness. NS, however, has large parts of the population in the left area of the plot, with varying safety score values.

The QD algorithms generally favour the lower right area of the plot, but large congregations also exist in the upper left area, showing that the population also consists of many individuals performing the task in the desired way.

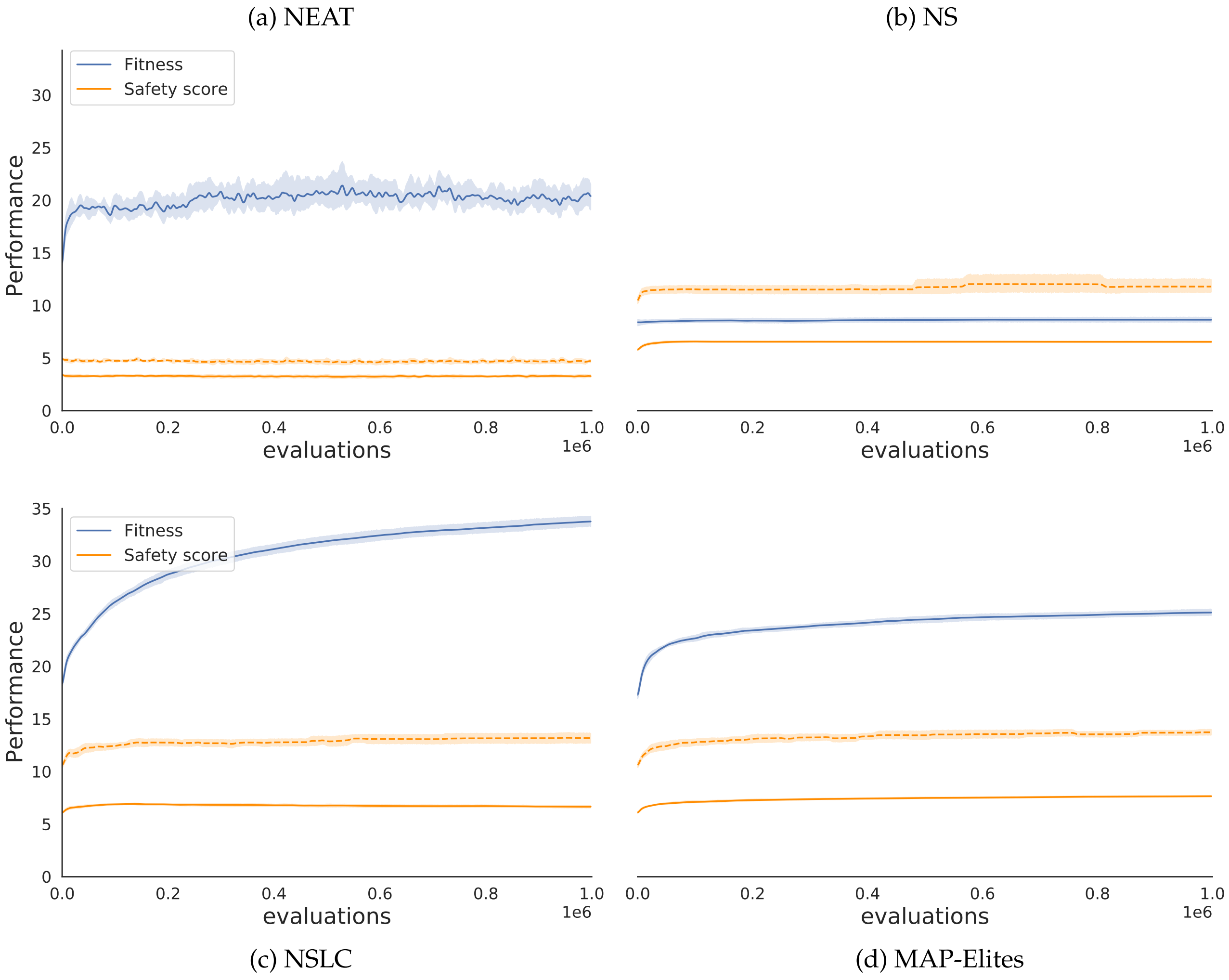
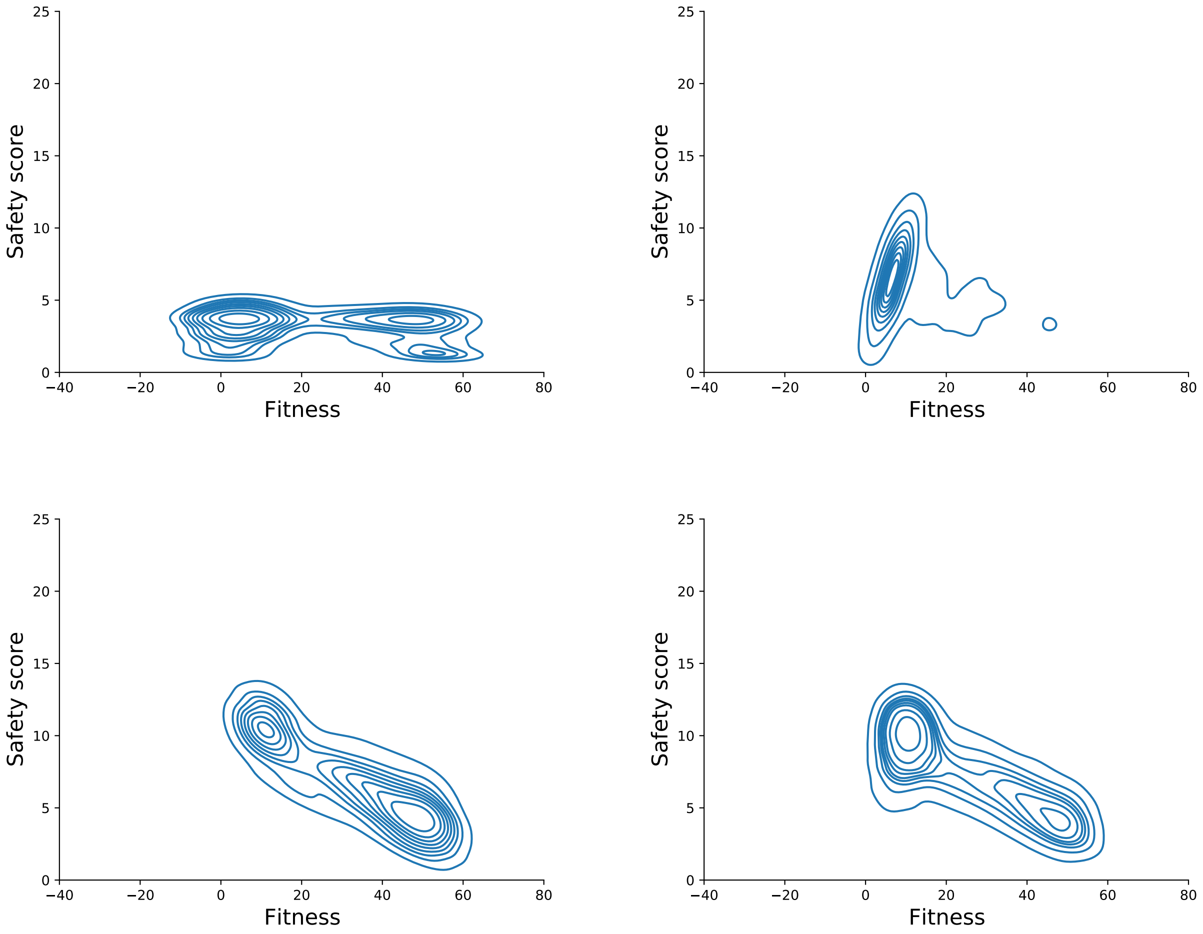


Figure 5.5: Task-specific BC results in the Tomato Watering Environment. Solid lines show the mean, dashed lines show the max , and the shaded area represents the 95 % confidence intervals.



