Daniel Sumann, BSc

# Method of Hyperparameter Optimization of a Genetic Algorithm applied in Critical Scenario Generation for Autonomous Vehicles

**Master's Thesis**

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

**Graz University of Technology**

Supervisor

Univ.-Prof. Dipl.Ing. Dr.techn. Franz Wotawa

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl.Ing. Dr.techn. Franz Wotawa

Seiersberg-Pirka, January 2023

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

# Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

# Abstract

With the advent of autonomous driving, ensuring the correct implementation and safety of these systems becomes crucial. Relying solely on real-world driving might not be able to accumulate enough mileage; thus testing autonomous driving functions will need to be supported using simulations and software. This master's thesis will deploy a search-based approach for finding critical traffic situations in a simulation environment. This will be done by utilizing the dispersed search power of genetic algorithms. The genetic algorithm will be responsible for controlling vehicles in the simulation in a way, that will generate critical situations for the autonomous driving function. For the purposes of this master's thesis, the tested vehicle lacks complete autonomous driving capabilities. Instead, a behavior tree guides it through the simulated world.

After establishing the used tools and discussing their implementation, this master's thesis will focus on optimizing the control parameters of the genetic algorithm. Design of experiment, namely the Taguchi method was applied for that purpose. This optimized algorithm will be subsequently compared in different start scenarios to a genetic algorithm with control parameters taken from the literature as well as random search. The results of this evaluation show, that utilizing the Taguchi method is a very effective way of tuning a genetic algorithm on the given search problem. The optimized genetic algorithm displayed significantly better performance compared to random search and to a genetic algorithm which utilizes hyperparameters suggested by existing literature. Only in the case of a small number of actors, the performance between both genetic algorithms was similar.

# Kurzfassung

TODO...

# Acknowledgements

I would like to express my deep gratitude to my supervisor Univ.Prof. Dipl.-Ing. Dr.techn. Franz Wotawa for his guidance, feedback and support.

This master's thesis was a company collaboration with AVL List GmbH in Graz, so I thank my colleagues Florian Klück, Lorenz Klampfl, David Kaufmann and Gabriel Muresan for their shared expertise as well as numerous constructive discussions.

Finally, I thank my family and my partner Johanna for their support and patience throughout the entire process of completing this master's thesis.

# Contents

**Abstract** v

**1 Introduction** 1
1.1 Research Questions . . . 2
1.1.1 Research Question 1 . . . 2
1.1.2 Research Question 2 . . . 3

**2 Foundations** 5
2.1 Genetic Algorithm . . . 5
2.1.1 Chromosomes and Genes . . . 7
2.1.2 Population Size . . . 8
2.1.3 Selection . . . 8
2.1.4 Crossover . . . 9
2.1.5 Mutation . . . 11
2.1.6 Adaptive Control Parameters . . . 11
2.2 Behaviour Tree . . . 12
2.2.1 Control Flow and Execution Nodes . . . 13

**3 Implementation** 15
3.1 Traffic Manager . . . 15
3.1.1 Action Interface . . . 15
3.2 Genetic Algorithm . . . 17
3.2.1 Maximum Number of Generations . . . 18
3.2.2 Encoding . . . 18
3.2.3 Cost Function . . . 22
3.3 Behaviour Tree . . . 23
3.3.1 Structure . . . 23

**4 Hyperparameter Tuning** **25**
4.1 Simulation Setup . . . . . 25
4.2 Population . . . . . 26
4.2.1 Identifying Suggested Hyperparameter Settings from Existing Literature . . . . . 26
4.2.2 Comparison of Population Size . . . . . 27
4.3 Design of Experiment . . . . . 29
4.3.1 Taguchi Method . . . . . 30
4.3.2 Selection of a Suitable Standard Orthogonal Array . . 32
4.3.3 Result Analysis . . . . . 35

**5 Evaluation** **47**
5.1 Start Scenario 1 . . . . . 48
5.2 Start Scenario 2 . . . . . 49
5.3 Start Scenario 3 . . . . . 51
5.4 Start Scenario 4 . . . . . 52

**6 Conclusion** **55**
6.1 Research Questions . . . . . 55
6.1.1 Research Question 1 . . . . . 55
6.1.2 Research Question 2 . . . . . 56
6.2 Outlook and Future Work . . . . . 56
6.2.1 Additional Actions . . . . . 56
6.2.2 Oracle Functions . . . . . 57

**Bibliography** **59**

**Appendix A.** **69**

**Appendix B.** **71**

# List of Figures


3.1 Action Interface structure . . . . . 17
3.2 Time . . . . . 19
3.3 Time + NPC . . . . . 20
3.4 implemented behaviour tree . . . . . 24

4.1 mean and error bars per population size . . . . . 28
4.2 linear graph of $L_{16}(2^{15})$ taken from Yang and El-Haik [34] . . 34
4.3 modified linear graph . . . . . 35
4.4 percentage contribution . . . . . 39
4.5 Main Effects . . . . . 40
4.6 interaction plot . . . . . 41
4.7 elite selection comparison over GA generations . . . . . 44
4.8 elite selection comparison results . . . . . 45

5.1 start scenario 1: Default GA vs Optimized GA vs Random Search . . . . . 48
5.2 start scenario 1: comparison of GAs . . . . . 49
5.3 start scenario 2: Default GA vs Optimized GA vs Random Search . . . . . 50
5.4 start scenario 2: comparison of GAs . . . . . 50
5.5 start scenario 3: Default GA vs Optimized GA vs Random Search . . . . . 51
5.6 start scenario 3: comparison of GAs . . . . . 52
5.7 Start Scenario 4: Default GA vs Optimized GA vs Random Search . . . . . 52
5.8 start scenario 4: comparison of GAs . . . . . 53

1 start scenarios 1-2 . . . . . 69
2 start scenarios 3-4 . . . . . 70

# List of Tables


3.1 probability of actions . . . . . 20
3.2 Integer encoding: probability of parameters per action . . . . 21
3.3 Dictionary encoding: probability of parameters per action . . 22

4.1 summary of literature review . . . . . 27
4.2 population size results - mean over 5 repetitions . . . . . 28
4.3 control parameters (factors) with corresponding settings (levels) - (*uniform) . . . . . 32
4.4 $L_{16}(2^{15})$ Taguchi orthogonal array taken from Roy [35] . . . . 33
4.5 merging rules taken from Roy [35] . . . . . 35
4.6 Building 4 Level columns from 2 Level columns . . . . . 36
4.7 final version of orthogonal array . . . . . 37
4.8 ANOVA results . . . . . 38
4.9 S/N ANOVA results . . . . . 43

1 hyperparameter tuning: taguchi experiment results . . . . . 71
2 evaluation results Optimized GA . . . . . 72
3 evaluation results Default GA . . . . . 72
4 evaluation results Random Search . . . . . 72

# 1 Introduction

Automated driving has made considerable advancements in the last 15 years, yet significant technical challenges still persist [1]. Fully developed and integrated, it will have a lasting impact on mobility, road safety and society at large [2].

Automated driving systems are considered safety-critical and need responsible and thorough testing in order to mitigate traffic accidents and human harm. As outlined by Klück [3], the main requirements of automated driving systems are that they "must operate under all circumstances" and, crucially, "must be safe". Self-driving systems have the potential for harm, as their failure may lead to catastrophic accidents. Errors can be difficult to detect, due to the complexity of the environment, in which automated driving systems operate. High quality testing is subsequently needed in order to ensure safety on the road.

> *"Thoroughly validating and verifying automated or autonomous driving functions is inevitable for assuring to meet quality criteria for safety-critical systems." [4]*

According to Kalra et al. [5], achieving statistical safety for autonomous driving systems necessitates a significant number of miles driven. Accumulating this amount might not be feasible by driving solely in the real world. Klück [3] adds, that unguided mileage collection may provide only "limited quality assurance". During real world driving, the number of challenging situations is low, given most everyday situations can be driven without trouble.

Different complexity levels of driving automation are defined by the Society of Automotive Engineers [6] with a level 5 system being required to perform the entire task of diving completely without a human driver. Lower levels

take over only parts of the driving, such as parking and steering. Examples include driver support features such as Automatic Emergency Braking (AEB) or Automatic Lane Keep Systems (ALKS). Testing these lower levels is similarly vital.

Klück [3] proposed search based testing (SBT) is a valuable tool for improving the robustness and revealing systematic failures of automated driving systems. SBT generates inputs for a system under test in order to provoke some unwanted behaviour. It utilizes a problem-specific cost function that guides the search towards areas that have a high probability of failures. SBT can be applied to complex and large search spaces, where other, less focused, testing tools like full factorial or random testing might struggle.

Genetic algorithms (GAs) are a viable tool of search based testing and were already successfully utilized in the domain of automated driving, as evidenced by notable studies [4, 7, 8, 9]. This master's thesis will employ a genetic algorithm in order to generate critical driving scenarios for testing self-driving technology in vehicles. While generating these scenarios is the objective, the main task of the thesis will evolve around the implementation of the genetic algorithm as well as the optimization of its hyperparameters. In the following chapters, the vehicle under test is refereed to as 'EGO' vehicle, while all other road users are denoted as 'NPCs'.

## 1.1 Research Questions

The following two research questions will provide a guideline for this master's thesis. Both questions evolve around examining and comparing the performance of a genetic algorithm on generating critical scenarios for autonomous vehicles. The specifics of the actual cost function are explained in Chapter 3.

### 1.1.1 Research Question 1

> *Is a genetic algorithm suitable for generating critical driving scenarios compared to random search?*

While the mentioned papers provide strong suggestions that a genetic algorithm has a significant advantages over random search, a direct comparison is still crucial. Are situations possible where a genetic algorithm struggles to find critical scenarios compared to a random search approach?

### 1.1.2 Research Question 2

> *Can the performance of a genetic algorithm be improved by optimizing the control parameter using the Taguchi method?*

To tune the hyperparameter of the genetic algorithm, design of experiment, specifically the Taguchi method, was employed. In an effort to test, if significant performance improvements are possible, the optimized GA will be compared to a GA using control parameters as recommended by existing literature. This evaluation aims to understand, whether the Taguchi method can indeed lead to notable performance advancements for genetic algorithms.

# 2 Foundations

This chapter will provide an overview of both genetic algorithms as well as behavior trees. Both tools are required by the proposed approach. The genetic algorithm will search for critical situations by controlling all NPCs, while the behavior tree will be specifically applied to the EGO vehicle.

## 2.1 Genetic Algorithm

Pioneered by Holland [10], genetic algorithms emulate the process of natural selection and the darwinian principle known as 'survival of the fittest'. Genetic algorithms are a subclass of evolutionary algorithms (EAs) and explore the solution space using a population of individuals [11]. Each individual contains one chromosome, which serves as a candidate solution. Using genetic operations, the individuals mate among themselves and mutate independently.

The selection of an individual is determined by a fitness function, which defines the search problem. Each individual has a fitness value corresponding to the performance of its chromosome [12]. Individuals that perform poorly on the given problem die out, while individuals deemed 'strong' propagate. Genetic algorithms optimize iteratively, with each iteration referred to as a generation. Chromosomes consist of individual genes and form a solution to the search problem. Through successive generations using genetic operations, gene values and position are optimized, resulting in progressively improved solutions [13]. Generational genetic algorithms, where the entire population is replaced each generation, will be utilized in this master's thesis. In contrast, steady state genetic algorithms only replace a small fraction of the population at a time [13].

This search method, on the basis of biological principles, enables a global and dispersed search through its population, which avoids various shortcomings of local search techniques [14]. Particularly on challenging search spaces with multiple local optima, a GA is less prone to getting stuck on a suboptimal solution [12, 15, 16]. Genetic algorithms offer advantages for complex optimization and non-deterministic polynomial problems [17].

> *"GAs can find good solutions within a large, ill-defined search space, and can be readily adapted to a wide variety of search problems" [11]*

Grefenstette [14] further emphasis its ability to outperform gradient techniques on challenging problems with high-dimensional, noisy or discontinuous fitness functions by efficiently exploiting a "relatively simple selection mechanism." The primary advantage of a GA in tackling these problems stems from its capability to explore the search space using its entire population [17]. A basic GA, as proposed by Holland [10] can be defined as follows:

```
simple_genetic_algorithm()
{
    initialize_population();
    evaluate_population();
    for(int i = 0; i < num_of_generations; i++)
    {
        select_individuals_for_next_population();
        perform_crossover();
        perform_mutation();
        evaluate_population();
    }
}
```

The population is initialized randomly, and its individuals are subsequently evaluated using the fitness function. The following steps are iteratively repeated until some stopping criterium is triggered. Initially, individuals are chosen using a selection operation, which takes their fitness value under consideration. Subsequently, a crossover rate determines which individuals mate using a crossover function. This operation serves the purpose of exchanging information between the chromosomes. In order to add variation and diversity, the resulting crossover offspring undergoes small changes

using a mutation function. A mutation rate decides which individuals undergo this operation. The newly generated population is then evaluated for the next iteration. Different stopping methods like time limit, fitness limit, minimum convergence rate or maximum number of generations are available [12].

As discussed by Hussain and Muhammad [17], genetic algorithms face an exploration vs exploitation dilemma. If the algorithm converges too quickly to a solution, most of the search space might not have been explored yet, thus increasing the probability of getting stuck in a local optimum. In contrast, a low convergence rate might be time consuming and result in inefficient utilization of computational resources.

> *"Finding a balance between exploration and exploitation has been a difficult-to-achieve goal from the beginning." [18]*

Attaining this balance requires a suitable choices on both genetic functions and their applied probabilities. Due to no real consensus on the best control parameter settings, their optimal selection proves to be difficult [18].

### 2.1.1 Chromosomes and Genes

Both chromosomes as well as genes require suitable encodings in order to fit the search task. Each chromosome, composed of a list of genes, must represent a complete solution of the specific problem. Finding appropriate encodings proved to be, besides the hyperparameter tuning, the main challenge in setting up a genetic algorithm pipeline.

While the chromosome encoding is highly problem specific and heavily depends on the choice of gene encoding, there are various gene encodings suggested by existing literature. A simple and commonly used representation for genes is binary encoding, where a gene takes on the form of either zero or one. Further common encodings include octal and hexadecimal representations [13, 15]. Srinivas and Patnaik [13] also suggest integer representations in case a optimization problem has real-valued continuous variables, where the objects are linearly mapped to integers defined in a specific range. Various other encodings are available, often very problem

specific. For example tree encodings allows for genes to represent programming functions, leading to a subcategory of genetic algorithms called 'genetic programming' [15]. In cases where existing gene encodings do not fit the given problem, custom encodings can be employed.

### 2.1.2 Population Size

The population size, which defines the number of individuals per generation, directly impacts the performance of a genetic algorithm. Increasing the population size will enhance the degree of parallelism in the genetic algorithm, as each individual represents one distinct search point [11].

Research seams to be in agreement, that a small population leads to less diverse individuals and might provide an insufficient sample size, which can result in premature convergence to a local optimum. A large population size will allow the GA to perform a more informed search. However computation time will suffer due to the larger number of individuals per generation and might lead to slower convergence to an optimum [14, 15, 18]).

### 2.1.3 Selection

The selection operator chooses two parents for crossover and mutation operations until the list of offsprings has reached the desired population size. Individuals are chosen based on their fitness value, ensuring their increased representation from generation to generation. Weak solutions will be discarded over time [13]. The selection algorithm must satisfy two requirements. On the one hand, high selection pressure will lead to decreased diversity in the population resulting in premature convergence [15]. The algorithm will subsequently behave more like a local search method, a hill-climber or a greedy algorithm [18]. The initial low average fitness value of the population will, in combination with a few well performing individuals lead to them overtaking the population, drastically reducing diversity. On the other hand, low selection pressure will struggle to converge to an optimum. Different selection methods like stochastic uniform remainder, random selection, rank selection, roulette wheel selection and

tournament selection exist [12]. According to Hussain and Muhammad [17], who provide a extensive comparison of different selection techniques, the choice of selection methods significantly affects the performance of the genetic algorithm.

