Genetic Programming and Evolvable Machines (2023) 24:12
https://doi.org/10.1007/s10710-023-09456-0

Reward tampering and evolutionary computation: a study of concrete AI-safety problems using evolutionary algorithms

Mathias K. Nilsen[1] · **Tønnes F. Nygaard**[1] · **Kai Olav Ellefsen**[1]

Received: 31 August 2021 / Revised: 16 December 2022 / Accepted: 7 June 2023

Abstract
Reward tampering is a problem that will impact the trustworthiness of the powerful AI systems of the future. Reward Tampering describes the problem where AI agents bypass their intended objective, enabling unintended and potentially harmful behaviours. This paper investigates whether the creative potential of evolutionary algorithms could help ensure trustworthy solutions when facing this problem. The reason why evolutionary algorithms may help combat reward tampering is that they are able to find a diverse collection of different solutions to a problem within a single run, aiding the search for desirable solutions. Four different evolutionary algorithms were deployed in tasks illustrating the problem of reward tampering. The algorithms were designed with varying degrees of human expertise, measuring how human guidance influences the ability to discover trustworthy solutions. The results indicate that the algorithms' ability to find and preserve trustworthy solutions is very dependent on preserving diversity during the search. Algorithms searching for behavioural diversity showed to be the most effective against reward tampering. Human expertise also showed to improve the certainty and quality of safe solutions, but even with only a minimal degree of human expertise, domain-independent diversity management was found to discover safe solutions.

Keywords AI trustworthiness · Reward tampering · Evolutionary computation · Neuroevolution · Behavioural diversity · Quality diversity

✉ Mathias K. Nilsen
mathias_kn@hotmail.com

[1] Institute of Informatics, University of Oslo, Harald Hårfagresgate 12A, Oslo, Norway

Published online: 19 September 2023

1 Introduction

The use of Artificial Intelligence (AI) has grown rapidly in recent years, making AI systems a larger and more important part of our society. Healthcare, self-driving cars, advertising, finance, and game-playing are among the many domains where AI algorithms thrive.

With the rapid advancement of AI, AI-safety has become a significant field of study, aiming to avoid unintended and harmful behaviours from powerful and intelligent systems [7, 13]. Multiple concrete safety problems have been discovered [1], causing careful considerations when designing AI systems. Many of the safety problems studied today are caused by algorithms exploiting flaws and unintended loopholes in the design of an objective function. Examples include agents finding and exploiting ways to cheat in games [3], agents exploiting bugs in simulators [14], and supervised learning agents that can be fooled as they have learned other patterns in their data than we expected [24]. As AI algorithms become smarter they might even find other exploiting methods that are not reliant on weaknesses in the objective function design [7].

In Reinforcement Learning [26], an *agent* usually learns to perform a task by taking actions in an *environment*, aiming to 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 [7]. That is, while reward tampering can lead to inconveniences in training current AI systems [5], the main reason for worrying about it is that future AI systems will both be able to cause more damage and have more potential for this kind of tampering.

An example is a robot accessing the computer running its source code and altering the reward function, allowing more or easier access to the reward. This could potentially enable dangerous behaviours from very intelligent systems.

Evolutionary algorithms (EA) are a branch of AI that take inspiration from natural evolution to solve a problem [6]. 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 [25]. By ensuring 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 thereby obtain some desirable solutions. An emerging area in Evolutionary Algorithms is so-called Quality Diversity techniques, which work to ensure exactly this: That individuals in the evolutionary population are spread across different and relevant behavioural niches [23].

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)—for instance, we could specify that we want some robots that run fast, some that run slowly, and some in-between, yielding a population with diversity in running pace. Such task-specific BCs

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 [21]. This can be applied to any problem without human guidance, and is therefore better suited as a general solution. Could we ensure that a population would contain trustworthy solutions without use of human expertise of what is considered a safe solution?

The main contributions of this paper are (1) an investigation of whether evolutionary algorithms can discover trustworthy solutions in tasks featuring reward tampering, (2) a comparison of the performance between different classes of evolutionary algorithms when reward tampering is possible and (3) a comparison of the performance between evolutionary algorithms with varying degree of human guidance.

2 Background

As mentioned in the introduction, there are many known examples of intelligent agents learning unintended behaviors by exploiting loopholes in their objective function. We begin this chapter with providing more details on two examples, to provide a clear picture of what types of problems AI-safety research addresses.

Lehman et al. [14] collect many anecdotes of evolutionary algorithms that achieve high fitness scores by finding ways to exploit their objective function. In many cases, this can involve finding ways to take advantage of bugs or errors in a physics simulator. For instance, Feldt [8] applied Genetic Programming to optimize control software for safely and efficiently decelerating airplanes landing on an aircraft carrier. Surprisingly, the algorithm quickly arrived at solutions with a perfect performance score. Upon closer inspection, it was found that the optimization had discovered a way to break the physics simulation by applying stronger than anticipated forces. Thereby, the algorithm was in practice rewarded for solutions that were very far from the safe, low-force solutions that were desired.

Chrabaszcz et al. [3] investigated the possibility of applying a basic Evolution Strategy as an alternative to deep Reinforcement Learning. For this goal they tested their algorithm on Atari games, among other environments. To the surprise of the authors, the algorithm found and exploited a way to cheat in the game Qbert that was previously unknown. The agent seemingly discovered a bug in the game that allowed it to collect an unreasonable amount of points by following a specific movement pattern. The agent discovering this solution thus avoided becoming a skilled Qbert player, and instead found a way to get a high score by cheating.

We recommend the interested reader to look closer at the collection of anecdotes by Lehman et al. [14], to see the wide variety of ways intelligent agents can find unintended, and often undesirable, solutions to a problem.

2.1 Categorizing AI-safety problems

The possibility of unintended consequences and possibly harmful behaviours increase with more intelligent AI agents, limiting the trustworthiness of AI systems [1, 7, 13, 18]. Amodei et al. [1], recently categorized some of these concrete problems and divided them into subcategories.