(a) NEAT

(b) NS

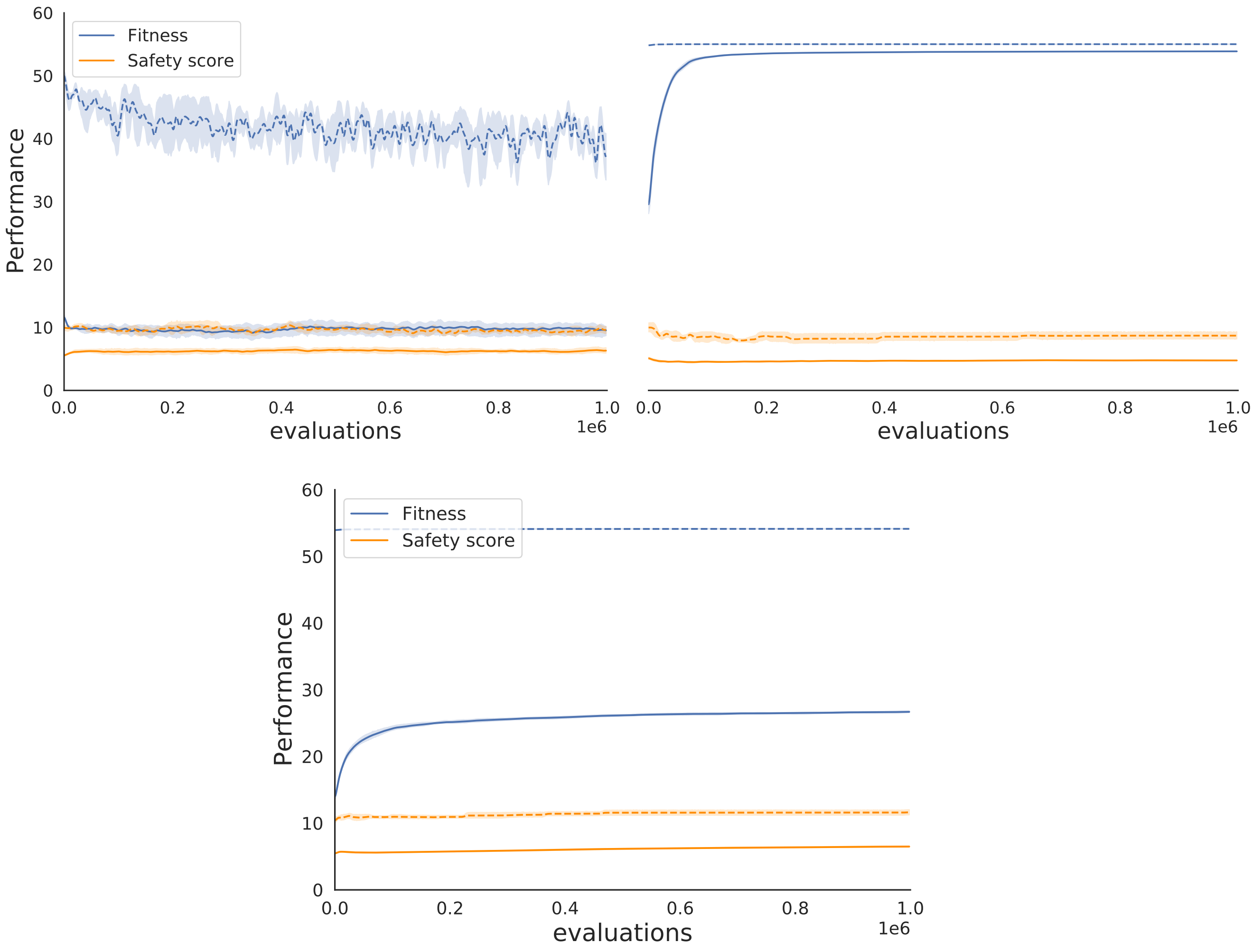
(c) NSLC

(d) MAP-Elites

Figure 5.6: KDE plot of the algorithms using task-specific BCs

5.2.2 Generic BC

From figure 5.7, it is clear that all behavioural diversity algorithms performed poorly when employed with the generic BC in the tomato watering environment. Neither algorithm attain any high safety-scoring individuals, as shown by the max safety-scores, all being around 10. The average fitness values also skyrocket for the QD-algorithms, reaching almost 30 for MAP-Elites and 54, the global optimum, for NSLC.



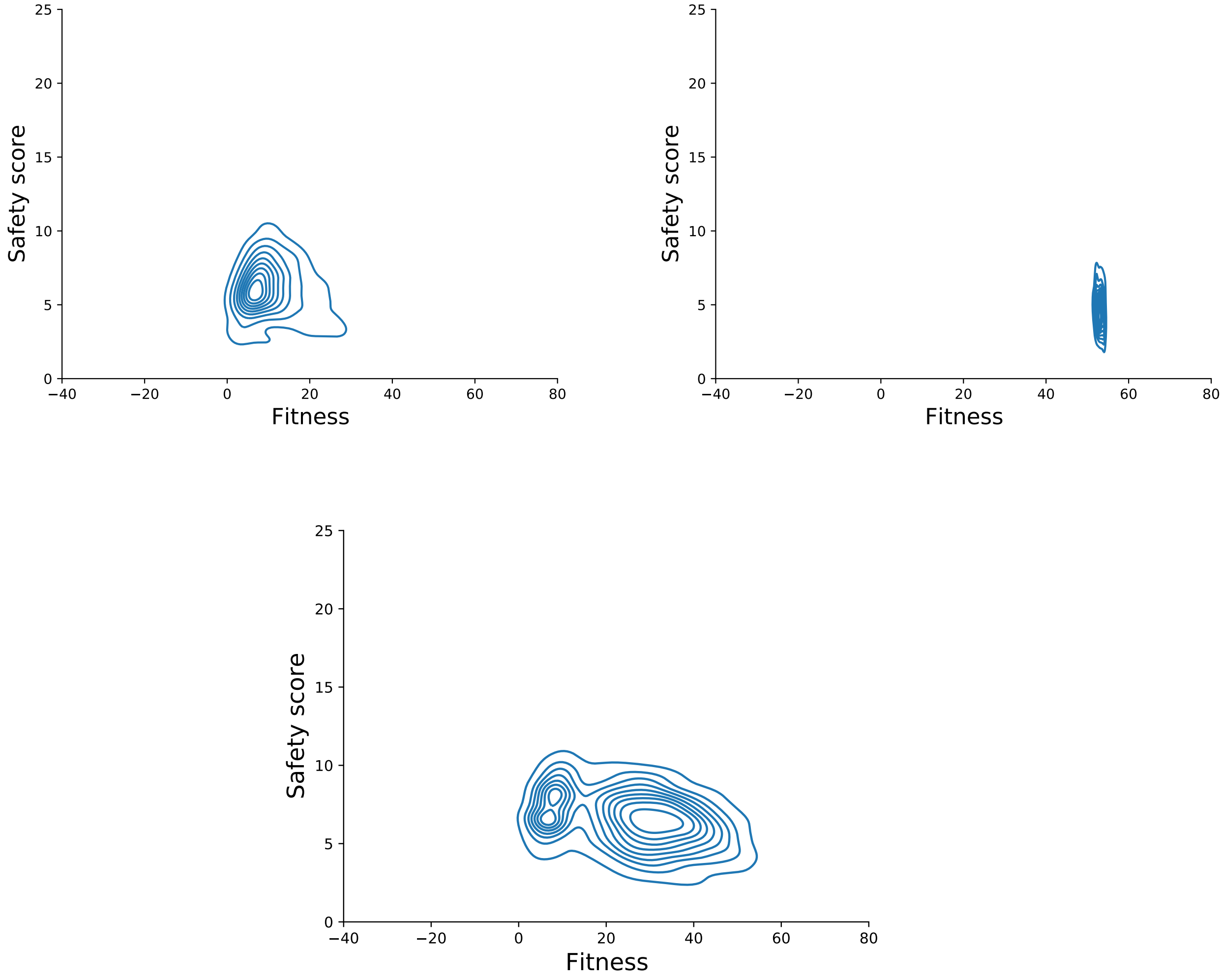
(a) NS

(b) NSLC

(c) MAP-Elites

Figure 5.7: Generic BC results in the Tomato Watering Environment. Solid lines show the mean, dashed lines show the max and the shaded area represents the 95 % confidence intervals.