Roulette wheel selection is a popular mechanism, where each individual corresponds to an area on a roulette wheel, based on its fitness value [10]. Grefenstette [14] points out the scaling problem as its major drawback. As the algorithm progresses, its fitness variance to mean ratio becomes increasingly small, leading to low selection pressure. This problem can be mitigated by using ranks instead of the fitness value [15]. Individuals will be sorted based on their performance and their relative ranks are then inserted into the roulette wheel instead.

Tournament selection is a popular alternative to roulette wheel selection. A parameter $t$ defines the size of a tournament. Until the desired number of offsprings is achieved, $t$ individuals will be chosen at random from the population. Only from this smaller list, the best individual is selected. A popular tournament size is 2, larger sizes might enhance competition among individuals, however they can also has an impact on the diversity of the population [17]. In a comparison of different selection methods by Jinghui Zhong et al. [19], tournament selection was deemed to be the most performant, converging more quickly than for example roulette wheel selection.

Elite selection is an additional method that can enhance a selection operation and was proposed by De Jong [20]. The best $n$ individuals are automatically chosen for the next generation. Elite selection can help reduce variance, as the highest performing individuals can not be removed by randomness.

### 2.1.4 Crossover

The crossover operation serves as the mating process between two individuals chosen from the population by the selection operation. The resulting offspring is a combination of both parent genes. Grefenstette [14] highlights, that first, “it provides new points for further testing within the hyperplanes

already represented in the population" and second, that it "introduces representatives of new hyperplanes into the population".

Different types of crossovers can be implemented, with the simplest approach being the single-point crossover. In this method, a point is chosen at random along the length of the chromosome. The two parents swap their genetic information at this position, generating two new offsprings [15]. Extension are the two- or $k$-point crossover operations, where chromosomes are divided by $k$ segments, which will get exchanged. A second crossover operation is named uniform crossover, where gene swaps between parents are randomly decided by a given probability, independent to the exchange of other genes [15]. Lim et al. [21] demonstrate various other crossover operations.

Srinivas and Patnaik [13] proposed the notion of positional and distributional bias. If a crossover operation has a positional bias, the probability of a bit to be swapped depends mainly on its position. Examples of this are the mentioned single and $k$-point crossover operators. The uniform crossover can be found at the other end of the spectrum, as it has a maximal distributional bias, completely disregarding any positional information. While ignoring gene positions results in higher disruptiveness, which has potential drawbacks, it becomes more exploratory in homogeneous populations where $k$-point crossovers might struggle. Srinivas and Patnaik [13] suggest that uniform crossover is more useful in small populations. Larger populations inherently are more diverse, making k-point crossover more suitable.

Not only the crossover type has to be chosen, a crossover rate also requires declaration. This value defines how likely a single crossover operation is to be applied on the selected individuals. Choosing a suitable crossover probability is, again, influenced by the exploration-exploitation balance. Higher crossover rates will introduce new structures quickly into the population; however, good sequences of genes might get disrupted. A low crossover rate will result in a low exploration, which leads to stagnation [14].

### 2.1.5 Mutation

The mutation operation is applied after crossover to maintain the diversity of the gene pool. Through small, random changes, the variability of the population increases. Each individual can be exposed to mutation, irrespective of their fitness value. A mutation rate chooses the individuals to be mutated. An additional individual mutation rate selects which specific genes of the chosen chromosome are mutated [13]. Depending on the gene encoding, the mutation operation might vary. In case of binary gene encoding, mutation will only flip certain bits. For custom encodings, the mutation operation can be tailored accordingly.

> *"If the mutation is not considered during evolution, then there will be no new information available for evolution." [15]*

The settings for mutation rates again need to be chosen carefully. If they are too high, the genetic algorithm might transform into random testing. In case they are set too low, the population will not be able to maintain diversity as no new genetic material is reintroduced [7, 14]).

### 2.1.6 Adaptive Control Parameters

Various pieces of literature [17, 22, 23] suggest adaptively controlling selection methods, crossover rates and mutation rates over the duration of a genetic algorithm. The main goal is to mitigate the previously mentioned exploration-exploitation problem. In the beginning, exploration is desired, thus low selection pressure as well as high crossover and mutation rates are required. However towards the end of the genetic algorithms duration, convergence to an optimum is preferred, meaning more 'elite' selection methods as well as less variation due to crossover and mutation [13].

Marsili Libelli and Alba [22] proposed a method where different mutation rates are used depending on the fitness of the individual. They claim that a higher mutation probability for a low fitness population results in a more efficient search. Hussain and Muhammad [17] argue for a selection operation, where initially low selection pressure is applied while increasing it

towards the end, stating that this approach will help mitigate both mentioned competing criteria, namely premature convergence as well as slow convergence.

Contrary, De Jong [18] is more critical on adaptive control parameters. Although they might improve the performance of a genetic algorithm for specific problems, genetic algorithms are already quite robust in practice. Adding more tunable parameters will not make the task of the researcher easier.

> *"The purist might argue that inventing feedback control procedures for EAs is a good example of over-engineering an already sophisticated adaptive system." [18]*

He adds, that "perhaps the most interesting thing to note today, after more than 30 years of experimenting with dynamic parameter setting strategies, is that, with one exception, none of them are used routinely in every day practice. The one exception are the strategies used by the ES community for mutation step size adaptation." In order to keep the number of tunable parameters low, the approach implemented by this master's thesis will not utilize adaptive control parameters.

## 2.2 Behaviour Tree

Behavior trees were invented and developed by the computer game industry as a control structure of non-player characters [24]. Since then, their scope of application has expanded into various fields, including robotics, smart homes, power grids, autonomous vehicles [25]. Sprague et al. [26] demonstrated their effectiveness as a robust control architecture for autonomous underwater vehicles. A behavior tree is a directed tree, with root, child, parent and leaf nodes. Control flow nodes are non-leaf nodes, while leaf nodes are known as execution nodes. The execution of a BT occurs through ticks, propagating down down the tree, starting from the root node at a specified frequency. When receiving a tick, all nodes execute and can return either "Running", "Success" or "Failure" [24].

Their high modularity and reactiveness are often stated as their main advantages. The modularity stems from the fact, that different parts in the BT can be changed and modified without influencing the rest of the tree. This aligns with the reactivity demonstrated by the Behavior Tree. Per tick, safety tests can be performed before entering the execution logic of the tree, allowing the system to react immediately to a non-safe situation. Adding complex details to this execution sub-tree will not jeopardise the safety tests [26].

### 2.2.1 Control Flow and Execution Nodes

The control flow nodes decide which actions and condition nodes to execute next. Three categories of control flow nodes - Sequence, Fallback and Parallel - exist. The two execution nodes are Actions and Conditions, both are responsible for carrying out the actual behavior of the behavior tree [24].

- Control flow nodes
  - **Sequence nodes:** These nodes execute their children nodes sequentially, routing the ticks to each child as long as they return "Success". If every child returns "Success," the sequence node itself returns "Success." If any child returns "Failure" or "Running", the sequence node immediately returns the corresponding value without routing the tick to the next child.
  - **Fallback nodes:** Fallback nodes execute their children one after another until one child returns "Success" or "Running", at which point will the fallback node returns "Success" or "Running" accordingly. As long as the childs status is "Failure", they will continue routing the ticks to the next child. If all children return "Failure", the fallback node will also returns "Failure".
  - **Parallel nodes:** These nodes execute all children simultaneously. Given a parameter $m$, they decide on its return status. If more than $m$ children succeeded, it will return "Success". If enough children fail to make "Success" impossible, they return "Failure"; otherwise, they return "Running."
- Execution nodes

- **Action nodes:** These nodes execute a command upon receiving a tickand can return "Success," "Failure," or "Running." Typically, action nodes execute the behavior of an agent, such as "open_door()".
- **Condition nodes:** These nodes also execute a command after receiving a tick. Contrary to the Action Nodes, they can only return "Success" and "Failure". Condition nodes are designed to check conditions, such as "is_door_closed()". They cannot execute over multiple ticks and thus can never return "Running".

# 3 Implementation

In addition to an overview of the simulation environment, the implementation of both the genetic algorithm and the behavior tree will be discussed in this chapter.

## 3.1 Traffic Manager

A custom-developed Traffic Manger is employed to simulate a traffic environment, which is subsequently controlled by a genetic algorithm. The Traffic Manager is treated as a 'black box' in this master's thesis, with only a brief introduction provided. In essence, it simulates simulate traffic using a start scenario, where the positions and types of actors (vehicles and pedestrians) are defined. Throughout all simulations conducted in this thesis, the Traffic Manager was configured with a frequency of 100Hz and the simulation duration set to 35 seconds.

Each simulation involves at least one EGO vehicle, with the option to include any number of Non-Player Characters (NPCs). The EGO vehicle can be either partly or completely supervised by autonomous driving functions. The goal of the genetic algorithm is to subsequently test these function for requirements like safety and comfort. This is achieved by controlling the NPCs in a way to generate critical scenarios.

### 3.1.1 Action Interface

In order to control the behaviour of all actors inside the simulation, actions can be requested via the Action Interface, which is provided by the Traffic

Manager. An action initiates a certain behaviour from an actor and can be set at any timestep[1]. Pedestrians and vehicles have different sets of possible actions. If no action is applied, the actor will behave in a normal manner inside the simulation. This implies that the actor will follow its assigned path until a new action changes its behaviour. It will stick to traffic rules and will break in case of an obstacle. The following list shows all actions which were available for the genetic algorithm at the time of this master's thesis.

- JunctionSelection
  - vehicle_id: int, step: int, junction_selection_angle (radians): float
  - Vehicles will choose which direction to take at junctions based the junction_selection_angle.
- LaneChange
  - vehicle_id: int, step: int, direction: float, distance: float, delay (optional): float
  - Initiates a lane-change based on its given parameters.
- AbortLaneChange
  - vehicle_id: int, step: int
  - Will abort a current lane-change.
- ModifyTargetVelocity
  - vehicle_id: int, step: int, percentage: float
  - Modifies the internal target velocity of the vehicle by a percentage. If set to 0, the vehicle will stop.
- TurnHeading
  - pedestrian_id: int, step: int
  - The pedestrian will turn 180 degrees and walk in the opposite direction
- CrossRoad
  - pedestrian_id: int, step: int
  - The pedestrian will cross the road immediately.

[1] depending on the autonomous driving function under test, the Action Interface might be disabled for the EGO vehicle

- CrossAtCrosswalk
  - pedestrian_id: int, step: int
  - The pedestrian will cross the road at the next crosswalk.

The genetic algorithm only controls NPCs, while a behaviour tree is used for the EGO vehicle itself. Both methods utilize the Action Interface to guide their respective actors. An overview is provided in Figure 3.1.

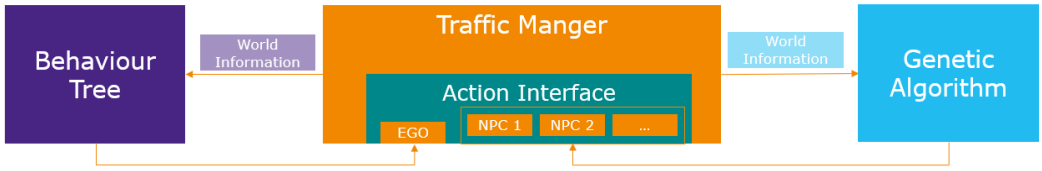



Figure 3.1: Action Interface structure

## 3.2 Genetic Algorithm

The genetic algorithm will search for sequences of actions that will result in the most interesting scenarios according to its cost function. To implement the genetic algorithm, the Python library DEAP[2] was chosen, as it is a popular tool in academia and allows for high customisability. The algorithm has full access to set actions for all NPCs. Searching for sequences of actions, the GA tries to optimize a cost function.

A few default settings for the genetic algorithm were chosen. For example, it was decided, that the genetic algorithm can set an action per actor every 50 steps, which translates to 0.5 seconds (simulation runs at 100hz). In other words, every 50 steps of the simulation is 1 timestep for the genetic algorithm. If the GA decides to not set an action, "NoAction" will be used as a placeholder.

[2] https://deap.readthedocs.io/en/master

### 3.2.1 Maximum Number of Generations

A fixed maximum number of generation is a commonly used stopping criteria. While more complex methods like an "adaptive convergence rate" might lead to better performance, it was decided that the additional complexity of having to choose a suitable convergence rate outweighs its potential benefits. Especially considering that a genetic algorithm already has a lot of different control parameters that need tuning. Performance is a big consideration in this thesis, thus a lower maximum number of generations is preferred. During testing, a generation size of 30 was almost always sufficient and will be used in all of the upcoming testing.

### 3.2.2 Encoding

When implementing a genetic algorithm, it is necessary to use an encoding that fits to the problem at hand. In this context, each chromosome is required to include all actions of one whole simulation. Due to the deterministic nature of the Traffic Manager, it is sufficient to define a simulation by only the action sequence from a chromosome and the initial state, which is defines by a start scenario. The different encodings presented in Section 2.1.1 were not directly applicable to the given problem. Therefore, a custom encoding for both chromosomes and genes needed to be generated.

#### Chromosome

Two different encodings proved to be a good fit. Both use the position of a gene inside the chromosome to define the time step of an action. Which chromosome encoding is the most adequate will be discussed and tested in Chapter 4.

**Time** The first encoding is called 'Time'. In this encoding, each gene corresponds to 1 time step (1 gene per 0.5 seconds) and needs to contain all actions of all NPCs at that particular moment. A gene is a list with a size equal to the number of NPCs. Each object in this list represents an

action. The index of an action inside a gene corresponds to the NPC ID. For example, an action positioned in the gene at index 2, will be applied to NPC 2. If the gene is at position 6 in the chromosome, the action will be set at second 3.0 in the simulation. A visualization is seen in Figure 3.2.

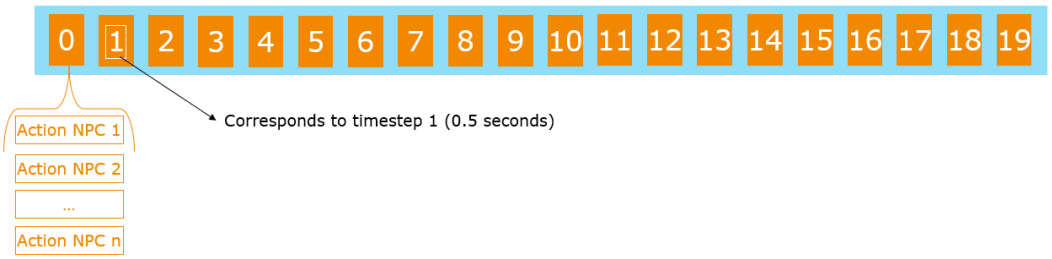



Figure 3.2: Time

Given the previously stated simulation time of 35 seconds, each chromosome has a length of $35 * 2 = 70$ genes. The crossover operation can only move all actions of a time step at once. Modifications in the timeline between actions of the same time step will only happen through a mutation operation.