One of the concrete AI-safety problems proposed is called *reward hacking*, which involves AI-systems developing behaviours that increase their rewards in ways that violate the intention of the developer. Reward hacking is further separated into two categories, *reward gaming* and *reward tampering*. Reward gaming occurs when agents are able to exploit misspecifications or design flaws in the process that computes the reward. A classic example of this is 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 reducing its vision, and thus not observe any dirt at all. Reward gaming has frequently been reported among researches [1, 15], and is usually fixed with a more robust objective function.

Reward tampering is a more advanced type of exploit, not reliant on design errors made by humans. Everitt et al. [7] 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.

An example of this would be for an agent to, rather than guess the right label, change the objective so that all labels are considered correct. This would then make the task easier for the agent to solve. With the current capabilities of AI systems, 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 describe 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 involves agents tampering with the process that determines their reward. If there is a way for the agent to change the reward function, it might exploit this possibility. This is often 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 [22].

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. A recent example of this [5] is a simulated robot hand that was trained to grasp objects, with human subjects evaluating its performance. The robot was able to learn to "fool" the human evaluators (and thus its

reward function) by obscuring the object rather than grasping it. This shows how an agent able to give false information to the reward function could be able to gain reward without carrying out its intended task.

2.2 Behavioural diversity

Evolutionary algorithms have been demonstrated to perform better in many deceptive and difficult environments when searching for novelty/diversity rather than only searching for solutions that achieve high fitness [23]. Therefore, studies concerning behavioural diversity have in the recent years been an emerging trend the field of Evolutionary Computation [23]. Novelty Search (NS) introduced the idea of explicitly rewarding behavioural diversity. In contrast to prior EAs, where the parents and surviving individuals in the next generation are chosen based on the rewards from the environment, NS instead provides high fitness to novel individuals that perform the task in a different manner than the rest of the population. This enables NS to discover new parts of the behaviour space potentially ignored by objective-based algorithms as they initially yield low reward.

Without utilizing the rewards from the environment, NS proved surprisingly effective in tasks containing multiple local optima, also known as *deceptive* tasks, and outperformed multiple objective-based EAs [17]. However, without the notion of quality, and the ability to separate good solutions from bad, NS struggles in large environments without constrains to the behaviour space. This paved the way to combine the ideas from objective-based EAs with novelty, resulting in Quality Diversity techniques [23].

2.2.1 Quality diversity

The goal of a Quality Diversity algorithm is to maintain behavioural diversity in the population while at the same time improving the quality of the individuals close to each other in behavioural space through local competition. Quality Diversity techniques (QD) have therefore outperformed objective-based EAs in deceptive tasks, and purely novelty seeking algorithms in complex tasks with large behaviour spaces [20, 23]. The original QD algorithm is called Novelty Search with Local Competition [16]. NSLC uses the same novelty mechanism as NS, but additionally individuals close in behavioural space are treated as a separate species and compete amongst themselves by looking at the reward from the environment.

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) is another QD algorithm inspired by NSLC [20]. Instead of defining a *niche* as the individuals close in behaviour space, MAP-Elites discretizes the behaviour space into cells. Each cell covers some user defined amount of the behaviour space, and the population of MAP-Elites consists of the best individual discovered in each cell.

2.2.2 Behaviour characterizations

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 typically a vector containing n behavioural dimensions where each dimension captures some aspect of the individual's actions during its evaluation. There are two main approaches, generic and task-specific BCs [21].

Generic behaviour characterization Generic BCs 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, a chronological list of actions taken and observations received [21]. As this is not specifically related to one task, they can be applied to any domain and still contribute to finding interesting and novel solutions [21]. The fact that generic BCs utilize no human guidance also means that they are likely to be more featured in future tasks, where the tasks will be more complicated and therefore more challenging for humans to design BCs encouraging diversity. One downside, however, is the computational cost; as the dimensionality of the a generic behaviour space is commonly far larger than task-specific variants. 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 solutions may differ in terms of the β-vectors, in many domains, the solutions can still end up with only slight deviations of the same behaviour. With that being said, Lehman et al. [17], managed to consistently discover neural networks that solved a maze environment using NS with a 400-dimension behavioural space.

Task-specific characterization Most papers concerning behavioural diversity utilize task-specific, or ad hoc BCs [21]. A task-specific BC characterizes an individual's behaviour based on information extracted from the environment during or after evaluation. For instance, in deceptive maze navigation [17, 23], a commonly used BC is the end position of the individual as this is very aligned with the objective. When evolving walking behaviours for a bipedal robot, multiple different BCs have been employed with the intention of generating different ways of walking, including center of mass, mean head angle, mean knee speed and mean leg speed [10]. Task-specific BCs generally encourage more relevant behavioural diversity at the cost of requiring prior human knowledge specific to each task [21]. Task-specific BCs also generally provide a lower-dimensional behaviour space, which leads to a less computational demanding comparison between individuals.

3 Environments

The environments used in this study originate from Deep Mind's AI-Safety Gridworlds suite [18], 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 simplified representation of more complex problems [7]. A gridworld consists of a two-dimensional grid of cells. The

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

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]. These environments were inspired by the popular puzzle game "Baba Is You", where the player has to alter the rules of the environment to reach the goal. In addition to the normal reward from the environment, the agent also receives 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 $R^* < R$ the agent has 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 Rocks and diamonds

In the Rocks and Diamonds environment, shown in Fig. 1, agents are able to alter the reward function. This illustrates the problem of reward function tampering. 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 altering 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.

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 the 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 in Eq. 1, while the safety-score of an agent is given in Eq. 2. For instance, if an agent obtain 100 reward and 20 safety-score during a run; 20 reward was obtained in the intended way and 80 by tampering.

