AUTHOR, OLDDEGREE

Evaluation of a Genetic Algorithm on generating critical Scenarios in a Traffic Simulation

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Telematik

submitted to

Graz University of Technology

Supervisor

Dr. Some Body

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Some One

Graz, November 2013

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

This is a placeholder for the abstract. It summarizes the whole thesis to give a very short overview. Usually, this the abstract is written when the whole thesis text is finished.

Contents

List of Figures

1 Introduction

better and longer introduction

This Thesis will use a genetic algorithm in order to generate critical driving scenarios for testing ADAS/AD functionality 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 hyperparameter.

1.1 Research Questions

1.1.1 Research Question 1

Is a genetic algorithm suitable for generating critical driving scenarios compared to a random generation?

1.1.2 Research Question 2

Is it possible to improve the performance of a genetic algorithm using the taguchi method?

1.2 ADAS/AD

probably talk about ADAS/AD here...

2 Foundations

2.1 Genetic Algorithm

Pioneered by John Holland (1974), genetic algorithms (GAs) imitate the process of natural selection and the darwinian principle called "survival of the fittest". Genetic algorithms are a subclass of evolutionary algorithms and explore the solution space using a population of individuals (Mills, Filliben, and Haines, 2015). Each individual contains one chromosome, which serves as a candidate solution. Using genetic operations, the individuals mate among themselves and mutate independently.

Find paper

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 (Majumdar and Ghosh, 2015). Individuals that do 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, which form a solution to the search problem. Through successive generations using genetic operations, gene values and position will be optimized, resulting in progressively improved solutions (Srinivas and Patnaik, 1994).

This masters thesis will utilize generational genetic algorithms, where the entire population is replaced each generation. In contrast, steady state genetic algorithms only replace a small fraction of the population at a time (Srinivas and Patnaik, 1994).

This search method on the basis of biological principles allows for a global and dispersed search through its population, which avoids various shortcomings of local search techniques (Grefenstette, 1986). Especially on challenging

search spaces with multiple local optima, a GA is less prone to get stuck on a substandard solution (Katoch, Chauhan, and Kumar, 2021, Xia et al., 2019, Majumdar and Ghosh, 2015). Genetic algorithms offer advantages for complex optimization and non-deterministic polynomial (NP-hard) problems (Hussain and Muhammad, 2020).

> *"GAs can find good solutions within a large, ill-defined search space, and can be readily adapted to a wide variety of search problems" (Mills, Filliben, and Haines, 2015)*

Grefenstette, 1986 further emphasis its ability to outperform gradient techniques on difficult problems with high-dimensional, noisy or discontinuous fitness functions by efficiently exploiting a "relatively simple selection mechanism." The main advantage of a GA in tackling these problems comes from its capability to explore the search space using its entire population (Hussain and Muhammad, 2020). Further advantages are the "high implicit parallelism", which make Genetic algorithms "numerically very efficient" (Marsili Libelli and Alba, 2000).

ref holland

Fundamentally, a genetic algorithm uses a list of simple functions which are inspired by evolution. A basic GA proposed by can be defined as follows.

```
1   simple_genetic_algorithm()
2   {
3       initialize_population();
4       evaluate_population();
5       for(int i = 0; i < num_of_generations; i++)
6       {
7           select_individuals_for_next_population();
8           perform_crossover();
9           perform_mutation();
10          evaluate_population();
11      }
12  }
```

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. Different stopping

methods like time limit, fitness limit, minimum convergence rate or maximum number of generations are available (Majumdar and Ghosh, 2015).

The individuals are chosen using a selection operation, which takes their fitness value under consideration. A crossover rate subsequently determines individuals which 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 on which individuals this operation is applied. The newly generated population will then get evaluated for the next iteration.

As discussed by Hussain and Muhammad, 2020, genetic algorithms suffer from an exploration vs exploitation dilemma. If the algorithm converges too quickly to a solution, most of the search space might not have yet been explored, thus increasing the probability of getting stuck in a local optimum. In contrast to that, 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." (K. De Jong, 2007)*

Due to no real consensus on the best control parameter settings, their optimal selection proves to be difficult, which will be discussed in Section 4.

2.1.1 Chromosomes and Genes

explain genes and chromosomes a bit more

The encoding of genes proved to be, besides the hyperparamter tuning, the main challenge of setting up a genetic algorithm pipeline. A simple and commonly used representation for genes is binary encoding. As suggested by its name, a gene can take on the form of 0 or 1. Further common encodings are octal and hexadecimal representations (Srinivas and Patnaik, 1994, Katoch, Chauhan, and Kumar, 2021).

Srinivas and Patnaik, 1994 suggest to use integer representations in case a optimization problem has real-valued continuous variables. The objects are linearly mapped to integers defined in a specific range. It is also possible to

again represent these integers as binary encodings. 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" (Katoch, Chauhan, and Kumar, 2021).

In case no encodings fits the given problem, a custom encoding can be used. Here the genetic operation crossover and mutation might have to be tailored accordingly.

2.1.2 Population Size

Controlling the population size, which defines the number of individuals per generation, has a direct impact on the performance of a genetic algorithm. Research seams to be in agreement, that a small population leads to less diverse individuals and might provide an insufficient sample size, which can lead to a premature convergence to a local optimum. 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 as well as slower convergence to an optimum (Grefenstette, 1986, Katoch, Chauhan, and Kumar, 2021, K. De Jong, 2007).

Increasing the population size will increase the degree of parallelism, as each individual represents one search point (Mills, Filliben, and Haines, 2015).

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 selected based on their fitness value, ensuring their increased representation from generation to generation. Weak solutions will be discarded over time (Srinivas and Patnaik, 1994).

The selection algorithm needs to satisfy two requirements. On the one hand, high selection pressure will lead to decreased diversity in the population

resulting in premature convergence (Katoch, Chauhan, and Kumar, 2021). The algorithm will subsequently behave more like a local search method, a hill-climber or a "greedy" algorithm (K. De Jong, 2007). The initial low average fitness value of the population will in combination with a few good performing individuals lead to them overtaking the population, drastically reducing diversity. On the other hand, a 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 (Majumdar and Ghosh, 2015). According to Hussain and Muhammad, 2020, who provide a extensive comparison of different selection techniques, the choice of selection method highly affects the performance of the Genetic Algorithm.

Roulette wheel selection is a popular mechanism, where each individual corresponds to an area on the roulette wheel, based on its fitness value (). Grefenstette, 1986 points out the scaling problem as a 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 (Katoch, Chauhan, and Kumar, 2021). Individuals will be sorted based on their performance and their relative rank is inserted into the roulette wheel.

cite Holland, else: Hussain and Muhammad, 2020

Tournament selection is a popular alternative to roulette wheel selection. First, a tournament size t has to be chosen. 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 it also has an impact on the diversity of the population (Hussain and Muhammad, 2020). In a comparison of different selection methods by Jinghui Zhong et al., 2005, tournament selection was deemed to be the most performant, converging more quickly then for example roulette wheel selection.

Elite selection is an additional method which can enhance a selection operation, proposed by K. A. De Jong, 1975. The best n individuals are automatically chosen to go into the next generation. This can help improve

performance, 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). Its resulting offspring is a combination of both parent genes. Grefenstette, 1986 highlights, that "first, it provides new points for further testing within the hyperplanes already represented in the population" and second, that "crossover introduces representatives of new hyperplanes into the population".

Different types of crossovers can be chosen, the most simple approach being the single-point crossover. Here a randomly chosen point along the length of the chromosome is chosen. The two parents will swap their genetic information at this point, generating two new offsprings (Katoch, Chauhan, and Kumar, 2021). Extension are the two- or multipoint crossover operations, where chromosomes are divided by k segments, which will get exchanged. They eliminate a disadvantage of "the single-point crossover bias toward bits at the ends of strings" (Srinivas and Patnaik, 1994).

A second crossover operation is the uniform crossover where gene swaps between parents are randomly decided by a given probability, independent to the exchange of other genes (Katoch, Chauhan, and Kumar, 2021).

Srinivas and Patnaik, 1994 proposed the notions of positional and distributional basis. If a crossover operation has a positional bias, the probability of a bit to be swapped depends mainly on its position. Examples are the mentioned single and multipoint crossover operators. Uniform crossover can be found at the other end of the spectrum, as it has a maximal distributional bias, completely disregarding any positional information. Ignoring of the genes position results in a higher disruptiveness, which has potential drawbacks. However it will be more exploratory in homogeneous populations where k-point crossovers might struggle to explore. Srinivas and Patnaik, 1994 suggest that uniform crossover is more useful in small populations. Larger populations inherently are more diverse, making k-point crossover more suitable.

Various other crossover operations are analysed in paper .

Find good research paper, maybe Katoch, Chauhan, and Kumar, 2021

Not only the crossover type has to be chosen, a crossover rate also needs definition. This value defines, how likely a single crossover operation is to be applied on the selected individuals. A suitable crossover probability again is 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 (Grefenstette, 1986).

2.1.5 Mutation

The mutation operation is applied after crossover as a means 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 it 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 (Srinivas and Patnaik, 1994). 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 needs to be tailored accordingly.

> *"If the mutation is not considered during evolution, then there will be no new information available for evolution."(Katoch, Chauhan, and Kumar, 2021)*

The settings for mutation rates again needs to be chosen carefully. If they are set 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. (Klampfl, Klück, and Wotawa, 2023, Grefenstette, 1986).

2.1.6 Adaptive Control Parameters

Various literature suggests adaptively controlling selection methods, crossover rates and mutation rates over the duration of a genetic algorithm (...). The

insert papers

main goal is to mitigate the previously mentioned exploration-exploitation problem. In the beginning, exploration is desired, meaning low selection pressure as well as high crossover and mutation rates are required. However to the end of the genetic algorithms duration, convergence to an optimum is desired, meaning more "elite" selection methods as well as less variation due to crossover and mutation (Srinivas and Patnaik, 1994).

Marsili Libelli and Alba, 2000 proposed a method, where different mutation rates are used depending on the fitness of the individual. They claim, that a higher mutation probability for low fitness population results in a more efficient search. Hussain and Muhammad, 2020 argue for a selection operation, where initially a low selection pressure is applied while increasing it to the end, stating that this approach will help mitigate both mentioned competing criteria, namely premature convergence as well as slow convergence.

Contrary, K. De Jong, 2007 is more critical on adaptive control parameters. Although adaptive control parameter 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." (K. De Jong, 2007)*

K. De Jong, 2007 adds, that "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 Masters Thesis will not utilized adaptive control parameters.

