Martin Wieczorek

Evolutionary Algorithm for Parameter Optimization of Context Steering Agents

---

FAKULTÄT FÜR INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

Evolutionary Algorithm for Parameter Optimization of Context Steering Agents

Master Thesis

Martin Wieczorek

June 16, 2020

Supervisor: Prof. Dr.-Ing. habil. Sanaz Mostaghim

Advisor: Dr.-Ing. Heiner Zille

Advisor: Dr.-Ing. Christoph Steup

Martin Wieczorek: *Evolutionary Algorithm for Parameter Optimization of Context Steering Agents*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2020.

Abstract

Movement is a crucial aspect for agents navigating in a simulation. Several steering approaches emerged over time. Traditional steering provides easy to implement but simple movement pattern. Swarm simulation has a high focus on the movement of the swarm as a whole. Context steering, in contrast, tries to find a good movement solution for individuals by considering all of its surroundings. Context steering was made modular and highly parameterizable so that it can cope with many different situations. The more modules are used to create a movement pattern, the harder it gets to parameterize them. In this work, context steering is combined with automatic parameter configuration to overcome this problem. Several parameter configuration approaches are investigated, and evolutionary algorithms are considered to be the most promising ones. Thus, an evolutionary algorithm is designed that can find parameter configurations for context steering agents, and the context steering approach is extended to be used by an evolutionary algorithm. Several experiments are designed to evaluate the found solution quality. Different aspects are integrated to enable the system to be user-guided. Furthermore, a robustness criterion was developed that turns out to be useful for agents that shall deal with a set of different scenarios.

Preface

I want to give my gratitude to everyone who supported me in writing this thesis. Actually, there are too many people involved to name everyone, but I will try nevertheless. Let's start with the people who offered a topic to me that was so fascinating that I could not resist working on it. Thanks to Martin Kirst and Franz Pieper. I want to thank Sanaz Mostaghim for giving me the opportunity to write this thesis at the chair of computational intelligence of the Otto von Guericke University Magdeburg. A special thanks to Heiner Zille for giving me direction and plenty of advice on writing a thesis, and to Christoph Steup for improving it even more. Thanks to my coworker at Polarith, especially Martin Zettwitz. Thanks to my family for supporting me all the time and believing in me. And of course, I want to thank my friends Max, Kay, Bernd, Max, Denny, Danny, Vivek, Ali, Calli, and everyone else I forgot for giving me fun and refreshing time whenever I was exhausted of writing.

Contents

List of Figures IX

List of Tables XI

1. Introduction and Motivation 1

1.1. Goals . . . 2

1.2. Structure . . . 2

2. Basics and Related Work 3

2.1. Movement . . . 3

2.1.1. Overview . . . 3

2.1.2. History of Context Steering . . . 4

2.1.3. Context Steering . . . 6

2.1.4. Multi-objective Decision Making . . . 9

2.1.5. State of the Art . . . 11

2.2. Evolutionary Algorithm . . . 13

2.2.1. Encoding . . . 13

2.2.2. Fitness . . . 15

2.2.3. Selection for Reproduction . . . 15

2.2.4. Crossover . . . 16

2.2.5. Mutation . . . 17

2.2.6. Environmental Selection . . . 18

2.2.7. Termination Criterion . . . 18

2.2.8. State of the Art . . . 19

2.3. Robustness . . . 24

2.3.1. State of the Art . . . 27

2.4. Summary . . . 28

3. Methods **29**
3.1. Robustness for Context Steering 29
3.2. Context Steering 30
3.2.1. Behaviors and Parameters 30
3.2.2. Adjustments 35
3.3. Algorithm Choice and Design 39
3.4. Fitness Function 43

4. Benchmark Implementation **49**
4.1. General Settings 49
4.2. Scenes 49

5. Experiments and Evaluation **53**
5.1. Evaluation Non-Determinism 53
5.2. Evaluation of an Agent 53
5.3. Settings 54
5.4. Experiment 1 54
5.4.1. Description 54
5.4.2. Results 56
5.4.3. Interpretation 60
5.5. Experiment 2 60
5.5.1. Description 60
5.5.2. Results 61
5.5.3. Interpretation 65
5.6. Experiment 3 65
5.6.1. Description 65
5.6.2. Results 66
5.6.3. Interpretation 67

6. Conclusion and Future Work **69**
6.1. Conclusion 69
6.2. Future Work 70

Appendices **73**

A. Scenario Variations **75**

Bibliography **79**

List of Figures

2.1. Basics: Traditional seek behavior . . . 5
2.2. Basics: Deadlock example . . . 5
2.3. Basics: Sensor and context map . . . 7
2.4. Basics: Behavior mapping . . . 8
2.5. Basics: Behavior combination . . . 8
2.6. Basics: Pareto dominance and front . . . 10
2.7. Basics: Evolutionary algorithm cycle . . . 14
2.8. Basics: One-point crossover . . . 17
2.9. Basics: Robustness Type I . . . 26
2.10. Basics: Robustness Type II . . . 27

3.1. Context steering: seek and flee mapping . . . 31
3.2. Context steering: avoid mapping . . . 32
3.3. Context steering: magnitude multiplier . . . 32
3.4. Context steering: value mapping . . . 34
3.5. Contest steering: URQ Mapping . . . 37
3.6. Evolutionary algorithm: encoding . . . 41
3.7. Evolutionary algorithm: algorithm design . . . 43

4.1. Implementation: scene 1 . . . 50
4.2. Implementation: scene 2 . . . 50
4.3. Implementation: variation 1 scene 2 . . . 51
4.4. Implementation: scene 3 . . . 52

5.1. Experiment 1: trails of scene 1 . . . 56

5.2. Experiment 1: trails of scene 2 58
5.3. Experiment 1: trails of scene 3 59
5.4. Experiment 2: trail of Scene 1 62
5.5. Experiment 2: Scene 1 - Fitness distributions 63
5.6. Experiment 2: trail of Scene 2 63
5.7. Experiment 2: Scene 2 - Fitness distributions 64
5.8. Experiment 2: trail of Scene 3 65
5.9. Experiment 2: Scene 3 - Fitness distributions 66

A.1. Implementation: variations scene 2 76
A.2. Implementation: variations scene 3 77

List of Tables

3.1. Parameters of context steering behaviors and their boundaries . 31

5.1. Experiment 1: Scene 1 - Fitness values 57
5.2. Experiment 1: Scene 1 - Mann-Whitney U Test 57
5.3. Experiment 1: Scene 2 - Fitness values 57
5.4. Experiment 1: Scene 2 - Mann-Whitney U Test 58
5.5. Experiment 1: Scene 3 - Fitness values 59
5.6. Experiment 1: Scene 3 - Mann-Whitney U Test 60
5.7. Experiment 2: Behavior Comparison 61
5.8. Experiment 2: Test Statistic 61
5.9. Experiment 3: Scene 2 - Robustness Analysis 67
5.10. Experiment 3: Scene 3 - Robustness Analysis 67

1. Introduction and Motivation

Movement is one of the essential aspects of believable non-player characters (NPC) in video games. Traditional steering systems provide agent movements that work well if there is a group of NPCs that is perceived as one entity. Minor flaws of single agents are not that important since they are often not observed or do not matter for the overall movement. However, when focusing on individual agents, e.g., when fighting them one on one or in a live simulation game, every faulty behavior erases the illusion of an intelligent NPC. To prevent these flaws, a system called context steering was developed, which enables agents to make more accurate and believable movement decisions. Similar to traditional steering, it provides a set of different behaviors that can be combined to create more advanced and difficult movement patterns. The difference is that the final decision-making is not based on the set of individual decisions from each behavior like it is done in traditional steering, but on the whole contextual information that each behavior gathered from its surroundings. To ensure broad applicability, each behavior has several parameters for fine-tuning. For an agent that consists of several behaviors, this can be up to hundreds of parameters that need to be adjusted. This can be a very time-consuming task, especially since some of the parameters influence each other.

The goal of this thesis is to reduce the work a user of context steering has with fine-tuning all parameters. A more abstract way to tell the context steering system how an agent shall behave would be more user friendly. This could be something as simple as drawing the desired path into the scene, and the context steering system finds itself the parameters that are needed to create such a movement. This parameter optimization task could be solved, for example, by an evolutionary algorithm that has already been used to solve many optimization tasks, including algorithm parameter optimization.

1.1. Goals

This thesis aims to evaluate if evolutionary algorithms are a feasible method to find parameter sets for context steering agents so that they behave believably and robustly. To achieve this goal, the following problems need to be considered:

- A robustness definition for an evolutionary algorithm for context steering needs to be defined.
- An encoding that fits the needs of context steering has to be developed.
- A fitness function for context steering that leads to successful and robust solutions needs to be found. Furthermore, this fitness function shall enable the user to guide the evolution towards a desired behavior.
- It has to be examined if the algorithm can automatically find a set of useful parameters for context steering behaviors.
- The robustness definition needs to be analyzed according to its property to lead to solutions that apply to a broader range of scenarios.

1.2. Structure

In the next chapter, basic information and corresponding related work are presented. This covers movement, especially the concept of context steering, evolutionary algorithms, and robustness of optimal solutions. The third chapter describes the used methods in more detail. A robustness concept for context steering optimization is introduced. Context steering behaviors and their parameters are described. Limitations of the existing context steering were shown, and solutions to overcome these are presented. Finally, the used evolutionary algorithm and its operators are explained. The fourth chapter introduces the used evaluation scenarios with all its parameters and variations. Chapter five shows how the experiments are designed, the resulting data, and how this could be interpreted. In the last chapter six, a conclusion for this work is made, and future work will be discussed.

2. Basics and Related Work

This chapter covers all the background information that is needed to understand the topic of this thesis. It starts with movement and especially context steering and is followed by evolutionary algorithms and robustness.

2.1. Movement

This section explains what context steering is and how it works. Context steering is a type of steering that not only considers the made decisions of single behaviors when combining them but also the context in which they were made. However, to understand what steering is in general and how it is located in the general topic of movement, this section starts with a short overview of movement types.

2.1.1. Overview

Movement as a word can have a lot of different meanings that are mostly related to changes. This could be the change in prices of products or the change of a person's view towards a specific topic. Nevertheless, in the domain of this work, movement is the change of position of an agent or object. How this change of position occurs can be subdivided into three different aspects: locomotion, steering, and path planning.

Locomotion covers everything that can be considered as agent dependent motions that are needed to make the agent actually move. For humans, animals, and humanoid robots, this would mean the motion or actuation of the individual body parts to change location or perform actions in one place. Alternatively, for vehicles, it would be something as simple as turning the wheels or spinning a rotor to change the position.

Path planning is a more abstract task that enables an agent to find a way from its current position to a given goal, mostly by defining sub-goals that are easier and more clear to achieve.

Steering is the connecting piece between locomotion and path planning. It utilizes the motion and actual movement of locomotion to move the agent towards the goal or sub-goal given by the path planning. This is done by deciding in which direction the movement shall happen to reach the goal. Based on the type of steering, this can also incorporate more complex tasks like local obstacle avoidance.

2.1.2. History of Context Steering

Reynolds described in [31] the different components of movement for an agent. He divides movement into action selection, steering, and locomotion, similar to the aspects of movement in subsection 2.1.1. Action Selection is a higher-level decision mechanism that detects, e.g., a change in world state and reacts to it. Locomotion, in contrast, is the accurate actuation of each body part to enable a character's motion, e.g., walking, running, or even standing in an idle position. His main work is about steering, which is in between action selection and locomotion. It takes the goals from action selection, decomposes it into a series of simple subgoals, and utilizes locomotion to do the actual movement. Furthermore, he describes different steering behavior like seek, flee, wander, avoid, pursue, evade, and arrive. Figure 2.1 illustrates how the resulting movement for a seek behavior looks like. The agent is currently moving along its current movement direction. The seek behavior creates a desired direction towards the target. The resulting steering direction is computed by the subtraction of the desired direction and the current movement direction. If a force is applied to that steering direction, it will lead the agent towards the target. How this exactly looks like depends on the controller that the agent uses. Either the direction is directly applied, and the agent rotates in an instant towards the target, or the controller has some turning limitations, and the change in movement is done step by step. Here, the path for a controller with physical limitations is shown.

More complex behaviors can be achieved by combining different behaviors. This is done by adding all resulting steering directions of each behavior. Other ways of combining behaviors could be averaging or prioritizing. Nevertheless,

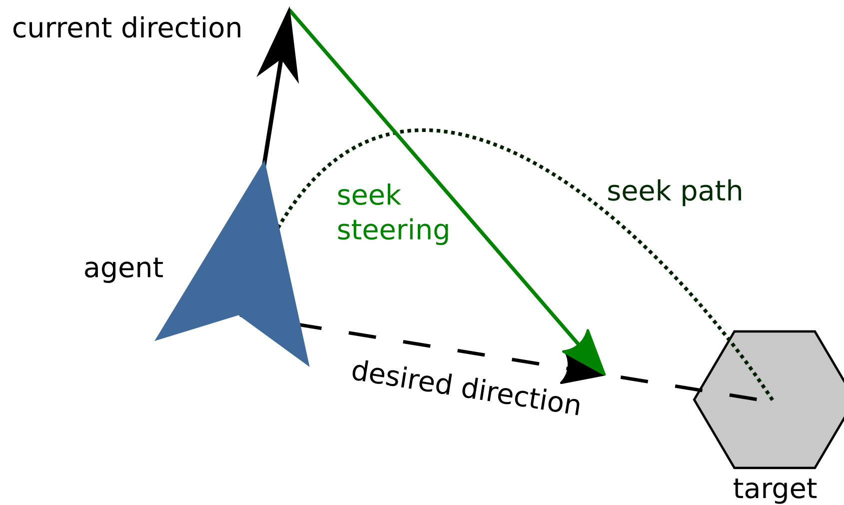


Figure 2.1.: The seek behavior of a traditional steering system is based on the agent's current position and movement and the target's position. The illustrated seek path is the result of the applied steering direction for a physics based agent. Graphic adopted from Reynolds [31].

the combination of merely steering directions can lead to unwanted behaviors like a deadlock (see Figure 2.2). Here, a deadlock is shown as a situation where the agent stops moving because the desired directions negate each other and result in a null vector. Another possibility is that the agents endlessly cycles in the same movement pattern without reaching the target. To prevent this kind of fault, additional code needs to be written to get highly specialized behaviors. This is contrary to the initial intention of having a modular and lightweight behavior system.

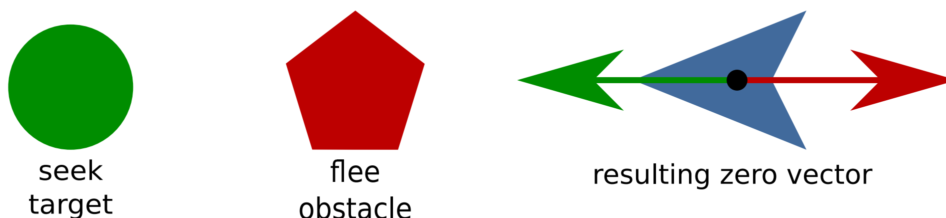


Figure 2.2.: Example for a deadlock scenario where seek and flee together result in a zero vector.

2.1.3. Context Steering

Aware of these issues, Fray developed context steering and published its concept in [13]. Context steering keeps the same goals as traditional steering: providing a set of small and lightweight behaviors that can be combined to achieve more complex behaviors. Nevertheless, compared to traditional steering, it provides not only the steering directions of each behavior but also the whole context in which the decision was made. Also, Fray extended this system from a single objective function to multiple objective functions.

Before explaining each part of context steering, a short overview is given. To perceive its surroundings, the agent has a sensor that consists of receptors. Each receptor has a direction vector and is linked to an element of a context map. This sensor enables the agent to get distance and direction-based information of all objects around it. This information is stored in a context map. Each behavior contributes its information into this context map. Based on the accumulated information, a final steering direction is made.

Context Maps

A context map is a one-dimensional array of scalar values. As already mentioned, each element of the context map is linked with a receptor of the sensor (illustrated in Figure 2.3). The direction vector each receptor has represents a possible movement direction of an agent. To get scalar values that can be written into the context map, the behavior needs to be sensitive to a particular set of objects. If an object the behavior searches for lies in the same direction as a receptor is pointing to, a scalar value is written into the corresponding element of the context map. How significant this value is, depends on the type of mapping that is used for creating the context values. For example, the closer an object is, the greater is the resulting context value.

Context Maps by Example

In Figure 2.4, a seek behavior is applied to the agent, and two objects A and B are within its vicinity so that the seek behavior can detect them. The distance between the agent and an object is described by the difference in positions: the object vector $vec_o = pos_{obejct} - pos_{agent}$. For all receptors, it is checked if

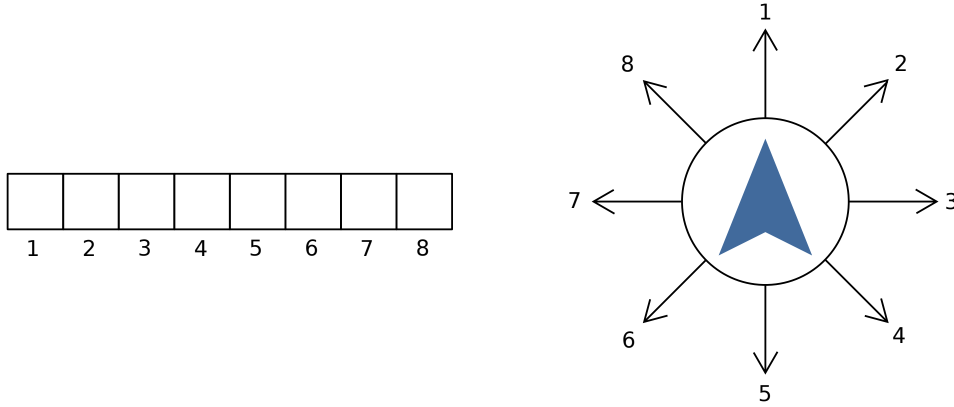


Figure 2.3.: Context map (left) and sensor (right) of an agent.

an object affects this receptor or not. This is done by the angle ω between the receptors vector v_r and the object vector.

$$\omega = cos^{-1} \frac{\langle \vec{v_r}, \vec{v_o} \rangle}{\|\vec{v_r}\| \|\vec{v_o}\|} \in [0, \pi] \tag{2.1}$$

If the angle is smaller than a given threshold $\theta \in [0, \pi]$, the receptor is affected by the object, and a context value z greater 0 is calculated.

$$z = \frac{\theta - \omega}{\omega} \|\vec{v_o}\| \in [0, 1] \tag{2.2}$$

Combining Behaviors and Context Maps

The example of context maps shows how a single behavior fills the values of a context map. For agents with more than one behavior, it needs to be defined how a behavior deals with an already filled context map. For the rest of this work, it is the case that if there is already a value in a context map element, then the greater one is chosen as the new context value. Other options are that the lower value is taken, or the values are added or subtracted.