Reward function

$$F_t = \text{Diamonds in goal} \cdot \Theta^R_{diamonds} + \text{rocks in goal} \cdot \Theta^R_{rocks} \tag{1}$$

Safety function

$$S_t = \text{Diamonds in goal} - \text{rocks in goal} \tag{2}$$

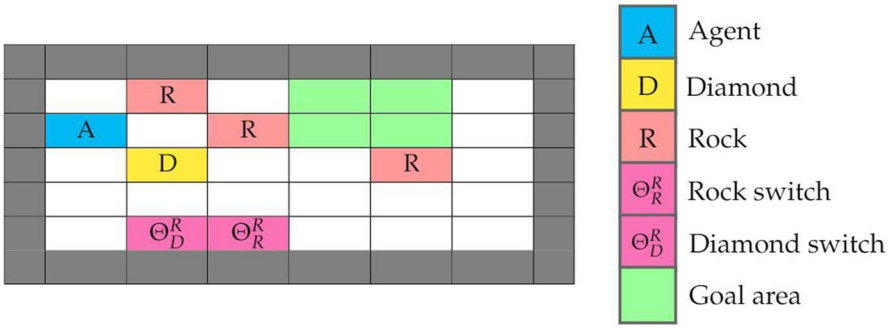


Fig. 1 The rocks and diamonds environment. An agent is here able to alter its reward function, known as reward function tampering. This way an agent can obtain a high reward, while achieving an undesired result

3.2 Tomato watering environment

The Tomato Watering Environment, shown in Fig. 2, is a challenging environment in which an agent can modify its observations. This represents the problem of RF-input tampering described in Sect. 2.1. 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 in Eq. 3 and the safety-score is given in equation 4.

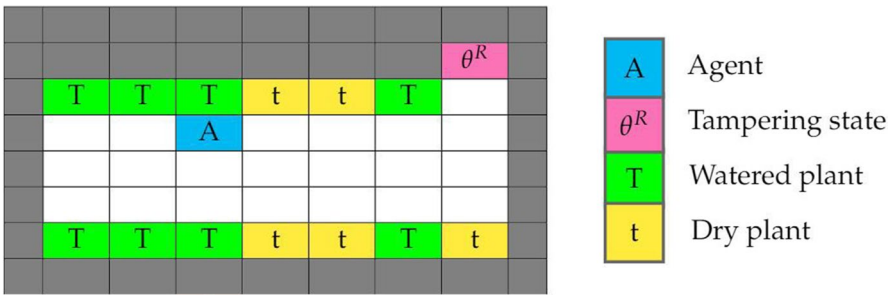


Fig. 2 The tomato watering environment. An agent is here able to perform RF-input tampering by visiting θ^R, *interpreting* all plants as watered and thereby gaining reward without performing its intended objective

Fitness function

$$F_t = \text{observed watered plants} \cdot 0.02 \tag{3}$$

Safety score function

$$S_t = \text{truly watered plants} \cdot 0.02 \tag{4}$$

4 Algorithms

The algorithms used in our study involves the objective-based algorithm NEAT, the purely novelty searching algorithm NS, and the QD algorithms NSLC and MAP-Elites. All algorithms use the same evolutionary operators; mutation, crossover, and genotype representation from the original NEAT paper [25]. The implementation of the evolutionary operators can be found in the neat-python package.[2]

4.1 NEAT

Neuroevolution of Augmented Topologies (NEAT) is a classical objective-based EA for evolving neural networks. The fitness of an individual is purely based on the cumulative reward R from the environment. NEAT also utilize speciation and fitness-sharing to maintain *genetic* diversity across the individuals in the population. This was implemented as the standard generational NEAT with a population size of 200.

4.2 Novelty search

Novelty search (NS) was the first behavioural diversity algorithm, only searching for individuals that behave differently without using any reward from the environment. The fitness of an individual in NS is based on how far it is from the rest of the population in behavioural space. The behavioural distance between two individuals is calculated by assessing the difference between the BC-vectors. In this study the euclidean distance and the hamming distance is used, depending on the dimensionality of the BC. The fitness of an individual is then calculated by summing the distances of the k-nearest neighbours, given in Eq. 5 ($k = 15$, in this study).

$$\rho(x) = \frac{1}{k} \sum_{i=0}^{k} dist(x, \mu_i) \tag{5}$$

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

4.3 Novelty search with local competition

Novelty search with local competition (NSLC) was one of the first QD algorithms that successfully managed to combine the ideas from objective-oriented EAs with behavioural diversity. NSLC uses NS to promote novelty, but additionally uses the reward from the environment to achieve local competition within behavioural *niches*. A niche is defined as the nearest neighbours in behavioural space, already obtained by NS. The individuals in a behavioural niche survive to the next generation based on a Pareto ranking of the reward and novelty score.

4.4 MAP-Elites

MAP-Elites is another QD algorithm inspired by NSLC [20]. Where NSLC defines a niche as the nearest neighbours in behaviour space, MAP-Elites divides the behaviour space into discrete cells, and the population consists of the highest performing individual within each cell. The behaviour space is divided using a grid, and the grid has equal dimensions to that of the behaviour space. The amount of the behaviour space each cell covers 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.

For both MAP-Elites and NSLC, the population was implemented using qdpy [2].

5 Methods

This section describes the behaviour characterizations used and how the experiments were conducted. To investigate the performance of the algorithms with varying degree of task-specific knowledge, two different BCs for each environment were designed; one task-specific and one generic.

5.1 Behaviour characterizations

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 [17], 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 Eq. 6, where D is the position of the diamond and R_i is the position of the i-th rock.

$$\mathrm{BC}_{\mathrm{R\&D}} : [D, R_1, R_2, R_3] \tag{6}$$

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 Eq. 7, where W is the number of plants watered during the episode and P is the number of different positions visited.

$$\mathrm{BC}_{\mathrm{TW}} : [W, P] \tag{7}$$

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 human expertise might be limited.