2.2 Behaviour Tree

insert a short introduction to BT

A behaviour tree is a decision tree.

3 Implementation

3.1 Traffic Manager

A custom developed Traffic Manger will be used to simulate an traffic environment, which can subsequently be controlled by the Genetic Algorithm. The Traffic Manager is not part of the Master's Thesis and will be regarded as a "blackbox", only receiving a brief introduction. Generally speaking, it will simulate traffic using a start scenario, in which the positions and types of actors (vehicles and pedestrians) are defined. A simulation always consist of at least one EGO vehicle. Additionally any number of NPCs can be added.

While the NPCs are only controlled by the Traffic Manager, the EGO vehicle can be either partly or completely controlled by an ADAS/AD function. The goal of the genetic algorithm is subsequently to test this function for safety, comfort etc... by controlling the NPCs in order to generate critical scenarios.

For all simulations done by this thesis, the Traffic Manager was set to 100Hz and the simulation duration set to 35 seconds.

3.1.1 Action Interface

In order to control the behaviour of all actors inside the simulation, actions can be requested over the Action Interface, which is provided by the Traffic Manager. An action will initiate a certain behaviour from an actor and can

be set to at any timestep[1]. Pedestrians and vehicles have a different set of possible actions. If no action is set, the actor will behave in a normal manner inside the simulation. This means that the actor will follow along its given path until a new action changes its behaviour.

The following list shows all actions provided by the traffic manager which were available for the genetic algorithm at the time of this master thesis.

- JunctionSelection
  - vehicle_id: int, step: int, junction_selection_angle: float (radiants)
  - Vehicles will chose which direction to take at junctions based on this angle.
- LaneChange
  - vehicle_id: int, step: int, direction: float, distance: float, (optional) delay: 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 it is 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.
- CrossAtCrosswalk

[1] depending on the ADAS/AD function under test, the Action Interface might be disabled for the EGO vehicle

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

The genetic algorithm will control only NPCs, while a behaviour tree is used for the EGO vehicle itself. Both methods use the Action Interface to control their respective actors. An overview is provided in figure 3.1.

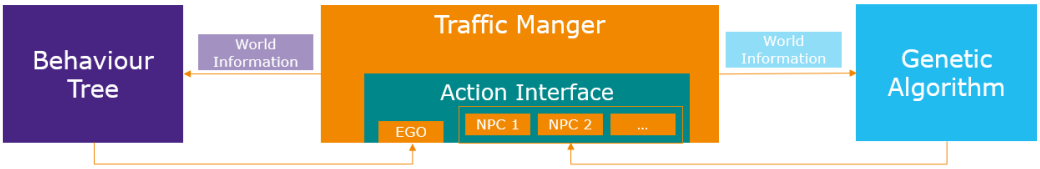

Figure 3.1: Action Interface Structure

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. For implementing the genetic algorithm, the Python library DEAP[2] was chosen. It is a popular tool in academia and allows for high customisability. The algorithm has full access over setting actions for all NPCs. Searching for sequences of actions, the GA tries to optimize a cost function. This section aims to explain how the genetic algorithm was implemented and which different control parameter are variable.

A few default settings for the genetic algorithm were chosen in the beginning. It was decided for example, that the genetic algorithm will 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

Using a fixed maximum number of generation is a commonly used stopping criteria. While more complex methods like a "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 pros. Especially considering that a genetic algorithm already has a lot of different control parameters that need tuning.

Performance is always 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 coming testing.

3.2.2 Encoding

When implementing a genetic algorithm, it is necessary to use an encoding that fits to the problem at hand. Here, 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 defining start scenario.

The different encodings presented in section 2.1.1 did not directly fit to the problem. 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. Each gene corresponds to 1 time step (so 1 gene per every 0.5 seconds), and needs to contain all actions of all NPCs at this particular moment. A gene is a list of the size of the number of NPCs. Each object in this lists equals 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.

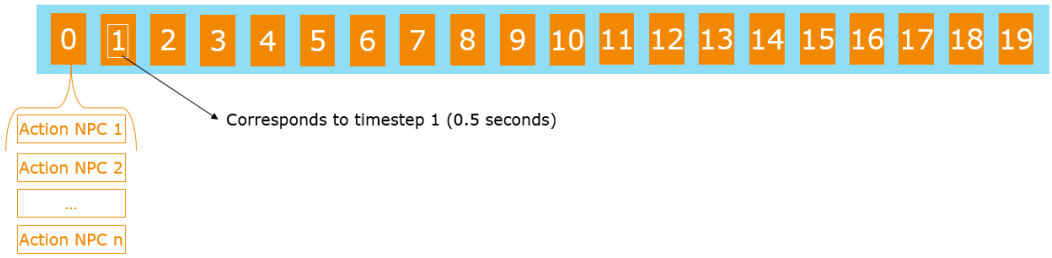

Figure 3.2: Time

Given the previously stated simulation time of 35 seconds, each chromosome has a length of $35 * 2 = 70$ genes. Crossover can now only move all actions of a time step at once, modifications in the timeline between actions of the same time step will only happen due to 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 ID of each NPC also needs to be encoded into the genes position of its chromosome. One might visualize this as unfolding the matrix build by the genes and chromosome in encoding Time into a linear array. Each actors actions will be 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. Figure3.3 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).

provide better explaination

However for this encoding to make sense, the crossover operations OnePoint and TwoPoint had to be modified. For each NPC, these crossover operations

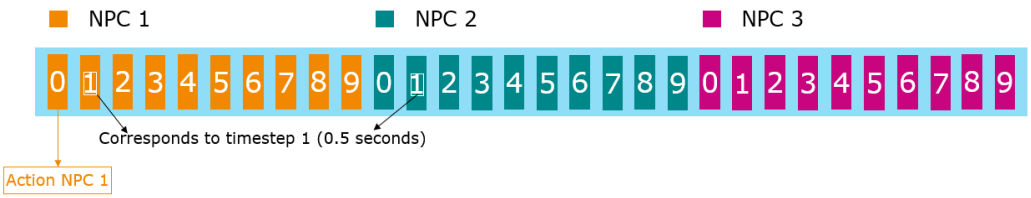

Figure 3.3: Time + NPC

are applied individually. If this was not the case, OnePoint crossover would introduce changes to actions of only one NPC.

Because the different crossover points are independently chosen, the position of the crossing differs between the NPCs. This fact 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 thus explain the type of object in these lists.

Integer The first encoding uses integer, which are translated into actions when the simulation is started. For each action, a range of integers is assigned, the larger the range, the more likely the action is chosen by the GA. These ranges were assigned based on intuition and trial and error. In Appendix these probabilities can be viewed.

ref appendix

Bad explanation, needs rewriting

Actions have parameters split into different ranges, according to which values make sense. For example ModifyTargetVelocity has five different percentages assigned, namely 50, 70, 100, 130, 160. The range of integers assigned to each of these parts is different as well. A percentage setting of 100 for example has the largest integer range assigned.

Actions will be split into 1 or more parts that get an range of integer assigned. Combing these ranges results in a continuos list of integers. The encoding is visualized in 3.4

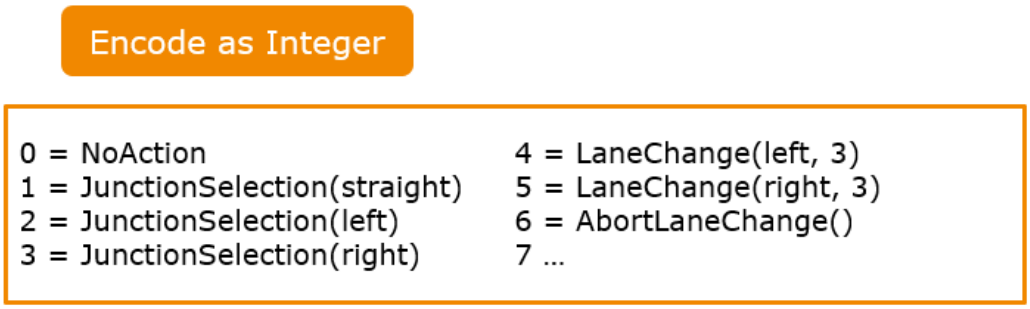

Figure 3.4: Integer

Dictionary The second encoding for genes corresponds exactly to the actual actions used in the simulations, requiring no translation. During generation of the individuals, each action is again selected based on different probabilities which are the same as for the integer encoding. For actions without parameters, the different encoding will make no difference. However actions with parameters no longer need to be split into different discrete settings. Now, 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. Figure 3.5 shows a visualization.

Better explanation

ref appendix

Again, these probability functions with settings were assigned based on intuition as well as trial and error. Detailed information can be seen in Appendix .

ref appendix

Compared to Integer encoding, this method allows for much higher granularity when selecting parameters. If the performance of a genetic algorithm differs between these settings will again be investigated in Chapter 4.

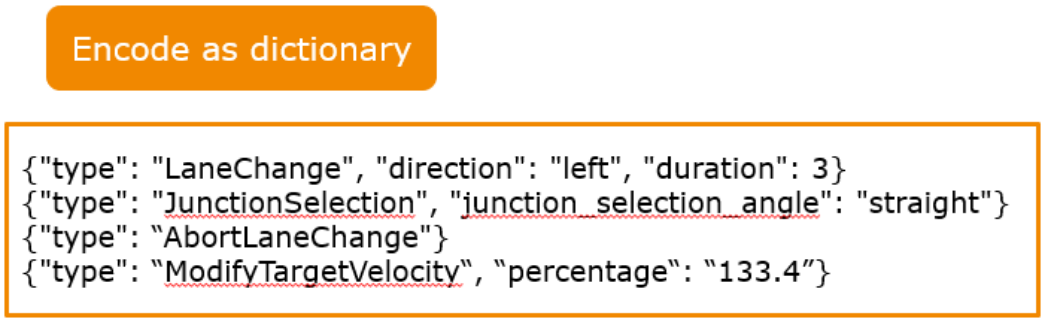

Figure 3.5: Dictionary

3.2.3 Cost Function

This Masters Thesis only utilized internal values from the Traffic Manager for the cost function. Namely if the ego vehicle has to initiate an emergency break. The emergency break values are booleans stored in a list called result["ego_emergency_stop"] which has a length of $100 * simulation\_duration\_seconds$ (because of 100hz).