**Time+NPC** The second encoding is called 'Time+NPC'. Here, one gene corresponds to exactly one action. In order to not loose required information, the IDs of each NPC must also be encoded into the gene's position in its chromosome. One might visualize this as unfolding the matrix build by the genes and chromosome in the Time encoding into a linear array. Each actor's actions are listed one after another. Now, each gene has a length of 1 and each chromosome has a length of $35 * 2 * number\_of\_actors$, which makes them much longer compared to the previous encoding. Figure 3.3 provides a visualization. In this example, a gene at position 13 corresponds to an action for NPC 2 at time step 3 (which is at 1.5 seconds in the simulation).

However for this encoding to make sense, the OnePoint and TwoPoint crossover operations had to be modified in a way, that for each NPC, these crossover operations are applied individually. If this was not the case, OnePoint crossover would introduce changes to actions of only one NPC. Because these different crossover points are chosen independently, the position of the crossing now differs between the NPCs. This modification

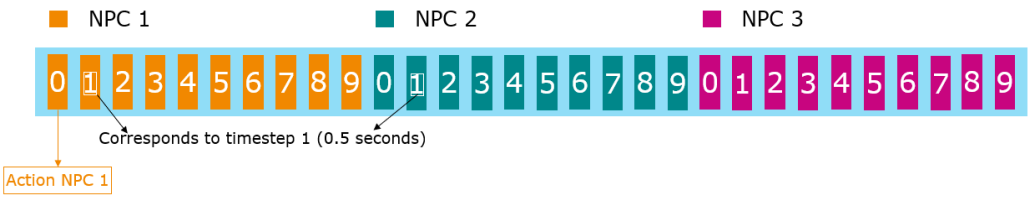



Figure 3.3: Time + NPC

allows the crossover operation to break up actions at the same time step, which was not possible with the Time encoding.

### Gene

Two different encodings for genes were implemented. A gene always consists of a list, which depending on the chromosome type either has a length of $number\_of\_actors$ (in case of Time) or of length 1 (in case of Time+NPC). The following two encodings explain the possible objects in these lists. In both encodings, the probability of each action remains the same and can be seen in Table 3.1.

| Vehicles | | Pedestrians | |
|---|---|---|---|
| ActionType | Probability | ActionType | Probability |
| NoAction | 65% | NoAction | 84% |
| JunctionSelection | 6% | PedTurnHeading | 2% |
| LaneChange | 10% | PedCrossRoad | 4% |
| AbortLaneChange | 2% | CrossAtCrosswalk | 10% |
| ModifyTargetVelocity | 17% | | |

Table 3.1: probability of actions

These probabilities were manually chosen based on experience. For example, setting PedTurnHeading to a high probability will see many pedestrians change their direction in front of the EGO vehicle multiple times, aiming to initiate emergency breaks by the EGO vehicle. However, this is only critical to a certain extend and the probability is accordingly set to a low value.

**Integer** The first encoding uses integers that are translated into actions when the simulation starts. For each action, a range of integers is assigned; the larger the range, the more likely an action is chosen by the GA. These ranges correspond to the probabilities given in Table 3.1.

Actions that have parameters need their assigned integer range to be manually split again. For example, it was decided, that ModifyTargetVelocity has five different percentage settings, namely 50, 70, 100, 130, 160. Therefore, each of these 5 settings needs a unique range of integers assigned, again with different lengths, in order to have different probabilities. A complete list can be seen in Table 3.2.

| ActionType | Parameter | Probability |
|---|---|---|
| JunctionSelection | straight | 34% |
| | left | 33% |
| | right | 33% |
| LaneChange | left | 50% |
| | right | 50% |
| ModifyTargetVelocity | 50 | 10% |
| | 70 | 20% |
| | 100 | 45% |
| | 130 | 20% |
| | 160 | 5% |

Table 3.2: Integer encoding: probability of parameters per action

Combining these ranges results in a continuos list of integers. The genetic algorithm will choose integers in this list for its genes. Before a simulation is started, these integers are then encoded back into actions.

**Dictionary** The second encoding for genes corresponds exactly to the actual actions used in the simulations, requiring no translation. Each action is again selected based on the same probabilities from Table 3.1. For actions without parameters, the different encoding will make no difference. However, actions with parameters are no longer split into different discrete settings. Instead, each parameter is chosen by a randomness function, which is individually selected per action. For example in case of the percentage parameter in

ModifyTargetVelocity, the values are selected from a gaussian distribution with $mu = 100$, $sigma = 25$ and a range limit between 0 and 300. An overview is provided in Table 3.3. Once again, these probability functions were assigned based on intuition as well as trial and error.

| ActionType | SelectionFunction | Settings |
|---|---|---|
| JunctionSelection | Choice | [straight, left, right] |
| LaneChange | Choice | [left, right] |
| ModifyTargetVelocity | Gauss Distribution | Mu: 100, Sigma: 25 |

Table 3.3: Dictionary encoding: probability of parameters per action

Compared to integer encoding, this method allows for much higher granularity when selecting parameters. However, because only one action type (ModifyTargetVelocity) has a continuos variable, this difference might be insignificant. The contrast between encodings will likely become more pronounced with an increasing number of implemented actions in the Action Interface. Chapter 4 will investigate, if the performance of a genetic algorithm differs between these two encodings.

### 3.2.3 Cost Function

This master's thesis utilizes only internal values from the Traffic Manager for the cost function, specifically if the ego vehicle had to initiate an emergency break. As long as the EGO vehicle has no emergency stop initiated, the cost value will increase. If a maximum duration of 3 seconds is exceeded, the cost value will increase as well. Only in cases where none of these two conditions is fulfilled, no cost will be added. This penalty of the maximum duration was introduced to reduce the number of instances, where the ego vehicle is constantly brake-checked[3] by the leading vehicle. The cost values are not easily interpreted, so these numbers are modified for this master's thesis using Equation 3.1.

[3] "the unsafe action of applying a car's brakes to dissuade a driver who is following too closely" according to https://www.dictionary.com/e/slang/brake-check/

$$cummulated\_emergency\_break\_duration = (3500 - cost)/100 \quad (3.1)$$

The constant 3500 is used to account for the 100Hz sampling rate and the duration of 35 seconds. The division by 100 transforms the value into seconds. A higher cumulated emergency brake duration is preferable. It is important to consider that this value might not exactly correspond to the actual cumulated emergency brake duration time from the simulation, due to the penalty applied by the cost function for emergency breaks that last longer then 3 seconds.

## 3.3 Behaviour Tree

The EGO vehicle will be controlled by a behaviour tree, which is implemented using the Python library py_trees[4]. The general idea is to have the EGO vehicle move in a relatable manner trough the world. It will attempt to dodge standing or slow-moving obstacles as well as decide which turn to take at junctions.

### 3.3.1 Structure

While the behaviour tree has access to the same Action Interface (see Section 3.1.1) as the genetic algorithm, it needs a higher level of integration with the Traffic Manger. The genetic algorithm is only interested in the resulting emergency break values, but the behaviour tree needs access to internal functions during the simulation. Conditions like NoObstacle() or LaneChangePossible() execute functions from the Traffic Manager to get needed information on the world around the EGO vehicle. Figure 3.4 shows the implemented behaviour tree.

Each tick, the branch junction selection will first get executed, which will switch between the settings "straight", "left" and "right" every 3 seconds. This

[4]https://py-trees.readthedocs.io/en/release-2.2.x/

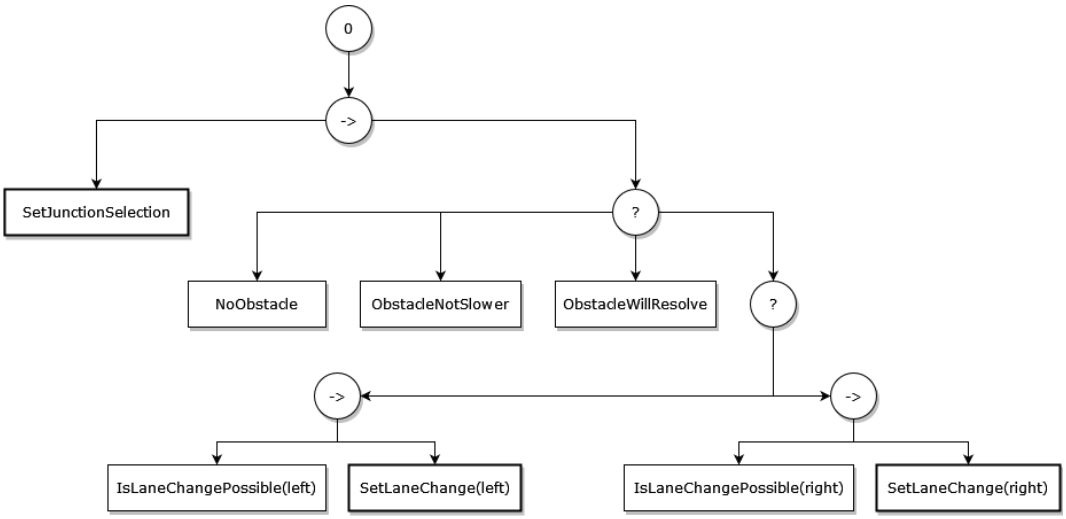



Figure 3.4: implemented behaviour tree

serves the purpose of generating more interesting situations, as otherwise, the ego vehicle will only move straight. A random junction selection would not be possible, as this would make the simulation non-deterministic. While implementing a navigation system was investigated, it was decided against doing so, as the added complexity did not seem to significantly improve the resulting simulations.

The next branch focuses on dodging obstacles. First, the condition NoObstacle() determines , if an obstacle is on the path of the EGO vehicle. Second, a threshold of 50% in ObstacleNotSlower() determines if a lane change makes sense. Third, the ObstacleWillResolve() function has the responsibility to check if the obstacle is 1. inside a junction or 2. roughly at a 90 degree angle. In both cases, no takeover will be performed. In case all of the three functions return false, a final function LaneChangePossible() will check, if a lane change to the left is possible and if this is not the case, if a lane change to the right is possible. Depending on its result, a lane change to the suggested direction is performed. LaneChangePossible() might not allow a lane change if 1. there is no lane or 2. if the lane is blocked by an obstacle.

# 4 Hyperparameter Tuning

The performance of a genetic algorithm is significantly influenced by its control parameters. Performant settings on one particular fitness landscape might not be on appropriate a different one [18]. This chapter will focus on tuning the genetic algorithm to perform well on the given cost function. First, a choice on the optimal population size is made. Afterwards, the Taguchi method is used for tuning the remaining hyperparameter.

## 4.1 Simulation Setup

Two workstations were available for running all of the following simulations. The first workstation had an Intel Core i7-9700K and a GeForce RTX 2070 SUPER with 32 GB of RAM. The second workstation had an Intel Core i7-6850K as well as two Nvidia GeForce GTX 1080, also equipped with 32 GB of RAM. On both systems, Kubuntu 20.04 LTS was the operating system.

As defined in Chapter 3, each genetic algorithm will run for 30 generations. Additionally one simulation will have a duration of 35 seconds. On a single workstation, approximately 3 hours and 50 minutes are needed to complete one genetic algorithm with a population size of 96. This time indication, however, is influenced by the number of actors. The street map 'Town 10' from the driving simulator Carla[1] will be used for all experiments. It has an adequate size, yet is not too big, and allows for interesting manoeuvrers. To minimize the number of needed tests, a single start scenario was utilized for tuning the control parameter. It is referred to as 'start scenario 1' can

[1] https://carla.readthedocs.io/en/latest/map_town10/

be seen in the Appendix at Figure 1. In Chapter 5, the performance of the tuned genetic algorithm on different start scenarios will be investigated.

## 4.2 Population

Finding a suitable population size is of high importance to a genetic algorithm. On one hand, a population that is too small might result in less diverse runs of the genetic algorithm, on the other hand, if the population size is too high, the simulations will become too costly (see Section 2.1).

In order to evaluate the best population size, other hyperparameters first have to be fixated. Grefenstette [14] suggests that a range of control parameter will already lead to acceptable performance, yet optimal performance needs tuning. De Jong [18] complements these findings, adding that the 'sweat spot' for control parameters of genetic algorithms is reasonably large and easy to find. A default set of static parameter values is generally speaking sufficient. Following this advice, the most suitable population parameters will now be evaluated by fixating the remaining hyperparameters to a small range of suggested values from the literature.

### 4.2.1 Identifying Suggested Hyperparameter Settings from Existing Literature

After reviewing various literature regarding control parameter of genetic algorithms, no clear consensus emerged. Mills et al. [11] came to a similar conclusion, mentioning the inconsistencies between findings during their literature review and highlighting the conflicting evidence regarding "key GA control parameter". Table 4.1 aims to provide a short, though not exhaustive, overview on different control parameter settings used in the literature. This compilation does not claim to cover the entire scope of available research in this domain, rather it served as a focused effort to identify usable hyperparameters.

A value between 30-200 is a commonly recommended as a population size. In order to reduce the needed computation time, a population size

| Parameter Set | Pop | Cross | Mut | Sel |
|---|---|---|---|---|
| De Jong [20] | 50 | 0.6 | 0.001 | ? |
| Mills et al. [11] | 200 | ? | Adaptive | SUS |
| Grefenstette [14] | 30 | 0.95 | 0.01 | ? |
| Grefenstette [14] | 80 | 0.45 | 0.01 | ? |
| Almanee et al. [9] | 50 | 0.8 | 0.2 | ? |
| Srinivas and Patnaik [13] | 30-100 | 0.9 | 0.01 | ? |
| Fazal et al. [27] | 50 | 0.5 | ? | Tourn |
| Dao et al. [28] | 200 | 0.7 | ? | Roul |
| Naruka et al. [29] | 200 | 0.4 | ? | Roul |
| Jinghui Zhong et al. [19] | 50-250 | 0.1-0.9 | 0.05-0.25 | ? |

Table 4.1: summary of literature review

of 96 is defined as a maximum for the future evaluation. Crossover rates are mostly be in a range of 0.6-0.9. When it comes to the mutation rate, a low value is commonly advised. For example Grefenstette [14] suggest poor performance using a rate over 0.05. Using a low mutation rate is also suggested by Whitley [30] and Jinghui Zhong et al. [19]. On the other hand, Boyabatli [31] found higher mutation rates for their application to be more suitable. Srinivas and Patnaik [13] differentiate between higher and lower population numbers, claiming that a smaller population needs higher mutation rates in order to maintain a sufficient diversity.

### 4.2.2 Comparison of Population Size

Based on the described research, population sizes of 32, 48, 64 and 96 will be compared. Crossover rates are set to 0.6 and 0.8. For mutation, 0.01 and 0.2 will be discussed. Further, tournament selection of 2 and 4 is used. Individual mutation probability will stay at 0.1. Chromosome encoding is set to Time and gene encoding is set to Integer. Each run will be executed 5 times in order to reduce randomness and to make the results more robust. Each simulation will last for 30 generations. A list of all settings with the mean over 5 repetitions per population size can be seen in Table 4.2.