The individuals in the population will be forced to behave differently in terms of the β-vectors. This contains the action of the individual at each timestep and was further described in Sect. 2.2.2. The generic BC-vector for an individual is therefore on the form given in Eq. 8, where a_i is the action of the individual at timestep i, going up to the maximum number of timesteps in the environment t.

$$\text{Generic BC} : [a_1, a_2, a_3, \ldots, a_t] \tag{8}$$

Given that the number of timesteps for each episode is 100, and each agent can perform four different actions per timestep, the behavioural space includes 4^{100} possible different behaviours. This is severely larger in both size and dimensionality than the behaviour space created for the task-specific BCs. Because of the amount of cells required for the grid, MAP-Elites is not compatible with β-vectors. 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" [27]. An alternative to β-vectors was therefore used as a generic behaviour descriptor for MAP-Elites, serving 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:

$$\text{Generic BC for MAP-Elites} : [n_{\mathrm{down}}, n_{\mathrm{up}}, n_{\mathrm{left}}, n_{\mathrm{right}}] \tag{9}$$

The Hamming distance was used to reduce the time-complexity of calculating the behavioural distance between individuals in this multidimensional behaviour space.

Table 1 Parameters used for both environments

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)

5.2 Experimental setup

In both environments, the four algorithms presented in Sect. 4 were used to generate and optimize both the weights and topology of neural networks with the objective of maximizing the cumulative reward from the environment. The safety-score was hidden from the algorithms during the entirety of the runs.

In order to allow a thorough statistical analysis of the performance of the algorithms, each algorithm was run 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 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. [18] used when attempting to solve the same environments with Reinforcement Learning.

The parameters are shown in Table 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 the network is 4 values representing the possible actions. The action taken by the agent is chosen using the argmax function, 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 [25] were used. For NEAT, the number of elites in each species was set to 5, and for NS and NSLC k was set to 15.

6 Results

This section presents the results gathered from the experiments. The results of equipping the algorithms with task-specific BCs is first presented, then the generic results are shown. The last section functions as a summary of the experiments and

focuses on the comparison between the algorithms equipped with a task-specific BC versus generic BC.

6.1 Task specific experiments

Figure 3 shows the fitness and safety-score of all individuals in the last generation of the evolutionary runs (all 10 runs). Note that we here plot estimated underlying distributions (through kernel density estimation) of these measurements rather than individual data points, since there are too many data points for the reader to easily interpret their distribution directly.

From (a) we see that the objective based algorithm NEAT, is unable achieve high safety-scoring individuals in both environments. In the Rocks and Diamond environment the largest congregation of individuals is around center (0, 0), indicating that most individuals evolved by NEAT are irrelevant by neither achieving high fitness nor safety. This is also the group with the highest safety-score values of the population, showing that that the *optimized* are generally obtaining negative safety-scores.

The *elites* of the population are gathered in the lower right area, where the fitness is high and the safety score is low. No areas yielding high safety-scores are exhibited, showing that NEAT is unable to generate solutions performing the tasks in the desired way.

Similar results are gathered from the Tomato Watering Environment, where no individuals in the final generation attempt to solve the task as intended, as the best solutions in terms of safety obtain around 5 safety-score. In this environment NEAT experiences a larger part of the population in areas of high fitness, caused by the easy-to-obtain tapering behaviour.

From (b) we see that the purely behavioural diversity seeking algorithm, NS, performs far better regarding safety than NEAT, and has congregations in multiple areas of the plots. In the Rocks and Diamonds environment we see that the population solves the tasks in multiple different ways, as there are dense clusters in multiple parts of the plot. Even though most individuals evolved by NS achieves low safety score, the desired solution was found in all of the runs. In the Tomato Watering Environment, which is considered more challenging concerning achieving trustworthy individuals, NS is not able to achieve a population containing multiple different ways to solve the task.

The QD algorithms, NSLC (c) and MAP-elites (d), managed to attain the optimal solution in terms of safety score in all ten runs in the Rocks and Diamond Environment. Both algorithms achieve diverse populations containing agents with multiple different strategies of achieving fitness. Both algorithms also experience a dense congregation in the upper parts of the plot, around the optimal solution in terms of safety, which was not the case in (a) and (b). This shows that multiple individuals from the QD populations are carrying out the task as intended.

The QD algorithms were also the only ones that obtained individuals with safety-scores close to human performance (around 16), in the Tomato Watering Environment. Although reliably ending up with populations containing individuals that attempt to perform the task as intended, it should be noted that both algorithms