Besides combining multiple behaviors into one context map, the context steering system Fray developed is also able to handle multiple objective functions, which means that multiple context maps can be combined to a final steering direction. Figure 2.5 illustrates Fray's example of combining context maps.

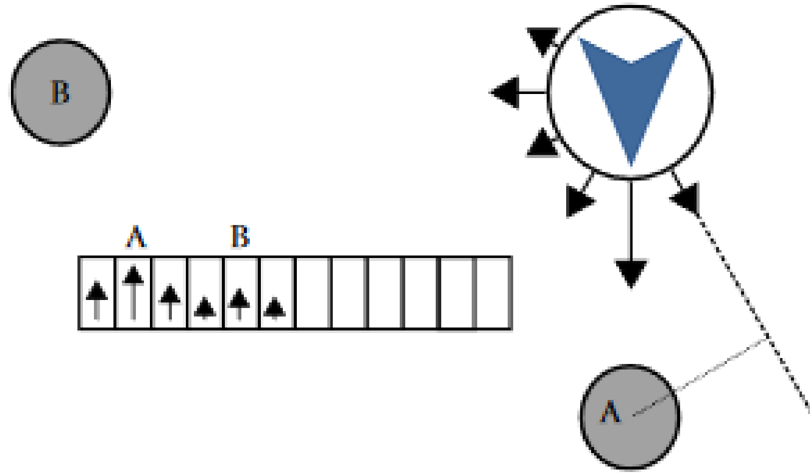


Figure 2.4.: Example of mapping information from a seek behavior into a context map. A and B are interest objects at different distances towards the agent. Graphic adopted from Fray [13].

Obstacles that need to be avoided are detected and stored in the danger map (i), and possible paths an agent can take are stored in the interest map. Next, the lowest value in the danger map is searched, which is zero in this scenario. So all values greater zero are masked out. This mask is also applied to the interest map (ii). Out of the remaining values of the interest map (iii), the largest one is chosen as the steering direction (iv).

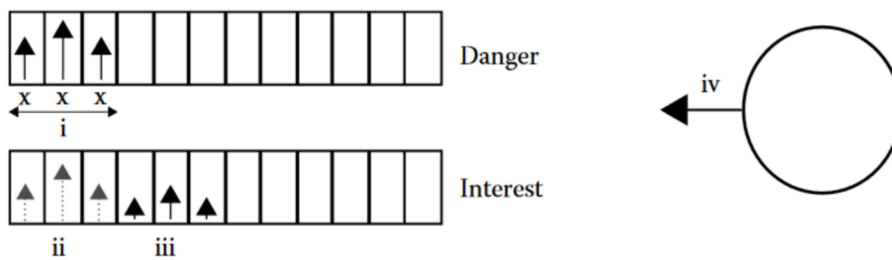


Figure 2.5.: Combination of two context maps for a steering direction. Graphic taken from Fray [13].

2.1.4. Multi-objective Decision Making

When using context steering with more than one context map, one has to decide how these maps are combined to find a final steering direction. The example from Figure 2.5 was Fray's approach to solve this problem. M. Kirst decided to use another approach, multi-objective optimization (MCO). Therefore the combination of several context maps can be formulated as a multi-objective optimization problem, as shown by Miettinen [26].

Definition 2.1 *(Multi-objective optimization problem):*

$$\begin{aligned} &\textit{minimize } \{f_1(x), f_2(x), ..., f_k(x)\} \\ &\textit{subject to } x \in S, \end{aligned} \tag{2.3}$$

with $k(\geq 2)$ being the number of objective functions that shall be minimized simultaneously and S as the search space.

This is done assuming that there is no single solution that is optimal with respect to all objective functions. Thus, the objective functions can be considered at least partially conflicting.

For a context steering scenario with the two context maps Interest and Danger, there are respectively the two objective functions f_i for the Interest map values and f_d, which for the Danger map values. The corresponding MCO-problem was taken from Kirst's work [20] and can be formulated as follows:

$$\begin{aligned} &\text{minimize } \{f_d(x), -f_i(x)\} \\ &\text{subject to } x \in 1, 2, ..., r, \end{aligned} \tag{2.4}$$

Where r denotes the context map resolution. $f_i(x)$ is negated since the problem is described as a minimization task and the maximal interest value shall be found.

For multi-criteria problems, a concept called Pareto-dominance was developed to find optimal solutions. For a minimization task, Pareto-dominance is defined as follows:

Definition 2.2 *(Pareto Dominance). A solution x is dominant to solution y if x is better or equal to y in all objectives i and better than y in at least one objective j.*

$$x \prec y \Leftrightarrow f_i(x) \leq f_i(y), \forall i \in 1, ..., k \land \exists j : f_j(x) < f_j(y) \tag{2.5}$$

An optimal solution is one that is not dominated by any other solutions. Since there are several non-dominated solutions for a multi-criteria optimization problem, these are combined in a set that is also called Pareto-front. Figure 2.6 shows these concepts for a minimization task. Solution s_2 dominates all solutions that are worse than itself in all objectives. Although it is a dominant solution, it gets dominated by solution s_1. s_1, in contrast, is not dominated by any other solution, and thus, it belongs to the first non-dominated Pareto-front F_1.

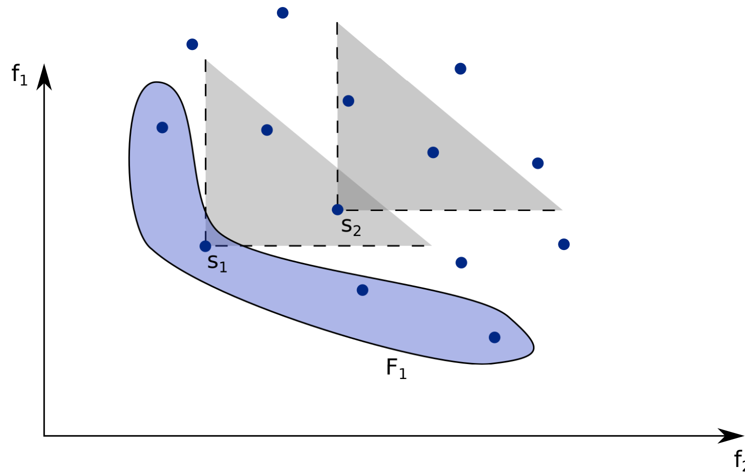


Figure 2.6.: Example for Pareto-dominance and Pareto-front for a minimization task of objectives f_1 and f_2.

Since the solutions in the Pareto-front are all optimal, all of them can be used to solve the MCO-problem. Nevertheless, one of them has to be chosen. Therefore several approaches can be used: weighted sum, ε-constraint, and a hybrid approach. For this work, only the ε-constraint is of importance, and only this one will be explained because it is the method of the context steering tool that is used for this work. Furthermore, M. Kirst's experiments show that this approach leads to better performance when changing some context steering parameters like sensor resolution, the angle threshold, or the constraint itself.

ε-constraint

In the ε-constraint method, one objective is chosen to be optimized, and all other objectives act as a constraint for the optimization. For our context

steering scenario with two objectives, the ε-constraint problem looks like this:

$$\begin{aligned} \text{minimize} \quad & -f_i(x) \\ \text{subject to} \quad & f_d(x) \leq \varepsilon_d, \\ & x \in 1, 2, ..., r \end{aligned} \tag{2.6}$$

Here, the interest objective is the one that will be optimized and the danger objective acts as a constraint. Only solutions whose danger value is less than the constraint ε_d are allowed to be chosen. If no solution satisfies the constraint, the one with the smallest danger value is chosen.

2.1.5. State of the Art

Besides context steering, there are other approaches that keep the believable behavior of an agent in mind. G. Berseth and M. Kapadia present a steering concept that considers the agent's locomotion abilities [6]. This footstep-based algorithm is sensitive to motion limitations so that turning or sliding feet are not possible while they are in contact with the ground.

A. Gerdelan and C. O'Sullivan propose a system where an evolutionary algorithm is used to tune the rules in the rule base of a fuzzy controller for steering [15]. In fuzzy steering, distance and angles are classified in human-like terms. Instead of meters, a fuzzy set like near, medium, and far is used. For angles in degrees, it would be narrow, mid, and wide. The rule base is a list that matches every possible combination of fuzzy inputs (distance and angle) to a valid fuzzy output (steering direction and speed). Their evaluations show that this system is able to produce a better controller configuration than the hand-tuned reference set.

Traditional steering has some limitations that are visible if an individual agent is inspected more closely. In simulations were a lot of agents move as a flock this behavior does not destroy the immersion since the flock acts as the individual under observation. While context steering concentrates on better performance of individual agents, there are many algorithms that try to improve crowd simulations.

PLEdestrian [16] from S.J. Guy et al. is an example of a crowd steering algorithm that uses the principle of least effort (PLE) and a biomechanically energy-efficient trajectory to move agents in a multi-agent simulation.

Other algorithms use the social force model for pedestrian dynamics developed by D. Helbing and P. Molnár [17]. This model describes the movement of pedestrians and how they are influenced by their surroundings. They have a certain goal they want to reach and thus have a force towards it. Then there are other things, like walls or strangers, that create a repelling force and interesting things, like friends or attractions, that create an attraction force so that the pedestrian is pulled towards it.
An algorithm that uses this concept is developed by P. Saboia and S. Goldenstein [32]. Here the social force model is modified by a mobile grid so that pedestrians are able to avoid blocked or crowded areas.

In times where machine learning is of great interest, there are also data-driven approaches for crowd simulation. A. Bera et al. present an algorithm that extracts trajectories from video material and learns the parameters for a pedestrian motion model [5].

2.2. Evolutionary Algorithm

Evolutionary algorithms are nature-inspired metaheuristics to solve numerical and combinatorial optimization problems that are used when no other efficient solution algorithm is known. They provide only approximate solutions for an optimization problem. Kruse et al. gave the following definition for an optimization problem [22]:

Definition 2.3 *(Optimization problem). A Tuple* $(\Omega, f, \prec)$ *describes an optimization problem: with a search space* Ω*, a fitness function* $f : \Omega \to \mathbb{R}$ *assigns a quality value to each solution, and a relation operator* $\prec \in \{<, >\}$

At the start of this section, a short overview of evolutionary algorithms is given, and afterward, all aspects are explained in more detail.

Evolutionary algorithms are population-based approaches to solve optimization problems. This means that they create individuals that serve as a possible solution to the given problem. An individual is made of genes in which its properties are encoded. How these genes look like and what each gene means is described by the encoding. These solutions are tested on their ability to solve the problem by a fitness function. A termination criterion decides if the algorithm is finished or another generation of individuals is tested. If the algorithm does not end, and the next generation is required, this generation has to be created. This is done by selecting individuals based on their acquired fitness value and an operation called crossover in which individuals exchange genes to create new individuals. Crossover is not the only option to change the genes of an individual. Mutation can randomly change single or multiple genes of an individual. After the new individuals are created, the next generation will be formed. Mostly it is advisable to take the best or at least a specific ratio of the best individuals into the next generation and some individuals that keep the gene pool diverse. This cycle repeats until the termination criterion is satisfied.

2.2.1. Encoding

Like in nature, individuals in evolutionary algorithms are made out of genes. These genes together form the chromosome of an individual. In contrast to nature, there is only one chromosome necessary. The combination of genes in a chromosome is also called the genotype. It contains the information which

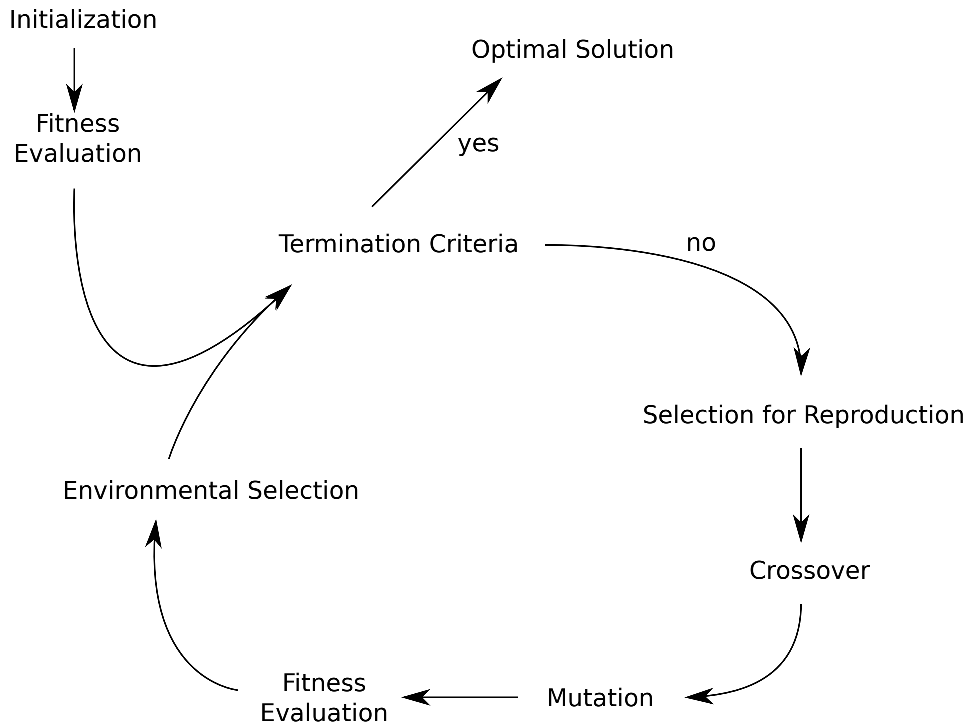


Figure 2.7.: Structure of an evolutionary algorithm.

genes, gene types, and value boundaries are used. Out of this information, a real individual can be constructed that can be tested on its ability to solve the given problem. The characteristics and properties of an individual can be summarized in the term phenotype. As already mentioned, there are different types of genes, and thus, a variety of different encoding types are possible.

The binary encoding uses bits to encode the information of an individual. They are also called bitstrings since the genes are chained into a single string. Bitstrings can be used in various ways that depend on the task to solve. The knapsack problem is a task in which the representation can also be used as a solution. This is done by encoding each object as a bit where 0 means the object is not put into the knapsack and 1 that the object is put into the knapsack. Another way to use binary vectors is to encode single numbers as bitstrings and process these bits in the algorithm. To use these vectors again as a solution, they need to be decoded back to a number. So a certain type of mapping between representation and the search space is needed. For numbers, this would be the standard binary decoding or gray code. Gray code has the property that it reduces the hamming distance between consecutive numbers to

one. When using bitstrings to encode continuous search spaces, it is advisable to use discretization and encode intervals by bit combinations.

There is also the possibility to encode the genes in a vector of real-valued numbers. Either pure integer values or floating-point numbers can be used. For using integer values, there is also the option that these encode categorical options. Thus, the genes are limited to the number of options available. In contrast, floating-point numbers offer the possibility to directly work with them instead of converting them into a discrete interval. In general, it is practical to use an encoding that is inherently given by the task so that no additional conversion is necessary. Of course, it is also possible to mix the different gene types in a chromosome and have a mixed encoding.

2.2.2. Fitness

The fitness of an individual is a value that indicates how good this solution is at solving the task. The fitness function calculates it. In nature, the fitness function would be the environment in which an individual has to live and survive. If it is good at surviving, it has the possibility to create offspring. In evolutionary algorithms, the fitness function can be a mathematical function that needs to be optimized or a simulation that needs to be solved. Besides telling how good an individual is, the fitness and fitness functions have other important characteristics and properties. The most important one is that the fitness function should be a continuous function that has no jumps in it. This is important because then small changes in the decision variables will also cause small changes in the fitness value. If these changes improved the fitness value, changes further into that direction could maybe lead to even more significant improvements. This property enables a solution to accumulate small improvements by doing just small changes. This makes evolutionary algorithms not purely random but more like a guided random walk.

2.2.3. Selection for Reproduction

The selection for reproduction is a fitness-dependent process. Individuals with better fitness have a higher chance of surviving and reproducing. This concept is also part of most of the selection mechanisms. Examples are roulette wheel selection, stochastic universal sampling, and tournament selection [22].

Roulette wheel selection is also called fitness-proportional selection since individuals with better fitness have a higher chance of being selected. For problems where a maximal fitness is searched, the selection probability of individual i is equal to its contribution to the accumulated fitness of the generation G.

$$p_{sel}(i) = \frac{f(i)}{\sum_{j \in G} f(j)}$$

Stochastic universal sampling (STU) is a variation of roulette wheel selection that was developed to reduce the selective pressure. Selective pressure describes the property of a selection method, how much it favors individuals with better fitness. Here, instead of selecting one individual at a time, several are selected. If there is an individual that dominates the selection process because it has a fitness value that is hugely better than all other, it would be chosen nearly all the time. With STU, other individuals have a higher chance of being selected than by roulette wheel selection.