In case an emergency break lasts longer than 3 seconds, a penalty will be applied. This penalty was introduced as it mitigated results, where the ego vehicle is constantly brake-checked[3] by the leading vehicle.

explain code a bit

The following code snipped shows the cost calculation.

```
1  SEPS_PER_SECOND = 100
2  # allow emergency breaks to last only 3 seconds
3  MAX_DURATION = 3 * STEPS_PER_SECOND
4
5  cost = 0
6  duration_counter = 0
7  for i in range(len(result["ego_emergency_stop"])):
8      if not result["ego_emergency_stop"][i]:
9          # base cost in case of no current emergency break
10         cost = cost + 1
11         duration_counter = 0
```

[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/

```
12      else:
13          if duration_counter > MAX_DURATION:
14              # increase cost if emergency break max
                    duration is exceeded
15              cost = cost + 10
16          duration_counter += 1
17  return cost
```

The resulting cost values however are a not easy to interpret, thus these numbers are modified using equation 3.1.

$$cummulated\_emergency\_break\_duration = (3500 - cost)/100 \tag{3.1}$$

The constant 3500 is chosen to account for the 100hz sampling rate and the duration of 35 seconds. The division by 100 transforms the value into seconds. This value from now on be used instead. A higher *cummulated_emergency_break_time* 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, as the cost function applies a penalty for emergency breaks lasting longer then 3 seconds.

Optimizing for different cost functions, such as improving time to collision TTC would have been interesting as well. However at the time, no working TTC functionality was implemented. Consequently, only the emergency break cost function will be optimized by the genetic algorithm.

explanation

3.3 Behaviour Tree

The EGO vehicle will not be controlled by the genetic algorithm, rather by a behaviour tree. The general idea is to have an EGO vehicle moving in a relatable manner trough the world. It will try to dodge standing or slow moving obstacles. Every 3 seconds the behaviour tree switches between left, straight and right for the junction selection, to produce more interesting

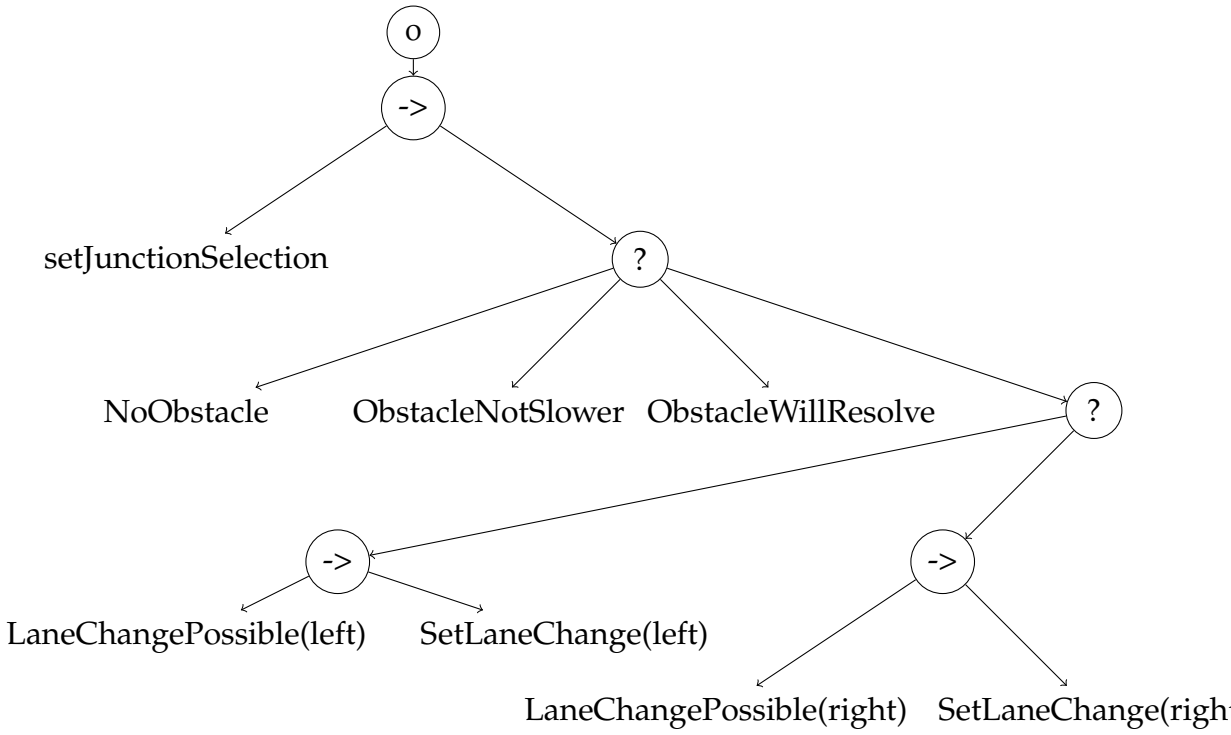

Figure 3.6: Used Behaviour Tree

scenarios, where the EGO vehicle is not always moving straight. Figure 3.3 shows the behaviour tree implemented.

While the behaviour tree has access to the same Action Interface (described in 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 the resulting emergency break values, the behaviour tree however needs access to internal functions during the simulation. Conditions like NoObstacle() or LaneChangePossible() execute internal Traffic Manager functions to get needed information on the world around the EGO vehicle.

4 Hyperparameter Tuning

The performance of a genetic algorithm is significantly influenced by its control parameters. Performant settings on a particular fitness landscape might not be appropriate a different one (K. De Jong, 2007).

explain No Free Lunch Theorem better

According to the "No Free Lunch Theorem", no single algorithm will outperform all other algorithms on a single class of problem (K. De Jong, 2007). K. De Jong, 2007 further states, that "the No Free Lunch results place an obligation on the EA practitioner to understand something about the particular properties of the problems that (s)he is trying to solve that relate to particular choices of EA parameter settings." A human-in-the-loop approach is needed to fine-tune parameter settings to a particular problem.

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 a 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 with 32 GB of RAM. On both systems, Kubuntu 20.04 LTS was the operating system.

As has been defined in Chapter 3, each genetic algorithm will run for 30 generations, additionally 1 simulation will have a duration of 35 seconds. On a single workstation, it takes approximately 3 hours and 50 minutes to

complete one genetic algorithm with a population of 96. This time indication is also influenced by the number of actors used.

Town10 from Carla will be used as the map for all simulations. It has an adequate size, yet is not too big and allows for interesting manoeuvrers.

not sure where to show the start scenarios, they are currently in the appendix

In order to reduce the required number of tests, only one starting scenario was used for all of the tuning of the control parameters. Start scenario 1 can be seen in appendix 6.1.1. 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, especially considering the limed processing resources available. 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 test for the best population size, other hyperparameters first have to be fixated. While reviewing the literature, trends of general settings for genetic algorithms can be found. Grefenstette, 1986 suggests that a range of control parameter will already lead to acceptable performance, yet optimal performance needs tuning. K. De Jong, 2007 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 Suggested Hyperparameter from the Literature

After reviewing various literature regarding control parameter of Genetic Algorithms, no clear consensus emerged. Mills, Filliben, and Haines, 2015

come to a similar conclusion, mentioning the inconsistencies between findings during their literature review, highlighting the conflicting evidence regarding "key GA control parameter".

Table 4.1 aims to provide a short, tough 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.

Table 4.1: Summary of Genetic Algorithm Hyperparameters

Parameter Set	Pop	Cross	Mut	Sel	Elite
K. A. De Jong, 1975	50	0.6	0.001	?	1
Mills, Filliben, and Haines, 2015	200	?	Adaptive	SUS	8%
Grefenstette, 1986	30	0.95	0.01	?	1
Grefenstette, 1986	80	0.45	0.01	?	1
Almanee et al., 2021	50	0.8	0.2	?	?
Srinivas and Patnaik, 1994	30-100	0.9	0.01	?	?
Fazal et al., 2005	50	0.5	?	Tourn	5
Dao, Abhary, and Marian, 2016	200	0.7	?	Roul	1
Naruka et al., 2019	200	0.4	?	Roul	10
Jinghui Zhong et al., 2005	50-250	0.1-0.9	0.05-0.25	?	?

Generally speaking, a low mutation rate is recommended. For example Grefenstette, 1986 suggest poor performance using a rate over 0.05. Using a low mutation rate is also suggested by Whitley, 1994 and Jinghui Zhong et al., 2005. On the other hand, Boyabatli and Sabuncuoglu, 2004 found higher mutation rates for their application to be more suitable. Srinivas and Patnaik, 1994 differentiates between higher and lower population numbers, claiming that a smaller population needs higher mutation rates in order to maintain a sufficient diversity.

analyze other control parameters a bit more

4.2.2 Comparison of Population Size

This leads to a difficult decision in choosing the right parameters. Based on the extensive research, population sizes of 32, 48, 64 and 96 will be compared. Crossover rates are set to 0.8 and 0.6. 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 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 4.2.2.

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

Figure 4.1: results from testing population size - mean of 5 repetitions

In figure 4.2, the results per population are plotted. The line is corresponds to the mean, while the bars show the spread (min to max) of all 5 repetitions.

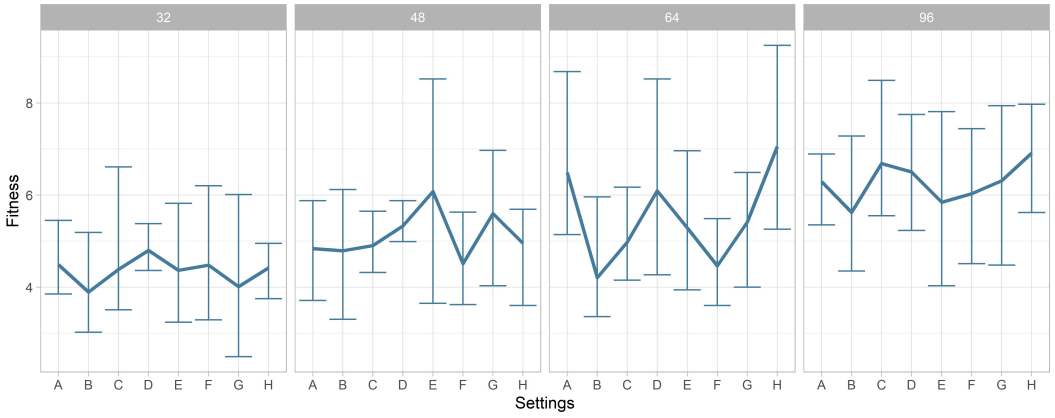

Figure 4.2: mean and error bars per population size

say that 64 had the best results, however there was too much variance

A higher spread in the results can be seen when looking at small 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 to be more important to keep the variation low.

explain decision a bit more

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 tune the control parameter of the genetic algorithm, various different strategies can be used. Using automated approaches like "grid search", "bayesian optimization, "simmulated annealing or "hyperband" might lead to good results with minimal effort (tuning hyperparameter of these search algorithms is still needed), however they each require a high number of runs (K. De Jong, 2007). K. De Jong, 2007 even suggest using a second, higher level Genetic Algorithm for the optimization process.

find more references

Due to performance considerations, many optimization methods did not fit the requirements. Executing one run for 30 generations currently 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 each experiment has factors, of which each consists of at least two settings, with the actual number of settings being called levels (Yang and El-Haik, 2009). 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." (Yang and El-Haik, 2009)*

Afterwards, main effects and interactions can be calculated to find the best settings per factor. Using ANOVA (Analysis of Variance) it is possible to identify the significance of each factor and interaction, which enables their ranking. 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 manually selected factor levels. Looking at the proposed factors in table 4.3.1, 1024 runs[1] are required, which 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, 2009 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.