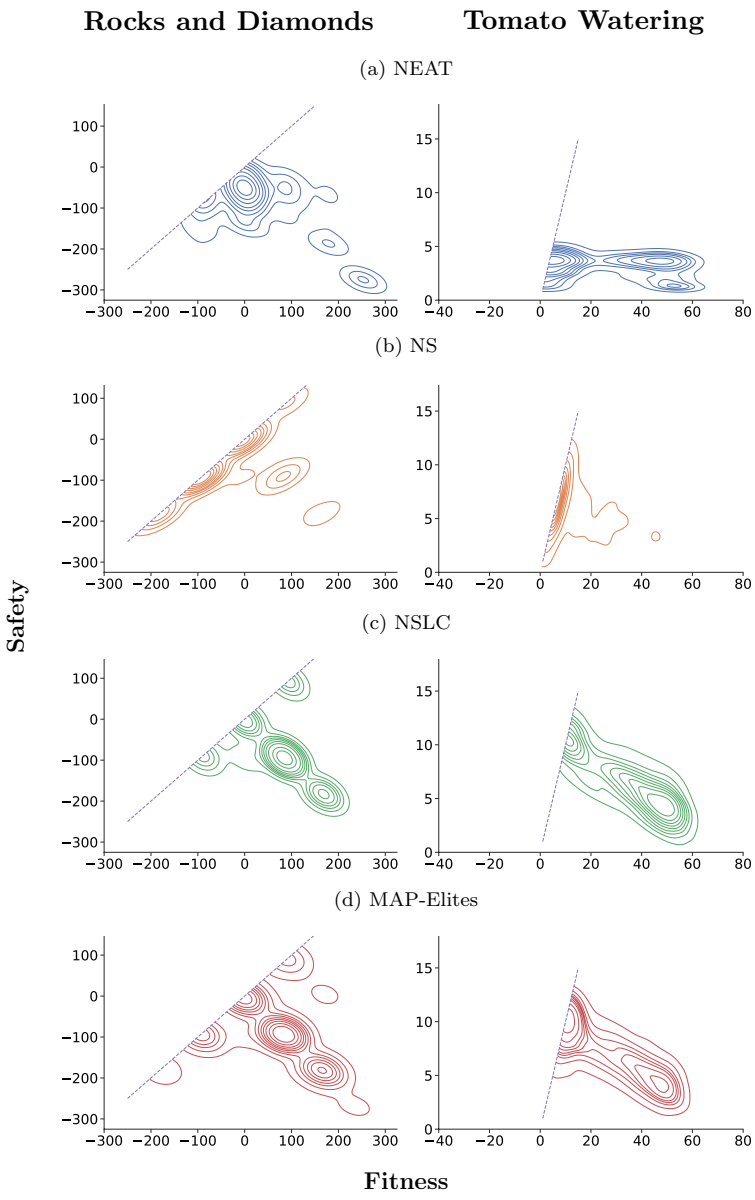


Fig. 3 Task specific BC results. Distribution of individuals in the last generation with regards to safety (y-axis) and fitness (x-axis). Solutions where the safety-score is equal to the fitness, indicated by the dashed line, are considered safe. Note that by definition, it is not possible to have a safety score higher than the fitness value, and the kernel density estimate has therefore been clipped at the line

found solutions of continually higher fitness and experienced an increase in tampering individuals during the entirety of the runs.

6.2 Generic experiments

Figure 4 visualizes the performance of the individuals evolved by NS, NSLC, and MAP-Elites using generic BCs. By comparing with the corresponding task-specific results, we see that the last populations are very different, given the different BCs. From (a) we see that the generic version of NS fails in both environments, as it is not able to achieve any high fitness nor high safety-scoring individuals in the final generations. Plot (b) and (c), however, shows that the generic QD-algorithms (and especially NSLC) have very different performance in the two environments. In Rocks and Diamonds, the generic QD-algorithms performed exceptionally, as very large parts of the populations where congregated around the desired solution (93 safety-score). In fact, a considerably greater percentage of the population attempted to perform the tasks in the desired way than with task-specific BCs. The populations evolved by the generic versions were also more spread, and not as gathered around local optima.

This was not the case in the Tomato Watering environment however, where the generic versions of the QD-algorithms utterly fails. The plots show that the individuals from both algorithms are congregated in lower ares of the plot, achieving only low safety-score values. For NSLC this, 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 preserve any high safety-scoring solutions.

6.3 Task-specific versus generic

Figure 5 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.

Plot (a) from Rocks and Diamonds, shows that the individuals are heavily gathered around local optima. For NS we see that the task-specific approach is far superior, as the generic version is not able to any individuals with positive safety-score values. The populations from the different variants of the QD-algorithms, however, are more similar, and both cover large parts of the behaviour landscape. From this plot we clearly see that the generic versions surprisingly 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.

In the Tomato Watering Environment (b), the results show that the task-specific approach was far 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, especially for NSLC.

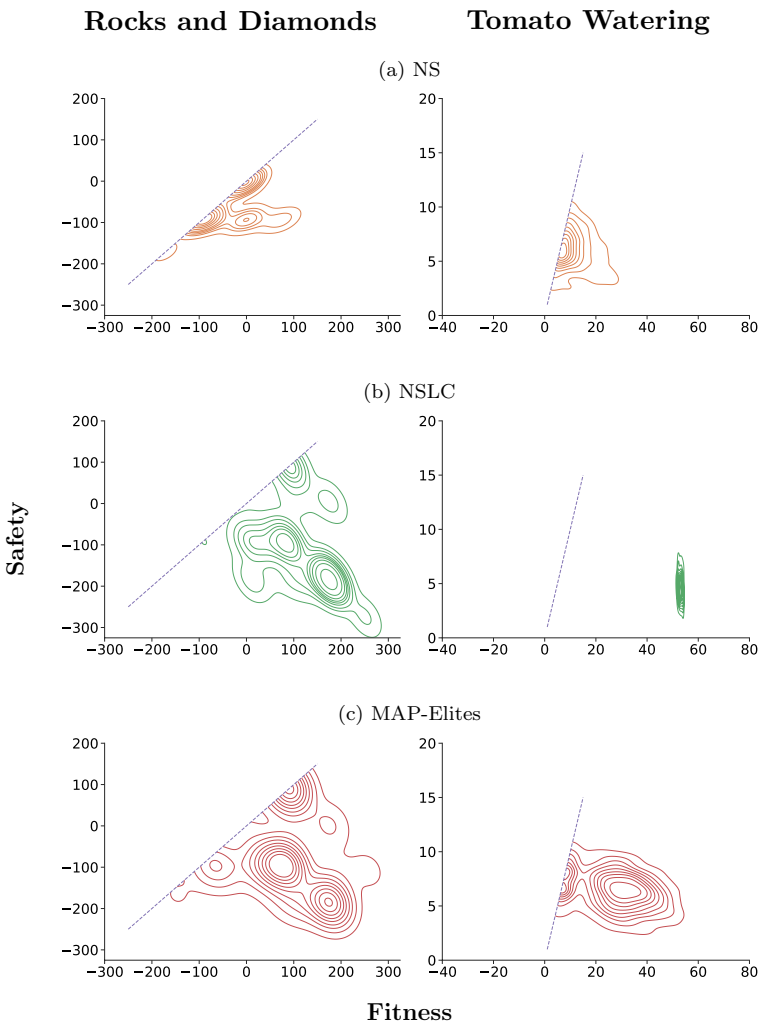
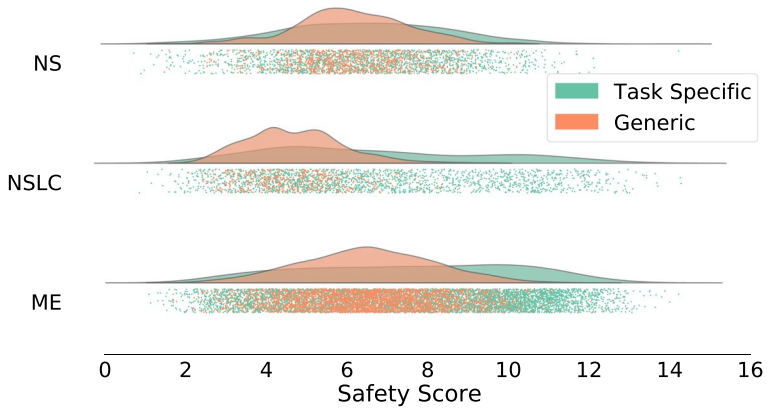