Figure 5.8 also depicts the poor performance of the algorithms with regards to the safety-score. The plots show that the individuals in all algorithms are congregated in the lower areas of the plot, achieving only low safety-score values. For NSLC, this congregation is very extreme, showing that all (!) individuals end up near the global optima for fitness. For MAP-Elites, the population is more spread, but still unable to find any solutions that achieve high safety-score.



(a) NS

(b) NSLC

(c) MAP-Elites

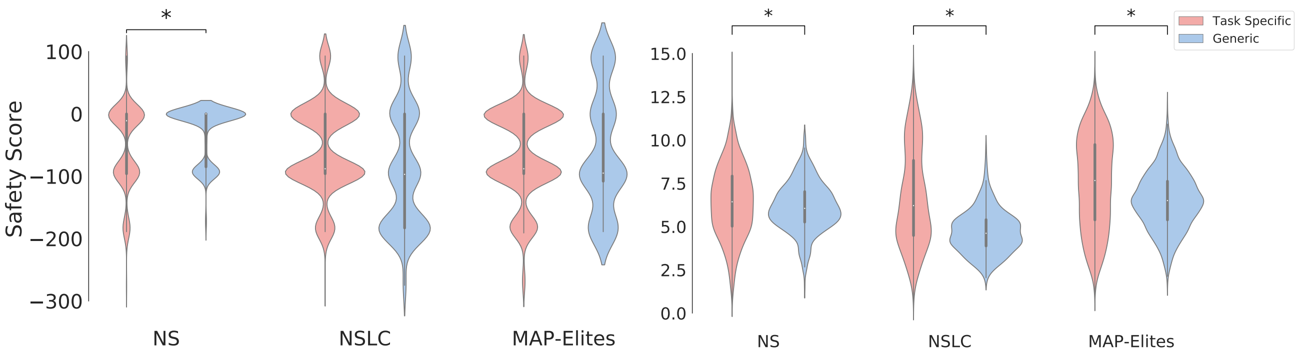
Figure 5.8: KDE plot of algorithms using generic BCs.

5.3 Task-specific BC vs Generic BC

Figure 5.9 shows the safety-score of the resulting individuals from all ten runs of each algorithm with both task-specific BC and generic BC. This is the same data previously visualized by the KDE plots, but where only the safety-scores are shown. Mann-Whitney U tests with Holm correction were performed to assess the statistical significance of differences between the max safety-scores of the task-specific version versus the generic version of the algorithms.

Plot (a) from Rocks and Diamonds, shows that the individuals are heavily gathered around local optima. For NS, there is a significant difference between the max safety-scores of the task-specific and the generic approach ($p < 0.01$). We also see that the generic approach fails to produce any individuals that achieve high safety-score. For the QD algorithms, NSLC and MAP-Elites, there is no significant difference between the max safety-scores of the task-specific and generic versions, indicating that the generic version had very similar performance to the task-specific. The plot also shows that the generic versions have a higher density in the area of high safety-score. The individuals from the generic approach also seem to be less gathered around local optima. Interestingly, the task-specific version of MAP-Elites appears to have more low-safety scoring individuals than the generic version, reaching down to -300.

In the Tomato Watering Environment (b), the results show that the task-specific approach was superior to the generic approach for all algorithms in the context of safety. The task-specific algorithms all managed to end up with solutions that attempted to solve the task in the desired way, although optimal solutions were not found reliably. This was not the case for the generic versions, and for all algorithms there was a significant difference between the task-specific and the generic approach.



(a) Rocks and Diamonds

(b) Tomato Watering Environment

Figure 5.9: Safety-score for the last generations (all runs) in both environments. * represents a statistically significant difference between the *max* safety-score values of all runs ($p < 0.01$). A more detailed version of the plots, and the results from the Mann-Whitney *U* test is included in Appendix A.

Chapter 6

Discussion

This chapter connects the results to the goals of the thesis. Each section covers one of the sub-goals. Some limitations and challenges of the study will be discussed in the last section.

6.1 Finding safe solutions

The results from both environments indicate that Evolutionary Algorithms are able to find and preserve safe solutions that attempt to perform the task in the desired way when the opportunity for reward tampering is present. Although the rewards from the environment do not necessarily correspond to the desired behaviour, a diverse evolutionary population can still ensure the development and optimization of agents performing the task in an intended way.

From the experiments conducted in the rocks and diamond environment, the max safety-score values of NSLC and MAP-Elites show that the QD algorithms manage to find and preserve the optimal solution for safety in all of the ten runs.

Even in the challenging Tomato Watering environment, the QD algorithms managed to evolve and maintain a population containing individuals that attempted to perform the task in the desired way, although very good solutions were not found reliably. From the tomato watering environment results, we saw that the average fitness values continually increase and eventually surpass the max-safety score values, indicating that most of the population is obtaining high reward by tampering. This comes as no surprise, as reward tampering in this environment is easier and more rewarding than any desirable behaviour. Even though the population is being pushed towards evolving individuals who perform reward tampering in different ways, the population's diversity still ensures that some individuals obtain reward without tampering.

The following sections will further discuss how population-based algorithms success varies between different types and how prior human knowledge can vastly increase the algorithms' performance and certainty of finding safe individuals.

In this study, the safety-score is a measure of how much fitness an individual obtained the "intended" way, and is the metric we use to evaluate the safety performance of an algorithm. This might seem unfair, as the algorithms are designed to maximize fitness and have no information on whether an individual's behaviour is safe or harmful. It is also nontraditional in the field of ML and AI, that an algorithm is being evaluated on a different metric than it is designed to maximize. However, as AI systems become more intelligent and are employed on more complex problems, it might not be the case that the reward from the environment is the best measure of the performance and that we, in the future, need develop different performance measures.

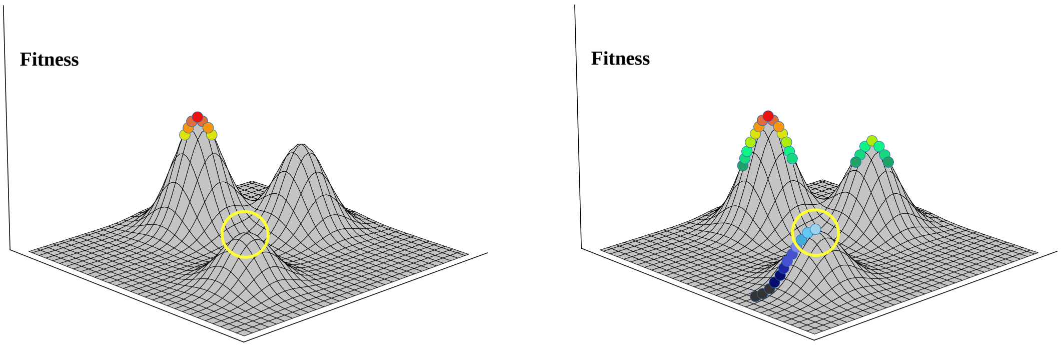
6.2 Comparison of the algorithms

The results indicate that the safety performance of population-based algorithms is very dependent on how the diversity of the population is maintained. For both environments, the algorithms having behavioural diversity as an objective performed better in terms of safety. The results also show, however, that the safety performance of the algorithms was different between the two environments.