In tournament selection, individuals compete against each other in small tournaments. Here, a group of individuals is chosen at random from the generation, and the best one is selected. The tournament size T decides how significant the selective pressure of this method is. The two limiting forms of this method are a tournament size T of one or equal to the number of individuals in the Generation N. With only one individual in a tournament, this resembles random selection, and there is no selective pressure. With a tournament size of N, only the best individual of the whole population will be selected, and thus, a maximal selective pressure is applied. Mostly, a value between one and N is chosen.

2.2.4. Crossover

Crossover is a process that is inspired by sexual reproduction. Here, the genes of different individuals are mixed to create a new individual. The first version was inspired by the Mendelian inheritance, where children get a combination of the parents' genes (see Figure 2.8). Examples for these crossover types are the one-point crossover, the more general n-point crossover, uniform crossover, or shuffle crossover [36]. For the different encoding types, different forms of crossover emerged. For floating-point genes, an arithmetic crossover can be

used, where the genes are averaged or processed in a different form before passing them to the children. Encodings that require permutations as individuals use crossover forms that create new permutations out of the parent genes. The number of parents is not just limited to two. Sometimes three or more parents can be involved. Some crossover types also consider the individuals' fitness values to create offspring that are closer to the fittest individual.

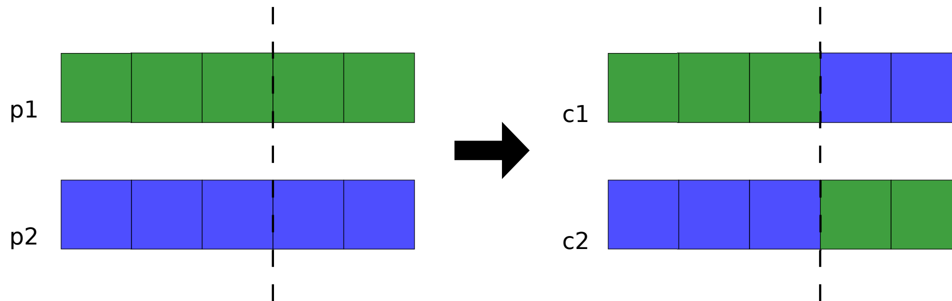


Figure 2.8.: Example for a one-point crossover where parents p1 and p2 exchange genes to create children c1 and c2.

A commonly used crossover type is the Simulated Binary Crossover (SBX) [10] . SBX takes the genes from both parents as well as the gene boundaries and a parameter η to create child genes that can be an interpolation as well as an extrapolation of the parent's genes. The η value decides how similar the child's genes are compared to the parent's genes. For a large η value the children resemble their parents and for a low value they are more different.

2.2.5. Mutation

Mutations are small changes in an individual's genes that occur randomly based on a mutation probability p_m. Depending on what gene type it is, there are different ways for possible changes. Binary genes can be mutated by flipping one bit to the inverse value. For integer genes, a random offset can be added to the value, or if the gene encodes categorical values, one of the remaining values can be chosen at random. Also, for float genes, a random offset can be added to the current value. Here, the difference is that the random offset is determined by a probability density function. An example is the Gauss distribution that is used in a multi-gene Gauss mutation[3].

In a multi-gene Gauss mutation, each individual can be mutated multiple times, depending on p_m. Here, p_m is used as the mutation probability for

an individual. In consequence, the mutation probability for a single gene depends on p_m and the number of genes an individual has. So that genes of an individual with fewer genes have a higher probability of being mutated. Therefore, a random value from the Gauss distribution is chosen and added to the current gene value. How big this random value is, depends on the mean value μ and the standard deviation σ of the Gauss distribution.

2.2.6. Environmental Selection

In evolutionary algorithms, the generation size N is fixed, so one has to choose which individuals survive. There are two important aspects that one need to keep in mind: first progressing to better fitness, second keeping the gene pool diverse.

For reaching a better fitness value over time and generation, one can choose a fixed number E of best individuals of the current generation and take them to the next generation. This concept is called elitism. After choosing the best individuals, the next generation needs to be filled up to the generation size N. Therefore several options are available. In the (μ, λ) - selection, there are μ parent individuals that create λ offspring. Here, the remaining individuals are chosen only from the offspring. Another way would be the $(\mu + \lambda)$ - selection where μ parents create λ offspring and the remaining individuals are chosen from both. In both variants, it is the case that if the selection pool (μ or $\mu+\lambda$) is greater than the remaining individuals that need to be selected $(N-E)$, there has to be a selection mechanism. This could be at random to keep a diverse gene pool, individuals with the best fitness, or other selection mechanisms like roulette wheel, STU, or tournament selection.

2.2.7. Termination Criterion

The steps of an evolutionary algorithm described in the previous sub-chapters are part of a loop that enables the evolutionary algorithm to search for an optimal solution. However, at some time, the algorithm needs to stop. The termination criterion defines when the algorithm shall stop. One way could be to let the algorithm run until a certain solution quality is reached. Nevertheless, it is not always clear at what fitness value the solution is good enough to solve the real task. So another way is to let the algorithm run for a fixed

amount of time or number of generations. This way, the used time is known, and if the solution quality is not good enough, the process can be continued.

2.2.8. State of the Art

There is a wide range of optimization techniques available that can lead to a good parameter set. In this work, a basic evolutionary algorithm was chosen. However, there are several other kinds of evolutionary algorithms and other optimization methods that shall be mentioned.

Evolutionary Algorithms

M. Affenzeller and S. Wagner present in their works [1] and [2] a Self Adaptive SEgregative Genetic Algorithm with Simulated Annealing aspects (SASEGASA). Besides finding an optimal solution, this algorithm is designed to avoid premature convergence of a population so that local optima are better avoided in the final solution. To achieve this goal, the initial population is split into several sub-populations that evolve independently from each other until each of them converged to a local optimum. For the convergence, a special crossover mechanism is used. Only children that surpass the worse of their parent individuals are chosen for the next generation. If a child is worse than both parents, a new child needs to be created. If a maximum number of children per generation is reached, this sub-population is considered as converged. If all n sub-populations are converged, they are reunited into n-1 sub-populations. This cycle repeats until all individuals are contained by only one population.

M. Potter and M. De Jong introduced a concept called cooperative coevolution (CC) [30]. The main idea of this concept is that a complex task can be decomposed into several subcomponents. For an individual of an evolutionary algorithm, this means that its genes are separated from each other into groups called species, and these groups evolve independently from each other. Since, for most problems, these subcomponents have some interdependencies, the species need to be recombined to a complete solution for the fitness evaluation. To get the fitness values of one species type, a solution is constructed based on representatives of the other species. The representative solution is completed by the individuals of the tested species, and thus a relative fitness contribution for each individual can be calculated. This is also done for all other species to estimate their individuals' fitness contributions.

Z. Yang et al. use this concept in a work [38] where they want to use evolutionary algorithms for a high-dimensional problem with up to 1000 dimensions. Their developed framework is tested on several classical benchmark functions and test functions provided by CEC2005 Special Session. These results were compared to conventional EAs, Self-adaptive differential evolution with neighborhood search (SaNSDE), and other cooperative coevolution algorithms like FEPCC and DECC-O. The result is that the proposed framework outperforms non-CC algorithms for high-dimensional problems and is better than the other CC algorithms for most of the test cases.

Another way to use this concept is shown by Yong and Miikkulainen [39]. They try to solve a multi-agent task by using cooperative coevolution. The task is a predator-prey scenario in which a group of predators has to catch the prey by cooperation. The actual controller for the predator is a neural network which is evolved by a CC-EA. Here the focus lies not on the problem decomposition but in evolving cooperative predators that can have different characteristics. So the subpopulations consist of predator individuals. These subpopulations are evolved independently from each other. For the evaluation, a predator from each subpopulation is taken, and it is tested if all together are able to catch the prey. The CC approach is tested against a central controller that controls all three predators together. The results show that controlling the predators by individual networks leads faster to better results than controlling them by a centralized controller. Furthermore, the need for communication and cooperation are evaluated. In the scenario without communication, the predators do not know each other's positions. Surprisingly the non-communication system learned a useful behavior faster. In the scenario without cooperation, the predators are trained in a way that they chase the prey alone. The results show that predators trained this way are not able to solve this task since they chase the prey all from the same direction.

Besides use cases for cooperative coevolution, there are several papers that concentrate on different aspects. Mahdavi et al. evaluate how important the initialization of the starting population is [25]. Therefore they present three sampling strategies: center-based normal distribution, central golden region, and hybrid random-center normal distribution. The performance is evaluated on the CEC-2013 LSGO benchmark functions. The results show that proposed sampling strategies for CC algorithms are superior to or at least comparable to a random initialization.

Other Optimization Techniques

There are also other optimization techniques besides evolutionary algorithms. Sequential Model-based Algorithm Configuration (SMAC) [18] is a method developed by F. Hutter et al. Here, a regression model of one or several problem instances is created and based on this model new configurations/solutions are created to refine the model even more. In this way, the model can be used to predict how configuration instances perform on the problem to find an optimal solution.

Another population-based approach is iterated racing for automatic algorithms configuration (irace) [24]. This is a racing algorithm where several parameter configurations are tested on a set of problem instances. After a certain amount of instances have been evaluated, the configurations are regularly checked if some of them perform worse than the others. If that is the case, these configurations are removed from the race. The race continues until either a minimum number of configurations remain or a maximum number of instances are tested. For the next race, new configurations are sampled in a way that they are in the vicinity of the surviving one from the last race. To be able to compare the new configurations with the best configurations from the previous race, they are tested on nearly the same amount of problem instances. These races continue until a certain amount of budget (number of evaluations, time) is consumed.

Particle swarm optimization (PSO) is a method for the optimization of continuous nonlinear functions developed by J. Kennedy and R. Eberhart [19]. Originally this method was inspired by social behavior and bird flock simulations. However, after changing these concepts to an optimizer, the particles/solutions looked more like a moving swarm than a bird flock. The particles move in the search space based on their positions and velocities. To efficiently explore the search space, the velocity is adapted based on the particles' previous best position, and the global best position found so far that all particles share with each other. The velocity update is the difference from the current position and the best positions, multiplied by a random factor. To test this method, it was used to train a neural network solving the exclusive-or problem. In another test, it was used to train a network to classify the Fisher Iris Data Set. In both experiments, PSO was as effective as the backpropagation method.

Some years after the development, PSO was a research topic of interest in which many papers were published. Poli et al. give an overview of changes and progress made in this topic [29]. A concept called inertia weight is presented

that acts like friction for particles. It can be used to adjust the degree of exploration through the whole experiment. A variation of the standard PSO is the fully informed PSO, where the particles do not only know their own best position but also from all of its neighbors. In the early stages of PSO, the population topology was proximity-based, but also static and dynamic topologies were examined. Different PSO variants are presented like the binary particle swarms where PSO is applied to binary problems, and the solutions are represented as bitstrings. Some variations are able to work on dynamic problems where the fitness function changes over time, and some variations are made to handle noisy fitness functions. Other variations focus on diversity control to avoid premature convergence in local optima.

Application Examples

Beside directly finding a problem solution, evolutionary algorithms can also be used to find hyperparameters. In their work [4], C. Banerjee et al. used an evolutionary algorithm to find good parameter values for a neural network, such as learning rate, batch size, number of epochs, and dropout regularization. This approach was compared with other parameter estimators and gave better results.

Another example of the use of evolutionary algorithms for parameter optimization present A. Sehegal et al. in their work [34] where they want to find parameters for reinforcement learning. Here, they use a genetic algorithm (GA) to optimize parameters of the Deep Deterministic Policy Gradient algorithm (DDPG) combined with Hindsight Experience Replay (HER) for a set of robotic manipulation tasks like slide, push, pick and place, and door opening. They showed that using a GA to find parameters lead to better performance that is also found faster for the defined tasks.

Aziz Kaba and Emre Kiyak use an evolutionary algorithm to improve the performance of a Kalman filter for nonlinear quadrotor attitude dynamics. The algorithm is used to estimate the measurement and noise covariance matrix. This approach is compared with other optimization algorithms like optimal Kalman filter, covariance - matrix adaptation, evolution strategy, and simulated annealing. The results show that this approach is usable, and a Monte Carlo analysis verified that it is better than the other algorithms.

Shape optimization is another of the many applications for evolutionary algorithms. This can be the shape of radio frequencies (RF) like it is done by M.

Kranjčević et al. [21]. The shape needs to optimized in a way that the RF is capable of accelerating charged particle beams.
Another way of shape optimization is presented by G. Persico [28]. Here, a surrogate-based evolutionary algorithm is used to find optimal shapes for blades of a turbine so that no shock waves occur.

P. García-Sánches et al. demonstrate that evolutionary algorithms can be used to develop artificial intelligence for collectible card games [14]. They also show that this approach is approach can compete with state-of-the-art techniques such as Monte-Carlo Tree search.

An approach similar to the one presented in this thesis is shown by G. Berseth et al. [7]. For crowd simulation, there are many steering algorithms so that agents can move from point to point while avoiding static and dynamic obstacles. The performance depends on internal parameters. Berseth et al. use the Covariance Matrix adaptation Evolution Strategy technique (CMA-ES) to find optimal parameters for a set of different crowd steering algorithms. This way, they can reduce turbulence at bottlenecks, produce emergent patterns, and improve the computational efficiency of the algorithms.

2.3. Robustness

Evolutionary algorithms, like the ones described in the previous section, are used to find optimal solutions. An optimal solution in terms of fitness is not always the best solution for a given task. Some applications can not guarantee that a solution will be used as specified. For example, there may be noise in the processes that disturb the decision variables of the solution so that it is a slightly different solution depending on the tolerances the process has. These changes could result in a bad performance that is not acceptable for the process so that for some tasks, only robust solutions are of interest.

K. Deb and H. Gupta described robustness as a property of a solution such that small perturbations in its variables/decision space will hardly affect the solution quality [11]. Branke suggested that, therefore, one should not only look at the solution itself but also at its neighborhood [8]. So that solutions on a high plateau will be preferred over solutions on a thin peak. A value that also considers the neighborhood of a solution is called effective fitness (f_{eff}). How much of the neighborhood is needed depends on the noise that will be added to a solution to consider it robust. A calculation could look like this:

$$f_{eff}(x) = \int_{[-\infty,\infty]^d} p(\delta) \cdot f(x+\delta) d\delta \tag{2.7}$$

Here, $p(\delta)$ is the probability density function for the disturbance δ and d the number of dimensions of the fitness function. Since this operation is expensive and will not be possible for problems of higher complexity, the effective fitness has to be estimated. This estimate of $f_{eff}(x)$ is called $f_{mod}(x)$ and there are several ways to get it:

- Repeat the evaluation with random perturbation and use the mean value out of these evaluations.
- Do a single disturbed evaluation. The returned value of a random disturbed solution is considered equivalent to the mean of its neighborhood.
- Repeat the evaluation only for the best individual or a range of best individuals. Again use the mean value as the effective fitness.
- For population-based approaches, already existing individuals can be used as the neighborhood. Either the current population can be used,

or the last x individuals can be stored. The estimate is calculated as follows:

$$f_{mod}(x) = \frac{\sum_y w(y) \cdot f(y)}{\sum_y w(y)} \tag{2.8}$$

$$w(y) = \max\{0, 1 - d \cdot b\} \tag{2.9}$$

with $w(y)$ being the weight of individual y, d being the distance between x and y, and b a parameter that adjusts how big the vicinity of x is.

Types of Robustness

K. Deb and H. Gupta defined different types of robustness for multi-objective solutions. In this work, those types are used for a single objective task.

Definition 2.4 *(Robust Solution of Type I): A Solution x^* is called a multi-objective robust solution of type I, if it is the Pareto-optimal solution of the following multi-objective minimization problem defined with respect to a δ-neighborhood (B_δ)*

$$\begin{aligned} &\textit{Minimize } (f_1^{eff}(x), f_2^{eff}(x), ..., f_M^{eff}(x)), \\ &\textit{subject to } x \in S, \end{aligned} \tag{2.10}$$

where $f_j^{eff}(x)$ is defined as follows:

$$f_j^{eff}(x) = \frac{1}{|B_\delta|} \int_{y \in x + B_\delta} f_j(y) dy. \tag{2.11}$$

Figure 2.9 shows a visual interpretation of this robustness type for a single objective minimization task. The solid line is the original function that needs to be optimized. The dashed line is the value of $f_{eff}(x)$. From Equation 2.11, one can see that this function accumulates the function values over a certain neighborhood and divides these values by the neighborhood size. It is similar to calculating the mean value of a discrete function, and thus it is a smoothed version of $f(x)$. In this function, B is the global optimal solution. Nevertheless, with respect to the definition of type I robustness, A would be the optimal solution, as illustrated.