In Figure 4.1, the results per population are plotted. The line corresponds

| Settings | Code | 32 | 48 | 64 | 96 |
|---|---|---|---|---|---|
| C: 0.6, M: 0.01, TS: 2 | A | 4.49 | 4.84 | 6.49 | 6.29 |
| C: 0.6, M: 0.01, TS: 4 | B | 3.89 | 4.79 | 4.21 | 5.63 |
| C: 0.6, M: 0.20, TS: 2 | C | 4.38 | 4.90 | 4.98 | 6.69 |
| C: 0.6, M: 0.20, TS: 4 | D | 4.80 | 5.33 | 6.09 | 6.50 |
| C: 0.8, M: 0.01, TS: 2 | E | 4.37 | 6.08 | 5.29 | 5.84 |
| C: 0.8, M: 0.01, TS: 4 | F | 4.48 | 4.51 | 4.46 | 6.03 |
| C: 0.8, M: 0.20, TS: 2 | G | 4.01 | 5.60 | 5.41 | 6.31 |
| C: 0.8, M: 0.20, TS: 4 | H | 4.42 | 4.95 | 7.06 | 6.91 |

Table 4.2: population size results - mean over 5 repetitions

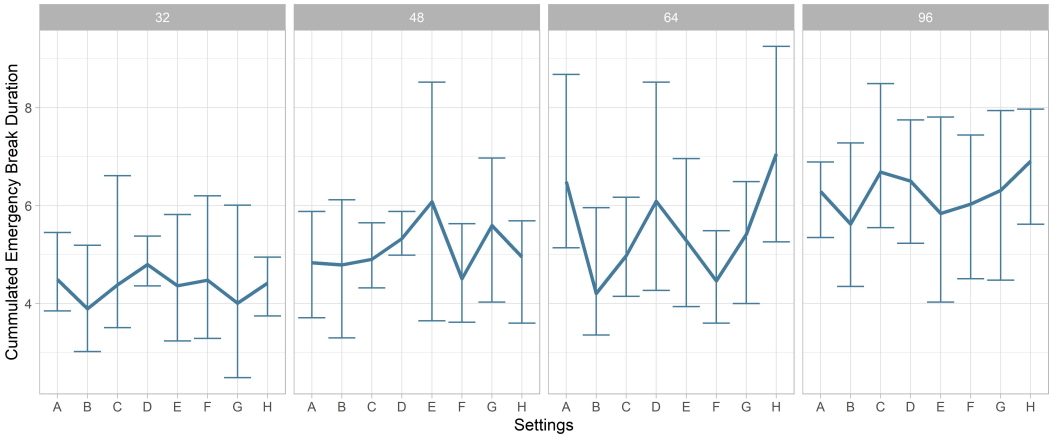



Figure 4.1: mean and error bars per population size

to the mean, while the error bars show the spread (min to max) of all 5 repetitions. A higher spread in the results can be seen when looking at the smaller population sizes. Considering these findings, a population size of 96 was chosen. While such a high value will result in a performance impact, it was deemed more important to keep the variation low.

## 4.3 Design of Experiment

This section aims to find optimal settings for the remaining control parameters of the given problem. Following the conclusion from the previous Section 4.2, a population size of 96 will be fixated. In order to further tune the control parameter of the genetic algorithm, various different strategies can be used. Using automated approaches like 'grid search' [32] or 'simmulated annealing' [33] might lead to good results with minimal effort. Grefenstette [14] even suggests using a second, higher level genetic algorithm for the optimization process. The tuning of hyperparameter for these algorithms is however still needed. Due to performance considerations, many optimization methods did not fit the requirements. Executing one run for 30 generations takes around 3:50 hours. The high variance between runs requires a certain number of repetitions for each setting. Although two different workstations were available, the time required to execute the needed number of runs for these automated tests would exceed the available time budged. For this section, 8 repetitions were used per setting, which makes one evaluation last 30 hours.

A different approach is called design of experiment (DOE), also known as statistically designed experiments. DOE tries to find the cause and-effect relationship between the factors and the output of experiments. It uses factorial design where the variables in an experiment are named 'factors'. Each factor consists of at least two settings, with the actual number of settings being called 'levels' [34]. Design of experiment needs manual expertise to define which factors are possibly of importance and which settings each factor should have.

> *"If the range of variable is too small, then we may miss lots of useful information. If the range is too large, then the extreme values might give infeasible experimental runs." [34]*

Subsequently, main effects and interactions can be calculated to find the best settings per factor. ANOVA (Analysis of Variance) will identify the significance of each factor and interaction. More details on these analysis tools will be provided in Section 4.3.3.

A full factorial design will test over all possible combinations of the selected factor levels. Looking at the proposed factors in Table 4.3, 1024 runs[2] are required, which is not feasible performance wise. A full factorial design has the drawback, that, as the number of factors k gets increased, the number of needed experimental runs increases exponentially, thus resulting in lengthy experiments. Yang and El-Haik [34] state that most of the results obtained by testing over all combinations are only used for estimating higher-order interactions, which are in most cases insignificant.

### 4.3.1 Taguchi Method

Various improvements to design of experiment have been but forward by Dr. Genichi Taguchi, such as reducing the influence of uncontrollable (noise) factors on processes and products as well as reducing variability [35]. This master's thesis will not discuss all of Taguchi's proposed considerations; for more detail Roy [35] as well as Yang and El-Haik [34] are recommended. Taguchi's proposed design of experiment is a fractional factorial design, which requires significantly fewer runs. In fractional factorial designs, only a fraction of all possible combinations is investigated [35]. While different fractional factorial designs are available, Taguchi was chosen, because he provides a simple and easy-to-follow procedure that requires only a minimal number of runs.

> *"There are many similarities between "regular" experimental design and Taguchi's experimental design. However, in a Taguchi experiment, only the main effects and two-factor interactions are considered. Higher-order interactions are assumed to be non-existent. In addition, experimenters are asked to identify which interactions might be significant before conducting the experiment, through their knowledge of the subject matter." [34]*

Taguchi predefined a number of different orthogonal arrays where each row contains the specific levels (i.e the settings) of one experiment, while the columns correspond to the factors. The researcher has the responsibility to

[2]number of runs calculated using: `https://datatab.net/statistics-calculator/design-of-experiments`

select an array based on the individual needs [36]. Using these orthogonal arrays instead of full factorial experiments will lead to a much smaller amount of simulation runs, while the full factorial experiments "might not provide appreciably more useful information" [35].

A big drawback of using Taguchi orthogonal arrays is the inability of evaluating higher-order interaction effects. Not only are these interaction effects impossible to evaluate, but they also have a negative influence on the performance. Orthogonal array experiments perform best in cases of minimal interaction between factors. While there is still a good chance of identifying the optimum condition, especially the performance estimate can be significantly off [35]. Although Yang and El-Haik [34] state that in many cases higher-order interaction effects in factorial designs are seldom significant, this might not be the case for genetic algorithms. For example, according to [18], control parameters of a genetic algorithm "interact in highly non-linear ways." If the proposed method is able to provide a suitable set of hyperparameter settings will be evaluated in Chapter 5.

**Selection of an Orthogonal Array** When choosing a suitable Taguchi orthogonal array, various factors have to be taken into account. According to Yang and El-Haik [34], a three step procedure needs to be followed:

1. Calculate the total degree of freedom (DOF).
2. Based on the following two rules, a standard orthogonal array should be selected:
    a) The total DOF need to be smaller than the number of runs provided by the orthogonal array.
    b) All required factor level combinations need to be accommodated by the orthogonal array.
3. Factors have to be assigned using these rules:
    a) In case the factor level does not fit into the orthogonal array, methods such as column merging and dummy levels can be used to modify the original array.
    b) Using the linear graph and interaction table, interactions can be defined.

c) In case some columns are not assigned, its possible to keep these columns empty.

For this design of experiment, seven factors (three 4-level factors and four 2-level factors) have been selected. The choice of factors and levels to choose was made based on experience gained on Section 4.2. In Table 4.3, every factor with the corresponding levels is listed.

| Factors | Code | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|---|
| CrossoverType | A | one point | two point | uni* 0.1 | uni* 0.5 |
| CrossoverRate | B | 0.2 | 0.5 | 0.8 | 0.9 |
| MutationRate | C | 0.01 | 0.1 | 0.3 | 0.5 |
| ChromosomeType | D | Time | Time+NPC | - | - |
| GeneType | E | Int | Dict | - | - |
| TournamentSize | F | 2 | 4 | - | - |
| IndMutationRate | G | 0.1 | 0.5 | - | - |

Table 4.3: control parameters (factors) with corresponding settings (levels) - (*uniform)

As already stated, Taguchi allows to only test for predetermined two-factor interactions; evaluating higher-order factor interactions is not possible [34]. Analysing interactions comes at the cost of degrees of freedom. An interaction between ChromosomeType and GeneType might be of interest and will thus be chosen. In order to minimized the required degrees of freedom (and correspondingly the required number of experiment runs), no additional interactions will be analysed.

### 4.3.2 Selection of a Suitable Standard Orthogonal Array

The total degree of freedom can be calculated using the rules provided by Yang and El-Haik [34]:

1. 1 DOF is always used for the overall mean.
2. Each factor has a DOF of NumberOfLevels - 1.
3. Two-factor interactions require a DOF of: $(n_{factor1} - 1)(n_{factor2} - 1)$ where $n$ = number of levels.

This leads to the following calculation for the needed three 4-level factors and four 2-level factors as well as the interaction ChromosomeType-GeneType:

$$\begin{aligned} DOF &= 1 + 3 * (4 - 1) + 4 * (2 - 1) + 1 * (2 - 1) * (2 - 1) \\ &= 15 \end{aligned} \tag{4.1}$$

A $L_{16}$ array seems suitable to accommodate the required 15 DOF, which can be seen in table 4.4.

| | $L_{16}(2^{15})$ | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NO. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| 5 | 1 | 2 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 |
| 6 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 |
| 7 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 |
| 8 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| 9 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |
| 10 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 |
| 11 | 2 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 1 | 2 | 1 |
| 12 | 2 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 1 | 2 | 1 | 2 |
| 13 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 |
| 14 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 |
| 15 | 2 | 2 | 1 | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 2 |
| 16 | 2 | 2 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 1 |

Table 4.4: $L_{16}(2^{15})$ Taguchi orthogonal array taken from Roy [35]

The 4-level factors need additional space, which will be generated using column merging, while the interaction will need to be assigned as well. Either an interaction table or linear graphs of the $L_{16}$ array can be used for both column merging and interaction assignment [37]. Both illustrate the interaction relationships in the orthogonal array [34]. The linear graph is

more straight forward and will be the selected approach. While there are multiple linear graphs for the $L_{16}$ array, Figure 4.2 describes the graph which best fits the requirements from Table 4.3. If no suitable graph is found, they can be modified using rules described by Danacıoğlu and Muluk [37].

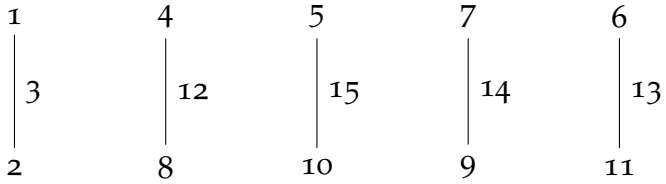



Figure 4.2: linear graph of $L_{16}(2^{15})$ taken from Yang and El-Haik [34]

In a Taguchi linear graph, both nodes and connections represent columns in the orthogonal array. The interaction between two nodes is described by the connecting line [38]. This is useful for both analysing interactions between columns as well as combining (merging) interacting columns in case a higher factor is needed.

**Column Merging** A, B and C are 4-level factors. The currently selected orthogonal array only fits 2-level factors. Through column merging, it is possible to extend columns to accommodate higher-order levels. As calculated in Equation 4.1, a 4-level column requires three degrees of freedom, thus three 2-level columns need to be merged. Column merging needs the to-be-merged columns to be part of an interaction group [34]. The available interaction groups are visualized by the linear graph in Figure 4.2. Three 2-level interaction columns need to be selected first. One column is discarded and the remain two columns are merged using the rules in Table 4.5.

The 4-level factor can then be assigned to this newly generated column. Because three 4-level factors are needed for the current experiment, a total of nine 2-level columns have to be merged.

**Assigning Interactions** The interaction between both 2-level factors is also assigned by utilizing the linear graph. An interaction between ChromosomeType and GeneType seems possible, thus D and E will be assigned to

| OLD COLUMN | | | NEW COLUMN |
|---|---|---|---|
| 1 | 1 | -> | 1 |
| 1 | 2 | -> | 2 |
| 2 | 1 | -> | 3 |
| 2 | 2 | -> | 4 |

Table 4.5: merging rules taken from Roy [35]

connected nodes in the linear graph. The resulting graph can be seen in Figure 4.3. An interaction between F and G can not be investigated, as the chosen orthogonal array is not able to fit the additional 1 degree of freedom (see Equation 4.1).

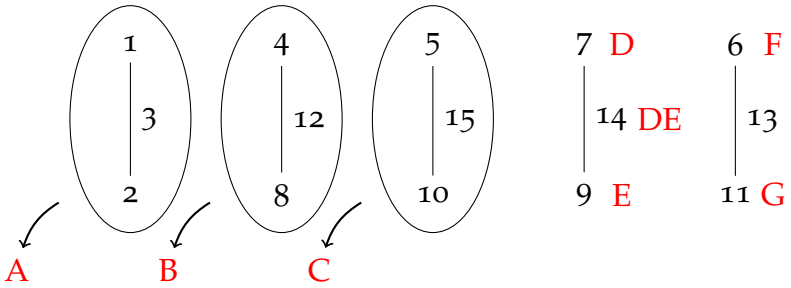



Figure 4.3: modified linear graph

Combining columns 1 2 3 to A, 4 8 12 to B and 5 10 15 to C using rules defined by Table 4.5 is done in Table 4.6. Removing the old and inserting the new columns in the table and transcoding 7 to D, 9 to E, 14 to DE, 6 to F and 11 to G results in the final Table 4.7.

### 4.3.3 Result Analysis

Table 4.7 will provide the settings for all needed test runs (the interaction column DE can be ignored until the evaluation). Transcoding all factors and levels to get the corresponding setting is done in Table 4.3. Every setting will be repeated 8 times to reduce randomness and gain information about variance. Running the genetic algorithm with these 16 different settings

| NO. | 1 2 ~~3~~ | 4 8 ~~12~~ | 5 10 ~~15~~ |
|---|---|---|---|
| 1 | ~~1 1~~ > 1 | ~~1 1~~ > 1 | ~~1 1~~ > 1 |
| 2 | ~~1 1~~ > 1 | ~~1 2~~ > 2 | ~~1 2~~ > 2 |
| 3 | ~~1 1~~ > 1 | ~~2 1~~ > 3 | ~~2 1~~ > 3 |
| 4 | ~~1 1~~ > 1 | ~~2 2~~ > 4 | ~~2 2~~ > 4 |
| 5 | ~~1 2~~ > 2 | ~~1 1~~ > 1 | ~~1 2~~ > 2 |
| 6 | ~~1 2~~ > 2 | ~~1 2~~ > 2 | ~~1 1~~ > 1 |
| 7 | ~~1 2~~ > 2 | ~~2 1~~ > 3 | ~~2 2~~ > 4 |
| 8 | ~~1 2~~ > 2 | ~~2 2~~ > 4 | ~~2 1~~ > 3 |
| 9 | ~~2 1~~ > 3 | ~~1 1~~ > 1 | ~~2 1~~ > 3 |
| 10 | ~~2 1~~ > 3 | ~~1 2~~ > 2 | ~~2 2~~ > 4 |
| 11 | ~~2 1~~ > 3 | ~~2 1~~ > 3 | ~~1 1~~ > 1 |
| 12 | ~~2 2~~ > 3 | ~~2 2~~ > 4 | ~~1 2~~ > 2 |
| 13 | ~~2 2~~ > 4 | ~~1 1~~ > 1 | ~~2 2~~ > 4 |
| 14 | ~~2 2~~ > 4 | ~~1 2~~ > 2 | ~~2 1~~ > 3 |
| 15 | ~~2 2~~ > 4 | ~~2 1~~ > 3 | ~~1 2~~ > 2 |
| 16 | ~~2 2~~ > 4 | ~~2 2~~ > 4 | ~~1 1~~ > 1 |

Table 4.6: Building 4 Level columns from 2 Level columns

each repeated 8 times took 10 days on the two workstations described in Section 4.1. The results are found in the Appendix at Table 1.