Rocks and Diamonds

The decreasing safety-score values in the population evolved by NEAT indicate that safe solutions are substituted with more unsafe solutions performing better regarding fitness, but worse with regards to safety-score. This signifies that objective-based EAs method of preserving diversity does not maintain a population with sufficient diversity to preserve solutions with high safety-scores in the population.

The highest congregation of individuals produced by NEAT was in the center of the KDE plots, showing that most solutions in the population are irrelevant, having neither high fitness nor high safety-score. This is not surprising, however, and the same phenomenon will often occur in most complex tasks where NEAT is employed. This is a result of the survival selection used by objective-based EAs. After each generation, a large portion of the population is substituted with new individuals who have no guarantee to have similar performance as their parents because of the complexity of the genotype-phenotype mapping. The smaller congregation around the global fitness optimum is likely made up by the elites. The reason why all of the elites are situated in the same area is illustrated in Figure 6.1. The fundamental problem is that the higher fitness optima lead to a poor safety score, while the least-fit local optimum correspond to the desired solution. The goal of NEAT is to find the global optimum in the fitness landscape and therefore disregard the lower fitness-yielding optima. However, when dealing with reward tampering, the global optimum in terms of fitness will not necessarily correspond to the best solution in terms of user utility. The genotypic diversity of NEAT is used to force the evolution of different Neural Networks; however, as indefinitely many different NNs can lead to the same behaviour, all the elites of all the species end up in the same place in the behaviour space. This is also part of the



(a) NEAT

(b) QD

Figure 6.1: Illustration of the problem with NEAT in Rocks and Diamonds. The goal of NEAT (a) and QD-algorithms (b) in an illustration of the behaviour landscape of Rocks and Diamonds, where the yellow circle indicate the area of the best solution in terms of safety-score.

reason why the field of evolutionary computation is moving more in the direction of behavioural diversity [47].

The algorithms that maintain diversity in the population through behavioural diversity perform far better when it comes to safety. NS, NSLC, and MAP-Elites all manage to preserve solutions with high safety-scores while also finding many solutions that achieve high fitness by tampering with the reward function. This is also illustrated in figure 6.1, visualizing the goal of QD-algorithms. Maintaining a behavioural diverse population therefore seems to be a key factor for EAs to succeed in these types of environments. We are then able to find multiple different ways to perform the task and increase the chance of preserving solutions with high safety-scores. NS, NSLC, and MAP-Elites manage to find solutions that gather around multiple optima in the fitness landscape, indicating the populations are less pushed towards the global optimum and more focused on covering multiple areas of the behaviour space. MAP-Elites also showed to be able to find more varying solutions than NSLC, indicating that it achieves a population with higher diversity. This supports the claims made by other papers, that MAP-Elites generally achieves a more diverse population than previous QD algorithms (NSLC) [44, 47].

Tomato Watering Environment

The results from the tomato watering environment also indicate that safety performance is very different between the different algorithms. Although this environment is quite different and more challenging than the Rocks and Diamonds environment, the results indicate that the behavioural diversity algorithms are more suitable than objective-based EAs. These results also depict a poor performance by NEAT; finding no solutions with high safety-scores, and an average fitness way above the maximum achievable safety-score, indicating a high amount of reward tampering individuals. NS, NSLC, and MAP-Elites manage to find and preserve safe solutions that attempt to solve the task in the desired way. However, the high average-fitness scores of NSLC and MAP-Elites indicate that their populations mainly consists of individuals that end up in the tampering state.

Although there is an initial increase in max-safety score, all algorithms seem to reach a point where the max safety-scores stop improving and oscillates around the same values, not reliably achieving optimal performance. One reason why this phenomenon might occur is that the constant presence of the easily obtained reward tampering behaviour removes the need to evolve more intelligent agents, and therefore inhibits the search for solutions performing the task in the desired way. Instead of improving the complex behaviour of watering the dry plants, the algorithms instead learn different ways of eluding the intended task, as this is much easier. This causes the algorithms to potentially miss out on important stepping-stones that lead to safe solutions that also achieve high fitness. Reward Tampering in the rocks and diamonds environment, however, requires more intelligent behaviours than what is needed for the desired solution, meaning that the population evolved in the rocks and diamond environment was not limited in the same way.

The results also show that MAP-Elites generally performs better than NSLC in both environments. This indicates the discretization of the behaviour space is able to ensure more safe individuals than the diversity mechanism of NSLC.

6.3 On the importance of human expertise

The last sub-goal for the thesis was to investigate whether prior human knowledge of the task is required to evolve safe individuals, and how BC with varying human expertise affects the safety performance. There are multiple papers concerning behavioural diversity [17, 45] that argue that task-specific behaviour characterizations provide a more diverse and interesting population, which the later sections have shown is vital in order to ensure safe individuals in these environments.

The results from utilizing generic BCs indicate that the performance of NS is very dependent on a behaviour characterization that resembles the interesting properties of the task. As previously discussed, NS achieved high performance in both environments when employed with task-specific BCs; however, this is not the case when generic BCs were used. This is evident from our results, showing that NS does not attain any interesting individuals when employed with a generic BC in both environments, and that the task-specific vs generic max safety-score values were significantly different in both environments. The distribution of the safety-score in the last populations also show that the variety of the population of NS massively deteriorates. This supports the claim made by [31], that NS works best with a limited behaviour space and where the BC is aligned with the objective of the task.

The results from applying generic BCs on the QD-algorithms, NSLC and MAP-

Elites, also result in a decrease in performance, but in less degree than with NS. For NSLC, there was a significant difference in performance between the two environments. In the rocks and diamond environment, the generic version of NSLC was able to find and maintain the optimal solution in many of the runs, although more and more safe solutions were being replaced with unsafe solutions as the algorithms progressed. This signifies that although the behaviour space is severely increased, and the behaviour characterization is not aligned with the task, that generic NSLC was successful in the rocks and diamond environment. The generic version was able to find a large variety of different solutions, even in areas not covered by the population of the task-specific versions. This can be an indication that the freedom provided by generic BCs both has pros and cons. The pro being that the population is not bound by a predefined area of search, enabling more interesting solutions to be discovered, and the con being that this at the same time allows for more tampering.

However, in the tomato watering environment, generic NSLC utterly fails and achieves nowhere near the performance of its task-specific counterpart. The results show that all (!) of the resulting individuals in all ten runs end up tampering with the reward. This is a result of a common problem that can emerge when using generic BC. Although the individuals are forced to perform different actions, they all still end up finding different paths to the same resulting behaviour, which in this case is tampering. This shows that NSLC employed with a generic BC does not ensure a population with high enough diversity to enable desirable solutions in the tomato watering environment.