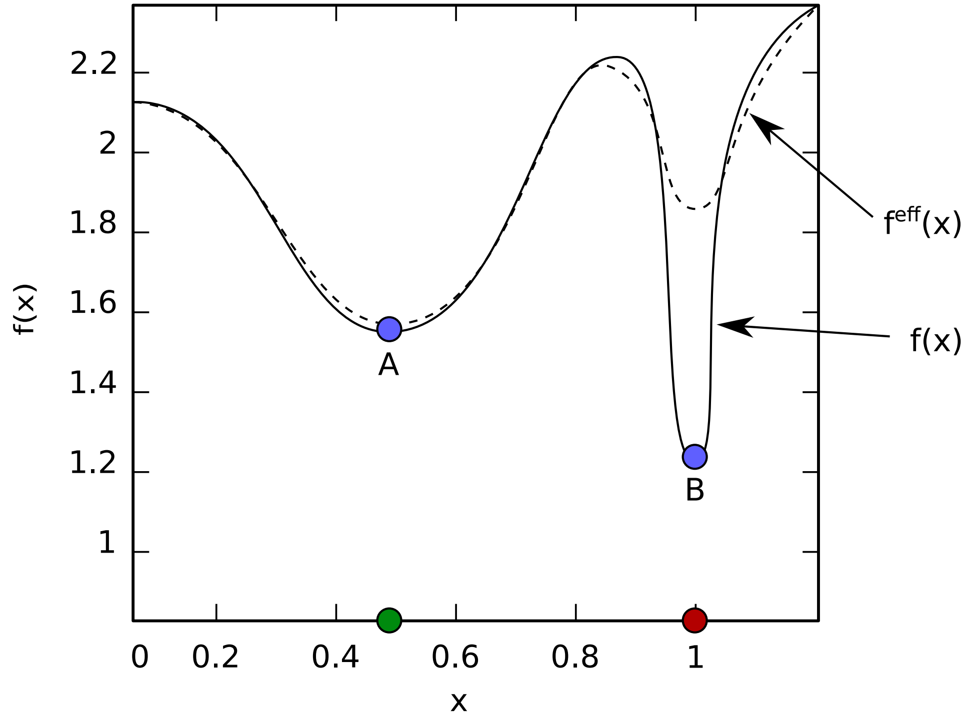


Figure 2.9.: Global vs. robust solution optimization

Definition 2.5 *(Robust Solution of Type II): For the minimization of a multi-objective problem, a solution x^* is called a robust solution of type II, if it is the Pareto-optimal solution to the following problem:*

$$\begin{aligned} &\textit{Minimize } (f_1(x), f_2(x), ..., f_M(x)), \\ &\textit{subject to } \frac{\|f^p(x) - f(x)\|}{\|f(x)\|} \leq \eta, \\ &x \in S. \end{aligned} \tag{2.12}$$

Figure 2.10 shows a visual explanation of type II robustness for a single objective task. Here, a solution is considered as a robust solution if a perturbed function $f^p(x)$ is only to a certain degree η worse than the actual fitness function. $f^p(x)$ could be f^{eff} like for type I or the worst fitness value inside a given neighborhood. If the perturbed function values $f^p(x)$ are worse than allowed by η, these solutions create an infeasible region which is considered as not robust enough for the given application. So the goal of type II robustness is to find the optimal solution out of all remaining feasible solutions.

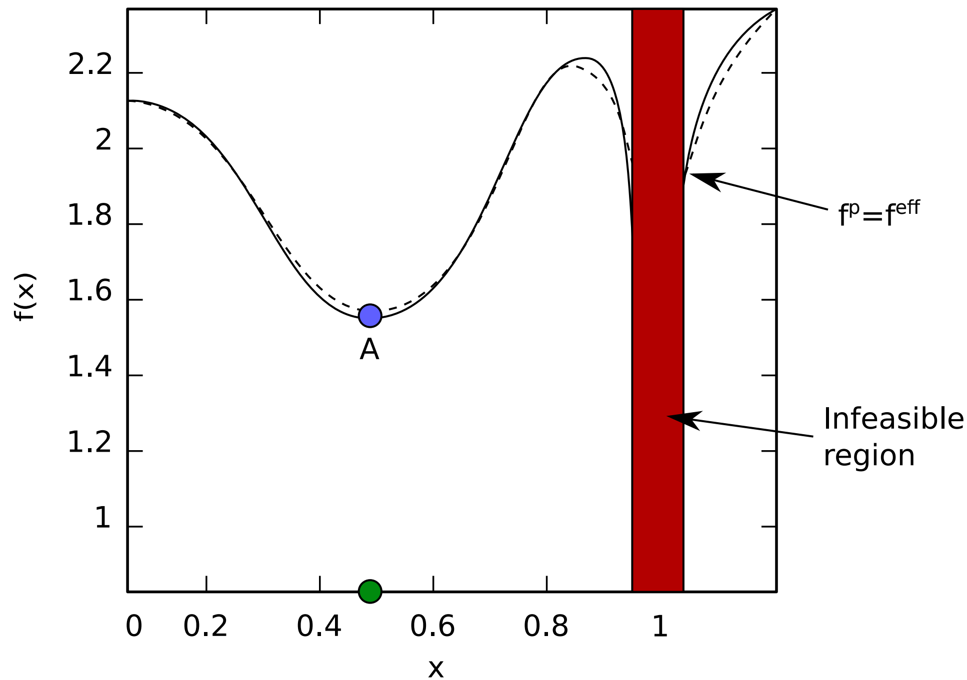


Figure 2.10.: Robustness of type II where an infeasible region marks all non-robust solutions.

2.3.1. State of the Art

Robustness is researched in a large variety. As mentioned in section 2.3 K. Deb and H. Gupta defined robustness for a multi-objective optimization in their work [11]. Furthermore, they used their definition not only to find one robust solution but a whole Pareto-optimal front.

J. Branke studied robustness for individuals in several of his works. He is starting with a definition of robust solutions and how to estimate them in more complex scenarios where the exact calculation is not possible[8]. Also, several ways of creating robust solutions are compared in regard to their efficiency [9].

Pérez et al. investigated the robustness of general video game playing agents in [27]. Therefore they took 4 out of 60 games of the general video game artificial intelligence (GVGAI) framework, two controllers from the framework, and four winner controllers out of the competitions 2014 and 2015. Each controller was tested 100 times in each of the five levels of the four games. So in total, each agent played each game 500 times to get ranked. The performance and changes in the ranking were evaluated under normal conditions, with a changed reward

system, and noise in the agent's applied decision and/or the decision in the forward model. The forward model is a simple simulation of the game so that the agent can predict what action it should use. The goal was to evaluate how the different controllers react to the different reward systems and noises and if there is a controller that is relatively robust towards these changes.

Niels van Hoorn et al. used multi-objective evolution to create a robust player imitating AI for racing scenarios in [37]. The developed controller was based on a recurrent neural network. This network was evolved using the NSGA-II algorithm. Three fitness measures were defined, of which two evaluate the imitation of player commands, and the third measured how well the car performed on a particular track. The controller was evolved on different tracks and tested on all 4 of them. What the authors consider as a robust agent was not exactly described. Probably the generalization of the agent was meant. So how well the agent performed on a track, it was not trained for.

2.4. Summary

This chapter covered the background information for movement, evolutionary algorithms, and robustness. A basic categorization of different movement aspects was presented. The development of context steering out of traditional steering was shown, and the context steering principles were explained in detail. In contrast to context steering, which focuses on individual agents, several swarm simulation variants were presented.
For evolutionary algorithms, the structure and components were described in detail. Several variants and their advantages were presented and also other population-based approaches for optimization.
Robustness as an optimization criterion was introduced, and different types of robustness were shown. Moreover, some example applications out of the current literature were shown.

In the next chapter, context steering, evolutionary algorithms, and robustness shall be combined to create a system that is able to find robust and believable context steering solutions.

3. Methods

The goal of this work is to evaluate if evolutionary algorithms can be used as parameter optimizer for context steering agents. Thus, this chapter explains the used context steering approach, including the most important parameters, and how they influence the agent's movement pattern. Additionally, it is explained what algorithm is used for this task and how it is structured. However, at first, an own robustness definition for context steering is made.

3.1. Robustness for Context Steering

A solution for context steering is the set of parameters $\vec{x}$ that define how each behavior itself works and how all behaviors are combined into a steering direction. This parameter set is fixed, and for simulations in a computer, it cannot be affected by noise except this is wanted. So robustness as insensitivity to perturbations would be of no use for a context steering system that does not allow noise. Thus, robustness for context steering is defined as follows:

Definition 3.1 *(Robust solution for context steering): A solution $\vec{x}^*$ is called a robust solution in terms of context steering, if it is the optimal solution to the following problem:*

$$\left.\begin{array}{l} \mathit{Minimize}(\mathit{Median}(f(\vec{x}, e_1), f(\vec{x}, e_2), ..., f(\vec{x}, e_M))) \\ \mathit{subject\ to}\ e_j \in E, \vec{x} \in S. \end{array}\right\} \tag{3.1}$$

Where E is a set of different but similar environments on which the solution $\vec{x}$ is tested. $f(\vec{x}, e)$ is the fitness function for context steering that is explained in detail in section 3.4.

Robustness for context steering means to find a parameter set that is fixed for an agent but allows the agent to perform nearly equally well in a set of different scenarios that are similarly structured.

3.2. Context Steering

The Polarith AI for Movement plugin for unity was used as a baseline for context steering. Only the parts that are important for this work will be explained here. For a detailed explanation of the plugin, see the documentation [35].

All the scenes in this work have the same structure. There is a 2D scenario that serves as an environment for an agent to move around. Besides the agent, this scenario consists of an interest object that the agent shall collect, one or more danger objects that the agent shall avoid and one or more user-defined paths to which the agent shall minimize its distance while moving around. To change the movement characteristics of the agent, it has some configurable behaviors attached. These behaviors detect a particular group of objects which are either interest objects or danger objects. The information the behaviors generate out of the detected objects is put into one of two context maps. One serves as the interest objective and the other one as the danger objective. Finally, both objectives are combined to find a final movement decision.

3.2.1. Behaviors and Parameters

For this thesis, the following behaviors were chosen: Seek, Flee, Avoid, Pursue, and Evade. First, these behaviors are explained. For each behavior, it is also described how objects generate objective values that are applied to a context map. After that, the parameters that all behaviors have in common are described, and also the parameters that are only used in a subset of behaviors are explained.

Behaviors

Seek/Flee: Whereas seek generates context values towards the detected object, flee generates these values in the opposite direction. These behaviors are illustrated in Figure 3.1.

Avoid: For the avoid behavior (see Figure 3.2), a plane is created with its normal vector towards the detected object. The better a receptor is aligned with the plane, and thus it is perpendicular to the detected object, the greater the context value it generates.

Pursue/Evade: Pursue and evade are similar to seek and flee. Pursue creates objective values towards the target object and evade away from it. The

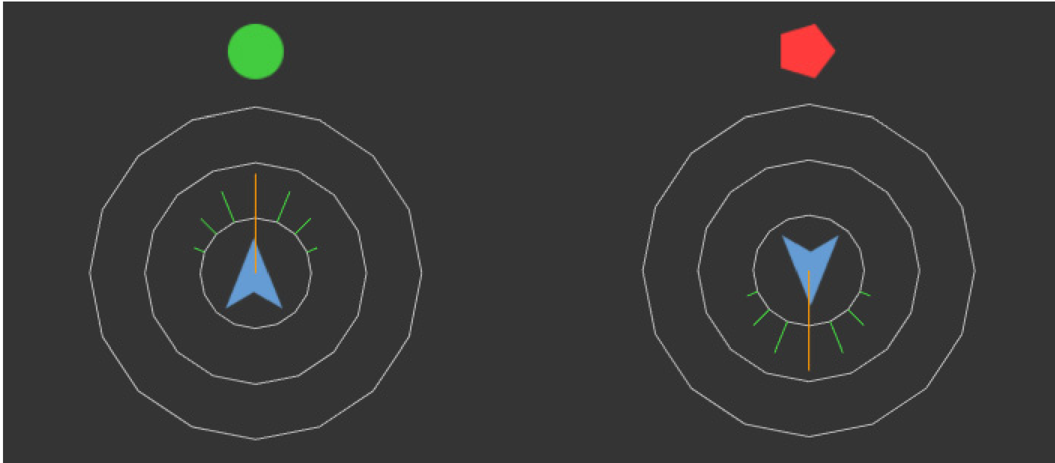


Figure 3.1.: Illustration of the mapping of seek(left) and flee(right), both mapped to the interest objective.

Parameter	Boundaries
Magnitude Multiplier	[0,10]
Sensitivity Offset	[-90,90]
Inner Radius	[0,5]
Outer Radius	[5,50]
Prediction Magnitude	[0,10]
Value Mapping	[0.33,3]
Radius Mapping	[0.33,3]

Table 3.1.: Parameters of context steering behaviors and their boundaries

difference to seek and flee is that not the actual position of the target object is used but a predicted future position along its movement direction.

General parameters

Table 3.1 shows parameters that all behaviors have in common. Next to the parameter name, the boundaries are displayed that are needed since bounded simulated binary crossover is used.

Magnitude Multiplier: The behavior-specific objective value is multiplied by this magnitude to amplify or weaken the effect of the behavior. This value affects how much influence a behavior has compared to other behaviors. It ranges from 0 to 10. At 0, the behavior can be seen as not active. The upper bound was chosen in a way that the behaviors can be adjusted to be

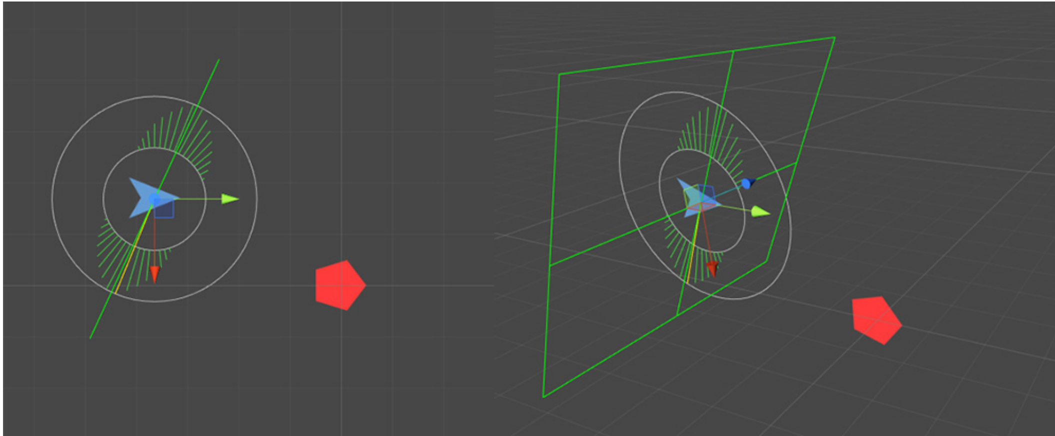


Figure 3.2.: The avoid behavior for a 2D scenario (left) and the plane that is constructed for context value generation (right).

very sensitive to its surrounding objects. Figure 3.3 illustrates the effects of different Magnitude Multiplier values for an inverse linear mapping. The higher the magnitude, the faster the context value reaches its maximum. A value of 0.5 will only give a maximum context value of 0.5, and thus, at a Magnitude Multiplier of 0.0, the behavior has no effect. At a value of 1.0, the maximal context value is reached at the inner radius. A value of 2.0 or 10.0 leads to a maximal context value at half or a tenth of the detection range.

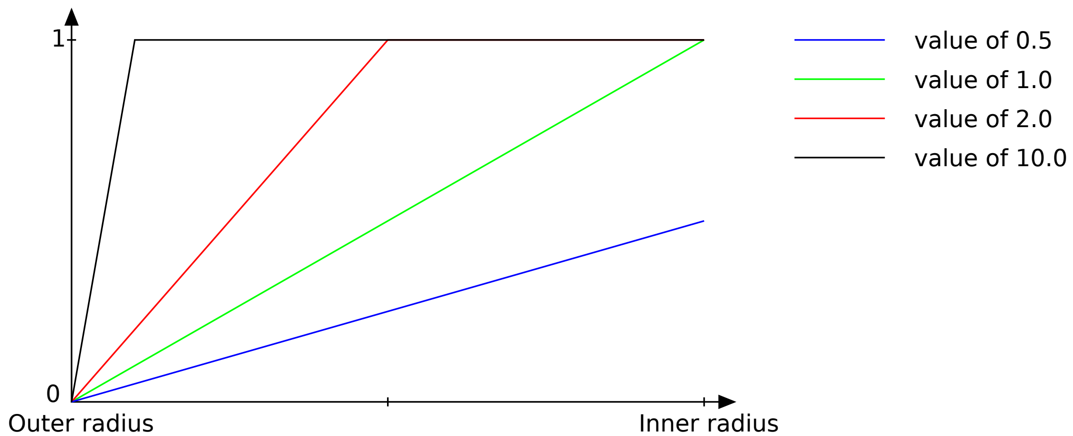


Figure 3.3.: The resulting context values for different magnitude multiplier values, plotted for an inverse linear mapping.

Sensitivity Offset: In subsection 2.1.3, it is described that the angle between the vector towards an object and the receptor vector affects how large the

objective value for this receptor is. The threshold that decides the maximum angle to have a minimal effect is by default at 90. The sensitivity offset is added to this threshold and thus changes how many receptors are affected by a behavior.

Inner Radius: The inner radius is the distance, after which the detection of objects starts.

Outer Radius: The outer radius is the maximal distance to which objects are detected. Together with the inner radius, it defines the range in which objects are detected and mapped into the context map.

Prediction Magnitude: The agent's point of perception is projected along its current movement direction by a fixed offset defined by this value.

Radius Mapping: The distance of an object that is detected by the agent needs to be mapped to a value between 0 and 1 based on the inner and outer radius. Therefore an Uniform Rational Quantization (URQ) mapping is used, which is configured by this value. How the URQ-mapping works can be seen in subsection 3.2.2.

Value Mapping: The objective values are not only mapped by the distance between the inner and outer radius but also by the angle difference to the receptor vector. For an inverse mapping that is used in this work, the objective value is maximal in terms of value mapping if the angle difference is zero. The objective value gets lesser, the closer the angle difference is to the threshold. The actual mapping is again done by an URQ-mapping that is configured by the value mapping value.

Behavior specific parameters

Seek/Flee: Seek and Flee have no additional parameters.