Short explaination

Fractional factorial experiments require less runs ...

> *"Techniques such as fractional (or partial) factorial experiments are used to simplify the experiment. Fractional factorial experiments investigate only a fraction of all possible combinations. This approach saves considerable time and money but requires rigorous mathematical treatment, both in the design of the experiment and in the analysis of the results." (Roy, 1990)*

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 and reducing variability (Roy, 1990). This master thesis will not discuss all of Taguchi's proposed considerations,

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

for more detail Roy, 1990 as well as Yang and El-Haik, 2009 is highly recommended.

Taguchi's proposed experimental design is a fractional factorial design, which requires significantly less runs. While different fractional factorial designs are available, Taguchi was chosen, because he provides a simple and easy to follow procedure which 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." (Yang and El-Haik, 2009)*

Using Taguchi design for finding optimal hyperparameter of a genetic algorithm has been successfully performed by Dao, Abhary, and Marian, 2016 as well as Naruka et al., 2019.

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 the factors (Hamzaçebi, 2021). Taguchi's orthogonal arrays "represent the smallest fractional factorials" (Roy, 1990). Only a fraction of combinations needs to be tested, drastically reducing computational needs.

The researcher has the responsibility to select an array based on the individual needs (Hamzaçebi, 2021). 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" Roy, 1990.

TODO: rewrite from here

" OFF and OLH experiment designs sample a full factorial design space in a balanced and orthogonal fashion. Balance ensures good effect estimates by reducing an estimator's bias and variability. Orthogonality ensures good estimates of two-term interactions. Balance is achieved by ensuring that each level of every factor occurs an equal number of times in the selected

sample. Orthogonality is achieved by ensuring that each pair of levels occurs an equal number of times across all experiment parameters in the selected sample. OFF and OLH designs exhibit good space-spanning properties, which aid screening, sensitivity, and comparative analyses. On the other hand, highly fractionated OFF or OLH designs can have poor space-filling properties, which are necessary for optimization analyses.

"Mills, Filliben, and Haines, 2015

As has been stated, probably the biggest drawback of using Taguchi orthogonal arrays is on the one hand to increased manual labour and on the other hand the fact, that higher order interactions ignored.

Roy, 1990 explains why this might not be a big problem: “Generally speaking, OA experiments work well when there is minimal interaction among factors; that is, the factor influences on the measured quality objectives are independent of each other and are linear. In other words, when the outcome is directly proportional to the linear combination of individual factor main effects, OA design identifies the optimum condition and estimates performance at this condition accurately. If, however, the factors interact with each other and influence the outcome, there is still a good chance that the optimum condition will be identified accurately, but the estimate of performance at the optimum can be significantly off. The degree of inaccuracy in performance estimates will depend on the degree of complexity of interactions among all the factors.”

This is complimented by Yang and El-Haik, 2009, who states, that: “During many years of applications of factorial design, people have found that higher-order interaction effects (i.e., interaction effects involving three or more factors) are very seldom significant. In most experimental case studies, only some main effects and two-factor interactions are significant.”

However, K. De Jong, 2007 claim, that "tuning EA parameters can itself be a challenging task since EA parameters interact in highly non-linear ways." It remains to be seen in it is possible to use this method for optimizing parameters of a genetic algorithm.

to here

Selection of orthogonal array When choosing a suitable Taguchi orthogonal array, various factors have to be taken into account. According to Yang and El-Haik, 2009, a three step procedure needs to be followed:

1. Calculate the total degree of freedom (DOF).
2. Base on the following two rules a standard orthogonal array should be selected:
    a) 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 level 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, 7 factors (3 factors of level 4 and 4 factors of level 2) have been selected. Which factors and levels to choose was done based on experience gained on section 4.2. When selecting levels, it is important to have them "as far away from either side of the current working condition as possible"(Roy, 1990). In table 4.3.1, every factor with the corresponding levels has been listed.

It is important to state, that Taguchi allows to only test for possible (pre determined) two-level interactions, evaluating higher level interactions is not possible (Yang and El-Haik, 2009). Analysing interactions comes at a cost of degrees of freedom. An interaction between ChromosomeType and GeneType might be of interest and will thus be chosen.

Factors	Code	Level 1	Level 2	Level 3	Level 4
CrossoverType	A	one point	two point	uniform 0.1	uniform 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	-	-

Figure 4.3: List of control parameters (factors) with correspond settings (levels)

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, 2009:

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

This leads to the following calculation for the needed 3 Factors of Level 4 and 4 Factors of Level 2 as well as the interaction between 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 4.3.2.

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 (NazanDanacıoğlu, 2005). Both illustrate the interaction relationships in the orthogonal array (Yang and

NO.	$L_{16}(2^{15})$														
	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

Figure 4.4: $L_{16}(2^{15})$ Taguchi orthogonal array taken from Roy, 1990

El-Haik, 2009). The linear graph is more straight forward and will be the chosen approach. While there are multiple linear graphs for the L_{16} array, 4.3.2 describes the graph which best fits the requirements from table 4.3.1. If no suitable graph is found, they can be modified using rules described by NazanDanacıoğlu, 2005.

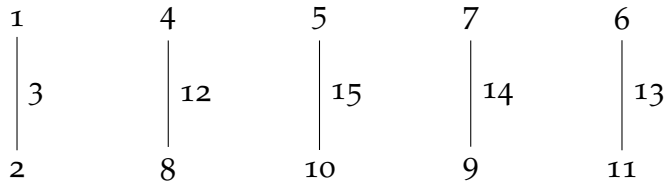

Figure 4.5: Linear Graph of $L_{16}(2^{15})$ taken from Yang and El-Haik, 2009

bad explanation

In a Taguchi linear graph, the nodes as well as the connections both represent

columns in the orthogonal array. An interaction between to columns that are represented as nodes "comes out to" the connecting line column Taguchi et al., 2005. This is useful for both analysing interactions between columns as well as combining (merging) interacting columns in case a higher factor is needed.

The column describing their connection will subsequently contain the interaction (Taguchi et al., 2005).

Column Merging A, B and C are both 4 level factors. The currently selected orthogonal array only fits 2 level factors. Column merging, it is possible to extend columns to accommodate higher order levels.

As calculated in 4.1, a four-level column requires three degrees of freedom, thus three two-level columns need to be merged. Column merging needs the to be merged columns to be part of an interaction group (Yang and El-Haik, 2009). The available interaction groups are visualized by the linear graph in figure 4.3.2.

So, 3 interaction 2-level columns need to first be selected. One column is discarded, the remain two columns are merged using the rules in tabular 4.6.

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

Figure 4.6: Rules taken from Roy, 1990

The four-level factor can then be assigned to this newly generated column. Because three four-level factors are needed for the current experiment, nine two-level columns need to be merged in total.

Assigning Interactions Interactions between two-level factors can be assigned using the linear graph as well. Here, select two connecting nodes.

An interaction between ChromosomeType and GeneType seems possible, thus D and E will be assigned to connected nodes in the linear graph. The resulting graph can be seen in 4.3.2.

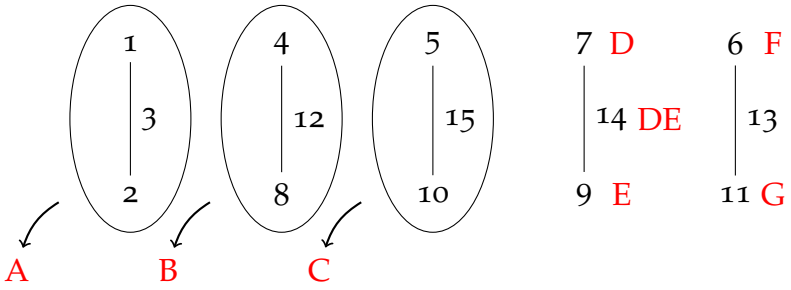

Figure 4.7: Modified Linear Graph to fit our needs

Combining columns 1 2 3 to A, 4 8 12 to B and 5 10 15 to C using rules defined by table 4.6 is done in 4.8.

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, 11 to G and 13 to FG results in the final table 4.9. This combination table will subsequently be used as settings for the simulation runs.

4.3.3 Result Analysis

Table 4.9 can now be used for running all the needed testcases (the interaction columns can be ignored until the evaluation). Transcoding all factors and levels to get the corresponding setting can be done using the table from 4.3.1. Every setting will be repeated 8 times to reduce randomness and gain information about variance. Running the genetic algorithm using these 16 different settings each repeated 8 times took 10 days on the two in Section 4.1 described workstations. The results are found in the appendix at 6.1.1.

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

Figure 4.8: Building 4 Level columns from 2 Level columns

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 (Yang and El-Haik, 2009). Calculating ANOVA is done using the programming language R[2] and the result can be seen in table 4.2.

Sum of squares of the model tell 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 models prediction versus what was actually observed (i.e the variance that can not be explained by the model) (A. P. Field, Miles, and Z. Field, 2012).

The F ratio (or value) measures the ratio of variance explained by the factor

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

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

Figure 4.9: Final version of used Taguchi orthogonal array

and the variation explained by the error term (A. P. Field, Miles, and Z. Field, 2012). 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 (A. P. Field, Miles, and Z. Field, 2012).

The higher number of DOF can be explained with our number of repetitions and can be calculated with the equation 4.2 (taken from Roy, 1990).