### ANOVA

ANOVA analysis (analysis of variance) will provide information on the magnitude of contribution of the main effects and interactions. The calculation of ANOVA on a Taguchi experiment is the same as for a classical design of experiment [34]. Calculating ANOVA was done with the programming language R[3] and the result can be seen in Table 4.8.

The sum of squares column tells how much of the total variation is explained by the model. The variance is broken down into the factors A-G along with the interaction D:E. The residual sum of squares shows the difference between the model's prediction versus what was actually observed (i.e the

[3]https://www.r-project.org/

| NO. | A | B | C | D | E | F | G | DE |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 2 |
| 3 | 1 | 3 | 3 | 2 | 1 | 2 | 1 | 2 |
| 4 | 1 | 4 | 4 | 2 | 2 | 2 | 2 | 1 |
| 5 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 2 |
| 6 | 2 | 2 | 1 | 2 | 2 | 2 | 1 | 1 |
| 7 | 2 | 3 | 4 | 1 | 1 | 1 | 2 | 1 |
| 8 | 2 | 4 | 3 | 1 | 2 | 1 | 1 | 2 |
| 9 | 3 | 1 | 3 | 2 | 2 | 1 | 2 | 1 |
| 10 | 3 | 2 | 4 | 2 | 1 | 1 | 1 | 2 |
| 11 | 3 | 3 | 1 | 1 | 2 | 2 | 2 | 2 |
| 12 | 3 | 4 | 2 | 1 | 1 | 2 | 1 | 1 |
| 13 | 4 | 1 | 4 | 1 | 2 | 2 | 1 | 2 |
| 14 | 4 | 2 | 3 | 1 | 1 | 2 | 2 | 1 |
| 15 | 4 | 3 | 2 | 2 | 2 | 1 | 1 | 1 |
| 16 | 4 | 4 | 1 | 2 | 1 | 1 | 2 | 2 |

Table 4.7: final version of orthogonal array

variance that can not be explained by the model) [39]. The F ratio (or value) measures the ratio of variance explained by the factor and the variation explained by the error term [39]. Simply speaking, how good is the model versus how bad is the model. Finally, the p-value (in column labelled Pr(>F)) shows how likely the size of the given F ratio is obtained in case there is no effect on the results. Commonly, if p is smaller than 0.05, the effect can be viewed as statistically significant [39]. The number of DOF is a result from the number of repetitions and can be calculated with the Equation 4.2 (taken from Roy [35]).

$$\begin{aligned} DOF &= totalNumberOfResults - 1 \\ &= numberOfTrials * numberOfRepetitions - 1 \\ &= 16 * 8 - 1 = 127 \end{aligned} \tag{4.2}$$

The multiple R-squared value of the model is 0.416 while the adjusted R-squared value is 0.344. Multiple R-squared gives a measure on how much

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| A | 3 | 23.89 | 7.96 | 6.59 | 0.0004 |
| B | 3 | 5.00 | 1.67 | 1.38 | 0.2532 |
| C | 3 | 34.32 | 11.44 | 9.46 | 0.0000 |
| D | 1 | 3.88 | 3.88 | 3.21 | 0.0759 |
| E | 1 | 0.35 | 0.35 | 0.29 | 0.5912 |
| F | 1 | 18.91 | 18.91 | 15.64 | 0.0001 |
| G | 1 | 6.98 | 6.98 | 5.77 | 0.0179 |
| D:E | 1 | 4.10 | 4.10 | 3.40 | 0.0680 |
| Residuals | 113 | 136.60 | 1.21 | | |

Table 4.8: ANOVA results

variability in the outcome is explained by the predictors [39]. Having only 41.6% does not seem optimal. Field et al. [39] state further that a model which generalizes well has an adjusted R-squared value that is similar to multiple R-squared, which is also not the case. The high error might possibly be explained by the huge search space in the scenario. Increasing the population and number of generations can lead to improvements, however at the cost of computational time. Whether the model, having this much of an error, will perform well compared to either a genetic algorithm build from values from the literature or compared to random search will be evaluated in Section 5.

Looking at the factors as well as the interaction, significant factors can be seen. The highest F value has the factor F (TournamentSize). The factor is significant with $F(1, 112) = 15.64$, $p < 0.001$. Next, the C (MutationRate) is significant with $F(3, 112) = 9.46$, $p < 0.001$. A (CrossoverType) is also significant $F(3, 112) = 6.59$, $p < 0.001$. Finally G (IndependendMutationRate) is significant with $F(1, 112) = 5.77$, $p < 0.05$. D (ChromosomeType) and the interaction D:E will be mentioned as well with $F(1, 112) = 3.21$, $p < 0.1$ and $F(1, 112) = 3.4$, $p < 0.1$ respectively. There is not enough evidence to suggest significance for the factors B (CrossoverRate) and E (GeneType). Particularly noteworthy is the insignificance of the CrossoverRate. The low influence of GeneType, however, might be explained by the fact, that it does not have an impact on the action selected apart from more granularity of the parameters when Dictionary encoding is used. To calculate the percentage

contribution of each factor, Equations 4.3 and 4.4 (gathered from from Yang and El-Haik [34]) are used.

$$SS_T = SS_A + SS_B + SS_C + ... + SS_{error} \tag{4.3}$$

$$contribution_A = SS_A / SS_T * 100 \tag{4.4}$$

The percentage contribution of all factors is plotted in 4.4 and again shows the high error of the model.

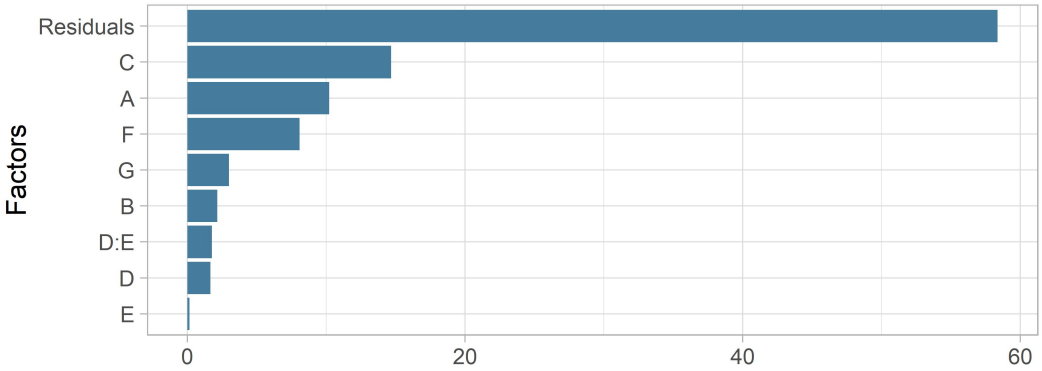



Figure 4.4: percentage contribution

### Main-effects and interaction chart

Identifying the optimal conditions needs analysis of the main effects per factor. They allow to predict the levels that lead to the best result [35].

> *"The main-effects chart is a plot of average responses at different levels of a factor versus the factor levels." [34]*

For every factor, the mean of all results per level is summed up and subsequently divided by the number of runs per level. The resulting main-effect charts are visualized in Figure 4.5. In case there is no interaction, the optimal setting is easily determined by using the main effects chart. Go over every factor in the chart and use the best level (in this experiment, the level with the highest value).

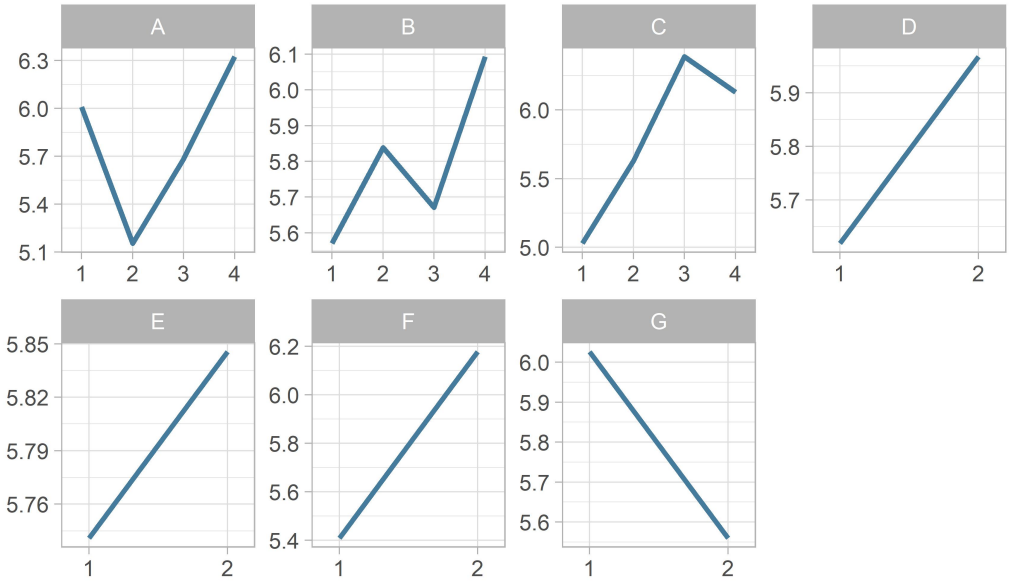



Figure 4.5: Main Effects

If interactions, however, exist, they might have an influence on the best settings and need further investigation [34]. The previously defined interaction is visualized in an interaction plot in Figure 4.6. The calculation is similar to calculating main effects, now the mean response for each level combination is summed up and divided by the number of runs per combination. The crossing of lines in the interaction plot indicates that an interaction between the two factors might exist [39]. The more parallel the lines are, the less likely an interaction. The magnitude of the angle between the lines corresponds to the degree of interaction presence [35].

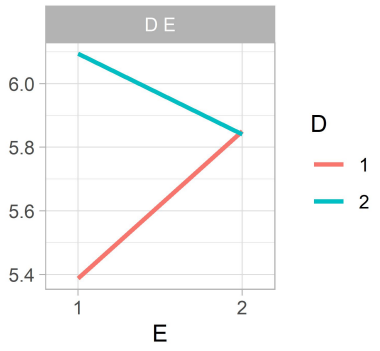



Figure 4.6: interaction plot

### Selection of optimal setting

When choosing the optimal setting, the first step is to look at the best main effects combination. For this experiment, the best combination according the main effects is the following: $A_4$, $B_4$, $C_3$, $D_2$, $E_2$, $F_2$, $G_1$.

Analysing the interaction graph and the ANOVA table, the interaction D:E however might be significant. The interaction plot in Figure 4.6 suggest $D_2$ and $E_1$ as the best combination. This is optimal, as $D_2$ is also recommended by the main effects. While $E_1$ does not correspond to its main effect, the low F value of the factor E suggests low significance anyway. Concluding this line of thought, the combination $A_4$, $B_4$, $C_3$, $D_2$, $E_1$, $F_2$, $G_1$ is preferred.

**Optimum performance calculation** To calculate the predicted results of a genetic algorithm with these settings, optimal performance calculation can be applied. Equation 4.5, which incorporates only the suggested main effects, and Equation 4.6, which includes the additional interaction term D:E. The calculation formulas are sourced from Roy [35].

$$\begin{aligned} Y_{opt} &= \overline{T} + (\overline{A_4} - \overline{T}) + (\overline{B_4} - \overline{T}) + (\overline{C_3} - \overline{T}) + (\overline{D_2} - \overline{T}) + \\ &\quad (\overline{E_2} - \overline{T}) + (\overline{F_2} - \overline{T}) + (\overline{G_1} - \overline{T}) \\ &= 8.06 \end{aligned} \tag{4.5}$$

$$\begin{aligned} Y_{opt} &= \overline{T} + (\overline{A_4} - \overline{T}) + (\overline{B_4} - \overline{T}) + (\overline{C_3} - \overline{T}) + (\overline{D_2} - \overline{T}) + \\ &\quad (\overline{E_1} - \overline{T}) + ([\overline{DxE}]_2 - \overline{T}) + (\overline{F_2} - \overline{T}) + (\overline{G_1} - \overline{T}) \\ &= 8.13 \end{aligned} \tag{4.6}$$

The inclusion of the interaction term D:E results in an improved performance estimation from 8.06 to 8.13. Consequently, the final optimal combination of control parameters is determined as follows: CrossoverType: Uniform 0.5, CrossoverRate: 0.9, MutationRate: 0.3, ChromosomeType: Time+NPC, GeneType: Integer, TournamentSize: 4 and IndividualMutationRate: 0.1.

The added higher granularity for the crossover function was preferred, which is provided by the chromosome encoding 'Time+NPC'. This encoding allows for the crossover operation to exchange actions between actors across different time steps. Notably, the choice of Uniform 0.5 as the CrossoverType as well as the high CrossoverRate and MutationRate is surprising. It remains to be seen, if the genetic algorithms will behave similar to random search or if the high rates will contribute to maintain diversity in its population.

### Signal-to-Noise Ratio

As discussed earlier, the ANOVA model has a high error, which suggests high randomness. Taguchi recommends using signal-to-noise ratio (S/N) to reduce the variability, as using only the mean of the results (which is used when calculating the main effects) does not take the variation into account [35]. S/N quantifies the ratio between the 'desired' signal to the 'undesired' noise. Essentially, a higher signal-to-noise ratio corresponds to reduced variance.

> *"The use of the S/N ratio offers an objective way to look at the two characteristics (consistency and average value) together." [35]*

S/N is calculated in two steps [35]. The first step involves determining the mean square deviation (MSD). Depending on the quality characteristic, a different equation has to be chosen. In this scenario, a higher result is better, thus Equation 4.7 is used for each repetition, with $y_1$ being the result of

repetition 1 and $n$ the number of repetitions. The signal-to-noise ratio is calculated in the second step in Equation 4.8.

$$MSD = (1/y_1^2 + 1/y_2^2 + 1/y_3^2 + ...)/n \tag{4.7}$$

$$S/N = -10log_{10}(MSD) \tag{4.8}$$

Generating the main effects and performing ANOVA is done using the S/N instead. As a side note, it's important to mention that only 15 DOF are available for the ANOVA analysis, as repetitions per run are merged when calculating S/N, which can be observed in Equation 4.9.

$$\begin{aligned} DOF &= totalNumberOfResults - 1 \\ &= numberOfTrials * 1 - 1 \\ &= 16 - 1 = 15 \end{aligned} \tag{4.9}$$

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| A | 3 | 5.14 | 1.71 | 3.54 | 0.3682 |
| B | 3 | 2.04 | 0.68 | 1.40 | 0.5397 |
| C | 3 | 10.59 | 3.53 | 7.28 | 0.2644 |
| D | 1 | 1.29 | 1.29 | 2.67 | 0.3498 |
| E | 1 | 0.39 | 0.39 | 0.81 | 0.5329 |
| F | 1 | 4.43 | 4.43 | 9.15 | 0.2033 |
| G | 1 | 2.28 | 2.28 | 4.70 | 0.2751 |
| D:E | 1 | 0.92 | 0.92 | 1.89 | 0.4005 |
| Residuals | 1 | 0.48 | 0.48 | | |

Table 4.9: S/N ANOVA results

Based on the p values obtained from the ANOVA results in Table 4.9, no factor is able to reject the null hypothesis, which states that a factor

has no significant effect. Considering this fact, further investigations were deemed unnecessary. The often recommend signal-to-noise ratio does not appear suitable for this experiment, thus the previously calculated settings in Paragraph 4.3.3 will remain unchanged.

### Elite Selection

Although the optimal hyperparameter setting are reviewed in Chapter 5, a problem was obvious when analysing results of a GA using the settings from Section 4.3.3. Setbacks in the fitness value between two generations happen frequently, likely due to the high crossover and mutation rates. In order to mitigate this problem, elite selection with a size of 2 was implemented. Per generation, the two best individuals are now copied into the next generation without modifications, which makes worse performance between generations not possible. It is important to note, that these two individuals can still be selected by tournament selection for modification, its just that a copy of them is automatically saved. Figure 4.7 compares both no elite selection with an elite selection of 2. For a few selected repetitions, the best individual fitness per generation is plotted.