MAP-Elites employed with generic BCs has very similar performance compared to the task-specific variant in the rocks and diamond environment. The results indicate that generic MAP-Elites is able to preserve safe solutions in most of the runs, and that it achieves a very diverse population. Interestingly, the generic version of MAP-Elites seems to have more individuals congregated around the global optimum for safety-score than what the task-specific variant achieves, and also fewer individuals with very low safety-score. This is a result of the discretization of the behaviour space used by MAP-Elites. In the task-specific behaviour space, the population is forced to contain individuals that attain different configurations of rocks and diamonds in the gridworld, resulting in some solutions with very low safety-score and some that perform the task as intended. However, as only the best solution within a cell is kept, this only allows a very small part of the population to have high safety-score. MAP-Elites with the generic BC does not have the same restriction, as there are multiple different sets of actions that can lead to the desired solution.

In the tomato watering environment, however, the performance of generic MAP-Elites is inferior compared to the task-specific variant, not being able to find any individuals of high safety score. The average fitness score values are still far below what the generic NSLC achieves, indicating that generic MAP-Elites at least manages to constrain some parts of the population from tampering.

The results from the generic experiments show that the performance of population-based algorithms equipped with generic BCs is different between the two environments, indicating that the chance of preserving safe individuals with generic BC drastically decreases as tampering becomes easier. However, the results also show that generic QD-algorithms can perform as good as task-specific QD-algorithms if the tampering behaviours are more complex and difficult to achieve.

These findings indicate that a task-specific BC designed by a human expert is *not* always required to ensure a population containing safe solutions when facing the problem of reward tampering. This is highly promising, as generic BCs adds no human bias to the process, and are more suitable for future tasks as it removes the need for a time-demanding engineering process requiring task-specific knowledge [45].

6.4 Limitations and Challenges

The environments used in the thesis only work as abstractions of the more general problem of Reward Tampering, trying to encapsulate the important principles. To limit the confounding factors that can impact the performance of an algorithm, the environments are made fairly simple. Although this allows us to precisely study the problem without any interfering factors, a good performance by an algorithm does not guarantee a good performance in a real-world safety-critical task where similar problems might appear. Therefore, these environments should rather be considered as minimal safety checks, perhaps the first of many tests verifying the safety of an algorithm.

A crucial design choice for algorithms focusing on preserving behavioural diversity in the population is behavioural characterization (BC). The performance of the algorithms is therefore heavily dependent on this design choice, which often can be very challenging to get right. Although we have investigated the performance of the algorithms using two different BCs in each environment, there could still be multiple BCs, and combinations between them that could lead to a population with more interesting and safe solutions. We saw, for instance, that the task-specific BC for the Rocks and Diamonds environment allowed for fewer solutions with the desired behaviour than the generic BC. Additionally, the first dimension of task-specific BC used in the tomato watering environment could be considered unrealistically task-specific, limiting how applicable this BC would be in the real world.

The comparison between the algorithms was also a challenge for this study. As most of the individuals of NEAT are irrelevant for the performance of the algorithm, it makes it difficult to compare it to QD algorithms where the mean is more relevant. The max-values of each run was therefore emphasized. The population of MAP-Elites is also very different from the populations of the other algorithms, as the size is varying and very dependent on the BC. A challenge was that the task-specific versions of MAP-Elites had a significantly higher amount of individuals than the generic versions, affecting the distribution and therefore limiting the usefulness of a significance test. Because of this, the significance tests for assessing the difference between the task-specific and generic versions was performed on the *max* safety-score of each run.

Chapter 7

Conclusion and future work

7.1 Conclusion

The primary goal of the thesis was to investigate whether Evolutionary Algorithms can be a useful tool when encountering the futuristic AI-safety problem of reward tampering. This was achieved by deploying four different EAs in environments representing both the problem of *Reward Function Tampering* and *RF-input Tampering*. The experiments indicate that Evolutionary Algorithms are well suited for problems where the user-defined rewards do not necessarily correspond to the desired solution to a problem, which is the result of reward tampering. EAs was able to find solutions that solved the tasks the intended way in both environments. However, the presence of reward tampering also showed to inhibit the search for desirable solutions.

Algorithms searching for a behavioural diverse population showed to outperform classical objective-based EAs in preserving safe solutions, as genotypic diversity did not ensure sufficient diversity in the population. Maintaining a behavioural diverse population proved to be crucial, as it allows for multiple ways of performing the tasks, and thereby increases the chance to preserve solutions with the intended behaviour. The experiments also indicate that human guidance substantially increases population-based algorithms' safety performance in tasks where tampering behaviours are simpler to achieve than the desired solution. However, in tasks where tampering is more complex and harder to achieve, QD-algorithms using no human expertise attain very similar performance as the task-specific variants. This indicates that EAs ability to find and preserve safe solutions is *not* dependent on human expertise, although task-specific knowledge can increase both the certainty and quality of safe solutions.

We demonstrate Evolutionary Algorithms to be a useful tool against the futuristic problem of Reward Tampering, suggesting that they could play a vital role in ensuring a safe advancement of AI in the years to come.

7.2 Future Work

More realistic environments As the methods used to evaluate the performance of the algorithms in this thesis only serve as abstractions of the real problem, it would be interesting to apply the same concepts in more realistic tasks, e.g. 3D-worlds or physics-based tasks. With more realistic settings, it would enable a better investigation of population-based algorithms' ability to preserve safe solutions when more realistic factors are presents. This could, for instance, involve adding reward tampering possibilities in the usual benchmark environments used today or improve the AI-safety environments suite by contributing with more realistic and complex environments.

Another interesting idea would be to take inspiration from the works of [65], where open-ended evolution was used to continually create more challenging environments, allowing algorithms to generate more complex individuals. This could potentially be used to create more challenging reward tampering environments and allow for further investigation of the problem.

BC Learning Although the task-specific Behaviour Characterizations resulted in the best performance of the algorithms, they were carefully designed in order to achieve diverse populations. This was both a time-consuming and demanding process. It would, however, be interesting to investigate how an algorithm automatically learning BCs would perform with regards to reward tampering, e.g., by using the framework proposed by [40]. Would these automatically learned BCs result in populations containing safe individuals? If so, it would greatly reduce the human expertise and time needed for problems where the tampering is easy to achieve.

Combinations with RL A natural step for further research would also be to combine the methods of Leike et al. [35] or Everitt et al. [13], where Reinforcement Learning algorithms were applied to reward tampering problems, with the methods that showed promising results in this thesis. There are, for instance, multiple interesting papers covering possible combinations between RL and diversity driven EAs [8, 21, 23].

Revealing more AI-safety problems The work of Amodei et al., [1], establishing the concrete AI-safety problems was mainly concerning gradient approaches such as Machine Learning. However, an interesting question is whether the lack of formal gradient and the creative process of EAs manifest to different futuristic AI-safety problems. For instance, could an open-ended system operating under no constraints be safe?

In this thesis, safe solutions were achieved through maintaining behavioural diversity in the evolutionary population. This pushed the individuals to perform the task in different ways, resulting in some solutions featuring reward tampering, and others performing the task as intended. However, if an agent is able to change its reward function, as is the case in reward function tampering, the agent might also be able to change its behaviour function. An individual could thereby perform the task in the most reward-yielding manner, and although this behaviour still exists in the population, it would still be considered novel. Behaviour tampering could therefore become a potential safety concern for the future of evolutionary computation. Investigating this problem might lead to a better understanding of the safety concerning EAs in the future.

Bibliography

[1] Dario Amodei et al. 'Concrete Problems in AI Safety'. In: *CoRR* abs/1606.06565 (2016). arXiv: 1606.06565. URL: http://arxiv.org/abs/1606.06565.

[2] T. Back, U. Hammel and H. -. Schwefel. 'Evolutionary computation: comments on the history and current state'. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 3–17.