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

	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 4.2: ANOVA results

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 variability in the outcome is explained by the predictors (A. P. Field, Miles, and Z. Field, 2012). Having only 41.6% does not seem optimal. A. P. Field, Miles, and Z. Field, 2012 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 in this scenario. The high error might possibly be explained by the huge search space in the scenario. Increasing the population and number of generations might lead to improvements, however at the cost of computational time. If 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 main effects can be seen. The highest F value has the main effect F (TournamentSize). The effect is significant with $F(1, 112) = 15.64$, $p < 0.001$. Next, the effect 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 that suggests significant main effects for the factors B (CrossoverRate) and E (GeneType).

Especially the fact, that the CrossoverRate does not show significant effects is surprising. The low influence of GeneType however might be explained by the fact it does not have a huge impact on the action selected apart from more granularity of the parameters when Dictionary encoding is used.

To look at the percentage contribution of each factor the formulas 4.3 and 4.4 (gathered by Yang and El-Haik, 2009) are used.

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

Example calculation for factor A.

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

The percentage contribution of all factors is plotted in 4.3.3 and again shows the high error of the model.

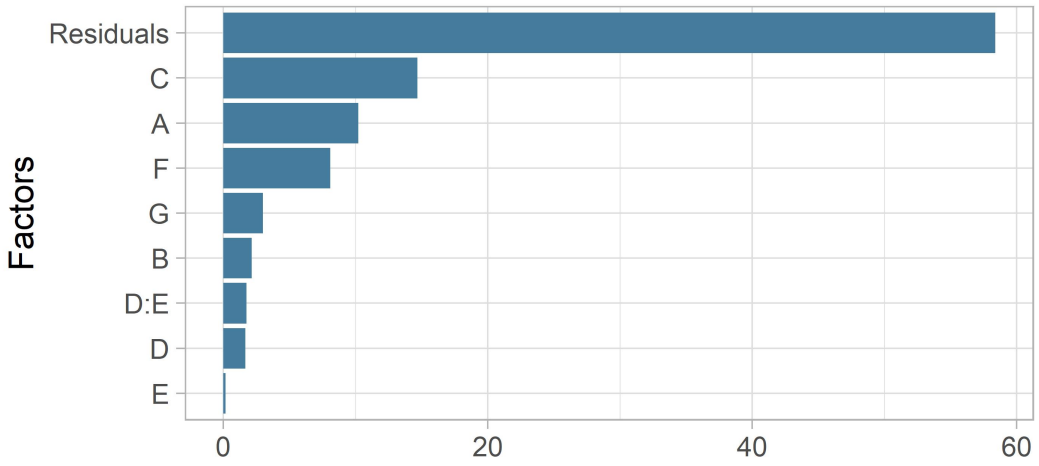

Figure 4.10: Percentage Contribution

Main-effects and interaction chart

Identifying the optimal conditions needs analysis of the main effects per factor. They allow to predict the factors, that lead to the best result (Roy, 1990).

> *"The main-effects chart is a plot of average responses at different levels of a factor versus the factor levels." (Yang and El-Haik, 2009)*

Show calculation example for D

For every factor, sum up the mean of all results per level, then divide by the number of runs per level.

The resulting main-effect charts can be seen in figure 4.3.3.

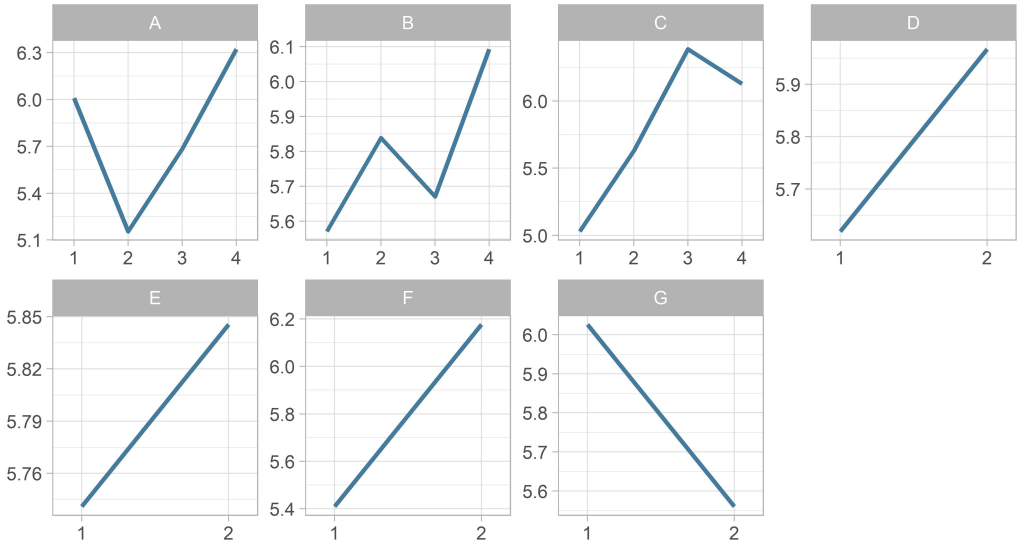

Figure 4.11: Main Effects

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 case of this experiment, the level with the highest value).

If interactions exist, they might have an influence on the best settings and need further investigation (Yang and El-Haik, 2009). To investigate previously defined interaction an interaction graph can be used. The calculation is similar to calculating main effects and shown in figure 4.3.3.

Show calculation example for DE

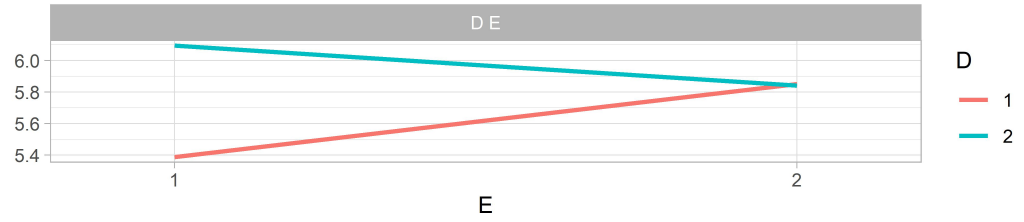

Figure 4.12: Test of interactions

The crossing of lines indicates, that an interaction between the two factors might exist (A. P. Field, Miles, and Z. Field, 2012). The more parallel the lines are, the less likely an interaction. Magnitude of the angle between the lines corresponds to the degree of interaction presence, according to (Roy, 1990).

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: A4, B4, C3, D2, E2, F2, G1.

Looking at the ANOVA table, the interaction D:E however seams to have significance, especially compared to E. According to (A. P. Field, Miles, and Z. Field, 2012), it makes little sense of analyse or further interpret main effects if the interaction effect is significant (which it almost is with $p < 0.1$), thus this interaction will be integrated.

The test of interaction in figure 4.3.3 suggest D2 and E1 as the best combination. This is optimal, as D2 is also suggested by the main effects. While E1 does not correspond to the suggested main effect, the low F value suggests low significance for E anyway. Concluding this line of thought, the combination A4, B4, C3, D2, E1, F2, G1 seems to be optimal.

Optimum performance calculation To have a understanding on the predicted results of an algorithm with these settings, optimal performance calculation can be used. Equation 4.5 only uses the optimal main effects while equation 4.6 has the interaction D:E applied. The calculation formulas are provided by Roy, 1990.

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

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

Using the interaction D:E, the performance estimation improves from 8.06 to 8.13. Finally the optimized settings are as follows: CrossoverType: Uniform 0.5, CrossoverPropability: 0.9, MutationPropability: 0.3, ChromosomeType: Time+NPC, GeneType: integer encoding, TournamentSize: 4 and IndividualMutationPropability: 0.1.

Talk about the settings

signal-to-noise (S/N) As previously discussed, the ANOVA model has a high error, which suggests high randomness. Taguchi recommends using signal-to-noise (S/N) ratio to reduce the variability, as using only the mean of the results does not take the variation into account (Roy, 1990). The greater the signal-to-noise ratio, the smaller the variance. Roy, 1990 further states, that the "use of the S/N ratio offers an objective way to look at the two characteristics (consistency and average value) together."

According to Roy, 1990, S/N is calculated in two steps, first the mean square deviation (MSD) is needed. Depending on the quality characteristic, a different equation has to be chosen. In this case, a higher result is better, thus equation 4.7 is used for each repetition, with y being a result and n the number of repetitions.

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

S/N is further calculated using equation 4.8.

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

Generating main effects and ANOVA is now done using S/N instead. Now, only 15 DOF for the ANOVA analysis are available, as repetitions per run get merged into 1 result (in contrast to 4.2). Which can be seen in equation 4.9.

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

The resulting ANOVA table which can be seen in 4.3.

	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 4.3: S/N ANOVA results

When looking at the p values of this table, it is obvious that no factor can discard the null-hypothesis, which states that a factor has no significant

effect. Considering this, it was deemed to be not necessary to perform further investigations. Using the often recommend Signal to Noise ratio does not seem to suitable for this experiment, thus the settings recommended in Paragraph 4.3.3 will not be adjusted.

Elite Although the optimal hyperparameter setting will be discussed in chapter 5, a problem was obvious when analysing results of a GA using the settings from 4.3.3. Setbacks in the optimal cost between two generations happen frequently, likely due to the high crossover and mutation rates. In order to mitigate this problem, it was decided to implement elite selection with a size of 2. 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 the two best individuals can still be selected by tournament selection for modification, its just that a copy of them is saved. Figure 4.3.3 compares both no elite selection with an elite selection of 2. For a few selected repetitions, the best individual cost per generation is plotted.