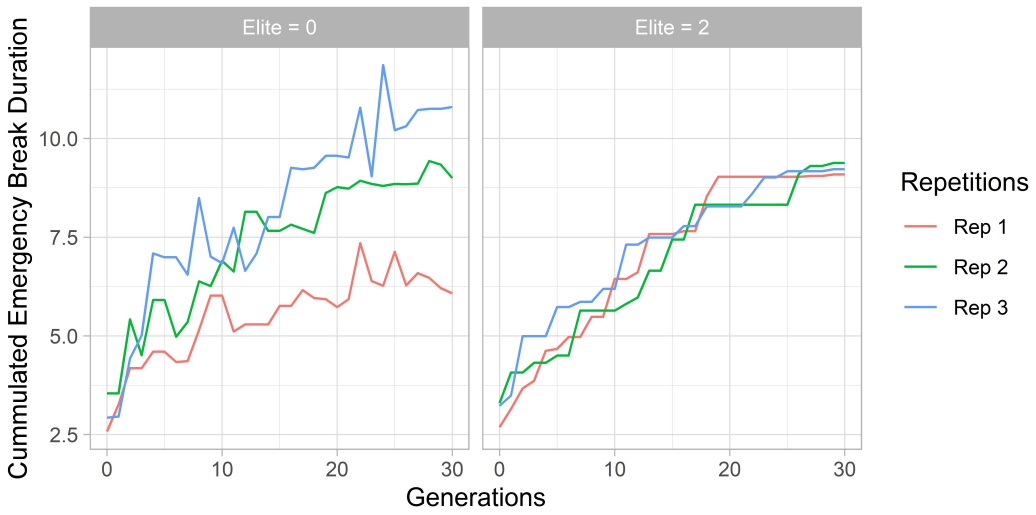



Figure 4.7: elite selection comparison over GA generations

For analysing the statistical significance in the differences in mean between a genetic algorithm using elite selection of 2 vs no elite selection, a t-test can be used. To account for possible violation of the assumption of homogeneity, a robust welch's t-test is applied, which adjusts the DOF accordingly [39]. On average, using elite improved the performance (M = 8.52, SE = 0.31), compared to using no elite (M = 7.92, SE = 0.55). This difference was not significant $t(14.21) = 0.96$, $p > 0.05$; however, it did represent a small-sized effect r = 0.25. A visual comparison of both GAs, each repeated 10 times, can be seen in Figure 4.8. Due to the existence of the aforementioned small effect as well as the reduced variance, it is concluded that elite selection of 2 is added to the settings from 4.3.3, which will be used in Chapter 5.

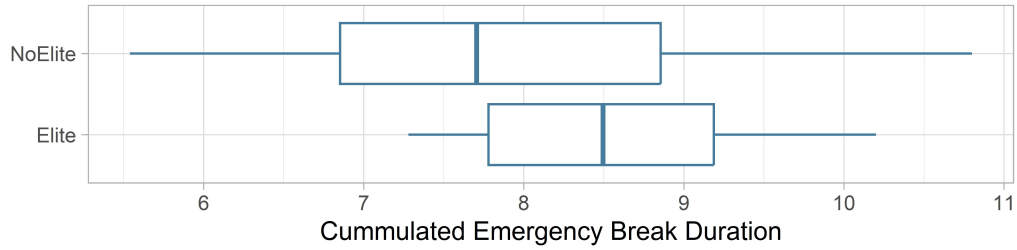



Figure 4.8: elite selection comparison results

**Effect Size** According to Field et al. [39] effect sizes provide an objective measure on the importance of an effect, where 0 means no effect and 1 means a perfect effect. They allow for a standardized measure and are not affected by sample size. He further recommends to utilize the widely used suggestions made by Cohen [40, 41] on defining between a large or small effect:

- r = .10 (small effect): The effect explains 1% of the variance.
- r = .30 (medium effect): The effect explains 9% of the variance.
- r = .50 (large effect): The effect explains 25% of the variance.

Effect sizes also be used to compare different algorithms in Chapter 5 as well and are calculated for a t-test using Equation 4.10.

$$r = \sqrt{\frac{t^2}{t^2 + DOF}} \tag{4.10}$$

# 5 Evaluation

This Chapter will compare the GA settings proposed in Section 4.3.3 with the best settings found in Section 4.2 as well as with random search. The three different algorithms that are going to be compared are as follows:

- **Default** Genetic Algorithm - using the best settings found in Section 4.2
  - CrossoverType: Two-Point, CrossoverRate: 0.8, MutationRate: 0.20, TournamentSize: 4, ChromosomeType: Time, GeneType: Integer, IndividualMutatioRate: 0.1 and EliteSelection: 0.
  - The algorithm will be run 10 times. The population of 96 needs is simulated 31 times per run (30 generations + initialization), which leads to $96 * 31 * 10 = 29,760$ simulations.
- **Optimized** Genetic Algorithm - using the recommended settings found in Section 4.3.3
  - CrossoverType: Uniform 0.5, CrossoverRate: 0.9, MutationRate: 0.3, ChromosomeType: Time+NPC, GeneType: Integer, TournamentSize: 4 and IndividualMutationRate: 0.1 and EliteSelection: 2.
  - The algorithm will be run 10 times. The population of 96 needs is simulated 31 times per run (30 generations + initialization), which leads to $96 * 31 * 10 = 29,760$ simulations.
- **Random** Search
  - Randomly choosing actions, using the probabilities from Table 3.1.
  - The algorithm will be run 10 times each with 2,976 simulations, always taking the maximum value as the result. 29,760 simulations were executed in total.

The evaluation is done over four different start scenarios which can be found in the Appendix at Figures 1 and 2.

## 5.1 Start Scenario 1

First, a comparison of the three algorithms was made on start scenario 1. This is the same start scenario that was used in for the optimizations in Chapter 4. The results are shown in Figure 5.1.

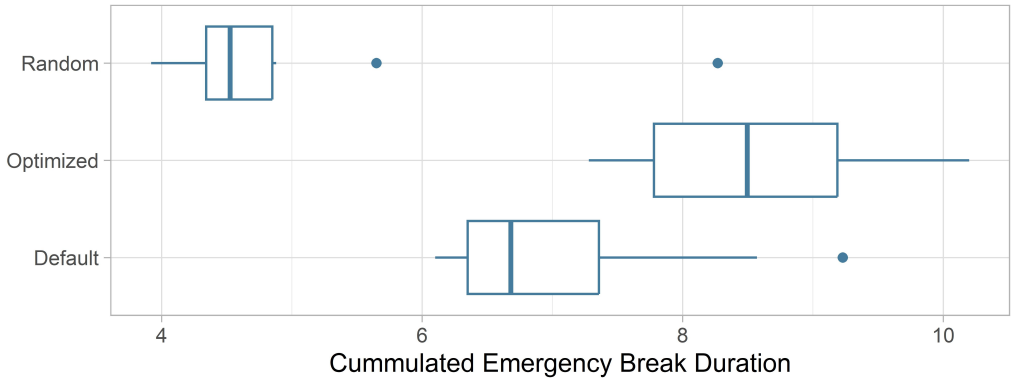



Figure 5.1: start scenario 1: Default GA vs Optimized GA vs Random Search

Analysing the graph, the Optimized GA clearly outperformed the Default GA as well as Random Search. Welch's t-test shows that, on average, greater fitness is achieved by using Optimized GA (M = 8.52, SE = 0.31) than by using Default GA (M = 7.09, SE = 0.34). This difference was significant $t(17.87) = 3.15$, $p < 0.01$. It did represent a large effect r = 0.60. Compared to Random Search (M = 4.943, SE = 0.4), the Optimized GA exhibits even higher performance, with a significant difference $t(16.9) = 7.12$, $p < 0.001$ and a large effect r = 0.87. The better performance of the Optimized GA was however unsurprising, as it was specifically trained on the used start scenario.

To further analyse both genetic algorithms, a look at their performance over the generations next to their diversity chart is of interest. Figure 5.2 plots the mean over 10 repetitions, the outlines show the first and third quantiles.

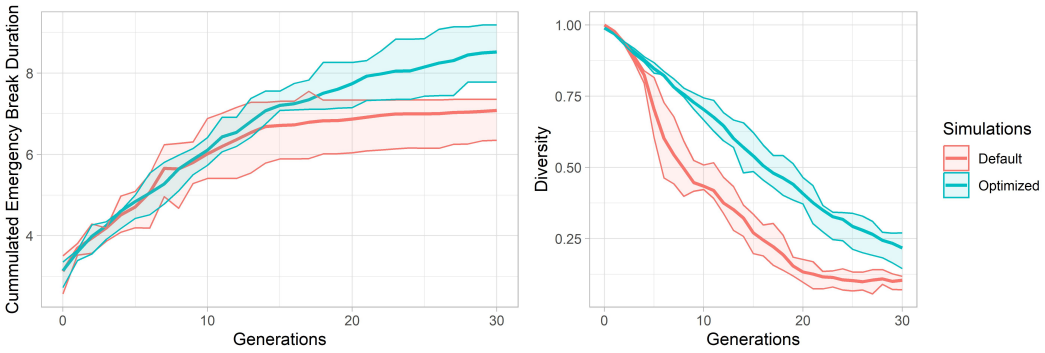



Figure 5.2: start scenario 1: comparison of GAs

After generation 10, the rate of improved fitness of the Default GA decreases compared to the Optimized GA. A combined early sharp decline in the diversity, suggests a connection. The Optimized GA shows to hold the diversity in the population longer, its rate of convergence is linear. The high crossover and mutation rates appear to help with the improved diversity. The graph also shows, that a higher number of generations might not be useful for improved performance, as after 30 generations, the Optimized GA does not seem to hold enough diversity to continue with adequate performance gains.

## 5.2 Start Scenario 2

Start scenario 2 has with nine vehicles and five pedestrians the same amount of NPCs as start scenario 1 and is described in more detail in the Appendix at Figure 1.

Figure 5.3 shows that the Optimized GA can also provide better results compared to the other two algorithms in start scenarios where it has not be trained on. The Optimized GA (M = 9.24, SE = 0.3) achieves better fitness on average compared to the Default GA (M = 7.46, SE = 0.31, with a significant difference $t(17.98) = 4.19$, $p < 0.001$ and a large effect of r = 0.7. The difference in the average fitness value compared Random Search (M = 5.13, SE = 0.13)

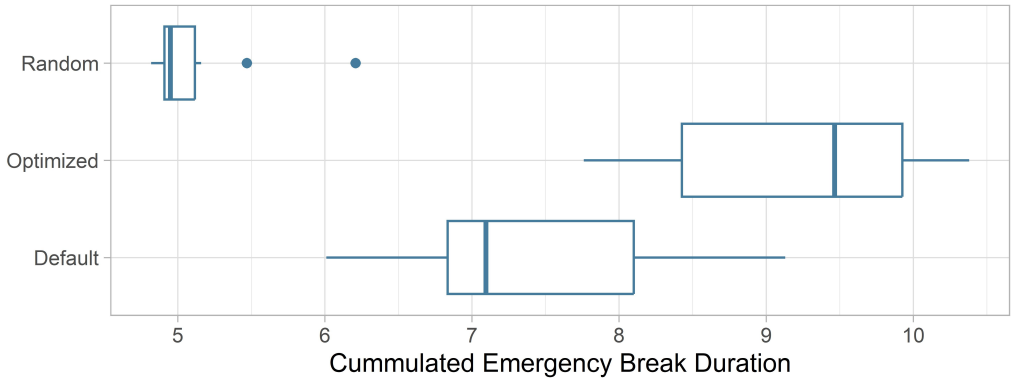



Figure 5.3: start scenario 2: Default GA vs Optimized GA vs Random Search

is significant as well with $t(12.57) = 12.7$, $p < 0.001$ and a large effect of $r = 0.96$.

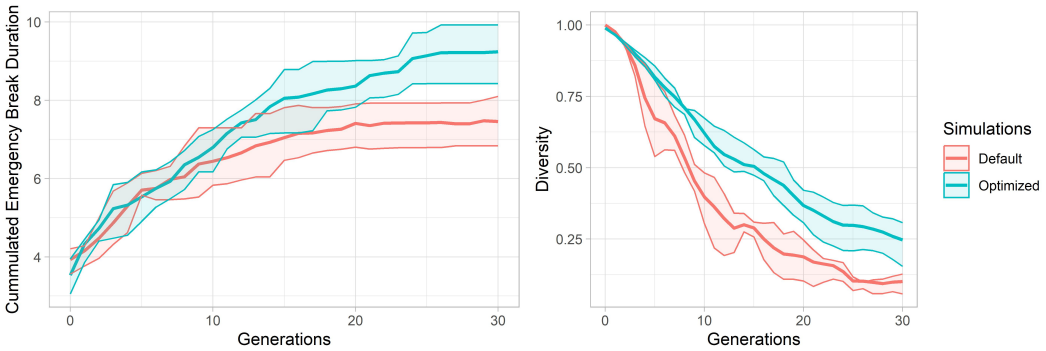



Figure 5.4: start scenario 2: comparison of GAs

The comparison shown in Figure 5.4 seems very similar to the comparison discussed in start scenario 1. On the one hand, the Default GA shows a slower performance increase, which starts already at generation 8. On the other hand, the diversity of the Optimized GA drops a bit sooner. Still both evaluations show, that the Optimized GA performs well in start scenarios with the given amount of NPCs.

## 5.3 Start Scenario 3

Start scenario 3 is described in more detail in Appendix at Figure 2. Five vehicles with three pedestrians are initialized, resulting in a simulation with only a small number of NPCs.


Figure 5.5: start scenario 3: Default GA vs Optimized GA vs Random Search

Looking at the graph, the Optimized GA again outperforms the Random Search, however there seems to be only marginal improvements compared to Default GA. A t-test confirms these findings. While on average, greater fitness is achieved by using Optimized GA (M = 6.66, SE = 0.38) than from using Default GA (M = 6.49, SE = 0.42), this difference was not significant $t(17.83) = 0.29$, $p > 0.05$ with an effect size of r = 0.07. Verifying the better performance of the Optimized GA compared to Random Search (M = 4.619, SE = 0.17) shows a significant difference $t(12.4) = 4.9$, $p < 0.001$ and a large effect size of r = 0.81. To further analyse both genetic algorithms, their performance over the generations next to their diversity chart is shown in Figure 5.6. The mean over 10 repetitions is plotted, the outline show the first and third quantiles.

The rate of improved fitness of the Default GA drops very similar to the Optimized GA. The early decline in the diversity is also not as pronounced as in the previous two comparisons. While the optimized GA shows to still hold the diversity in the population longer, this only has a minimal impact on its average fitness. The similarity in performance might be explained by

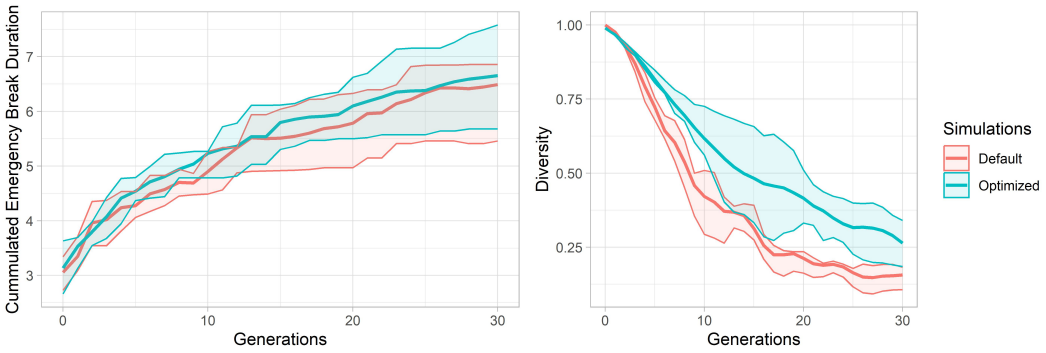



Figure 5.6: start scenario 3: comparison of GAs

the smaller search space, which is a result of the reduced number of NPCs. Here, high mutation and crossover rates might not have the previously pronounced advantage.