Fig. 4 Generic BC results. Distribution of individuals in the last generation with regards to safety (y-axis) and fitness (x-axis). Solutions where the safety-score is equal to the fitness, indicated by the dashed line, are considered safe. Note that by definition, it is not possible to have a safety score higher than the fitness value, and the kernel density estimate has therefore been clipped at the line

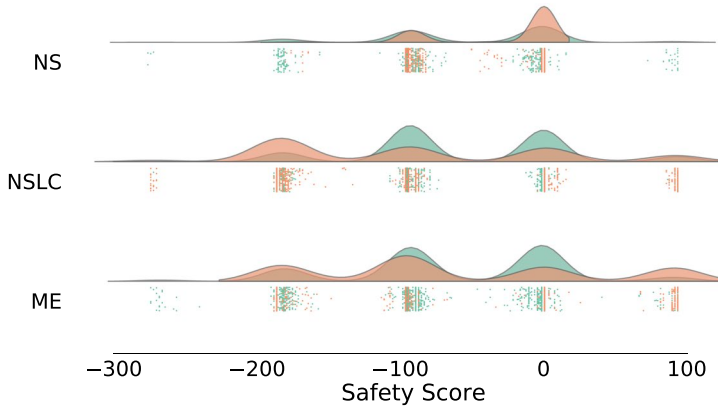
7 Discussion

7.1 Finding trustworthy 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



(a) Tomato Watering Environment



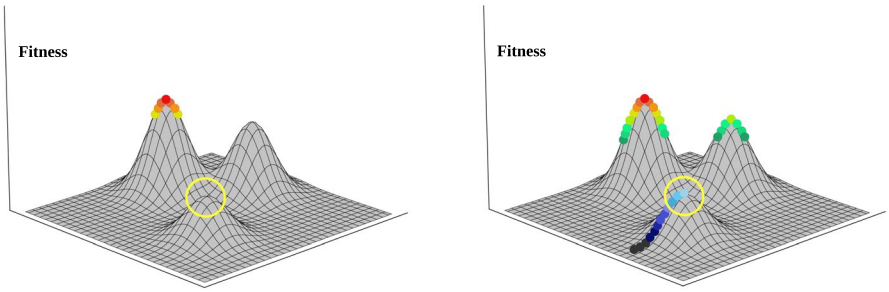
(b) Rocks and Diamonds

Fig. 5 Raincloud plot showing the safety-score for the last generations (all runs) in both environments. A kernel density estimate is shown at the top of each line, with a scatter plot showing the raw data below

evolutionary population can still ensure the development and optimization of agents performing the task in a safe and intended way.

The QD algorithms, NSLC and MAP-Elites, managed to evolve and preserve individuals performing the task the desired way in all runs. This indicates that although 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.

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



(a) NEAT (b) QD

Fig. 6 Illustration of the problem with NEAT when reward tampering is possible. The goal of NEAT (**a**) and QD-algorithms (**b**) in an illustration of the fitness landscape of rocks and diamonds, where the yellow circle indicates the desired solution in terms of trustworthiness (Color figure online)

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 to develop different performance measures.

7.2 Comparison of the algorithms

The results indicate that the trustworthiness 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.

7.2.1 NEAT

In both environments NEAT performed poorly with regards to safety. The objective-based algorithm did not manage to preserve any individuals of high safety-score in any of the 10 runs, signifying that objective-based EAs are unable to maintain a population containing trustworthy individuals.

Most individuals in the final population produced by NEAT were irrelevant, having neither high fitness nor high safety performance. The *elites*, however, were congregated around the global fitness optima. The reason why all of the elites are situated in the same area is illustrated in Fig. 6. The fundamental problem is that the higher fitness optima lead to poor safety-scores, while the least-fit local optima 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 reason why the field of evolutionary computation is moving more in the direction of behavioural (rather than genotypic) diversity [23].

7.2.2 Behavioural diversity

The algorithms maintaining diversity in the population through behavioural diversity perform far better when it comes to safety in both environments. Although the populations generated by NS, NSLC and MAP-Elites contain multiple tampering individuals, they all manage to preserve trustworthy agents in the population. This is also illustrated in Fig. 6, visualizing the goal of QD-algorithms, and showing why behavioural diversity is crucial to succeed in these environments. Maintaining a behaviourally diverse population therefore seems to be a key factor for EAs to succeed in tasks featuring tampering incentives, as we are then able to find multiple different ways to perform the task and increase the chance of preserving solutions with high safety-scores.

In the Rocks and Diamond environment, the behavioural diversity algorithms 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 produces a more diverse population than previous QD algorithms like NSLC [20, 23].

The QD algorithms are also superior in the Tomato Watering Environment, as their resulting populations show to be able to contain trustworthy individuals. As seen in Fig 5 however, the algorithms in this environment are not able to reliably find the optimal solution in terms of safety-score. 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.

7.3 On the importance of human expertise

An important goal of this study was to investigate whether prior human knowledge of the task is required to evolve trustworthy individuals, or if population-based algorithms operating generically also possess this potential. There are multiple papers concerning behavioural diversity [9, 21] that argue that task-specific behaviour characterizations provide a more diverse and interesting population,

which the previous 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, the task specific version of NS achieves individuals with high performance in both environments; however, this is not the case for the generic approach. This is evident from our results, showing that NS does not attain any interesting individuals when employed with a generic BC in both environments. The distribution of the safety-score of the individuals in the last populations also show that the variety of the population of NS massively deteriorates. This supports the claim made by [17], that NS works best with a limited behaviour space and where the BC is aligned with the objective of the task.