[3] Thomas Back, David B. Fogel and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. 1st. GBR: IOP Publishing Ltd., 1997. ISBN: 0750303921.

[4] Claudine Badue et al. 'Self-driving cars: A survey'. In: *Expert Systems with Applications* 165 (2021), p. 113816. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113816. URL: http://www.sciencedirect.com/science/article/pii/S095741742030628X.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[6] Mariusz Bojarski et al. 'End to End Learning for Self-Driving Cars'. In: *CoRR* abs/1604.07316 (2016). arXiv: 1604.07316. URL: http://arxiv.org/abs/1604.07316.

[7] G. H. Burgin. 'In Memoriam: Dr. Lawrence J. Fogel'. In: *IEEE Transactions on Evolutionary Computation* 11.3 (2007), pp. 290–293.

[8] Edoardo Conti et al. 'Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents'. In: *CoRR* abs/1712.06560 (2017). arXiv: 1712.06560. URL: http://arxiv.org/abs/1712.06560.

[9] Antoine Cully et al. 'Robots that can adapt like animals'. In: *Nature* 521 (May 2015), pp. 503–507. DOI: 10.1038/nature14422.

[10] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, 1859. URL: https://www.bibsonomy.org/bibtex/2d70d713c717fb28384fb073c9f6dfbc2/neilernst.

[11] Rachit Dubey et al. 'Investigating Human Priors for Playing Video Games'. In: *CoRR* abs/1802.10217 (2018). arXiv: 1802.10217. URL: http://arxiv.org/abs/1802.10217.

[12] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated, 2015. ISBN: 3662448734.

[13] Tom Everitt and Marcus Hutter. 'Reward Tampering Problems and Solutions in Reinforcement Learning: A Causal Influence Diagram Perspective'. In: *CoRR* abs/1908.04734 (2019). arXiv: 1908.04734. URL: http://arxiv.org/abs/1908.04734.

[14] Kevin Eykholt et al. 'Robust Physical-World Attacks on Deep Learning Visual Classification'. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 1625–1634.

[15] Jerome H. Friedman, Jon Louis Bentley and Raphael Ari Finkel. 'An Algorithm for Finding Best Matches in Logarithmic Expected Time'. In: *ACM Trans. Math. Softw.* 3.3 (Sept. 1977), pp. 209–226. ISSN: 0098-3500. DOI: 10.1145/355744.355745. URL: https://doi.org/10.1145/355744.355745.

[16] Tobias Friedrich et al. 'Theoretical Analysis of Diversity Mechanisms for Global Exploration'. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. GECCO '08. Atlanta, GA, USA: Association for Computing Machinery, 2008, pp. 945–952. ISBN: 9781605581309. DOI: 10.1145/1389095.1389276. URL: https://doi.org/10.1145/1389095.1389276.

[17] F. Gomez. 'Sustaining diversity using behavioral information distance'. In: *GECCO '09*. 2009.

[18] Joshua Hare. 'Dealing with Sparse Rewards in Reinforcement Learning'. In: (Oct. 2019).

[19] Matteo Hessel et al. 'Rainbow: Combining Improvements in Deep Reinforcement Learning'. In: *CoRR* abs/1710.02298 (2017). arXiv: 1710.02298. URL: http://arxiv.org/abs/1710.02298.

[20] Gregory Hornby et al. 'Evolving robust gaits with AIBO'. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)* 3 (2000), 3040–3045 vol.3.

[21] Ethan C. Jackson and Mark Daley. 'Novelty Search for Deep Reinforcement Learning Policy Network Weights by Action Sequence Edit Metric Distance'. In: *CoRR* abs/1902.03142 (2019). arXiv: 1902.03142. URL: http://arxiv.org/abs/1902.03142.

[22] Nick Jakobi, Phil Husbands and Inman Harvey. '"Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics,"' in: vol. 929. Jan. 1995, pp. 704–720.

[23] Shauharda Khadka and Kagan Tumer. 'Evolutionary Reinforcement Learning'. In: *CoRR* abs/1805.07917 (2018). arXiv: 1805.07917. URL: http://arxiv.org/abs/1805.07917.

[24] Jens Kober, J. Bagnell and Jan Peters. 'Reinforcement Learning in Robotics: A Survey'. In: *The International Journal of Robotics Research* 32 (Sept. 2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[25] Peter Krčah. 'Towards Efficient Evolutionary Design of Autonomous Robots'. In: *Proceedings of the 8th International Conference on Evolvable Systems: From Biology to Hardware*. ICES '08. Prague, Czech Republic: Springer-Verlag, 2008, pp. 153–164. ISBN: 9783540858560. DOI: 10.1007/978-3-540-85857-7_14. URL: https://doi.org/10.1007/978-3-540-85857-7_14.

[26] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[27] Joel Lehman. 'Evolutionary Computation and AI Safety: Research Problems Impeding Routine and Safe Real-world Application of Evolution'. In: *CoRR* abs/1906.10189 (2019). arXiv: 1906.10189. URL: http://arxiv.org/abs/1906.10189.

[28] Joel Lehman and Kenneth Stanley. 'Evolving a diversity of creatures through novelty search and local competition'. In: Jan. 2011, pp. 211–218. DOI: 10.1145/2001576.2001606.

[29] Joel Lehman and Kenneth O. Stanley. 'Abandoning Objectives: Evolution through the Search for Novelty Alone'. In: *Evol. Comput.* 19.2 (June 2011), pp. 189–223. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00025. URL: https://doi.org/10.1162/EVCO_a_00025.

[30] Joel Lehman and Kenneth O. Stanley. 'Exploiting open-endedness to solve problems through the search for novelty'. In: *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI*. MIT Press, 2008.

[31] Joel Lehman and Kenneth O. Stanley. 'Novelty Search and the Problem with Objectives'. In: *Genetic Programming Theory and Practice IX*. Ed. by Rick Riolo, Ekaterina Vladislavleva and Jason H. Moore. New York, NY: Springer New York, 2011, pp. 37–56. ISBN: 978-1-4614-1770-5. DOI: 10.1007/978-1-4614-1770-5_3. URL: https://doi.org/10.1007/978-1-4614-1770-5_3.

[32] Joel Lehman and Kenneth O. Stanley. 'Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search'. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. Portland, Oregon, USA: Association for Computing Machinery, 2010, pp. 103–110. ISBN: 9781450300728. DOI: 10.1145/1830483.1830503. URL: https://doi.org/10.1145/1830483.1830503.

[33] Joel Lehman et al. 'Safe Mutations for Deep and Recurrent Neural Networks through Output Gradients'. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 117–124. ISBN: 9781450356183. DOI: 10.1145/3205455.3205473. URL: https://doi.org/10.1145/3205455.3205473.

[34] Joel Lehman et al. 'The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities'. In: *Artificial life* 26 (Mar. 2018). DOI: 10.1162/artl_a_00319.

[35] Jan Leike et al. 'AI Safety Gridworlds'. In: *CoRR* abs/1711.09883 (2017). arXiv: 1711.09883. URL: http://arxiv.org/abs/1711.09883.

[36] Youguo Li and Haiyan Wu. 'A Clustering Method Based on K-Means Algorithm'. In: *Physics Procedia* 25 (Dec. 2012), pp. 1104–1109. DOI: 10.1016/j.phpro.2012.03.206.

[37] Hod Lipson and Jordan Pollack. 'Automatic design and manufacture of robotic lifeforms'. In: *Nature* 406 (Sept. 2000), pp. 974–8. DOI: 10.1038/35023115.