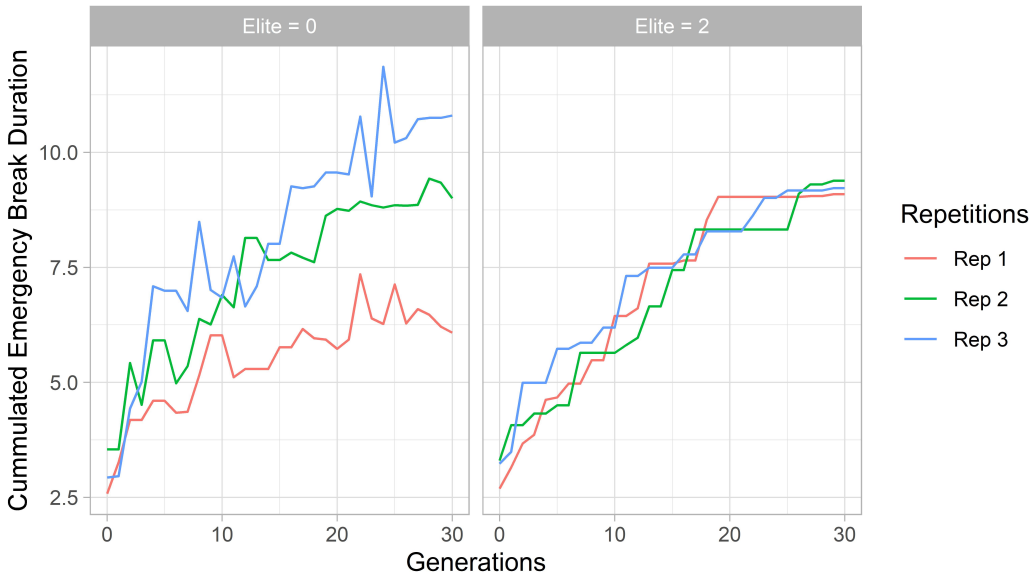

Figure 4.13: Elite Comparison

For analysing the statistical significance in the differences in mean between the a genetic algorithm using elite selection of 2 vs no elite selection, a t-test can be used. In order to ignore possible violation of the assumption of homogeneity, a robust welch t-test is applied which adjusts the DOF accordingly (A. P. Field, Miles, and Z. Field, 2012).

On average, using elite improved the performance (M = 8.52, SE = *), compared to using no elite (M = 7.92, SE = *). 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 comparison of both settings each repeated 10 times can be seen in figure 4.3.3.

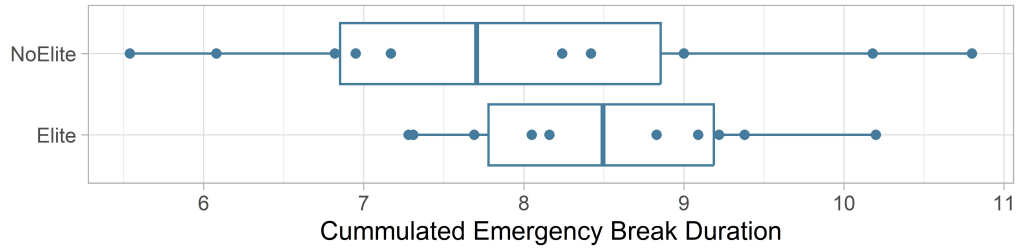

Figure 4.14: Comparison Elite vs No Elite

Due to the existence of the mentioned small effect, it is concluded that the slightly modified version of the settings from 4.3.3 will be used in Chapter 5, where elite selection is set to 2.

Effect Size According to A. P. Field, Miles, and Z. Field, 2012 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 use the widely used suggestions made by Cohen (1988, 1992) on defining between a large or small effect:

Find paper

- 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 will be further 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 4.3.3 with the best settings found in 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 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 to be 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 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 to be simulated 31 times per run (30 generations + initialization), which leads to $96 * 31 * 10 = 29,760$ simulations.
- **Random** Search
  - For randomly choosing actions, the same dict probabilities from appendix are used.
  - 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.

ref appendix

The evaluation is done over 4 different start scenarios which can be found in the appendix at 6.1.1

5.1 Start Scenario 1

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

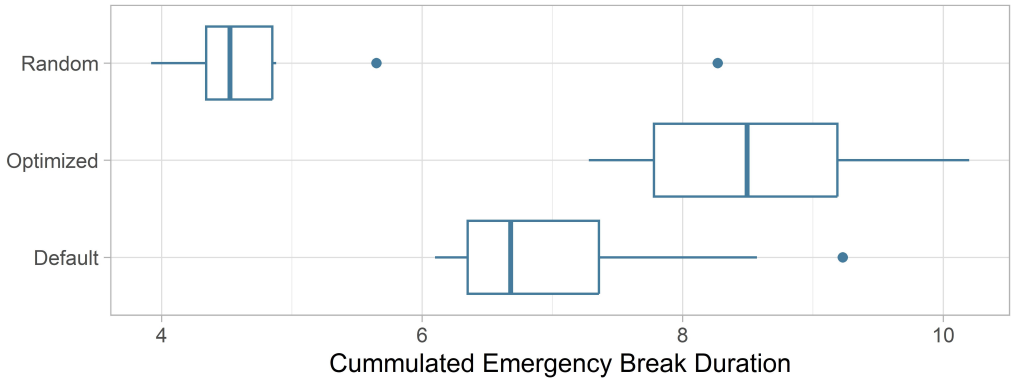

Figure 5.1: Start Scenario 1: Default GA vs Optimized GA vs Random Search

Looking at the graph, the Optimized GA clearly outperformed the Default GA as well as Random Search.

Welchs t-test shows that on average, greater fitness is achieved by using Optimized GA (M = 8.52, SE = ?) than from using Default GA (M = 7.09, SE = ?). This difference was significant $t(17.87) = 3.15$, $p < .01$. It did represent a large effect r = 0.60. Compared to Random (M = 4.943, SE = ?), the Optimized GA has even higher dominance, with a significant difference $t(16.9) = 7.12$, $p < .001$ 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 5.1 plots the mean over 10 repetitions, the outline show the first and third quantiles.

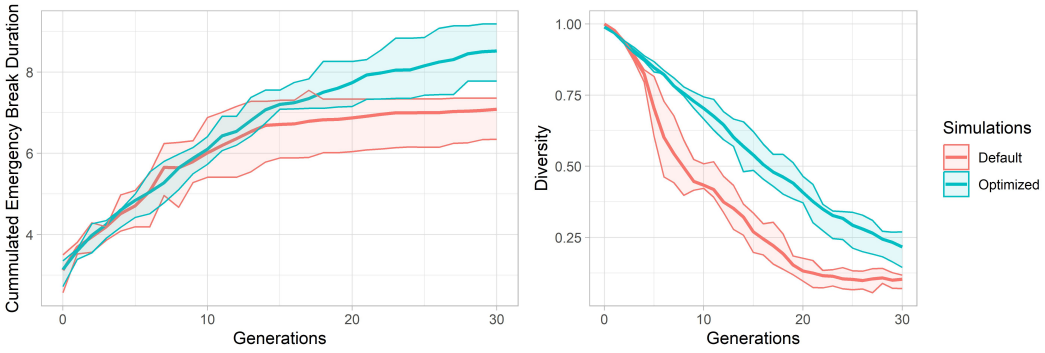

Figure 5.2: Start Scenario 1: Comparison of GAs

After generation 10, the rate of improved fitness of the Default GA drops 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. Its high crossover and mutation rates seem 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

9 Vehicles and 5 pedestrians. same number as start scenario 1.

simulations have not yet finished

5.3 Start Scenario 3

Start scenario 3 is described in more detail in Appendix 6.1.1. 5 vehicles with 3 pedestrians are initialized, resulting in a simulation with only a small

number of NPCs.

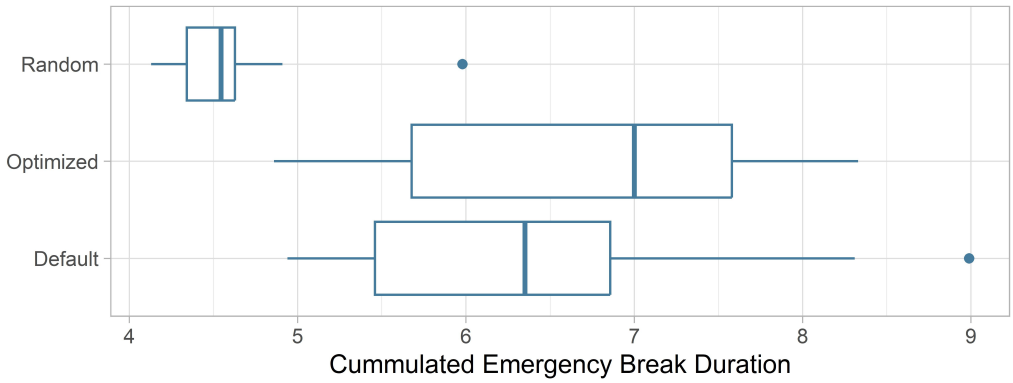

Figure 5.3: Start Scenario 3: Default GA vs Optimized GA vs Random Search

Looking at the graph, the Optimized GA again outperforms the Random Search, however there seams 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 = ?) than from using Default GA (M = 6.49, SE = ?). This difference was however not significant $t(17.83) = 0.29$, $p > .05$ and an effect size of r = 0.07. Verifying the better performance of the Optimized GA compared to Random Search (M = 4.619, SE = ?) shows a significat difference $t(12.4) = 4.9$, $p < .001$ 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 again shown in figure 5.3. The mean over 10 repetitions is plotted, the outline show the first and third quantiles.

The rate of improved fitness of the Default GA now drops very similar to the Optimized GA. The early decline in the diversity is also not as pronounced as in the previous 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 the smaller search space due to the smaller number of NPCs. Here,

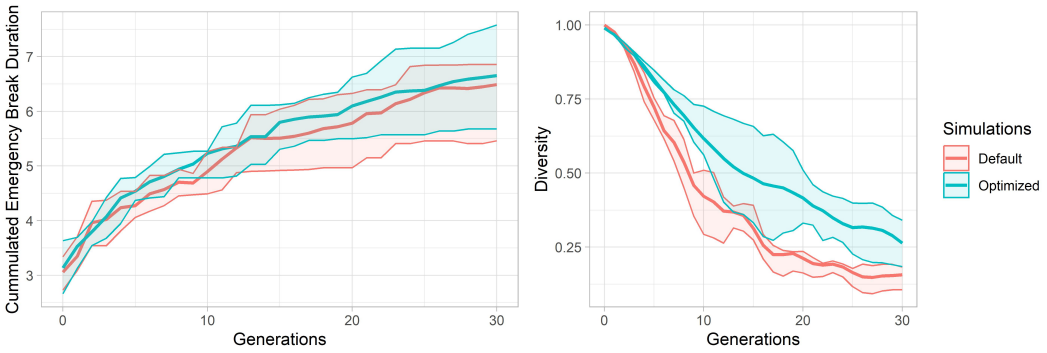

Figure 5.4: Start Scenario 3: Comparison of GAs

high mutation and crossover rates might not have the previous pronounced advantage.

5.4 Start Scenario 4

Start scenario 4 is can be seen in Appendix at 6.1.1. 18 vehicles with 10 pedestrians are initialized, resulting in the simulation with the most NPCs.