Avoid:

Parameter	Boundaries
Plane Bend	[0,90]

Plane Bend: Plane bend rotates the plane towards the detected object. At 90 degrees, the plane is rotated so much that receptors that point towards the detected object are aligned with the plane and generate the greatest context values.

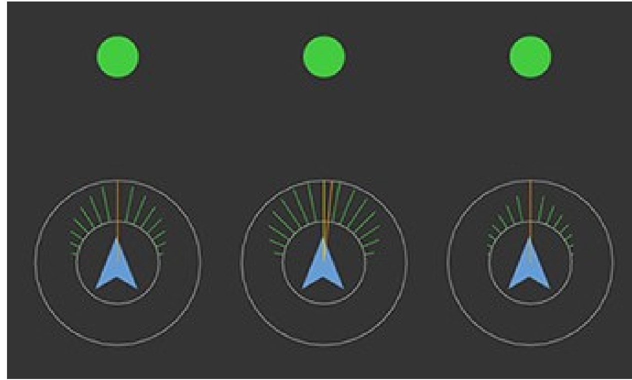


Figure 3.4.: The effects of different mapping types for value mapping. From left to right: inverse linear, inverse quadratic, inverse square root.

Pursue/Evade:

Parameter	Boundaries
Max Prediction Time	[0,5]

Max Prediction Time: The position of the target object is predicted based on its current position, velocity, and maximum prediction time (this value).

Other Parameter

Context:

The context module is the central part of the context steering plugin. Here, all the context data from each behavior is gathered and combined to a final decision. Next to these processing aspects, the context module also has a parameter that is part of the encoding.

Parameter	Boundaries
Objective Constraint	[0,1]

This parameter is based on the ε-constraint explained in subsection 2.1.4. For each objective that is not marked as the main objective, a parameter is created that defines the constraint value for this objective till which a solution is still valid. For the scenarios in this work, there are the two objectives interest and danger. Interest is the main objective, so only a constraint value for danger exists.

Parameter Interplay

While calculating the context values and finding the optimal movement direction, there are some sets of parameters that influence each other in a way that is not directly obvious. This part aims to explain the interplay of these parameter sets.

Magnitude Multiplier, Radius mapping, Inner Radius, Outer Radius: As already explained, the Inner and Outer Radius define the detection range in which an agent is able to sense its surroundings. The Magnitude Multiplier decides how big the maximum context value is and how fast it can be reached when objects get closer to the agent, as shown in Figure 3.3. Moreover, the Radius Mapping affects the shape of the mapping function that is used to map all values between minimal and maximal context value.

Sensitivity Offset, Value Mapping: A similar interplay holds for these values. Value Mapping defines how the context values are mapped towards neighboring receptors. For an inverse mapping that is used, receptors that point towards the detected object get a maximal context value, and the further the receptor points away from the object, the lesser is its context value until a given threshold is reached and the receptor is not affected anymore. The Sensitivity Offset is exactly affecting this threshold, and thus it determines how far the mapping goes and how many receptors are affected.

3.2.2. Adjustments

Besides the use of the Polarith plugin, some adjustments were made to make it usable for the used evolutionary algorithm.

Mapping

The plugin itself provides predefined settings for the different mapping types (value and radius mapping). These are linear, squared, square root, and their inverse forms. Those predefined settings have two issues why adjustments were made. First, they need to be encoded as categorical genes, whereas all other values are real-valued. So all evolutionary operators would have to deal with mixed encoded individuals. Second and more important, the change in behavior when switching between different mapping types is too big to ensure a continuous fitness landscape without any jumps in it. This would lead to a bad optimization behavior since the evolutionary algorithm can no

longer accumulate small changes to increase the fitness of an individual. That is why the predefined mapping types are replaced by the Uniform Rational Quantization (URQ) mapping [33].

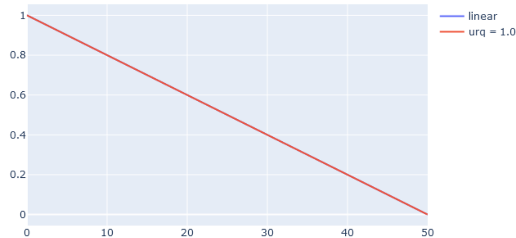
$$value_{mapped} = \frac{urq \cdot (value - min)}{urq \cdot (value - min) - (value - min) + (max - min)} \quad (3.2)$$

This is a modified version of the URQ-mapping that is able to map values in an arbitrary range, whereas the original function mapped values in a range from zero to a maximal value. What this function does is that it maps a value in a range from min till max to a value between zero and one, where min becomes zero and max becomes one. How the values in between are mapped is defined by the urq value. A value of one is equal to a linear mapping. A value greater one resembles a square root mapping, and a value between zero and one has similarities with a squared mapping.

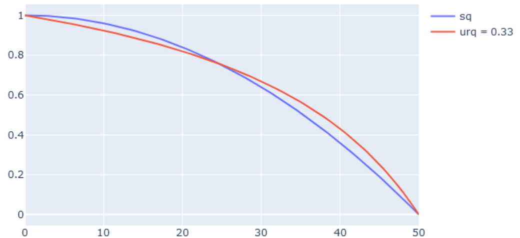
Instead of the predefined mapping types, the URQ-mapping can be used. To be more precise, the inverse URQ-mapping was chosen since the inverse mapping types are more appropriate for these context steering tasks. Figure 3.5a shows the mapping for a urq value of 1.0. This is equivalent to an inverse linear mapping. Because the predefined mapping types serve as a baseline, the URQ-mapping shall try to cover the same range of mapping possibilities. In Figure 3.5b, a value of 0.33 was chosen so that the mapping resembles an inverse squared mapping, and Figure 3.5c shows a mapping with a value of 3.0 which is similar to an inverse square root mapping. So that in total, the URQ mapping covers a range from 0.33 to 3.0 and is an adequate replacement for the predefined mapping types without causing big jumps in fitness when changing this value slightly.

Controller

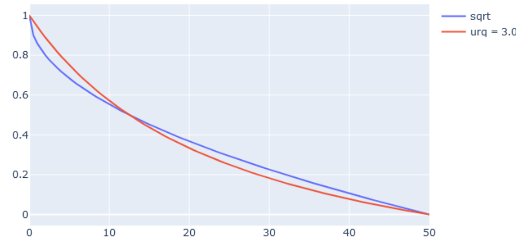
How the agent moves in a test scenario is not only crucial for the resulting behavior and movement of the agent, it is also essential for a reliable and repeatable evaluation of the algorithm. Unity offers several options on how to move characters and objects in its scenes. The easiest way is to set the position of an object directly. It is like to teleport the object to a different position. If the distance is small enough, it looks like the object moved on its own. Unity offers other approaches that are more realistic ways to move agents around.



(a) URQ value of 1.0 compared with an inverse linear mapping



(b) URQ value of 0.33 compared with an inverse squared mapping



(c) URQ value of 3.0 compared with an inverse square-root mapping

Figure 3.5.: URQ Mapping

By applying a velocity to an agent, Unity calculates the agent's position in the next frame and moves the agent to this position. For an even more realistic movement, a force can be applied so that acceleration and friction are also simulated. Nevertheless, not only the kind of movement is important but also the reliability of the evaluation. For games, it is common use to design a controller in a way that its movement distance is based on the time needed till the next frame so that the game feels the same on different hardware setups. Unity and the Polarith AI provide those controllers. While these types of controllers are suitable for enjoyable gaming experiences, they are bad in terms of repeatability and evaluation across several different PCs. A controller that is independent of the processing time ensures that runs that

are made on different machines are still comparable because, on both systems, the simulation does the same.

The controller for this work is simply applying new positions to agents and objects. The calculation is time-independent, which means that between two frames, only a fixed distance will be traveled. Also, the maximal turning rate is limited to a fixed value to mimic a physics-based controller to a certain degree. The distance between two frames is 0.1 Unity units towards the agent's forward direction, and a maximum of 5 degrees turning angle was chosen.

3.3. Algorithm Choice and Design

Algorithm Choice

For this work, an algorithm needs to be chosen that is able to find optimal solutions. Preferably global optima should be found, but also good local optima are sufficient. An important property needs to be that the evaluation of single individuals can run in parallel. More preferable would be to have an algorithm that can handle multiple individuals in a single evaluation instance instead of running one evaluation instance per individual.

Out of the approaches presented in subsection 2.2.8, the family of evolutionary algorithms satisfies the most important aspect of running several individuals in one evaluation instance. This is important because the simulation that runs in Unity needs a lot of time so that every type of parallelization that can save time will be chosen. SMAC and irace seem to be unable to handle multiple configurations in one evaluation instance. From the presented types of evolutionary algorithms, the basic form was chosen. Since this is a first test to evaluate if parameter optimization for context steering is possible, the more advanced versions of evolutionary algorithms are not chosen because they may have additional properties that may distort this knowledge gain. Based on the flaws of the basic version that will be detected, a more advanced version that has special properties to counteract these flaws can be chosen in future work.

Out of the presented related work, SMAC and irace are both optimization algorithms that do not belong to the family of evolutionary algorithms. So a comparison of the chosen approach with those two would be evident. Unfortunately, both algorithms have some drawbacks, that is why they were not chosen.
SMAC uses a surrogate model on which the different configurations are tested one after the other to iteratively improve the prediction model. Such a fast surrogate model for evaluation does not exist, so that with the evaluation that was used for this approach, SMAC would need a considerable amount of time that is not given for this project.
A similar problem occurs for irace. According to the user guide [23], irace is able to use parallelization. But this is only running several configurations in parallel, whereas with an evolutionary algorithm, a whole generation can be evaluated in parallel. As long as there is no possibility to evaluate several configurations in one irace instance run, this approach would need too much

time. At the time of this writing, the author is not aware of such a feature of irace.

For particle swarm optimization, a similar runtime as for evolutionary algorithms is expected since several evaluations of solutions can happen in parallel in one run. Nevertheless, there is a reason why evolutionary algorithms are preferred over PSO. For PSO, a gradual fitness landscape without many jumps is needed. In best cases, the fitness landscape is known a priori. This is due to the working of PSO. The particles are attracted to the best-known solution. When reaching the best position, they still have a movement momentum that lets them overshoot. By using this mechanic they should find better solutions, since they come from an area with worse fitness and, given the fitness function is gradual, they should move towards an area with better fitness, unless the best-known solution is an optimum. An evolutionary algorithm, in contrast, searches in a stochastic way. Since for the context steering scenario of this work, the fitness landscape is unknown, it is expected to get better results using an evolutionary algorithm.

Encoding

The encoding is the first essential decision of an evolutionary algorithm since some of the other operators are affected by this. For this work, the encoding has to fit the needs and structure of a context steering agent that was described in subsection 3.2.1. All parameters of the different behaviors are floating-point values, so a purely real-valued encoding is chosen. The encoding of an individual is similar to the behavior structure of an agent. For each behavior, a set of genes is created where each gene represents a parameter. Additionally to the parameter value also its boundaries are part of the genes since other operators need those boundaries. Important to know is that the order of behaviors of an agent are also part of the encoding because the first parameter of the first behavior creates the first gene in the chromosome. Switching two or more behaviors would create a completely different genotype, and thus, a different individual. For evaluation, the gene values are taken from the chromosome and are applied to the behaviors. The first gene in the chromosome contributes its value to the first parameter of the first behavior, and so on. That is why the order of the behaviors defines a unique individual.

Algorithm Design

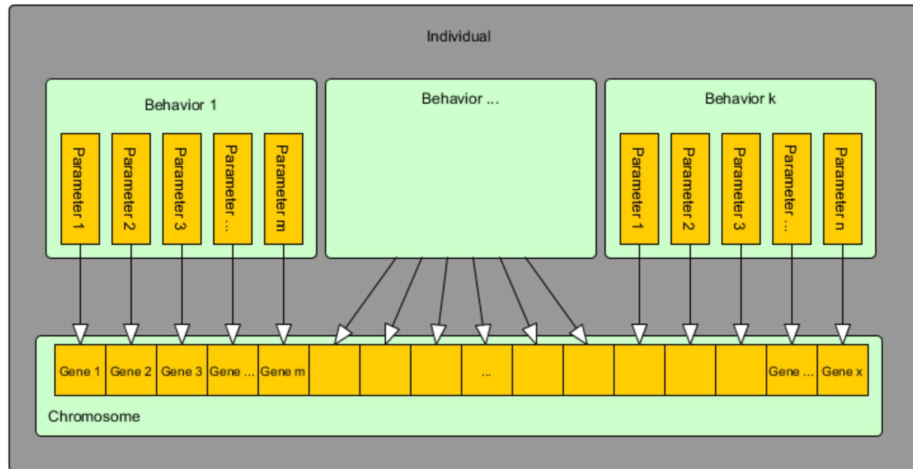


Figure 3.6.: Encoding of an individual for a context steering agent.

The implementation of the evolutionary algorithm that is used in this work is shown in Figure 3.7. It starts with a random initialization of the population that will be directly evaluated.

Tournament Selection is used to select the individual pairs that are used for crossover. Tournament selection with a tournament size of two was chosen to have a moderate selection pressure and thus to keep a diverse gene pool.

Simulated Binary Crossover was chosen as the crossover type since it is state of the art. The bounded version was chosen, and the implementation for that was taken from DEAP [12]. The pseudo-code taken from DEAP is shown in Algorithm 1.

Multi-gene Gauss Mutation is the mutation that was used in this work [3]. Again, the bounded version was chosen for this work. To make this a bounded approach, the random value is limited to a value so that the gene value is at maximum set to the gene boundaries. The Gauss distribution has a mean value μ of zero, and the standard deviation σ depends on the gene boundaries. For the Gauss mutation, a non-truncated version was chosen because limiting the mutation to an arbitrary small domain can lead to the problem that, in some cases, the mutation steps are too small to escape a local optimum.

Algorithm 1 Simulated Binary Crossover for a gene pair

1: **procedure** SBX($gene_1, gene_2, eta, x_l, x_u$)
2: $x1 = min(gene_1, gene_2)$
3: $x2 = max(gene_1, gene_2)$
4: $rand = random.random()$
5:
6: $beta = 1.0 + (2.0 * (x1 - xl)/(x2 - x1))$
7: $alpha = 2.0 - beta^{-(eta+1)}$
8: **if** $(rand <= 1.0/alpha)$ **then**
9: $beta_q = (rand * alpha)^{(1.0/(eta+1))}$
10: **else**
11: $beta_q = (1.0/(2.0 - rand * alpha))^{(1.0/(eta+1))}$
12: **end if**
13:
14: $c1 = 0.5 * (x1 + x2 - beta_q * (x2 - x1))$
15:
16: $beta = 1.0 + (2.0 * (xu - x2)/(x2 - x1))$
17: $alpha = 2.0 - beta^{-(eta+1)}$
18: **if** $(rand <= 1.0/alpha)$ **then**
19: $beta_q = (rand * alpha^{(1.0/(eta+1))}$
20: **else**
21: $beta_q = (1.0/(2.0 - rand * alpha))^{(1.0/(eta+1))}$
22: **end if**
23: $c2 = 0.5 * (x1 + x2 + beta_q * (x2 - x1))$
24:
25: $c1 = min(max(c1, xl), xu)$
26: $c2 = min(max(c2, xl), xu)$
27:
28: **if** $random.random() <= 0.5$ **then**
29: $gene_1 = c2$
30: $gene_2 = c1$
31: **else**
32: $gene_1 = c1$
33: $gene_2 = c2$
34: **end if**
35: **return** $gene_1, gene_2$
36: **end procedure**

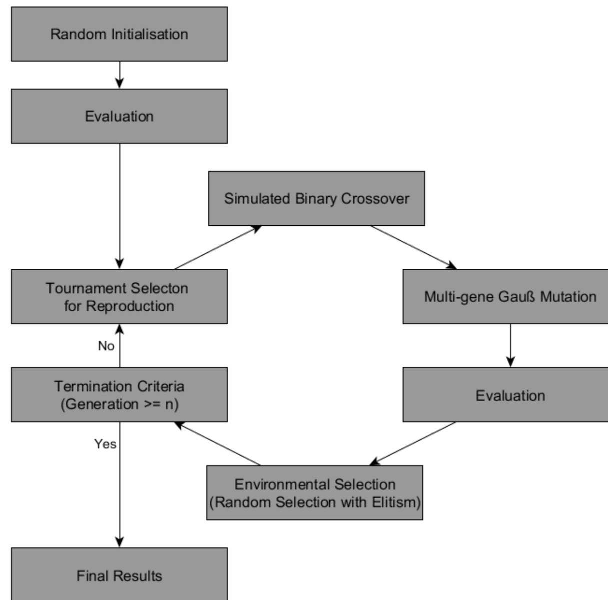


Figure 3.7.: Algorithm design: Structure of the chosen evolutionary algorithm.

The Environmental Selection is chosen to be a $(\mu + \lambda)$ - selection where random selection is combined with elitism. It was chosen this way to have a balance in taking the best individuals into the next generation by elitism and keeping a more diverse gene pool by selecting individuals at random.

The Termination Criterion is a maximal number of generation. Since no information about the expected solution quality is known, the simulation is limited by the number of generations.