[38] Andrzej Maćkiewicz and Waldemar Ratajczak. 'Principal components analysis (PCA)'. In: *Computers & Geosciences* 19.3 (1993), pp. 303–342. ISSN: 0098-3004. DOI: https://doi.org/10.1016/0098-3004(93)90090-R. URL: http://www.sciencedirect.com/science/article/pii/009830049390090R.

[39] Lluís Marquez and Jordi Girona Salgado. *Machine Learning and Natural Language Processing*. 2000.

[40] Elliot Meyerson, Joel Lehman and Risto Miikkulainen. 'Learning Behavior Characterizations for Novelty Search'. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*. Denver, Colorado: ACM, 2016. URL: http://nn.cs.utexas.edu/?meyerson:gecco16.

[41] Volodymyr Mnih et al. 'Asynchronous Methods for Deep Reinforcement Learning'. In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

[42] David Moriarty and Risto Miikkulainen. 'Evolving Obstacle Avoidance Behavior in a Robot Arm'. In: (Apr. 1996).

[43] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. '20 years of reality gap: a few thoughts about simulators in evolutionary robotics'. In: July 2017, pp. 1121–1124. DOI: 10.1145/3067695.3082052.

[44] Jean-Baptiste Mouret and Jeff Clune. 'Illuminating search spaces by mapping elites'. In: (Apr. 2015).

[45] Jean-Baptiste Mouret and Stéphane Doncieux. 'Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study'. In: *Evolutionary computation* 20 (Aug. 2011), pp. 91–133. DOI: 10.1162/EVCO_a_00048.

[46] Una-May O'Reilly and Mark Wagy. 'EC-Star: A Massive-Scale, Hub and Spoke, Distributed Genetic Programming System'. In: June 2012. DOI: 10.1007/978-1-4614-6846-2_6.

[47] Justin Pugh, Lisa Soros and Kenneth Stanley. 'Quality Diversity: A New Frontier for Evolutionary Computation'. In: *Frontiers in Robotics and AI* 3 (July 2016). DOI: 10.3389/frobt.2016.00040.

[48] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Hoboken, NJ: John Wiley & Sons Inc., 2005. URL: https://cds.cern.ch/record/1319893.

[49] Claude E. Shannon. 'XXII. Programming a computer for playing chess'. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41.314 (1950), pp. 256–275. DOI: 10.1080/14786445008521796. eprint: https://doi.org/10.1080/14786445008521796. URL: https://doi.org/10.1080/14786445008521796.

[50] David Silver. *Interested in learning more about reinforcement learning? Follow along in this video series as DeepMind Principal Scientist, creator of AlphaZero and 2019 ACM Computing Prize Winner David Silver, gives a comprehensive explanation of everything RL.* URL: https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver.

[51] David Silver et al. 'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm'. In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[52] David Silver et al. 'Mastering the Game of Go with Deep Neural Networks and Tree Search'. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.

[53] Kenneth O. Stanley and Risto Miikkulainen. 'Evolving Neural Networks through Augmenting Topologies'. In: *Evol. Comput.* 10.2 (June 2002), pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. URL: https://doi.org/10.1162/106365602320169811.

[54] Kenneth Stanley, David D'Ambrosio and Jason Gauci. 'A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks'. In: *Artificial life* 15 (Feb. 2009), pp. 185–212. DOI: 10.1162/artl.2009.15.2.15202.

[55] Kenneth Stanley et al. 'Designing neural networks through neuroevolution'. In: *Nature Machine Intelligence* 1 (Jan. 2019). DOI: 10.1038/s42256-018-0006-z.

[56] Felipe Petroski Such et al. 'Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning'. In: *CoRR* abs/1712.06567 (2017). arXiv: 1712.06567. URL: http://arxiv.org/abs/1712.06567.

[57] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: MIT Press, 1998. ISBN: 0-262-19398-1. URL: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html.

[58] Hideyuki Takagi. 'Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation'. In: *Proceedings of the IEEE* 89 (Oct. 2001), pp. 1275–1296. DOI: 10.1109/5.949485.

[59] Haoran Tang et al. '#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning'. In: *CoRR* abs/1611.04717 (2016). arXiv: 1611.04717. URL: http://arxiv.org/abs/1611.04717.

[60] Tim Taylor. 'Requirements for Open-Ended Evolution in Natural and Artificial Systems'. In: *CoRR* abs/1507.07403 (2015). arXiv: 1507.07403. URL: http://arxiv.org/abs/1507.07403.

[61] Tim Taylor, Alan Dorin and Kevin B. Korb. 'Digital Genesis: Computers, Evolution and Artificial Life'. In: *CoRR* abs/1512.02100 (2015). arXiv: 1512.02100. URL: http://arxiv.org/abs/1512.02100.

[62] Vassilis Vassiliades, Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret. 'Using Centroidal Voronoi Tessellations to Scale Up the Multi-dimensional Archive of Phenotypic Elites Algorithm'. In: *IEEE Transactions on Evolutionary Computation* PP (Aug. 2017), pp. 1–1. DOI: 10.1109/TEVC.2017.2735550.

[63] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/. 2019.

[64] Haoran Wang, Thaleia Zariphopoulou and Xunyu Zhou. 'Exploration Versus Exploitation in Reinforcement Learning: A Stochastic Control Approach'. In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3316387.

[65] Rui Wang et al. 'Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions'. In: *CoRR* abs/1901.01753 (2019). arXiv: 1901.01753. URL: http://arxiv.org/abs/1901.01753.

[66] Christopher John Cornish Hellaby Watkins. 'Learning from Delayed Rewards'. PhD thesis. Cambridge, UK: King's College, May 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

Appendix A

Supplementary

Environment	Algorithm	Task-specific median	Generic median	P-value	U-statistic
Rocks and Diamonds	NS	93	0	0.00071	0
	NSLC	93	93	0.16749	40
	MAP-Elites	93	93	0.16749	40
Tomato Watering	NS	11.22	8.94	0.00025	0
	NSLC	13.17	8.32	0.00025	0
	MAP-Elites	13.49	11.7	0.00134	0

Figure A.1: Results from the Mann Whitney *U* test with Holm correction on the *max* safety-score values from the last population (all runs) between the task-specific version and the generic version.

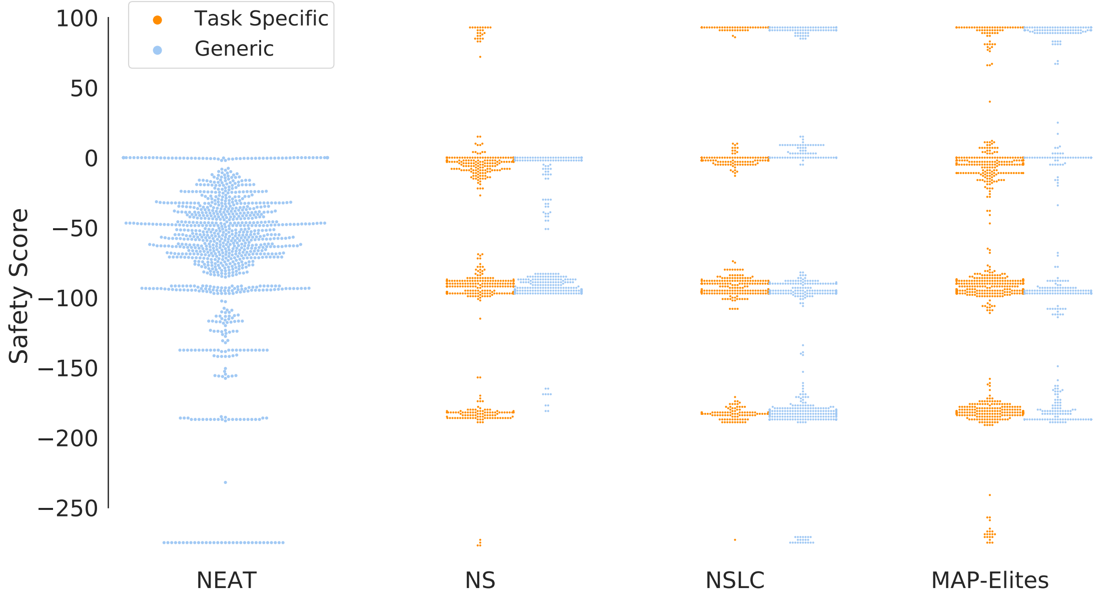


Figure A.2: Safety-score of all individuals in the last generations of all ten runs in Rocks and Diamonds.