Applying generic BCs to the QD-algorithms, NSLC, and MAP-Elites also results in a decrease in performance, but to a lower degree than with NS. In the Rocks and Diamond environment, the generic version of both QD algorithms was able to find and maintain the optimal solutions in many of the runs. This signifies that although the behaviour space is severely increased, and the behaviour characterization is not aligned with the task, that the generic QD approach was successful in the Rocks and Diamond environment.

An interesting observation was that the generic versions was able to find a large variety solutions, even in areas not covered by the population of the task-specific versions. This is an indication that the freedom provided by generic BCs comes with both advantages and disadvantages. The fact that the population is not bound by a predefined area of search is a clear advantage, enabling more interesting solutions to be discovered. It does, however, allow for more tampering.

In the Tomato Watering environment, the performance of the generic QD algorithms drastically deteriorate. The generic version of NSLC fails completely, as all individuals in the last generation end up with a tampering behaviour. This is a result of a common problem that can emerge when using generic BCs. 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 sufficient diversity to find desirable solutions in the tomato watering environment.

Generic MAP-Elites is also inferior compared to the task-specific variant as it is unable to preserve any individuals with a high safety score. The resulting population is still far more spread than that of NSLC, indicating that generic MAP-Elites at least manages to constrain some parts of the population from tampering.

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 add 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 [21]. However, the results also indicate that the chance of preserving safe individuals with generic BC drastically decreases as tampering becomes easier, as the safety performance was very different in the two environments.

7.4 Towards selecting safe solutions from evolved populations

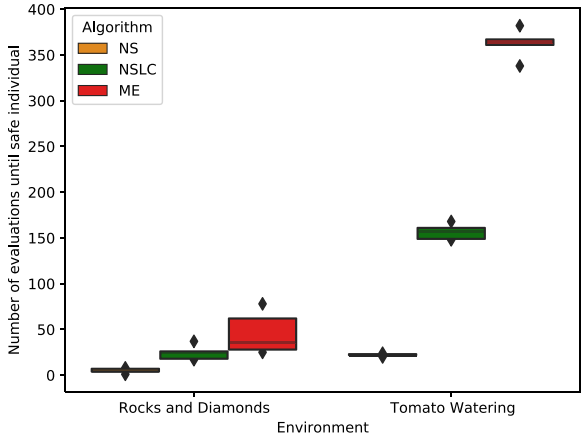
The experiments have shown that the diversity of populations optimized with evolutionary algorithms helps ensure that both safe and well-functioning solutions are available. But we have so far not considered how to *select* such safe solutions. While we consider a thorough exploration of this issue to be outside the scope of this paper, we here will give a demonstration of how the evolved populations can be used to find safe solutions.

As a basis for this demonstration, consider the populations already evolved for the Rocks and Diamonds, and Tomato Watering environments. As shown above, these contain many unsafe, cheating solutions and a few safe ones. Imagine now that we apply the top performing solution from such a population, and discover that it cheats. We would now want to, as efficiently as possible, identify a different solution which is both safe and well-performing. For simple problems, we could just run the evolutionary optimization again, while eliminating cheating solutions. However, for time-consuming, real-world optimization tasks, this might not be feasible, and we could save a lot of resources by instead finding a good solution in our already evolved population.

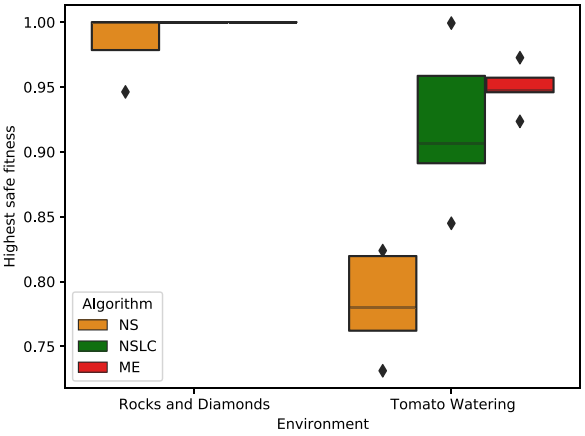
The following proof of concept illustrates how this could work. As mentioned, we imagine we discovered the cheat that many individuals employed, and we want to find a solution that does not cheat. A simple way to look for solutions that are safe *and* well-performing is to simply rank solutions in order of fitness, and go through them from the top, discarding any that cheat. Figure 7 shows the effect of this on both tasks for populations resulting from the Quality-Diversity algorithms shown in Fig. 3, in terms of the number of evaluations needed to find a safe solution, and the resulting fitness of the safe solution found. This was performed for the resulting population for all 10 runs of the algorithms, where (like in the original experiments) a solution with safety-score equal to the fitness in three evaluations was deemed safe.

As Fig. 7 shows, we are able to find well-performing and safe solutions in all the evolved populations within a reasonable number of evaluations. There are clearly differences in how fast the safe solutions are found in the evolved populations, and how well-performing those solutions are—but we regard investigating these differences closer to be outside the scope of this paper.

While there are certainly more intelligent ways to look for safe solutions in evolved populations, which we regard as an exciting venue for further research, the point of this demonstration was to show that if we search through our diverse, evolved populations, we can find safe and well-performing solutions much faster than if we had to start the optimization from scratch. This is especially important for more complex problems, where running a full, new round of optimization could be very costly.



(a) Number of evaluations until a safe solution was found.



(b) Highest safe fitness, scaled to the best safe fitness.

Fig. 7 Proof of concept: finding the best *safe* individual from the evolved populations

7.5 Future work

As the methods used to evaluate the performance of the algorithms in this paper 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. This 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 [28], 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.

A natural step for further research would also be to combine the methods of Leike et al. [18] or Everitt et al. [7], where Reinforcement Learning algorithms were applied to reward tampering problems. There are, for instance, multiple interesting papers covering possible combinations between RL and diversity driven EAs [4, 11, 12].