## 5.4 Start Scenario 4

Start scenario 4 can be seen in Appendix at Figure 2. Eighteen vehicles with ten pedestrians are initialized, resulting in the start scenario with the most NPCs.

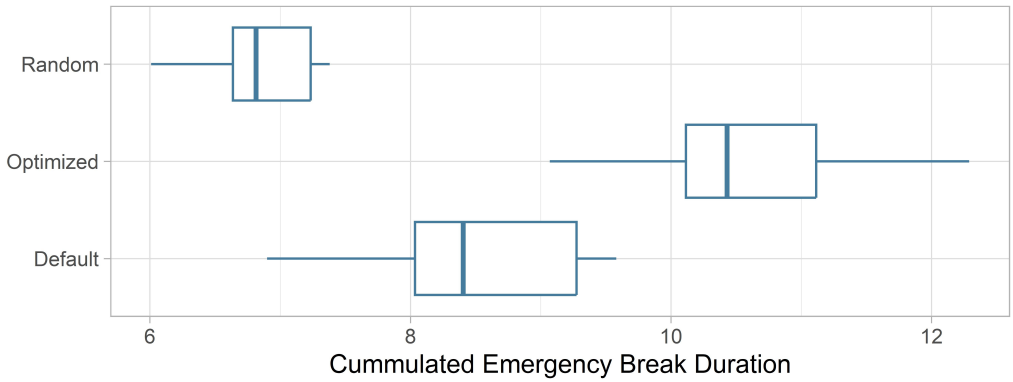



Figure 5.7: Start Scenario 4: Default GA vs Optimized GA vs Random Search

Figure 5.7 shows the Optimized GA clearly outperforming the Default GA as well as Random Search. The Optimized GA (M = 10.60, SE = 0.29) has significantly greater fitness then the Default GA (M = 8.46, SE = 0.28) with $t(17.98) = 5.30$, $p < 0.001$, and a large effect $r = 0.78$. Compared to Random Search (M = 6.86, SE = 0.14), the greater fitness of the Optimized GAs is significant with $t(11.71) = 12.7$, $p < 0.001$, and a large effect $r = 0.96$. Both genetic algorithms performance over the generations next to their diversity chart is compared in Figure 5.8.

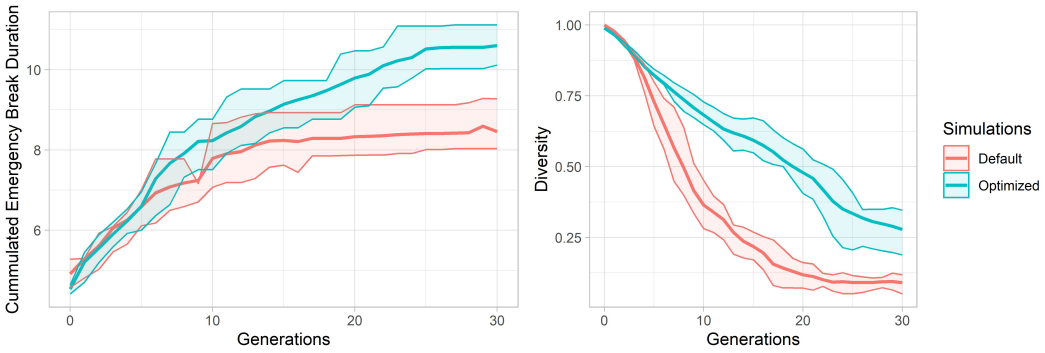



Figure 5.8: start scenario 4: comparison of GAs

Already at generation 5, the average rate of improved fitness of the Default GA drops compared to the Optimized GA. This decline is accompanied by an early, sharp reduction in the diversity. In contrast, the optimized GA shows to be able to hold the diversity much longer at a high level, its rate of diversity drop is linear. The results underline the statement made in Section 5.3, asserting that the Optimized GA excels at scenarios with lots of NPCs, where a vast search space is given.

# 6 Conclusion

This master's thesis begins by explaining the concept of both genetic algorithms and behavior trees in Chapter 2. The subsequent Chapter 3 outlines the design and implementation of both tools. The stated search goal is within the domain of autonomous vehicle testing, specifically in generating critical scenarios. To achieve this, the behavior tree aims to control the EGO vehicle in a realistic manner while the genetic algorithm maximises the cumulated emergency stop duration by commanding all NPCs in the scenario.

In Chapter 4, the thesis successfully optimized control parameters of a genetic algorithm through literature research and subsequent comparison testing in order to find a suitable population size. Utilizing a Taguchi orthogonal array, the remaining hyperparameter were selected and evaluated for their significance. Finally, in Chapter 5, the genetic algorithms were evaluated. The results showed that the optimized settings for a GA provide significant improvements in most start scenarios compared to general settings from existing literature as well as compared to random search. Only in the case of a small number of NPCs, the performance seemed to be on par with a genetic algorithm utilizing settings from the literature.

## 6.1 Research Questions

### 6.1.1 Research Question 1

> *Is a genetic algorithm suitable for generating critical driving scenarios compared to random search?*

Compared to random search, research question 1 definitely holds true. The optimized genetic algorithm will drastically improve performance when using the given cost function. In all four analysed starting scenarios, random search resulted in significantly worse results. There were no cases where a random search surpassed the mean result of the optimized genetic algorithm runs.

### 6.1.2 Research Question 2

> *Can the performance of a genetic algorithm be improved by optimizing the control parameter using the Taguchi method?*

Addressing the second research question, the answer is affirmative. The genetic algorithm optimized utilizing the Taguchi method showed significant improvements in three out of the four start scenarios compared to using control parameters suggested by the literature. The only exception occurred in a start scenario with a small number of NPCs, where the difference was not statistically significant. It can be concluded, that Taguchi orthogonal testing, which only needs a minimal amount of experiment runs, can lead to impressive performance improvements when it comes to optimizing the control parameter of a genetic algorithm.

## 6.2 Outlook and Future Work

Significant improvements to the genetic algorithm as well as to the Traffic Manger are possible. Two main paths seem to provide valuable enhancements.

### 6.2.1 Additional Actions

While the the provided actions by the Action Interface (see Section 3.1.1) already produce a vast search space, introducing additional actions might improve variation in the results and provide even more complex critical

scenarios. Examples for new actions include applying a lane offset, ignoring red traffic lights or simulating a blown tire. Implementing these actions will, in some cases, require only minimal effort and will allow a genetic algorithm to explore even more complex action sequences in its search.

### 6.2.2 Oracle Functions

The utilization of oracle functions to select interesting scenarios during the runtime of a genetic algorithm was already done by Almanee et al. [9]. Multiple oracle functions can be designed to test for different thresholds. For instance, one oracle function might assess , if the ego vehicle has exceeded the speed limit, a different function checks for crashes where the ego vehicle is at fault. Additional ideas include an oracle that evaluates if the comfort level of the passenger falls below a certain threshold. These functions will test each individual in the population. In case one or multiple functions return true, the individual scenario will be saved and automatically categorized according to the types of thresholds. The genetic algorithm however remains unaffected.

Oracle functions have the potential to extract multiple different scenarios from only one genetic algorithm run. The automatic categorization drastically improves the usefulness of this testing approach. Otherwise each search result needs to be evaluated manually for identifying errors in the autonomous driving system.

# Bibliography


[1] Jackie Ayoub et al. "From Manual Driving to Automated Driving: A Review of 10 Years of AutoUI." In: *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. AutomotiveUI '19: 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications. Utrecht Netherlands: ACM, Sept. 21, 2019, pp. 70–90. ISBN: 978-1-4503-6884-1. DOI: 10.1145/3342197.3344529. URL: https://dl.acm.org/doi/10.1145/3342197.3344529 (visited on 12/10/2023) (cit. on p. 1).

[2] Dimitris Milakis, Bart Van Arem, and Bert Van Wee. "Policy and society related implications of automated driving: A review of literature and directions for future research." In: *Journal of Intelligent Transportation Systems* 21.4 (July 4, 2017), pp. 324–348. ISSN: 1547-2450, 1547-2442. DOI: 10.1080/15472450.2017.1291351. URL: https://www.tandfonline.com/doi/full/10.1080/15472450.2017.1291351 (visited on 12/10/2023) (cit. on p. 1).

[3] Florian Klück. "Search-based Testing of Automated Driving Systems." PhD thesis. 2022 (cit. on pp. 1, 2).

[4] Hermann Felbinger et al. "Comparing two systematic approaches for testing automated driving functions." In: *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE). ISSN: 2378-1297. Nov. 2019, pp. 1–6. DOI: 10.1109/ICCVE45908.2019.8965209 (cit. on pp. 1, 2).

[5] Nidhi Kalra and Susan M. Paddock. "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" In: *Transportation Research Part A: Policy and Practice* 94 (Dec. 2016), pp. 182–193. ISSN: 09658564. DOI: 10.1016/j.tra.2016.09.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0965856416302129 (visited on 12/10/2023) (cit. on p. 1).

[6] SAE International. *SAE J3016 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Tech. rep. Visited on 01/07/2022. SAE International, 2021. URL: https://www.sae.org/standards/content/j3016_202104. Cit. on pp. 2, 22 (cit. on p. 1).

[7] Lorenz Klampfl, Florian Klück, and Franz Wotawa. "Using genetic algorithms for automating automated lane-keeping system testing." In: *Journal of Software: Evolution and Process* n/a (n/a), e2520. ISSN: 2047-7481. DOI: 10.1002/smr.2520. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2520 (visited on 03/29/2023) (cit. on pp. 2, 11).

[8] David Kaufmann et al. "Critical and Challenging Scenario Generation based on Automatic Action Behavior Sequence Optimization: 2021 IEEE Autonomous Driving AI Test Challenge Group 108." In: *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. 2021 IEEE International Conference on Artificial Intelligence Testing (AITest). Aug. 2021, pp. 118–127. DOI: 10.1109/AITEST52744.2021.00032 (cit. on p. 2).

[9] Sumaya Almanee et al. "scenoRITA: Generating Less-Redundant, Safety-Critical and Motion Sickness-Inducing Scenarios for Autonomous Vehicles." In: (2021). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2112.09725. URL: https://arxiv.org/abs/2112.09725 (visited on 10/19/2023) (cit. on pp. 2, 27, 57).

[10] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992. ISBN: 978-0-262-27555-2. DOI: 10.7551/mitpress/1090.001.0001. URL: https://direct.mit.edu/books/book/2574/adaptation-in-natural-and-artificial-systemsan (visited on 12/05/2023) (cit. on pp. 5, 6, 9).

[11] K. L. Mills, J. J. Filliben, and A. L. Haines. "Determining Relative Importance and Effective Settings for Genetic Algorithm Control Parameters." In: *Evolutionary Computation* 23.2 (June 2015), pp. 309–342. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/EVCO_a_00137. URL: https://direct.mit.edu/evco/article/23/2/309-342/986 (visited on 10/13/2023) (cit. on pp. 5, 6, 8, 26, 27).

[12] Abhishek Majumdar and Debashis Ghosh. "Genetic Algorithm Parameter Optimization using Taguchi Robust Design for Multi-response Optimization of Experimental and Historical Data." In: *International Journal of Computer Applications* 127.5 (Oct. 15, 2015), pp. 26–32. ISSN: 09758887. DOI: 10.5120/ijca2015906383. URL: http://www.ijcaonline.org/research/volume127/number5/majumdar-2015-ijca-906383.pdf (visited on 10/27/2023) (cit. on pp. 5–7, 9).

[13] M. Srinivas and L.M. Patnaik. "Genetic algorithms: a survey." In: *Computer* 27.6 (June 1994), pp. 17–26. ISSN: 0018-9162. DOI: 10.1109/2.294849. URL: http://ieeexplore.ieee.org/document/294849/ (visited on 10/13/2023) (cit. on pp. 5, 7, 8, 10, 11, 27).

[14] John Grefenstette. "Optimization of Control Parameters for Genetic Algorithms." In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (Jan. 1986), pp. 122–128. ISSN: 0018-9472. DOI: 10.1109/TSMC.1986.289288. URL: http://ieeexplore.ieee.org/document/4075583/ (visited on 10/13/2023) (cit. on pp. 6, 8–11, 26, 27, 29).

[15] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. "A review on genetic algorithm: past, present, and future." In: *Multimedia Tools and Applications* 80.5 (Feb. 1, 2021), pp. 8091–8126. ISSN: 1573-7721. DOI: 10.1007/s11042-020-10139-6. URL: https://doi.org/10.1007/s11042-020-10139-6 (visited on 03/28/2023) (cit. on pp. 6–11).

[16] Xuemin Xia et al. "Genetic algorithm hyper-parameter optimization using Taguchi design for groundwater pollution source identification." In: *Water Supply* 19.1 (Feb. 1, 2019), pp. 137–146. ISSN: 1606-9749, 1607-0798. DOI: 10.2166/ws.2018.059. URL: https://iwaponline.com/ws/article/19/1/137/39199/Genetic-algorithm-hyperparameter-optimization (visited on 10/06/2023) (cit. on p. 6).

[17] Abid Hussain and Yousaf Shad Muhammad. "Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator." In: *Complex & Intelligent Systems* 6.1 (Apr. 2020), pp. 1–14. ISSN: 2199-4536, 2198-6053. DOI: 10.1007/s40747-019-0102-7. URL: http://link.springer.com/10.1007/s40747-019-0102-7 (visited on 07/23/2023) (cit. on pp. 6, 7, 9, 11).

[18] Kenneth De Jong. "Parameter Setting in EAs: a 30 Year Perspective." In: *Parameter Setting in Evolutionary Algorithms*. Ed. by Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz. Red. by Janusz Kacprzyk. Vol. 54. Series Title: Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–18. ISBN: 978-3-540-69431-1 978-3-540-69432-8. DOI: 10.1007/978-3-540-69432-8_1. URL: http://link.springer.com/10.1007/978-3-540-69432-8_1 (visited on 10/13/2023) (cit. on pp. 7, 8, 12, 25, 26, 31).

[19] Jinghui Zhong et al. "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms." In: *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*. International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). Vol. 2. Vienna, Austria: IEEE, 2005, pp. 1115–1121. ISBN: 978-0-7695-2504-4. DOI: 10.1109/CIMCA.2005.1631619. URL: http://ieeexplore.ieee.org/document/1631619/ (visited on 10/23/2023) (cit. on pp. 9, 27).

[20] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan, 1975 (cit. on pp. 9, 27).

[21] Siew Mooi Lim et al. "Crossover and Mutation Operators of Genetic Algorithms." In: *International Journal of Machine Learning and Computing* 7.1 (Feb. 2017), pp. 9–12. ISSN: 20103700. DOI: 10.18178/ijmlc.2017.7.1.611. URL: http://www.ijmlc.org/index.php?m=content&c=index&a=show&catid=69&id=704 (visited on 12/15/2023) (cit. on p. 10).