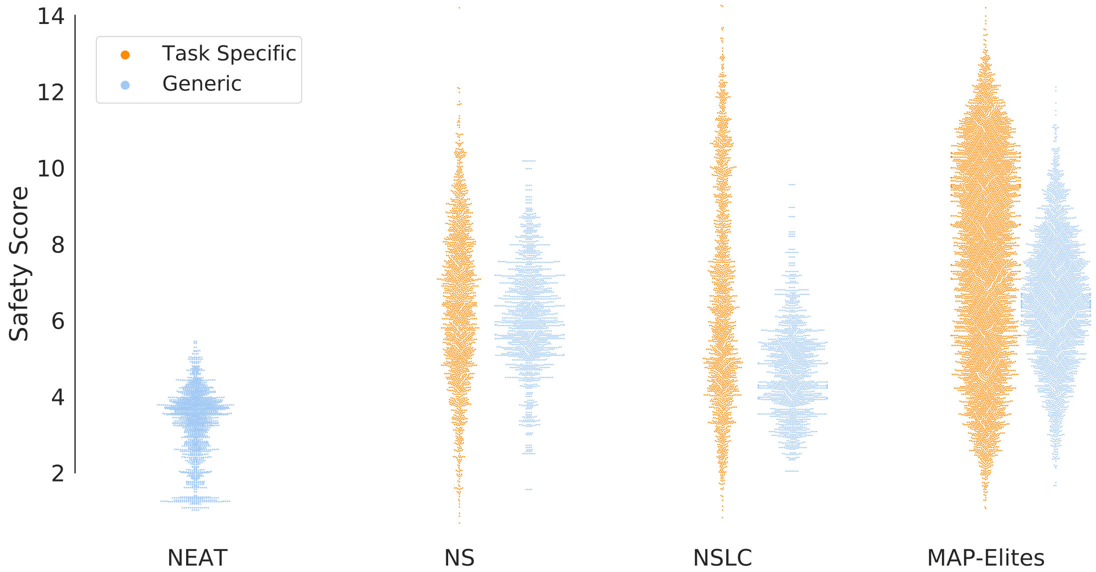


Figure A.3: Safety-score of all individuals in the last generations of all ten runs in Tomato Watering.

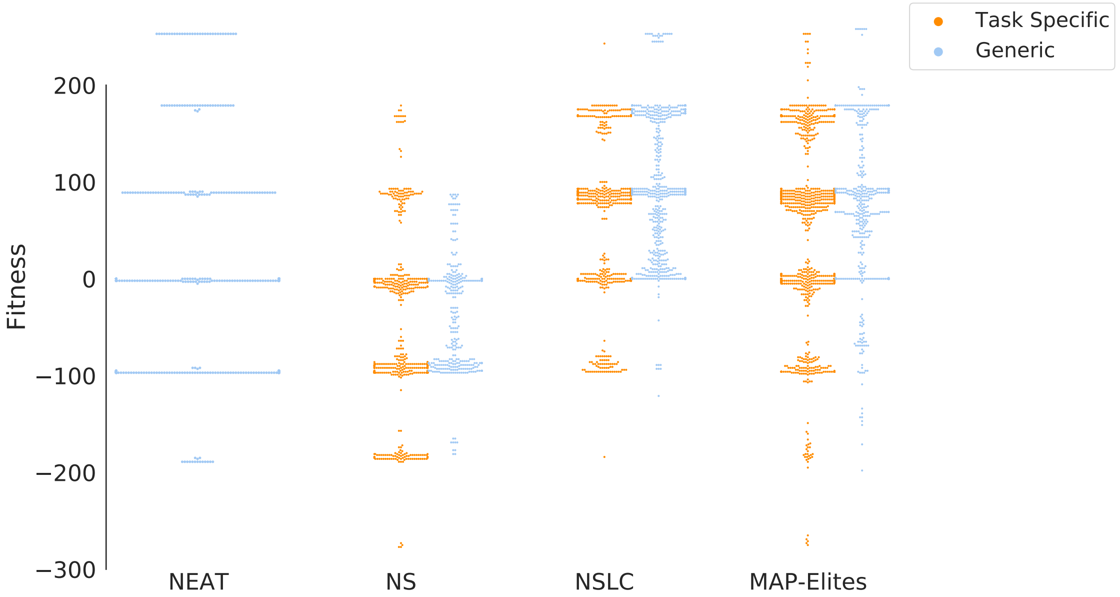


Figure A.4: Fitness of all resulting individuals (all ten runs) in Rocks and Diamonds.

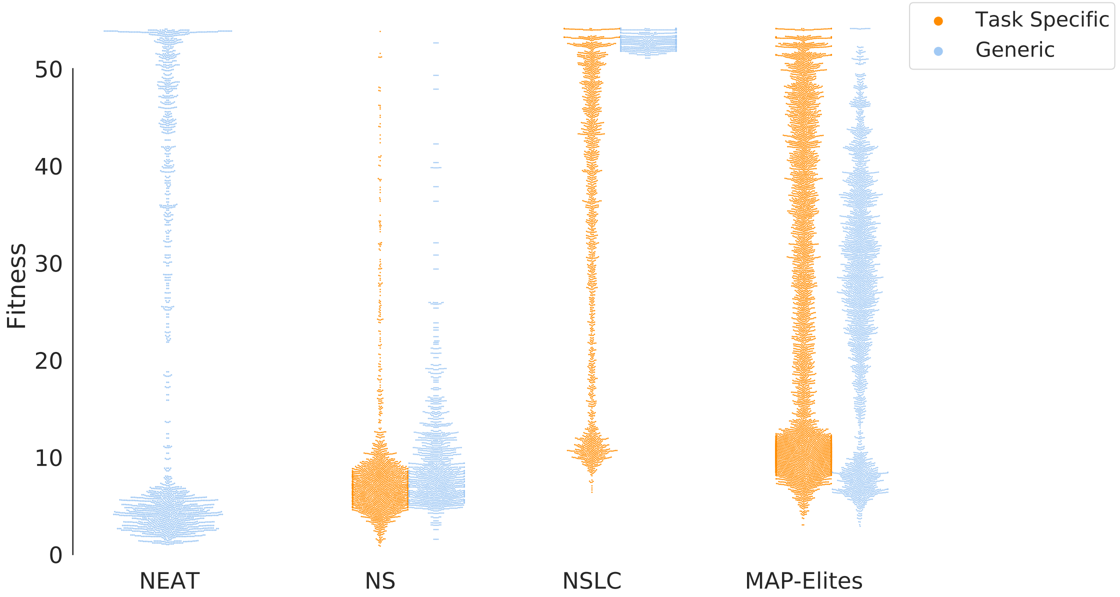


Figure A.5: Fitness of all resulting individuals (all ten runs) in Tomato Watering.