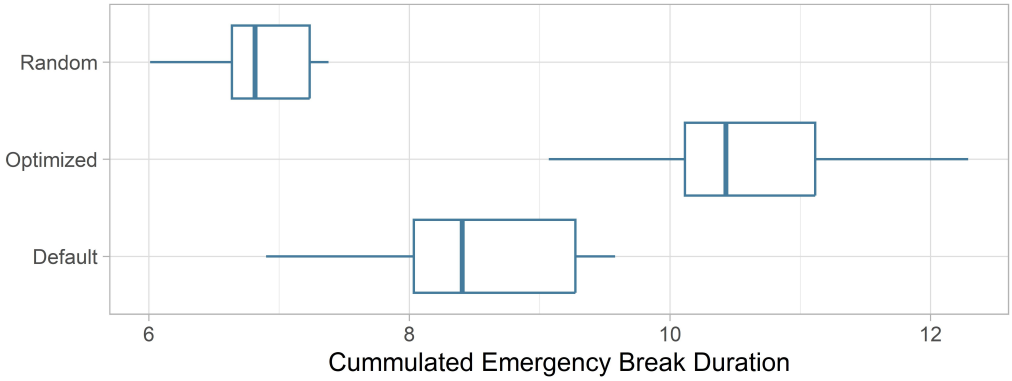

Figure 5.5: Start Scenario 4: Default GA vs Optimized GA vs Random Search

Figure 5.4 shows the Optimized GA clearly outperforming the Default GA as well as Random Search. The Optimized GA (M = 10.60, SE = ?) has

significantly greater fitness then the Default GA (M = 8.46, SE = ?) with $t(17.98) = 5.30$, $p < .001$ and a large effect r = 0.78. Compared to Random Search (M = 6.86, SE = ?), the greater fitness of the Optimized GAs is significant with $t(11.71) = 12.7$, $p < .001$ and a large effect r = 0.96.

Both genetic algorithms performance over the generations next to their diversity chart will be compared in figure 5.4.

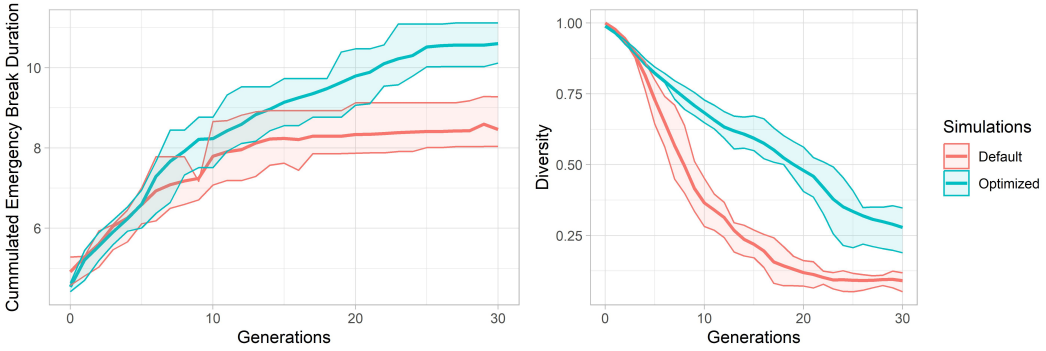

Figure 5.6: Start Scenario 4: Comparison of GAs

Already at generation 5, the average rate of improved fitness of the Default GA drops compared to the Optimized GA. This is accompanied by an early sharp decline in the diversity. The optimized GA again shows 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, that the Optimized GA excels at scenarios with lots of NPCs, where a vast search space is given.

6 Conclusion

write better and longer conclusion

The in this Master's Thesis proposed settings for a genetic algorithm show good performance on the given search problem.

The results of Chapter 5 prove, that the optimized settings GA shows significant improvements in most start scenarios compared to general settings from the literature as well as compared to random search. Only in the case of a small number of NPCs, the performance seems to be on par with the a genetic algorithm utilizing settings from the literature.

Compared to random testing, research question 1 definitely holds true. The optimized genetic algorithm will drastically improve performance when using the given cost function.

The second research question can be answered as well with yes. Taguchi orthogonal testing only needs a minimal amount of different experiments while leading to impressive performance improvements compared to using control parameters suggested by the literature.

6.1 Future Work

6.1.1 Oracles

add Responsibility-Sensitive Safety (RSS)

Better introduction

Using oracle functions to select interesting scenarios during the runtime of a genetic algorithm was already done by Almanee et al., 2021. Different oracle functions test for different thresholds in the simulation. Examples are 'overSpeedLimit', 'criticalLaneChange', 'criticalComfortLevel'. 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 remains unaffected.

Oracles have the potential to extract a multiple different scenarios from only one genetic algorithm run. The automatic categorization drastically improves the usefulness of this testing approach. Otherwise each result from the genetic algorithm needs to be evaluated manually for finding errors in the ADAS/AD function.

Appendix

Appendix A.