3.4. Fitness Function

Fitness and the fitness function is one of the fundamental parts of an evolutionary algorithm since this function is essential for the quality of found solutions. Ideally, the fitness function is a representation of the task that needs to be solved or at least an approximation. For example, if the task is to shape a car in a way that it reduces air friction to a minimum, the fitness function should be a simulation that can calculate the air friction for an arbitrary car shape. In this work, the task is to find a parameter set that enables a context steering agent to navigate through a game scene while collecting interest objects,

avoiding danger objects, and to stick as close as possible to a user-defined path that the agent can not perceive. So the fitness function is a simulation of the game scene in which different measurements are made to evaluate how good the agent solved the task. How exactly these measurements look like is covered in the rest of this section.

$$f(\vec{x}) = w_T \cdot f_T(\vec{x}) + w_D \cdot f_D(\vec{x}) + w_P \cdot f_P(\vec{x}) \tag{3.3}$$

The fitness function has a similar structure like the task definition. $f_T(\vec{x})$ is the fitness part responsible for collecting interesting objects or moving towards a target point. $f_D(\vec{x})$ measures how good the agent avoids dangerous objects. $f_P(\vec{x})$ penalizes agents that move away from the user-defined path. w_T, w_D, w_P are the weights of each fitness part. Each fitness part is designed in a way that the whole evolutionary algorithm is a minimization problem. So for each fitness part, the ideal value would be zero.

Fitness by target

The goal of this fitness part is to tell the agent that it has to collect as many targets as possible, ideally all. T specifies the set of all targets objects, and U is a subset of T, namely all uncollected targets. When all targets are collected ($U = \varnothing$), the agent gets a perfect fitness score of zero for this part.

$$f_T(\vec{x}) = \sum_{t \in U} \frac{1}{2}(1 + p_d(A, t)) \tag{3.4}$$

with a distance based penalty p_d

$$p_d(A, t) = \frac{d(A, t)}{d(S, t)} \tag{3.5}$$

Where $t \in U$ is each target position in a set of targets that were not reached by the end of the evaluation, $d(A, t)$ is the minimal distance between agent A and target t during that evaluation, and $d(S, t)$ is the starting distance of the agent towards t.

For each target that is not collected, there is a distance-based penalty. How big the penalty is, depends on the starting distance towards that target and how

close the agents got towards the target while moving around. The distance-based penalty can have a maximal value of one since the agent achieved at least the starting distance, even if the agent moves straight away from the target afterward. If the agent gets closer to the target, the distance-based penalty will be lower. Nevertheless, there is a fixed penalty for not reaching a target. A fixed penalty will cause a jump in the fitness function, which could be considered a bad design decision. Nevertheless, this jump will only occur between getting very close to the target and reaching it. Since there can not be any improvement after reaching the target, the jump will not cause any harm to the optimization procedure. Also, there should be a considerable difference between only getting close to a target or actually reaching it.

Fitness by danger

The danger part of the fitness function is designed to teach the agent not to get too close to dangerous objects.

$$f_D(\vec{x}) = \frac{1}{n_d} \cdot \sum_{t_d \in T_D} \frac{Threshold - minDist(t_d)}{Threshold} \tag{3.6}$$

Where $Threshold$ is the radius an agent is allowed to move towards a danger object before getting a penalty, T_D is the set of events in which the agent moved inside the threshold radius, n_d is the number of danger objects in the scene, and $minDist(t_d)$ is the minimal distance between agent and danger object while the agent moves inside the threshold radius.

The penalty for one event is normalized by the threshold value so it can be at maximum one. Additionally, all events are normalized by the number of danger objects so that a penalty of one is made if the agent crashes into each danger object with the maximum penalty. Theoretically, a value greater one could occur since another event is created if the agent re-enters a threshold radius of a danger object it entered before. However, penalizing an agent, even more, when it relentlessly crashes into the same danger object, again and again, benefits the goal of teaching the agent to avoid danger objects.

Fitness by path

The last part of the fitness shall influence the algorithm to find parameters that also affect how the agent behaves. By placing a path close to danger objects,

the agent acts braver if it finds a way to follow the path and simultaneously avoid the danger objects. A path far away from dangerous objects should result in a more cowardly behavior under the assumption that the agent sticks to the path, and the training is not dominated by one of the other fitness parts.

$$f_P(\vec{x}) = \frac{\sum_{t_{start}}^{t_{end}} d(A_t, P)}{\sum_{t_{start}}^{t_{end}} d_{max}} \tag{3.7}$$

Where t_{start} is the starting time of the evaluation, t_{end} is the end time of the evaluation, $d(A_t, P)$ is the smallest distance from the agent's position at time t to the path, and d_{max} the maximum distance an agent shall have from the path.

The distance to the path is accumulated over the entire evaluation and normalized by a distance value that is considered as bad behavior for a specific scene. However, the agent can move further away, which increases the penalty even more. The closer an agent sticks to the path over the complete evaluation, the better is the fitness value. Ideally, the agent follows the path directly and gets a value of zero. Nevertheless, this scenario is not very likely since the paths used in this work are linear paths, and the movement of the agent has some limitations that will not allow the agent to make too big turns in one position.

Contradiction in Fitness

The described fitness parts are all designed in a way that the evolutionary algorithm shall try to minimize each of them. For most scenarios, this will not be possible since the individual fitness parts can work contrarily. For example, if an interest object and a danger object have the same position, the agent can either collect the interest object and also touch the danger object or avoid the danger object and also not collect the target object, but minimizing both objectives is not possible. The same goes for the path. If danger objects are places on the path or target objects are not on the path, the agent can either follow the path and touch danger objects or miss target objects or leave the path to avoid danger objects and collect target objects. So it could be said that the goal of the fitness function is not to minimize all of the fitness parts individually but to find a tradeoff where the sum of all fitness parts is minimal. How this tradeoff looks like can be influenced by the weightings of the different fitness parts. With a stronger path weighting, the agents should

stick more strictly to the path and ignore the danger and target objects to a certain degree. A stronger target weighting results in an agent that seeks for target objects, even if it has to leave the path for reaching them or get close to danger objects that are near the target. A stronger danger weighting will lead to an agent that leaves the path or also avoids target objects if the danger objects are on the path or too close to a target.

4. Benchmark Implementation

This chapter is about the implementation of the benchmark scenes for the experiments. Furthermore, adjustable parameters are described, and the scene variations for the robustness experiments are introduced.

4.1. General Settings

For the experiments, three different scenes were created. Each scene consists of an agent that is trained to solve a given task, an interest object, one or more danger objects, and one or more paths. The agent's task is to collect the interest object while avoiding the danger objects. Furthermore, it shall try to minimize the overall distance to the path while moving around. Sometimes following the path is contrary to avoiding dangerous objects because these are placed directly on or close by the path. By avoiding dangerous objects, it is meant to stay further away from these objects than a given threshold radius. For all experiments, this threshold is 2.0 distance units in the scene. The distance from the path that serves as normalization value is scene dependent and is stated for each scene individually.

4.2. Scenes

Scene 1

The first scene is a simple seek scenario shown in Figure 4.1 to demonstrate the qualities of context steering in a simple case. There, an agent, a danger object, and an interest object are placed on a straight line. Also, the desired path is illustrated. Since the scene is symmetrical, the path is mirrored so that one path is above and one path is below the danger object. The normalization distance for the path fitness is set to 5.0 distance units.

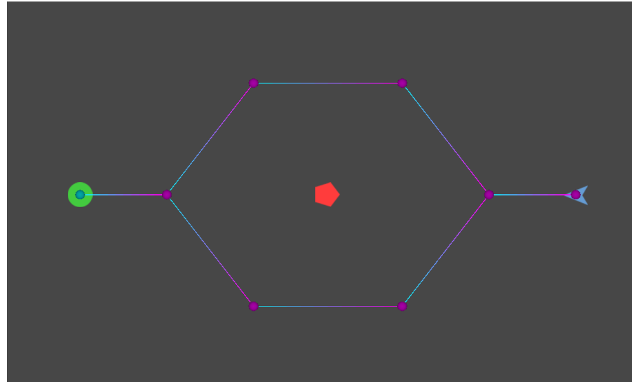


Figure 4.1.: A basic seek scenario for context steering.

Scene 2

Figure 4.2 shows the second scene. Here the simple seek scenario is extended by additional danger objects. This scene was created to test if an agent can be trained on several similar scenes to perform robustly on each of them. Therefore, the danger objects are randomly placed around their starting position, and some of them are deactivated at random. Out of these random samples, five variations were chosen on which the algorithm will be tested.

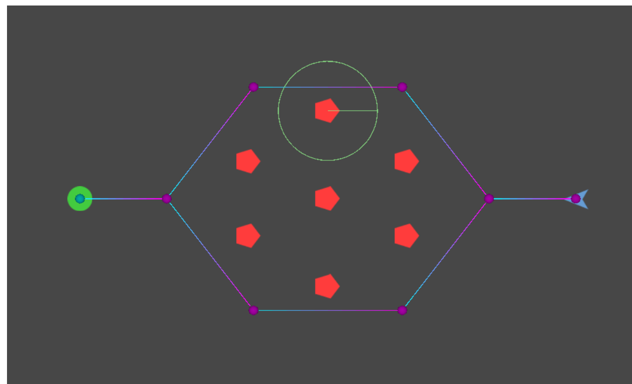


Figure 4.2.: A more complex seek scenario with several danger objects that are randomly place within a given radius.

Figure 4.3 shows the first variation of this scene. The remaining variations can be found in Appendix A. Some of these variations are similar to the simple seek scenario where the agent has to avoid the danger objects in a certain way to follow the path automatically, other have danger objects directly placed on the path, so the agent needs to leave the path or prefer the others path that

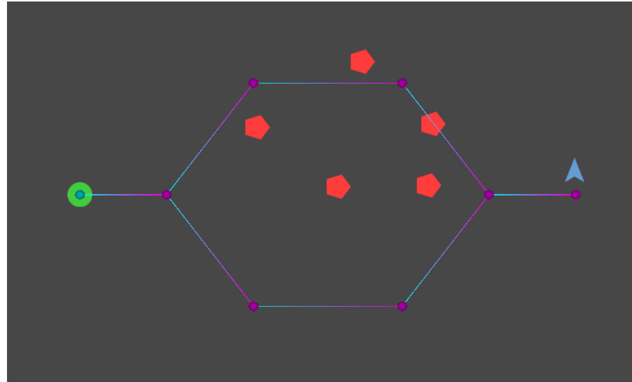


Figure 4.3.: First variation of the second scene.

is not blocked by danger objects. Figure A.1b shows a scene with only one danger object relatively far away from the center. This scene was chosen to see if an agent can find a solution for a scene with arbitrary less contextual information that still follows the path and if this solution performs equally well in the other variations.

Scene 3

Figure 4.4 shows a more complex scene to test if a parameter configuration can be found when moving danger objects are involved. All except the top danger object are oscillating horizontally around their starting position by a given distance and speed. They have alternating movement directions. If the first object starts moving to the left side, then the second moves to the right side. This continues until the second last object. The last danger object remains stationary. If by accident, an agent can avoid all oscillating danger objects by just moving forward, the stationary danger shall penalize this behavior. Here again, some variations were created regarding movement speed, traveling distance, and spacing between the danger objects.

Figure 4.4 also shows the standard setting for this scene. The danger objects move with a speed of 0.1 Unity units per frame over a distance of 10.0 units. Vertically they have a distance of 5.0 units to each other. For the other variations (see Appendix A), only the changed parameters are mentioned. Variation two has a 50 percent increased movement speed to see if faster moving objects still can be avoided. The third variation has the 50 percent increased speed and a wider moving distance of 20.0 units. In case the faster objects are a problem, this scene will show if the increased speed is the problem or the high

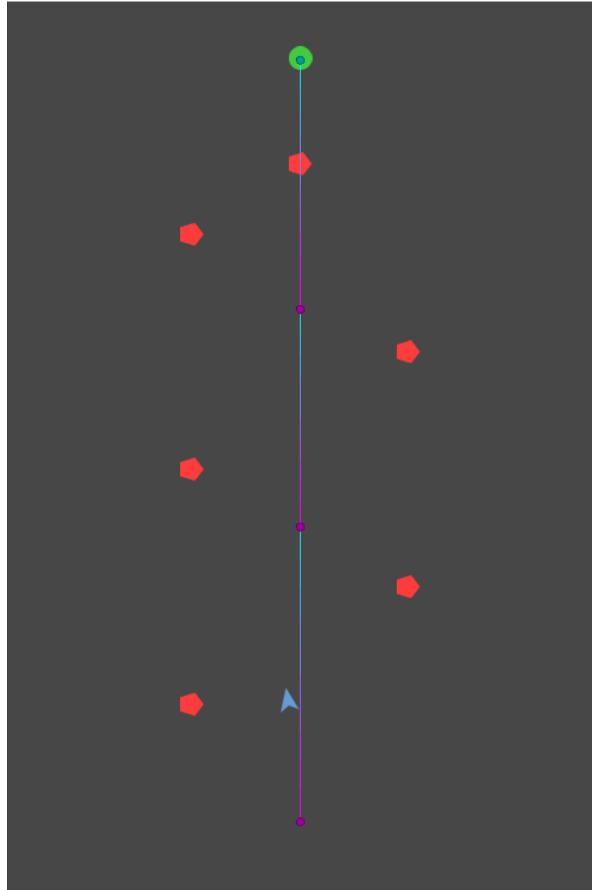


Figure 4.4.: Moving danger scenario

frequency of the objects because now this should be lower. In the fourth variation in Figure A.2c, the vertical spacing between the objects is reduced to 3.0 units. This gives the agent less space to navigate, and it needs to react faster. Furthermore, in Figure A.2d, all parameters are changed. At a spacing of 3.0 and a travel distance of 20.0, the objects move with 0.2 units per frame.

5. Experiments and Evaluation

The overall goal is to evaluate if an evolutionary algorithm can be used to optimize the parameters of a context steering agent. Since this is a superior goal, it will be part of all experiments.

5.1. Evaluation Non-Determinism

The simulation of an agent is not deterministic. There are slight differences between two runs so that the same agent gets different fitness values in different evaluation runs. This error seems to depend on the runtime of the simulation because between training where several agents run in parallel and reevaluation of a single agent the fitness difference can be much higher. Thus, for the results of the experiments, all individuals are reevaluated individually, and these fitness values are taken. To get an idea of the stochastic nature of an evaluation, an example agent was reevaluated 31 times. In training, it got a fitness score of 226.94. The mean value of the reevaluations is 233.70, with a standard deviation of 3.32.

5.2. Evaluation of an Agent

For each evaluation run, the agent and other movable objects are put to their starting positions. The agent has a fixed number of update steps to reach the interest object. If the interest object is reached, the run is considered as completed and directly ends. Otherwise, the evaluation ends when the maximal number of update steps is reached. At the end of the evaluation, the fitness score is calculated based on the movement of the agent during the run.

5.3. Settings

In this section, all parameters for the used evolutionary operators and general algorithm parameters are mentioned. The experiments in this work are all done with these settings. For other settings than these, it is expected that also the results will differ.

The evolutionary algorithm consists of a population of 100 individuals and runs for 100 generations. The mutation rate for the Gauss-mutation is 0.05 for an individual. A standard deviation of 0.2 of the gene boundaries was chosen. The crossover probability is 1.0, so each individual pair creates offspring, and the eta value of the simulated binary crossover is set to 20. The tournament size of the tournament selection is two. Each evaluation run is repeated three times to avoid assigning a good fitness value that occurred out of a stochastic behavior and does not represent the agent. Out of these three runs, the median value is chosen as the actual fitness. The best 10 percent of individuals of each generation are stored as elite and are taken to the next generation. There they can be used for recombination but are not mutated, so they stay as they are. For the robust setting, five different variations of a scene are created. The agent is trained on each variation for three runs. For each variation, the median is chosen as the fitness value of the variation. The total fitness value of the individual is the median value out of the five variation fitness values.

5.4. Experiment 1

5.4.1. Description

This experiment evaluates in what manner a different weighting of the fitness function will affect the resulting behavior. For this, agent configurations with hand-selected behaviors are chosen. Furthermore, the following weightings are chosen:

- An equally weighted fitness function where all parts have a weight of 1000.0.
- Three further weightings in which one part has a weight of 3000.0 and the two remaining keep the weight of 1000.0.

This results in 4 different weightings for each of the scenes. The equal weighting serves as a kind of baseline behavior. For the stronger path weighting, it is

expected that the agent will ignore danger objects that are placed near the path more than in other weightings. For a stronger danger weighting, the opposite is expected, in particular, that the agent strictly avoids danger objects even if it has to leave the path for a larger detour. Actually, the higher target weighting is expected to have no big impact in behavior for these scenes, since the scenes are designed that the agent is always able to reach the target if the parameter configuration is not completely wrong. So that the fitness for the target part should be zero, and thus the weighting has no effect on the final performance, compared to an equal weighting.

For the hand crafted agents following behavior configurations were chosen:
Scene 1:

- A seek behavior for detecting interest objects that are mapped to the interest objective.
- A seek behavior for detecting danger objects that are mapped to the danger objective.

Scene 2:

- A seek behavior for detecting interest objects that are mapped to the interest objective.
- A seek behavior for detecting danger objects that are mapped to the danger objective.

Scene 3:

- A seek behavior for detecting interest objects that are mapped to the interest objective.
- A seek behavior for detecting danger objects that are mapped to the danger objective.
- An avoid behavior for detecting danger objects that are mapped to the interest objective.
- An evade behavior for detecting danger objects that are mapped to the interest objective.

5.4.2. Results

Scene 1

Table 5.1 shows the individual fitness parts for the different weightings of the median solution out of 31 runs of scene 1. In all weightings, the agents are able to reach the target while avoiding the danger object. Because of this, the agents only focus on optimizing the path fitness. That is why all weightings result in similar fitness values. Figure 5.1 illustrates that all agents have similar behavior and choose only slightly different paths. Besides the fitness values of the median solution and a visual check for similarity, also a statistical test was made. The results of the Mann-Whitney U test in Table 5.2 show the p-value for a two-sided test for all possible weighting combinations. The Mann-Whitney U test checks if the given samples are out of the same distribution. A p-value of zero indicates different distributions, while a p-value of one indicates the same distribution. Most of the weighting pairs have a value above 0.5. Together with similar fitness values, this means that for scene 1, no weighting is superior to the others.