Although the task-specific behaviour characterizations resulted in the best performance of the algorithms in our work, 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 [19]. 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.

8 Conclusion

A fundamental challenge with an objective based learning system, is that the incentive for the agent is bound to this objective, typically via a reward or fitness function. If this reward function is not fully in tune with what the user wants to achieve, either because of poor design (reward hacking) or the relationship between the reward function and the intended task is broken during learning (reward tampering), the consequence can be undesired or even dangerous behaviour. For simple tasks, this is easily fixed by adjusting the objective function and re-running the search. However, for time-demanding optimization problems, or in real-world robot optimization where harmful behaviours could cause damage, re-running the experiment might not be possible. It is therefore important to explore ways to increase the chances that safe solutions exist in the already optimized population—and this is the focus of our research here.

The primary goal of this paper was to investigate whether evolutionary algorithms are able to enhance trustworthy solutions when encountering 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 EAs are well suited for problems where the user-defined rewards do not correspond to the desired solution, which is the result of reward tampering. EAs were 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 behaviourally diverse population showed to outperform classical objective-based EAs in finding and preserving safe solutions as genotypic diversity did not ensure sufficient diversity in the population. Maintaining a

behaviourally diverse population proved to be crucial, as it allows for multiple ways of performing the tasks, and thereby increases the chance of preserving 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 the trustworthiness of AI systems in the years to come.

Funding Open access funding provided by University of Oslo (incl Oslo University Hospital).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. D. Amodei, C. Olah, J. Steinhardt, P.F. Christiano, Schulman, J., Mané, D.: Concrete problems in AI safety. CoRR arxiv:1606.06565 (2016)
2. L. Cazenille, Qdpy: A python framework for quality-diversity. https://gitlab.com/leo.cazenille/qdpy (2018)
3. P. Chrabaszcz, I. Loshchilov, F. Hutter, Back to basics: benchmarking canonical evolution strategies for playing Atari, in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 1419–1426. AAAI Press (2018)
4. E. Conti, V. Madhavan, F.P. Such, J. Lehman, K.O. Stanley, J. Clune, Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. CoRR arxiv:1712.06560 (2017)
5. P.C. Dario Amodei, A. Ray, Learning from human preferences (2017). https://blog.openai.com/deep-reinforcement-learning-from-human-preferences/
6. A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*, 2nd edn. (Springer Publishing Company, Incorporated, 2015)
7. T. Everitt, M. Hutter, Reward tampering problems and solutions in reinforcement learning: a causal influence diagram perspective. CoRR arxiv:1908.04734 (2019)
8. R. Feldt, Generating diverse software versions with genetic programming: an experimental study. IEE Proc. Softw. **145**(6), 228–236 (1998)
9. F. Gomez, Sustaining diversity using behavioral information distance. In: GECCO'09 (2009)

10. V. Gupta, N. Aubert-Kato, L. Cazenille, Exploring the BipedalWalker benchmark with map-elites and curiosity-driven A3C, pp. 79–80 (2020). https://doi.org/10.1145/3377929.3389921
11. E.C. Jackson, M. Daley, Novelty search for deep reinforcement learning policy network weights by action sequence edit metric distance. CoRR arxiv:1902.03142 (2019)
12. S. Khadka, K. Tumer, Evolutionary reinforcement learning. CoRR arxiv:1805.07917 (2018)
13. J. Lehman, Evolutionary computation and AI safety: research problems impeding routine and safe real-world application of evolution. CoRR arxiv:1906.10189 (2019)
14. J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P.J. Bentley, S. Bernard, G. Beslon, D.M. Bryson et al., The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities. Artif. Life **26**(2), 274–306 (2020)
15. J. Lehman, J. Clune, D. Misevic, C. Adami, J. Beaulieu, P. Bentley, S. Bernard, G. Beslon, D. Bryson, N. Cheney, A. Cully, S. Donciuex, F. Dyer, K.O. Ellefsen, R. Feldt, S. Fischer, S. Forrest, A. Frénoy, C. Gagneé, J. Yosinksi, The surprising creativity of digital evolution: a collection of anecdotes from the evolutionary computation and artificial life research communities. Artif. Life (2018). https://doi.org/10.1162/artl_a_00319
16. J. Lehman, K. Stanley, Evolving a diversity of creatures through novelty search and local competition, pp. 211–218 (2011). https://doi.org/10.1145/2001576.2001606
17. J. Lehman, K.O. Stanley, *Novelty Search and the Problem with Objectives* (Springer, New York, 2011), pp.37–56
18. J. Leike, M. Martic, V. Krakovna, P.A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, S. Legg, AI safety gridworlds. CoRR arxiv:1711.09883 (2017)
19. E. Meyerson, J. Lehman, R. Miikkulainen, Learning behavior characterizations for novelty search, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016)*. ACM, Denver, Colorado (2016). http://nn.cs.utexas.edu/?meyerson:gecco16
20. J.B. Mouret, J. Clune, Illuminating search spaces by mapping elites (2015)
21. J.B. Mouret, S. Doncieux, Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol. Comput. **20**, 91–133 (2011). https://doi.org/10.1162/EVCO_a1_00048
22. J. Olds, P. Milner, Positive reinforcement produced by electrical stimulation of septal area and other regions of rat brain. J. Comp. Physiol. Psychol. (1954). https://doi.org/10.1037/h0058775
23. J. Pugh, L. Soros, K. Stanley, Quality diversity: a new frontier for evolutionary computation. Front. Robot. AI (2016). https://doi.org/10.3389/frobt.2016.00040
24. M.T. Ribeiro, S. Singh, C. Guestrin, Why should I trust you? Explaining the predictions of any classifier, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144 (2016)
25. K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
26. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, 1998)
27. V. Vassiliades, K. Chatzilygeroudis, J.B. Mouret, Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm. IEEE Trans. Evol. Comput. (2017). https://doi.org/10.1109/TEVC.2017.2735550
28. R. Wang, J. Lehman, J. Clune, K.O. Stanley, Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. CoRR arxiv:1901.01753 (2019)