[22] S. Marsili Libelli and P. Alba. "Adaptive mutation in genetic algorithms." In: *Soft Computing* 4.2 (July 2000), pp. 76–80. ISSN: 1432-7643.

DOI: 10.1007/s005000000042. URL: http://link.springer.com/10.1007/s005000000042 (visited on 11/21/2023) (cit. on p. 11).

[23] Giorgos Karafotias, Mark Hoogendoorn, and A. E. Eiben. "Parameter Control in Evolutionary Algorithms: Trends and Challenges." In: *IEEE Transactions on Evolutionary Computation* 19.2 (Apr. 2015), pp. 167–187. ISSN: 1089-778X, 1089-778X, 1941-0026. DOI: 10.1109/TEVC.2014.2308294. URL: http://ieeexplore.ieee.org/document/6747993/ (visited on 10/13/2023) (cit. on p. 11).

[24] Michele Collendanchise and Petter Ögren. *Behavior trees in robotics and Al*. Chapman & Hall/CRC artificial intelligence and robotics series 6. Boca Raton: CRC Press/Taylor, Francis Group, CRC Press is an imprint of the Taylor, and Francis Group, an Informa Business, 2019. 1 p. ISBN: 978-0-429-48910-5 978-0-429-95089-6 (cit. on pp. 12, 13).

[25] Matteo Iovino et al. "A survey of Behavior Trees in robotics and AI." In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096. ISSN: 09218890. DOI: 10.1016/j.robot.2022.104096. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889022000513 (visited on 12/05/2023) (cit. on p. 12).

[26] Christopher Iliffe Sprague et al. "Improving the Modularity of AUV Control Systems using Behaviour Trees." In: *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*. 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV). ISSN: 2377-6536. Nov. 2018, pp. 1–6. DOI: 10.1109/AUV.2018.8729810 (cit. on pp. 12, 13).

[27] M.A. Fazal et al. "Estimating groundwater recharge using the SMAR conceptual model calibrated by genetic algorithm." In: *Journal of Hydrology* 303.1 (Mar. 2005), pp. 56–78. ISSN: 00221694. DOI: 10.1016/j.jhydrol.2004.08.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0022169404003890 (visited on 10/23/2023) (cit. on p. 27).

[28] Son Dao, Kazem Abhary, and Romeo Marian. "Maximising Performance of Genetic Algorithm Solver in Matlab." In: *Engineering Letters* 24 (Feb. 2016) (cit. on p. 27).

[29] Bhagyashri Naruka et al. "Parameter Tuning Method for Genetic Algorithm using Taguchi Orthogonal Array for Non-linear Multimodal Optimization Problem." In: *International Journal of Recent Technology and Engineering (IJRTE)* 8.2 (July 30, 2019), pp. 2979–2986. ISSN: 22773878. DOI: 10.35940/ijrte.B2711.078219. URL: https://www.ijrte.org/portfolio-item/B2711078219/ (visited on 10/09/2023) (cit. on p. 27).

[30] Darrell Whitley. "A genetic algorithm tutorial." In: *Statistics and Computing* 4.2 (June 1, 1994), pp. 65–85. ISSN: 1573-1375. DOI: 10.1007/BF00175354. URL: https://doi.org/10.1007/BF00175354 (visited on 03/28/2023) (cit. on p. 27).

[31] Onur Boyabatli and Ihsan Sabuncuoglu. "Parameter selection in genetic algorithms." In: *Journal of Systemics, Cybernetics and Informatics* 4.2 (2004). Publisher: International Institute of Informatics and Cybernetics, p. 78 (cit. on p. 27).

[32] Leila Zahedi et al. *Search Algorithms for Automated Hyper-Parameter Tuning*. Apr. 29, 2021. arXiv: 2104.14677[cs]. URL: http://arxiv.org/abs/2104.14677 (visited on 12/22/2023) (cit. on p. 29).

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing." In: *Science* 220.4598 (May 13, 1983), pp. 671–680. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.220.4598.671. URL: https://www.science.org/doi/10.1126/science.220.4598.671 (visited on 12/22/2023) (cit. on p. 29).

[34] Kai Yang and Basem El-Haik. *Design for six sigma: a roadmap for product development*. 2nd ed. London: McGraw-Hill [distributor], 2009. 741 pp. ISBN: 978-0-07-154767-3 (cit. on pp. 29–34, 36, 39, 40).

[35] Ranjit K. Roy. *A primer on the Taguchi method*. Competitive manufacturing series. New York: Van Nostrand Reinhold, 1990. 247 pp. ISBN: 978-0-442-23729-5 (cit. on pp. 30, 31, 33, 35, 37, 39–42).

[36] Coşkun Hamzaçebi. "Taguchi Method as a Robust Design Tool." In: *Quality Control - Intelligent Manufacturing, Robust Design and Charts*. Ed. by Pengzhong Li, Paulo António Rodrigues Pereira, and Helena Navas. IntechOpen, Mar. 24, 2021. ISBN: 978-1-83962-497-1 978-1-83962-498-8. DOI: 10.5772/intechopen.94908. URL: https://www.intechopen.com/books/quality-control-intelligent-manufacturing-robust-

design-and-charts/taguchi-method-as-a-robust-design-tool (visited on 10/28/2023) (cit. on p. 31).

[37] Nazan Danacıoğlu and F. Zehra Muluk. "Taguchi Techniques for 2k Fractional Factorial Experiments." In: *Journal* 34.1 (2005), pp. 83–93. ISSN: 2651-477X-2651-477X (cit. on pp. 33, 34).

[38] Genichi Taguchi et al. *Taguchi's quality engineering handbook*. Hoboken, N.J. : Livonia, Mich: John Wiley & Sons ; ASI Consulting Group, 2005. 1662 pp. ISBN: 978-0-471-41334-9 (cit. on p. 34).

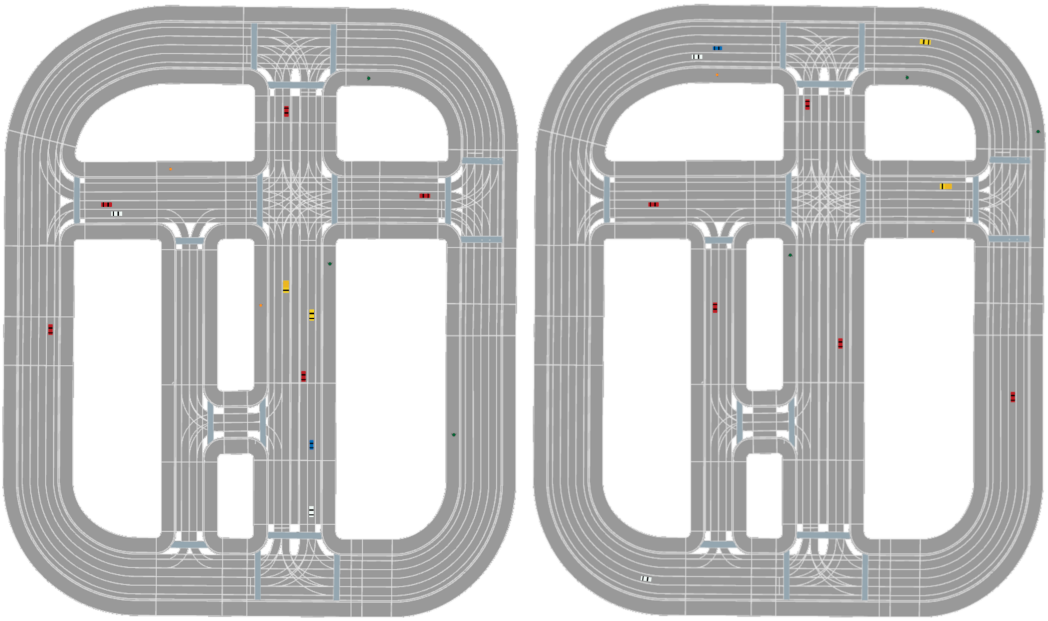
[39] Andy P. Field, Jeremy Miles, and Zoë Field. *Discovering statistics using R*. OCLC: ocn760970657. London ; Thousand Oaks, Calif: Sage, 2012. 957 pp. ISBN: 978-1-4462-0046-9 978-1-4462-0045-2 (cit. on pp. 37, 38, 40, 45).

[40] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. 2nd ed. OCLC: 844923079. Hillsdale, N.J.: L. Erlbaum Associates, 1988. ISBN: 978-1-134-74270-7 (cit. on p. 45).

[41] Jacob Cohen. "A power primer." In: *Psychological Bulletin* 112.1 (1992), pp. 155–159. ISSN: 1939-1455, 0033-2909. DOI: 10.1037/0033-2909.112.1.155. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-2909.112.1.155 (visited on 12/07/2023) (cit. on p. 45).

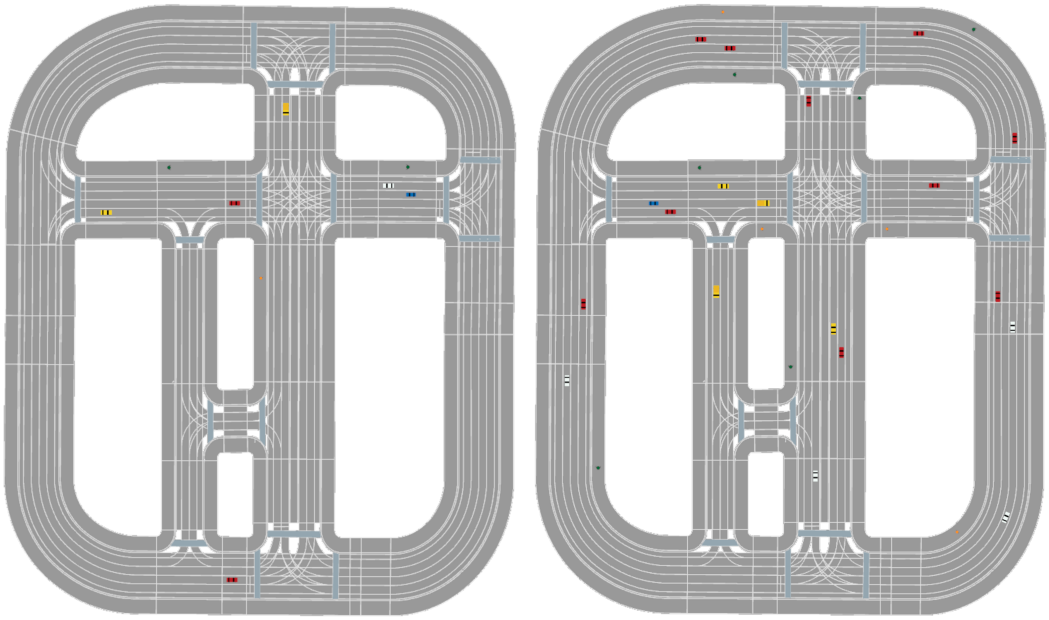
# Appendix

# Appendix A.



Start Scenario 1 Start Scenario 2

Figure 1: start scenarios 1-2



Start Scenario 3 Start Scenario 4

Figure 2: start scenarios 3-4

# Appendix B.

| NO. | rep1 | rep2 | rep3 | rep4 | rep5 | rep6 | rep7 | rep8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 3.76 | 3.90 | 4.75 | 4.23 | 4.32 | 5.75 | 3.94 | 3.95 |
| 2 | 8.06 | 5.20 | 4.75 | 5.04 | 4.63 | 5.79 | 4.32 | 6.00 |
| 3 | 8.62 | 7.89 | 8.76 | 6.44 | 8.77 | 6.68 | 5.96 | 7.22 |
| 4 | 7.65 | 6.95 | 6.49 | 5.35 | 6.24 | 7.97 | 6.52 | 6.42 |
| 5 | 4.26 | 5.45 | 4.55 | 4.20 | 3.80 | 5.29 | 6.05 | 7.02 |
| 6 | 5.26 | 5.21 | 5.71 | 5.59 | 5.48 | 5.64 | 3.61 | 5.47 |
| 7 | 3.92 | 4.01 | 4.51 | 4.80 | 4.08 | 4.43 | 4.10 | 6.05 |
| 8 | 6.60 | 5.69 | 5.79 | 5.79 | 5.43 | 5.03 | 6.11 | 6.05 |
| 9 | 4.93 | 5.05 | 5.17 | 4.91 | 6.53 | 5.04 | 7.66 | 5.73 |
| 10 | 5.84 | 6.72 | 4.87 | 7.13 | 6.82 | 5.74 | 4.66 | 6.78 |
| 11 | 4.93 | 6.21 | 4.10 | 4.67 | 5.94 | 5.19 | 3.91 | 3.96 |
| 12 | 5.54 | 4.84 | 7.10 | 5.83 | 5.96 | 5.17 | 6.02 | 8.94 |
| 13 | 11.22 | 7.88 | 5.94 | 7.00 | 5.88 | 7.05 | 6.40 | 6.66 |
| 14 | 6.05 | 7.40 | 7.50 | 6.51 | 9.58 | 5.03 | 5.35 | 5.09 |
| 15 | 6.58 | 4.35 | 4.50 | 7.21 | 6.38 | 5.77 | 5.45 | 6.08 |
| 16 | 4.35 | 6.66 | 8.57 | 4.44 | 4.49 | 4.89 | 6.72 | 5.37 |

Table 1: hyperparameter tuning: taguchi experiment results

| scenario | rep1 | rep2 | rep3 | rep4 | rep5 | rep6 | rep7 | rep8 | rep9 | rep10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7.28 | 7.31 | 9.09 | 9.38 | 8.16 | 8.05 | 7.69 | 10.20 | 9.22 | 8.83 |
| 2 | 7.76 | 9.42 | 8.28 | 9.51 | 9.96 | 10.27 | 10.38 | 8.87 | 9.82 | 8.12 |
| 3 | 7.67 | 7.34 | 6.86 | 4.86 | 5.00 | 7.14 | 8.33 | 5.51 | 6.18 | 7.66 |
| 4 | 10.19 | 11.57 | 10.67 | 10.71 | 9.07 | 11.25 | 10.19 | 10.09 | 12.29 | 9.97 |

Table 2: evaluation results Optimized GA

| scenario | rep1 | rep2 | rep3 | rep4 | rep5 | rep6 | rep7 | rep8 | rep9 | rep10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6.66 | 6.10 | 6.32 | 7.32 | 7.37 | 9.23 | 6.44 | 8.57 | 6.14 | 6.70 |
| 2 | 7.09 | 6.78 | 7.00 | 7.10 | 6.01 | 8.04 | 6.75 | 8.59 | 9.13 | 8.12 |
| 3 | 8.31 | 6.14 | 6.94 | 8.99 | 5.27 | 4.94 | 6.56 | 6.03 | 6.61 | 5.11 |
| 4 | 8.78 | 9.44 | 8.22 | 7.47 | 7.99 | 9.58 | 9.45 | 8.59 | 6.90 | 8.17 |

Table 3: evaluation results Default GA

| scenario | rep1 | rep2 | rep3 | rep4 | rep5 | rep6 | rep7 | rep8 | rep9 | rep10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3.92 | 4.41 | 4.19 | 4.33 | 4.38 | 4.88 | 8.27 | 4.76 | 5.65 | 4.64 |
| 2 | 5.16 | 4.91 | 4.82 | 4.89 | 4.91 | 5.47 | 4.97 | 6.21 | 4.93 | 4.99 |
| 3 | 4.53 | 4.63 | 4.35 | 4.62 | 4.13 | 4.14 | 4.91 | 5.98 | 4.56 | 4.34 |
| 4 | 7.01 | 6.01 | 6.63 | 7.31 | 6.85 | 6.66 | 7.35 | 6.78 | 7.38 | 6.57 |

Table 4: evaluation results Random Search