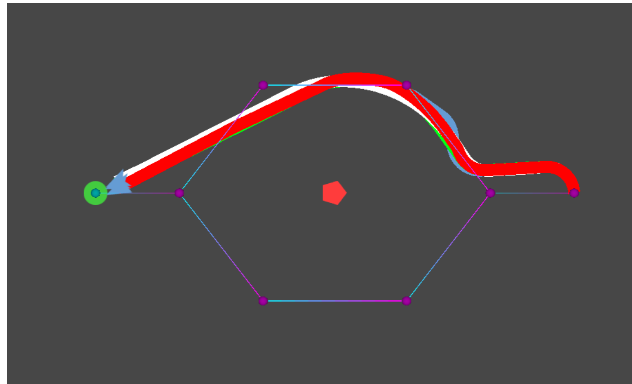


Figure 5.1.: Trails for agents trained on different weightings: equal is white, target is green, path is blue, danger is red.

Objectives	Weightings			
	w_{equal}	w_{target}	w_{path}	w_{danger}
$f_T(\vec{x})$	0.0	0.0	0.0	0.0
$f_P(\vec{x})$	92.62	93.42	93.59	91.89
$f_D(\vec{x})$	0.0	0.0	0.0	0.0
total	92.62	93.42	93.59	91.89
weighted	92.62	93.42	280.78	91.89

Table 5.1.: Results for different weightings of scene 1

p-value	w_{target}	w_{path}	w_{danger}
w_{equal}	0.42	0.55	0.72
w_{target}		0.89	0.75
w_{path}			0.89

Table 5.2.: Mann-Whitney U test for different weightings of scene 1

Scene 2

In scene 2, similar results are observed. In Table 5.3 the fitness parts of the median run are shown. Again, the targets are collected while avoiding all danger objects, which means that only the path fitness is optimized at the end of the training. Similar fitness values are achieved and an only slightly different path is taken, as shown in Figure 5.2. Also, the Mann-Whitney U test shows that most of the weighting combinations have a high p-value. The result is similar to scene one so that no weighting seems to be superior to the other weightings.

Objectives	Weightings			
	w_{equal}	w_{target}	w_{path}	w_{danger}
$f_T(\vec{x})$	0.0	0.0	0.0	0.0
$f_P(\vec{x})$	98.25	97.62	96.75	95.95
$f_D(\vec{x})$	0.0	0.0	0.0	0.0
total	98.25	97.62	96.75	95.95
weighted	98.25	97.62	290.25	95.95

Table 5.3.: Results for different weightings of scene 2

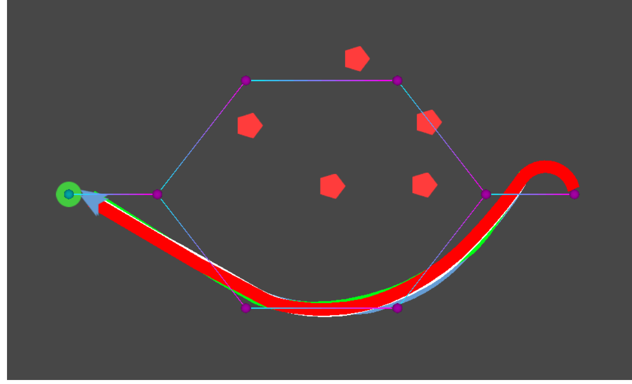


Figure 5.2.: Trails for agents trained on different weightings: equal is white, target is green, path is blue, danger is red.

p-value	w_{target}	w_{path}	w_{danger}
w_{equal}	0.93	0.70	0.33
w_{target}		0.86	0.48
w_{path}			0.83

Table 5.4.: Mann-Whitney U test for different weightings of scene 2

Scene 3

Scene 3 seems to create different movement patterns for different weightings. In terms of fitness this can be seen in Table 5.5. Most obvious is that the path weighting ignores more danger objects to be close to the path, and the danger weighting avoids danger objects at all costs and sacrifices an efficient path. The target weighting seems to avoid more danger objects at the cost of the path compared to the equal weighting. Only the agent of the danger weighting is able to completely avoid all danger objects, but this at the cost of a very high path fitness. This is also visible in Figure 5.3. The reference path is in the middle of the picture. The stronger path weighting is indicated by the blue line, which sticks to the reference path except for the last and stationary danger object. It seems that the moving danger objects are mostly ignored. The equal and target weighting (white and green line) tried to find a tradeoff between avoiding danger objects and being relatively close to the path. The danger weighting, in contrast, is mainly focused on avoiding danger objects by making a large turn around all danger objects at once, far away from the path indicated as a red line.

When testing if the fitness values were sampled from similar distributions, the Mann-Whitney U test shows that, in most cases, there are no similarities. The target weighting shows similarities with the path weighting and a small overlap with the equal weighting, but in general, it seems that the weightings create different results in scene 3.

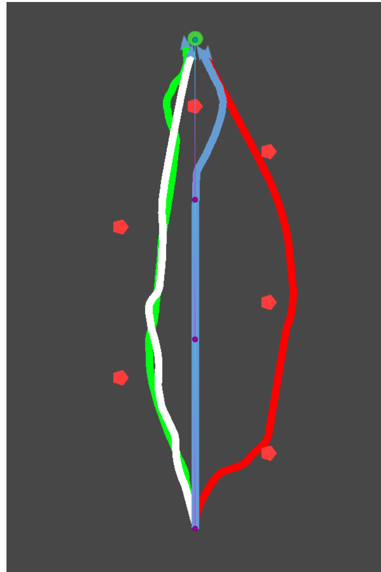


Figure 5.3.: Trails for agents trained on different weightings: equal is white, target is green, path is blue, danger is red.

Objectives	Weightings			
	w_{equal}	w_{target}	w_{path}	w_{danger}
$f_T(\vec{x})$	0.0	0.0	0.0	0.0
$f_P(\vec{x})$	174.40	285.82	31.99	412.64
$f_D(\vec{x})$	140.06	69.51	346.73	0.0
total	314.46	355.33	378.72	412.64
weighted	314.46	355.33	442.71	412.64

Table 5.5.: Results for different weightings of scene 3

p-value	w_{target}	w_{path}	w_{danger}
w_{equal}	0.22	0.02	0.005
w_{target}		0.48	0.07
w_{path}			0.15

Table 5.6.: Mann-Whitney U test for different weightings of scene 3

5.4.3. Interpretation

The results of this experiment show that different weightings of the fitness function can affect the movement behavior of the trained agent only if the scene is designed in a way that the individual fitness part work contrary to each other.

5.5. Experiment 2

5.5.1. Description

The next experiment evaluates how much knowledge is needed to enable the evolutionary algorithm to create successful context steering agents. In one case, the algorithms shall optimize hand-selected behaviors that are sufficient from an expert point of view. In the other case, the algorithm needs to deals with the whole set of behaviors where one would suggest that some of them work contrary to the given target behavior.

For the tests with all behaviors, the agents look the same in all three scenes. The following behaviors are used: seek, flee, avoid, pursue, and evade. A behavior can map the found context values to each of the two objectives: interest and danger. Furthermore, the objects that will be detected to create context values can be classified into two groups. Respectively to the objectives, these are interest and danger objects. If each object type is mapped to each objective, there are four variations for each behavior. That makes in total 20 different behaviors.

In general, it is expected that for simple scenes, it is easier to estimate what behaviors are used to fulfill the task. So that for the simple scenes (1 and 2), the handcrafted agents will perform equally well as the agents with all behaviors. For the more complex scene 3, a more advanced movement behavior is desired,

and this would need a broader combination of behaviors. It is not clear if the behaviors for the handcrafted agent are sufficient, and thus it is expected that the agent with all behaviors will perform better.

Out of the results from experiment 1, the weighting for scene 1 and 2 are set to an equal weighting. Since there is no real difference between the weightings, any other weighting would also be acceptable. For scene 3, the higher danger weighting is chosen since this was the only weighting in which the agent avoided all danger objects. These weighting settings are also used for the third experiment.

5.5.2. Results

Table 5.7 shows the fitness values of all scenes for the different behavior combinations. Surprisingly the agent with all behaviors is better than the handcrafted agent in scene 1. In scene 2, it is worse, although the task seems to be not that different from scene 1. In Scene 3, the agent with all behaviors is better, which was expected. The agent configurations with all behaviors are also able to avoid all danger objects in all three scenes.

	hand-selected behaviors	all behaviors	$f_{T_{all}}(\vec{x})$	$f_{P_{all}}(\vec{x})$	$f_{D_{all}}(\vec{x})$
Scene 1	92.62	65.00	0	65.00	0
Scene 2	98.25	191.58	0	191.58	0
Scene 3	412.64	236.18	0	236.18	0

Table 5.7.: Comparison of agents with hand-selected vs. all behaviors

These results are statistically significant as shown by the Mann-Whitney U test (see Table 5.8).

hand-selected vs all behaviors	p-value
Scene 1	4.0141e-07
Scene 2	0.0007
Scene 3	0.0555

Table 5.8.: Mann-Whitney U test for different amount of behaviors

Most surprising for this experiment is that for scene 2, the hand-selected solution is better, although, in scene 1, the configuration with all behaviors

achieved better results. It is surprising because both scenes are similarly structured, and thus, a similar solution quality was expected. The taken path and the complete training distribution will be shown for a better understanding of this difference.

Scene 1

Figure 5.4 shows the path of the median solution that was found for agent configurations with all behaviors attached. Compared to the found path of the first experiment (see Figure 5.1), it is obvious why a better fitness score was achieved. The solution with all behaviors results in a better approximation of the user-defined linear path and thus reduces the fitness score created by the path component of the fitness function.

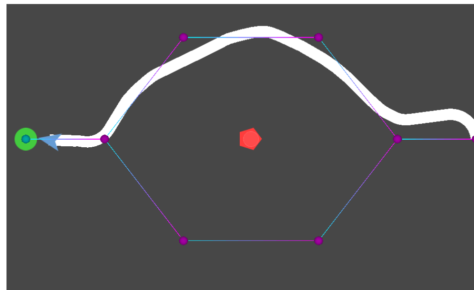


Figure 5.4.: Trail of an agent in Scene 1 that was trained with an equal weighting and all behaviors.

The fitness distribution illustrated in Figure 5.5 indicates that it is not just coincidence that the configuration with all behaviors achieves better results than the hand-selected configuration. Here, the fitness values of each run of both training methods are plotted in ascending order. In all runs, the configuration with all behaviors leads to a better solution than the hand-selected configuration.

Scene 2

In the second scene, it is the case that the configuration with all behaviors achieved worse results than the hand-selected configuration. The plotted path is illustrated in Figure 5.6. Compared with the path from the first experiment (see Figure 5.2), one can see where the major flaw of the found solution is. Directly at the start, the hand-selected solution turns counterclockwise, which is closer to the path. In contrast, the agent with all behaviors turns clockwise and increases the distance to the user-defined path.

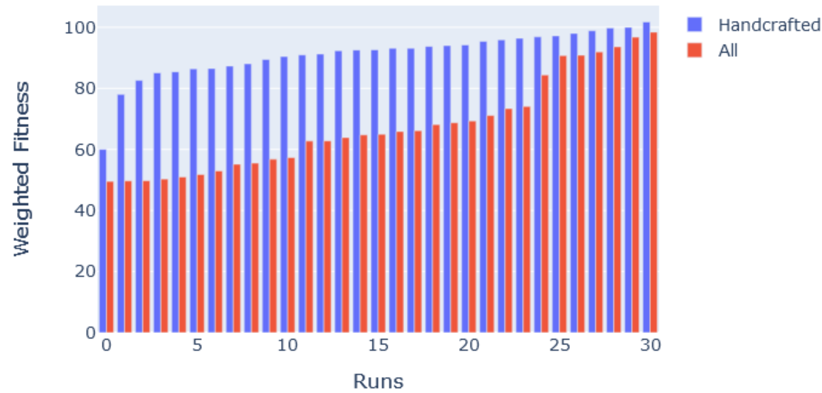


Figure 5.5.: Fitness distributions of 31 independent runs. Scene 1.

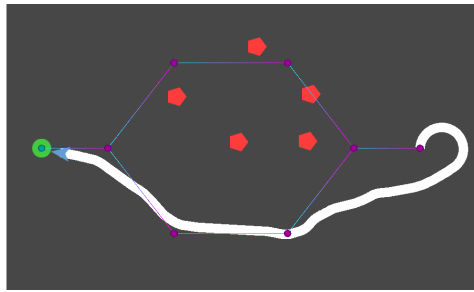


Figure 5.6.: Trail of an agent in Scene 2 that was trained with an equal weighting and all behaviors.

A look into the fitness distribution in Figure 5.7 shows that this problem is just a local optimal solution that was found until the end of the training. There are other training instances that found better solutions, even better than the best solution of the hand-selected configuration.

In general, one can see that the fitness distribution for the configuration with all behaviors has a high variance. This leads to the assumption that this scene has more local optima, and it is harder to train for a configuration with many behaviors.

Scene 3

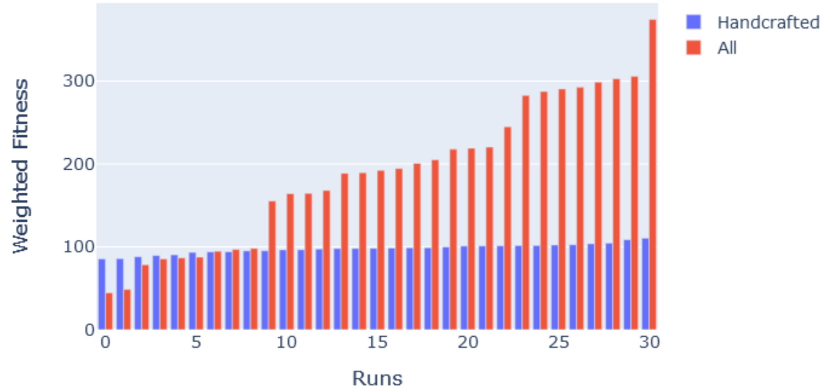


Figure 5.7.: Fitness distributions of 31 independent runs. Scene 2.

Figure 5.8 shows the found path for the configuration with all behaviors. Compared to the first experiment, the agent has found a more reactive behavior to avoid the moving danger objects at a smaller distance. This way, the agent is able to move more dynamically and closer to the user-defined path while still avoid the danger objects completely. Since this is an image of a dynamic scene, it may look like the agent violated the danger objective because some danger objects lie on the path. Nevertheless, at the time of moving along this path, the danger objects were at another position, and thus the agent's distance to the danger objects was large enough.

In the first two scenes, the hand-selected configuration had a more or less stable outcome with nearly always the same fitness score, and the configuration with all behaviors had a higher variance in the found fitness values. Here, both training methods have a high variance. This indicates that this scene is harder to train in general. Moreover, the fitness distribution shows that the configuration with all behaviors achieves overall better results.

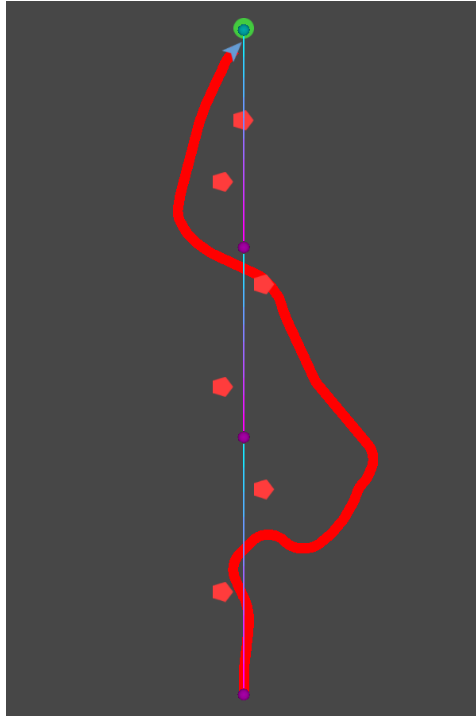


Figure 5.8.: Trail of an agent in Scene 3 that was trained with a higher danger weighting and all behaviors.

5.5.3. Interpretation

The results show that with more behaviors, a better configuration can be found. Sometimes this is done at the cost of more training time. In all scenarios, the target and danger part of the fitness is minimized to zero, and only the path part is left for optimization. To follow the path, an agent can only make turns that result in a curve, but the user-defined path is linear with hard edges. So it is impossible to follow the path perfectly. However, with more behaviors, the agent is able to do it more precisely.

5.6. Experiment 3

5.6.1. Description

The third experiment evaluates how the robustness of an agent can be affected by the way it is trained. Therefore five different variations of scene 2 and 3 are made and used for the agent's training. For each variation of a scene, an agent

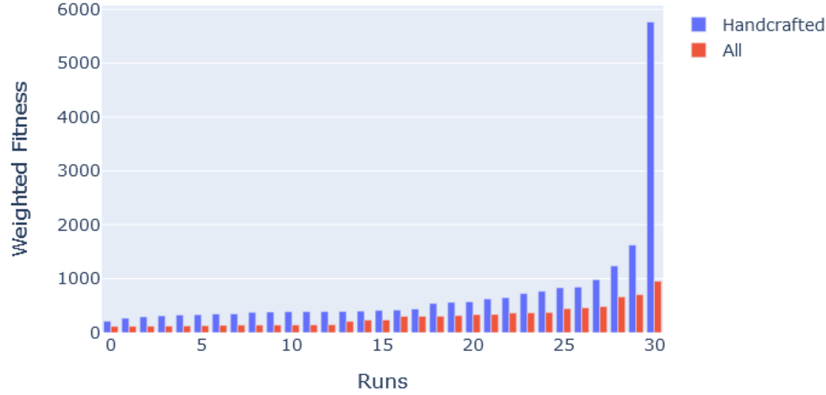


Figure 5.9.: Fitness distributions of 31 independent runs. Scene 3.

is trained solely on this variation. There is also an agent that is trained on all variations of a scene simultaneously. According to the robustness definition in section 3.1, this means that the agent is trained on each of the variations and get a fitness value for each variation. Out of these five fitness values, the median value is taken as the robust fitness of the agent. So for each scene, six differently trained agents exist. To compare them with each other, each agent is evaluated on each of the scene variations.

It is expected that the training for agents on a single scene variation should result in a quite good performance for this variation. The agent that is trained on all scene variations should at least have a good performance on three out of the five variations since the median value is taken as the fitness.

5.6.2. Results

The results of the robustness analysis are shown in Table 5.9 and Table 5.10. For both scenarios, the data shows the same results. The agents that are trained on one scene variation ($Agent1 - Agent5$) also have the best performance that was achieved in this variation($V1 - V5$), whereas the robust agent ($Agent0$) has a mediocre performance in most of the variations. Averaged over all scene variations, the robust agent achieves the best performance.

Scene 2	V1	V2	V3	V4	V5	average
Agent 0	419.24	213.55	227.02	355.16	215.98	286.19
Agent 1	191.58	378.55	384.89	473.74	395.76	364.90
Agent 2	333.33	138.04	287.48	287.25	536.45	316.51
Agent 3	546.03	698.79	117.19	1494.47	618.01	694.89
Agent 4	343.66	238.16	1580.83	241.37	207.62	522.32
Agent 5	337.02	333.73	617.22	269.01	154.93	382.50

Table 5.9.: Fitness values of differently trained agents tested on all variations of scene 2. Agent 0 was trained on all variations.

Scene 3	V1	V2	V3	V4	V5	average
Agent 0	626.39	314.92	266.91	393.74	310.58	382.50
Agent 1	236.18	362.94	920.61	335.88	489.44	469.01
Agent 2	978.92	227.49	1359.89	890.53	1603.54	1012.07
Agent 3	592.04	1015.18	74.11	715.71	598.50	599.10
Agent 4	2540.85	1424.84	305.08	214.78	806.06	1058.32
Agent 5	1231.19	618.35	560.91	553.90	121.58	617.18

Table 5.10.: Fitness values of differently trained agents tested on all variations of scene 3. Agent 0 was trained on all variations.

5.6.3. Interpretation

This experiment shows that when agents are trained for a scene, they can get quite good at solving this particular task. However, when some changes were made in the scenes, this performance can drop dramatically. In contrast, an agent that is trained on several scene variations robustly can yield better results in changing environments. This shows that with the presented type of robustness, a generalization effect can be achieved, at the cost of a performance decrease across all scenes.

6. Conclusion and Future Work

6.1. Conclusion

In this work, context steering is successfully combined with evolutionary algorithms to create a system that is able to find parameter configurations for context steering agents. Therefore, an encoding that fits the needs of context steering was developed. To support this encoding, some modifications were made to an existing context steering system. Some context steering parameters had to be transformed into real-valued terms to fit the chosen encoding of the evolutionary algorithm. Besides the encoding, a fitness function was developed that incorporates the goals of context steering and also considers a possibility for a user to influence the results. Furthermore, a controller design was proposed to enable a reliable training of the agents that is independent of the computation speed and thus can be used across several hardware setups. To test the developed system, several benchmark scenarios were created to test different properties of context steering agents. A robustness criterion was developed to test if the found parameter configurations can also be used in a set of different scenario variations. In several experiments, it has been discovered that a different weighting of the individual fitness parts can influence the agent's behavior. Moreover, the system can deal with many behaviors and utilize them to achieve better results than with a set of hand-selected behaviors. The robustness analysis shows that agents that are trained explicitly for robustness perform worse on the individual scene variations than the agents that were trained only on this variation. However, they have better performance across all variations. In general, one could say that it is possible to use evolutionary algorithms for parameter configuration of context steering agents. The presented work shows that for simple scenarios, interesting results can be achieved, and specific requirements must be fulfilled to enable a user-guided evolution.

6.2. Future Work

Although these first results look very promising, there is still a lot of optimization potential. For the encoding, not all context steering parameters were chosen. For a first evaluation, only the most important and most promising parameters were chosen. A missing parameter, for example, is the way how behaviors are combined into a final context map. For this work, it was fixed to take the maximal context value. Other options are that the values of all behaviors are added, subtracted, multiplied, divided, or that the lowest value is taken. To use these options for an evolutionary algorithm, a continuous change between these options need to be found, similar to how it was done for the URQ-mapping and the predefined mapping types.

To create authentic but different agents in one training session, the weighted sum approach for the fitness function could be replaced by a multi-criteria optimization. In this way, each fitness part is optimized in a single run. Nevertheless, this still works only if the scene is designed in a way that the fitness parts work contrary to each other.

To guide the evolutionary algorithm towards a better understandable solution, another fitness term can be introduced that considers a kind of energy function in which some behavior parameters like magnitude, sensitivity offset, or radius would be penalized. This could lead to agent configurations with a reduced magnitude for behaviors with low impact, or behavior parameters that have no effect and get just some arbitrary values.

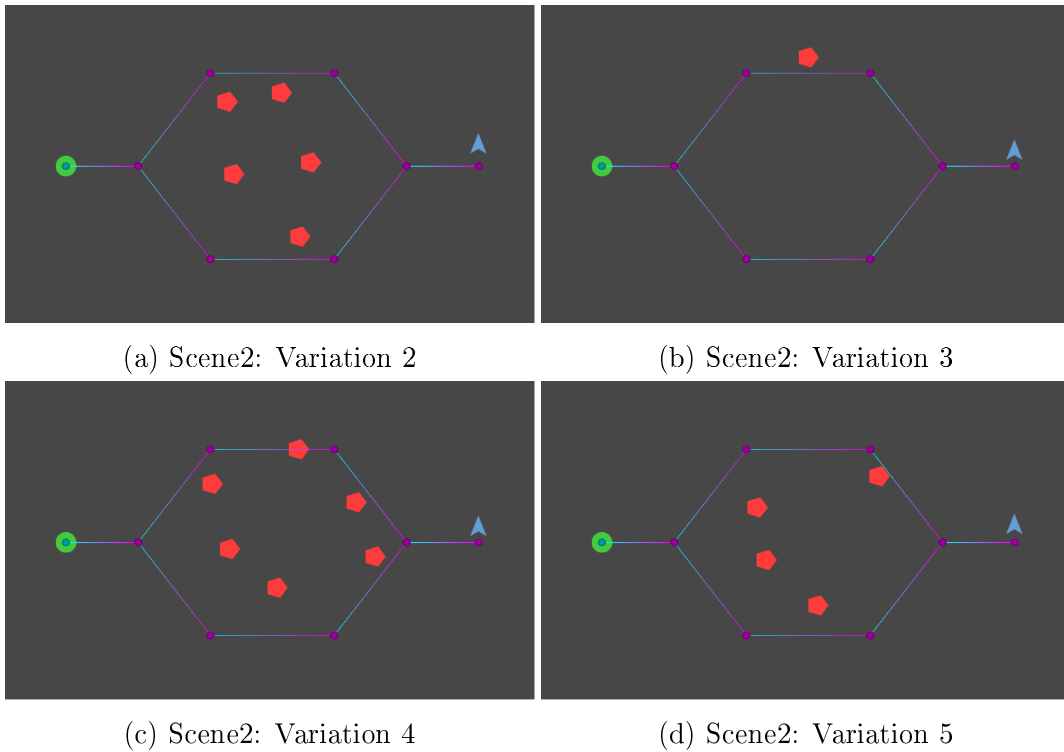
The robustness analysis already shows that this kind of training leads to agents that have a reliable performance across several scenes. Since the median was used, the training ignored flaws that were made in the two worst variations. Different robustness measurements could be tested to see if one of them leads to better overall performance. This could be a (weighted) mean value of all variations, the worst value, or a value that incorporates the standard deviation across all variations.

Besides further optimizations in the algorithmic parts of this work, the test scenarios can be optimized, too. The user-defined path was a linear one. Since the agent can only turn in curves, hard edges should be avoided for the user-defined path. A first suggestion for a better path representation is to use Bezier curves.

This work shows that this system is able to find solutions for static scenes with only a single target object. Also, a dynamic scene with fixed movement patterns was evaluated. For further experiments, other designs can be chosen to investigate the capabilities of this system even more. One point of interest would be to test if different paths for the same scene can be learned. This would require several training sessions for the same scene with different but reasonable paths. Since simple periodic movement patterns can be predicted, another experiment can test if an agent can be found that can predict more complex or arbitrary non-periodic movement patterns.

Appendices

A. Scenario Variations



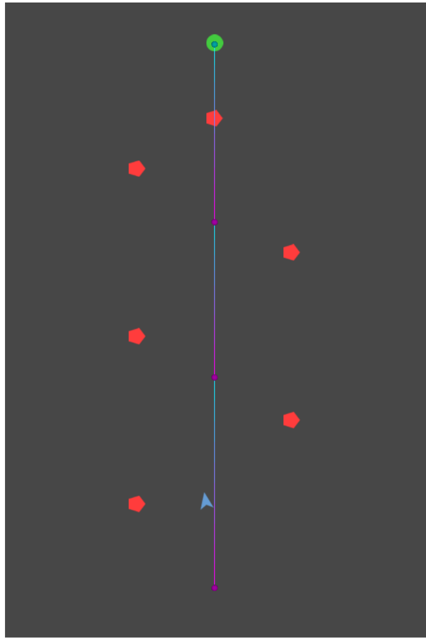
(a) Scene2: Variation 2

(b) Scene2: Variation 3

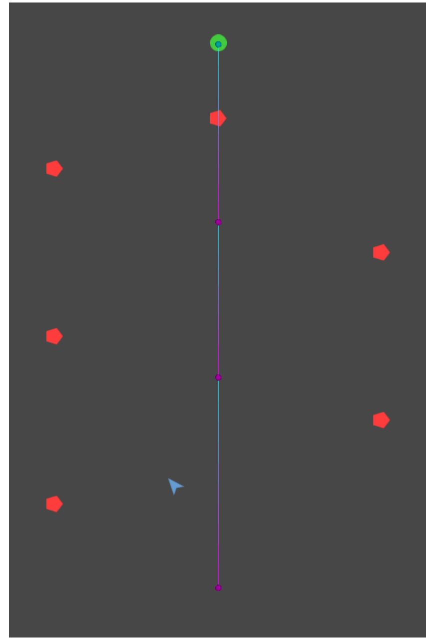
(c) Scene2: Variation 4

(d) Scene2: Variation 5

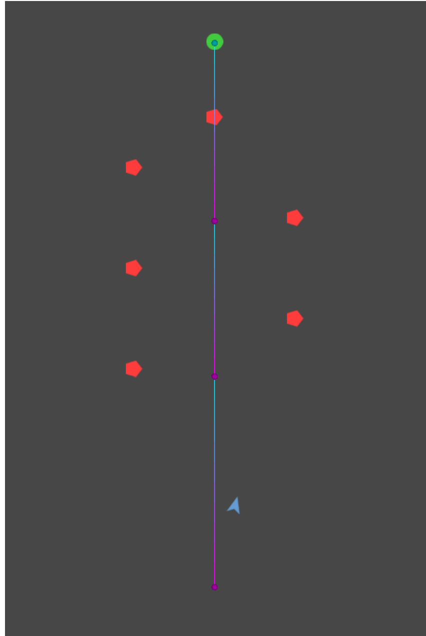
Figure A.1.: Variation 2, 3, and 4 of the second scene.



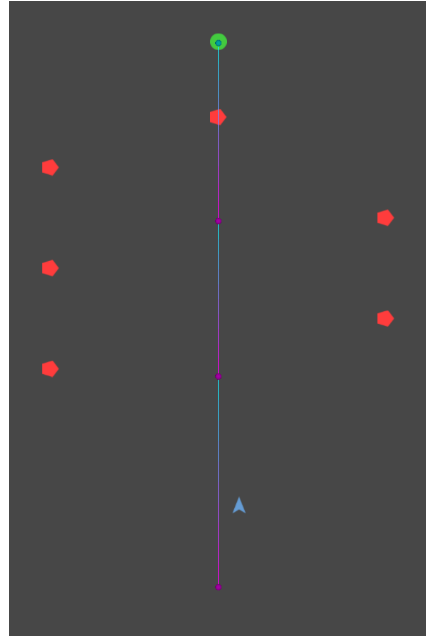
(a) Scene3: Variation 1 and 2



(b) Scene3: Variation 3



(c) Scene3: Variation 4



(d) Scene3: Variation 5

Figure A.2.: All variations of the third scene.

Bibliography

[1] Michael Affenzeller. Segregative genetic algorithms (sega): A hybrid superstructure upwards compatible to genetic algorithms for retarding premature convergence. *Int. J. Comput. Syst. Signal*, 2(1):16–30, 2001.

[2] Michael Affenzeller and Stefan Wagner. Sasegasa: A new generic parallel evolutionary algorithm for achieving highest quality results. *Journal of Heuristics*, 10(3):243–267, 2004.

[3] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[4] Chandan Banerjee, Sayak Paul, and Moinak Ghoshal. An evolutionary algorithm based parameter estimation using pima indians diabetes dataset. *International Journal on Recent and Innovation Trends in Computing and Communication*, 5(6):374–377, 2017.

[5] Aniket Bera, Sujeong Kim, and Dinesh Manocha. Efficient trajectory extraction and parameter learning for data-driven crowd simulation. In *Proceedings of the 41st Graphics Interface Conference*, pages 65–72, 2015.

[6] Glen Berseth, Mubbasir Kapadia, and Petros Faloutsos. Robust space-time footsteps for agent-based steering. *Computer Animation and Virtual Worlds*, 2015.

[7] Glen Berseth, Mubbasir Kapadia, Brandon Haworth, and Petros Faloutsos. Steerfit: Automated parameter fitting for steering algorithms. 2014.

[8] Jürgen Branke. Creating robust solutions by means of evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 119–128. Springer, 1998.

[9] Jürgen Branke. Efficient evolutionary algorithms for searching robust solutions. In *Evolutionary Design and Manufacture*, pages 275–285. Springer, 2000.

[10] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[11] Kalyanmoy Deb and Himanshu Gupta. Searching for robust pareto-optimal solutions in multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 150–164. Springer, 2005.

[12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[13] A. Fray. Context steering: behavior driven steering at the macro scale. In *Game AI Pro 2: Collected Wisdom of Game AI Professionals.*, pages 183–193. CRC Press, 2015.

[14] Pablo García-Sánchez, Alberto Tonda, Antonio J Fernández-Leiva, and Carlos Cotta. Optimizing hearthstone agents using an evolutionary algorithm. *Knowledge-Based Systems*, 188:105032, 2020.

[15] Anton Gerdelan and Carol O'Sullivan. A genetic-fuzzy system for optimising agent steering. *Computer Animation and Virtual Worlds*, 21(3-4):453–461, 2010.

[16] Stephen J Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming C Lin, and Dinesh Manocha. Pledestrians: A least-effort approach to crowd simulation. In *Symposium on computer animation*, pages 119–128, 2010.

[17] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[18] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*, 2010.

[19] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[20] M. Kirst. Multicriteria-optimized context steering for autonomous movement in games. *Master's thesis, Otto-von-Guericke University Magdeburg*, 2015.

[21] Marija Kranjčević, Andreas Adelmann, Peter Arbenz, Alessandro Citterio, and Lukas Stingelin. Multi-objective shape optimization of radio frequency cavities using an evolutionary algorithm. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 920:106–114, 2019.

[22] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. *Computational intelligence: a methodological introduction*. Springer, 2016.

[23] Manuel López-Ibáñez, L Pérez Cáceres, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package: User guide. In *Technical Report TR/IRIDIA/2016-004*. IRIDIA, Université Libre de Bruxelles, Belgium, 2016.

[24] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[25] Sedigheh Mahdavi, Shahryar Rahnamayan, and Kalyanmoy Deb. Center-based initialization of cooperative co-evolutionary algorithm for large-scale optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 3557–3565. IEEE, 2016.

[26] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[27] Diego Pérez-Liébana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M Lucas. Analyzing the robustness of general video game playing agents. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[28] Giacomo Persico. Evolutionary optimization of centrifugal nozzles for organic vapours. In *Journal of Physics: Conference Series*, volume 821, page 012015. IOP Publishing, 2017.

[29] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[30] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[31] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782. Citeseer, 1999.

[32] Priscila Saboia and Siome Goldenstein. Crowd simulation: applying mobile grids to the social force model. *The Visual Computer*, 28(10):1039–1048, 2012.

[33] Christophe Schlick. Quantization techniques for visualization of high dynamic range pictures. In *Photorealistic rendering techniques*, pages 7–20. Springer, 1995.

[34] Adarsh Sehgal, Hung La, Sushil Louis, and Hai Nguyen. Deep reinforcement learning using genetic algorithm for parameter optimization. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 596–601. IEEE, 2019.

[35] Polarith UG. Polarith ai documentation. `http://docs.polarith.com/ai/`, 2020. Accessed: 2020-02-28.

[36] AJ Umbarkar and PD Sheth. Crossover operators in genetic algorithms: A review. *ICTACT journal on soft computing*, 6(1), 2015.

[37] Niels Van Hoorn, Julian Togelius, Daan Wierstra, and Jurgen Schmidhuber. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation*, pages 652–659. IEEE, 2009.

[38] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[39] Chern Han Yong and Risto Miikkulainen. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX*, 2001.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Martin Wieczorek Magdeburg, June 16, 2020