Better images

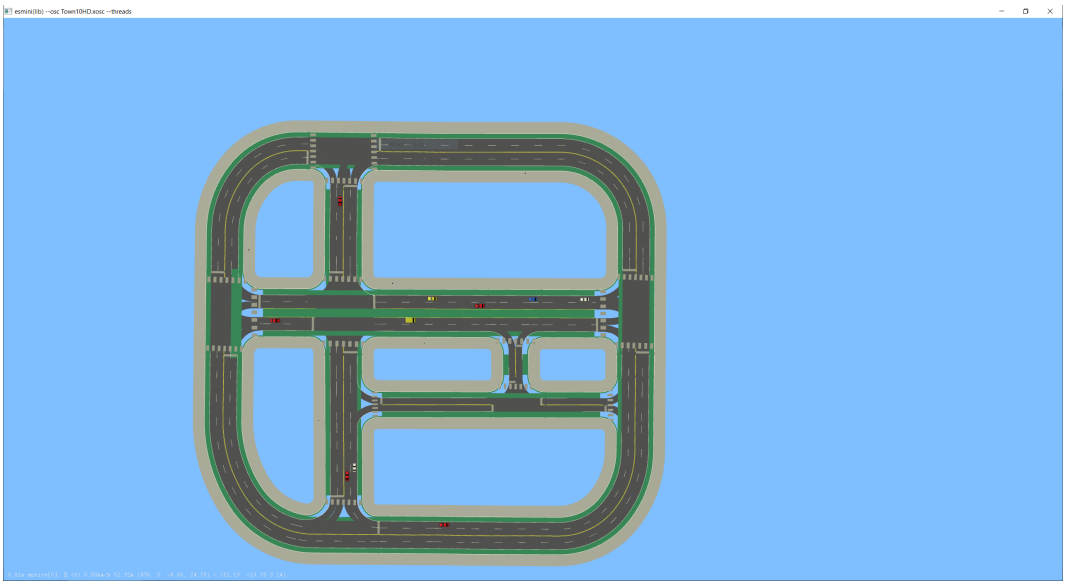

Figure 1: Start scenario 1

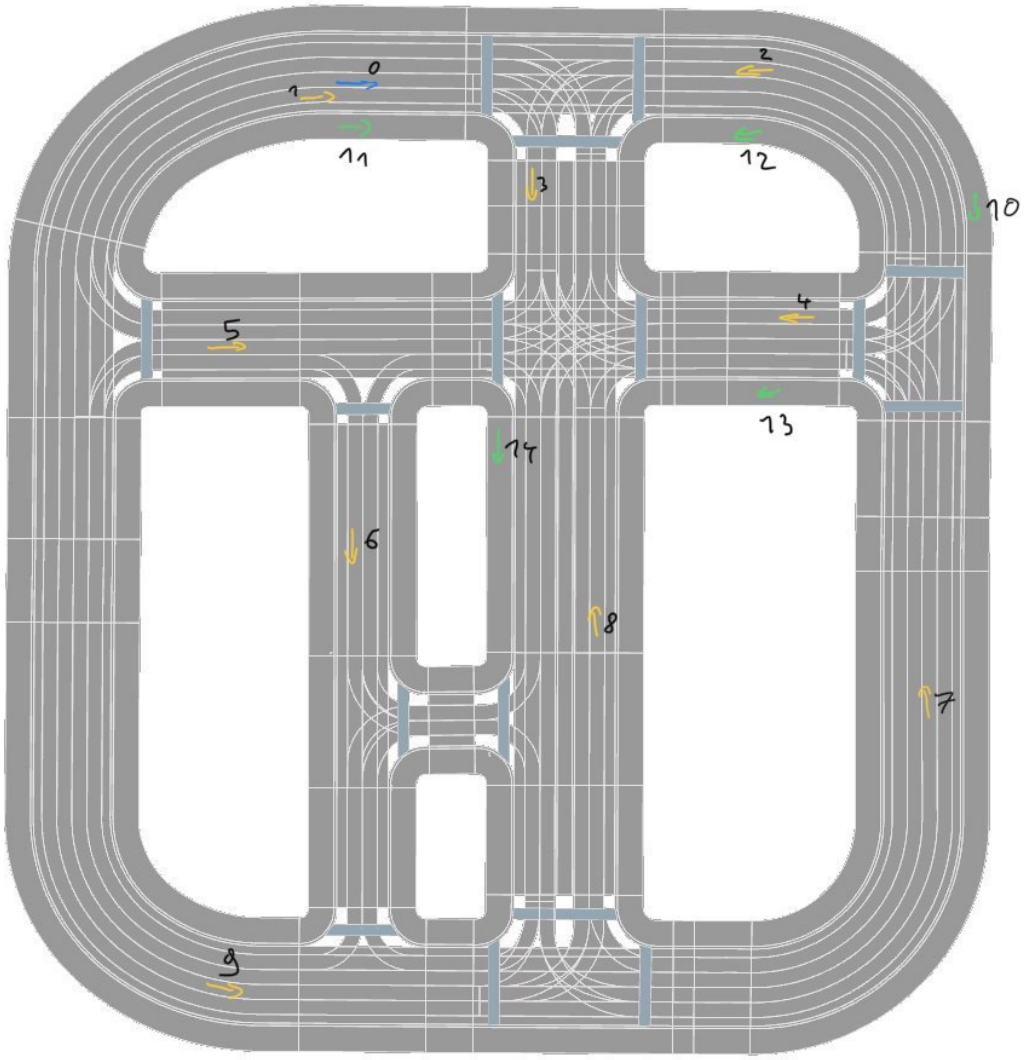

Figure 2: Start scenario 2

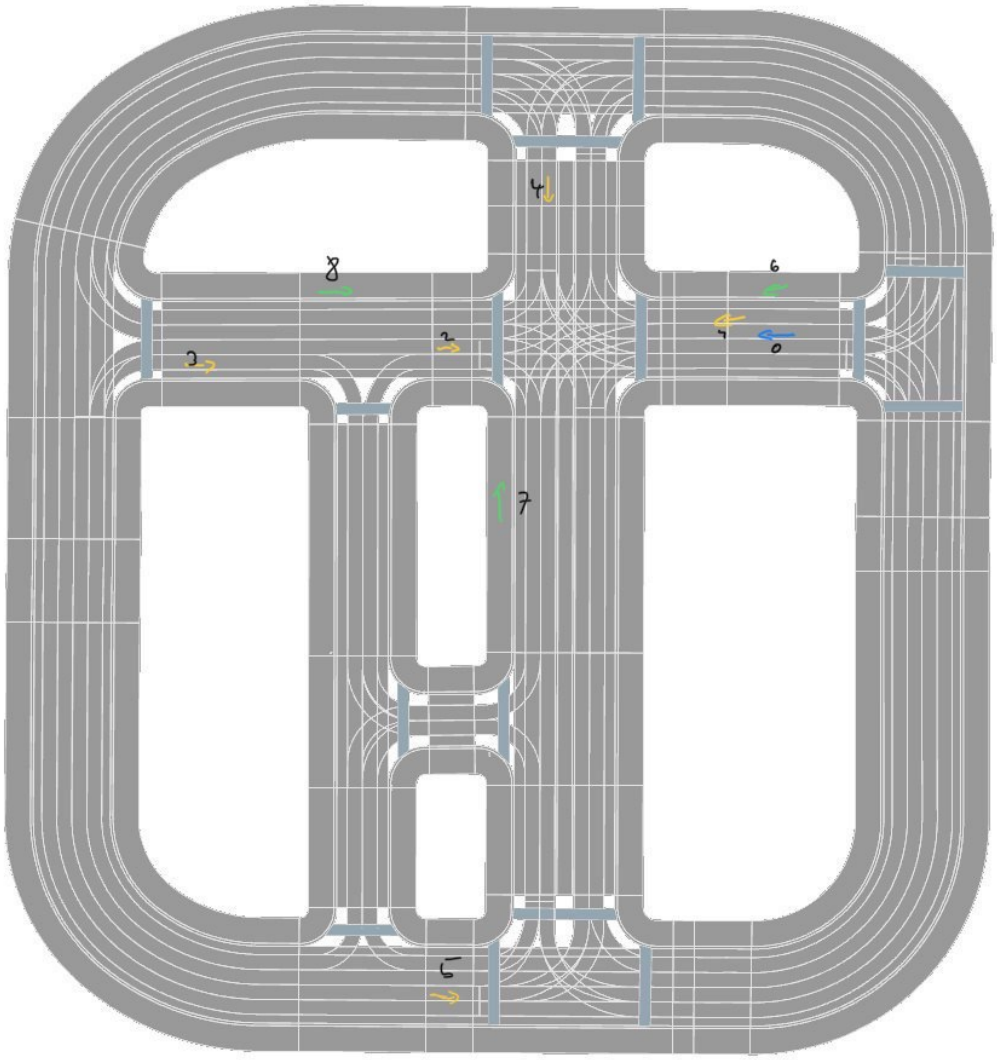

Figure 3: Start scenario 3

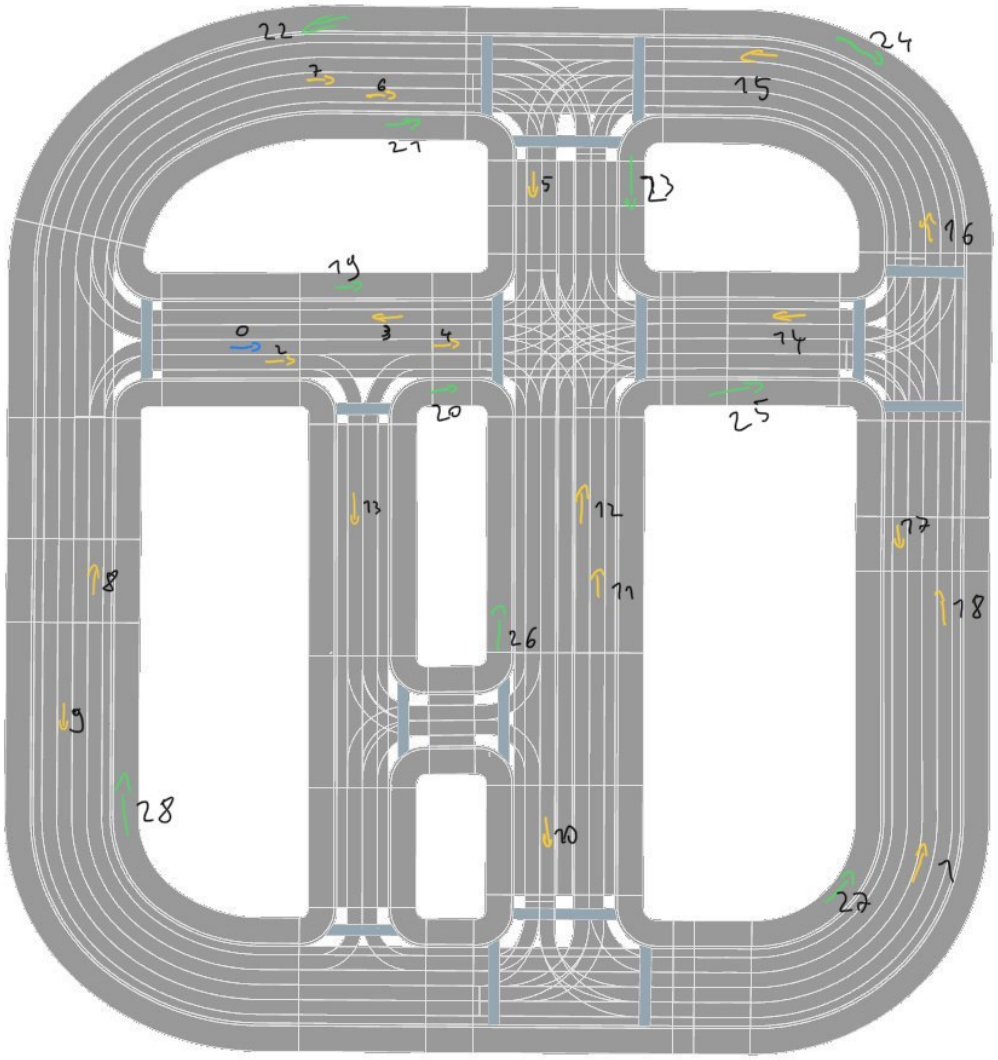

Figure 4: Start scenario 4

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

Figure 1: Taguchi experiment results

insert all results from evaluation

Appendix C.

Insert information of Gene action probabitlities

Bibliography

Almanee, Sumaya et al. (2021). "scenoRITA: Generating Less-Redundant, Safety-Critical and Motion Sickness-Inducing Scenarios for Autonomous Vehicles." In: 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. 23, 51).

Boyabatli, Onur and Ihsan Sabuncuoglu (2004). "Parameter selection in genetic algorithms." In: *Journal of Systemics, Cybernetics and Informatics* 4.2. Publisher: International Institute of Informatics and Cybernetics, p. 78 (cit. on p. 23).

Dao, Son, Kazem Abhary, and Romeo Marian (Feb. 2016). "Maximising Performance of Genetic Algorithm Solver in Matlab." In: *Engineering Letters* 24 (cit. on pp. 23, 27).

De Jong, Kenneth (2007). "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, 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. 5–7, 10, 21, 22, 25, 28).

De Jong, Kenneth Alan (1975). *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan (cit. on pp. 7, 23).

Fazal, M.A. et al. (Mar. 2005). "Estimating groundwater recharge using the SMAR conceptual model calibrated by genetic algorithm." In: *Journal of Hydrology* 303.1, 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. 23).

Field, Andy P., Jeremy Miles, and Zoë Field (2012). *Discovering statistics using R*. OCLC: ocn760970657. London ; Thousand Oaks, Calif: Sage. 957 pp. ISBN: 978-1-4462-0046-9 978-1-4462-0045-2 (cit. on pp. 34–36, 39, 43).

Grefenstette, John (Jan. 1986). "Optimization of Control Parameters for Genetic Algorithms." In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1, 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. 3, 4, 6–9, 22, 23).

Hamzaçebi, Coşkun (Mar. 24, 2021). "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. 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. 27).

Hussain, Abid and Yousaf Shad Muhammad (Apr. 2020). "Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator." In: *Complex & Intelligent Systems* 6.1, 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. 4, 5, 7, 10).

Jinghui Zhong et al. (2005). "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, 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. 7, 23).

Katoch, Sourabh, Sumit Singh Chauhan, and Vijay Kumar (Feb. 1, 2021). "A review on genetic algorithm: past, present, and future." In: *Multimedia Tools and Applications* 80.5, 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. 4–9).

Klampfl, Lorenz, Florian Klück, and Franz Wotawa (2023). "Using genetic algorithms for automating automated lane-keeping system testing." In: *Journal of Software: Evolution and Process* n/a (n/a). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2520, 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 p. 9).

Majumdar, Abhishek and Debashis Ghosh (Oct. 15, 2015). "Genetic Algorithm Parameter Optimization using Taguchi Robust Design for Multiresponse Optimization of Experimental and Historical Data." In: *International Journal of Computer Applications* 127.5, 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. 3–5, 7).

Marsili Libelli, S. and P. Alba (July 2000). "Adaptive mutation in genetic algorithms." In: *Soft Computing* 4.2, 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 pp. 4, 10).

Mills, K. L., J. J. Filliben, and A. L. Haines (June 2015). "Determining Relative Importance and Effective Settings for Genetic Algorithm Control Parameters." In: *Evolutionary Computation* 23.2, 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. 3, 4, 6, 22, 23, 28).

Naruka, Bhagyashri et al. (July 30, 2019). "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, 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 pp. 23, 27).

NazanDanacıoğlu, F. ZehraMuluk (2005). "TAGUCHI TECHNIQUES FOR 2^k FRACTIONAL FACTORIAL EXPERIMENTS." In: *Journal* 34.1, pp. 83–93. ISSN: 2651-477X-2651-477X (cit. on pp. 30, 31).

Roy, Ranjit K. (1990). *A primer on the Taguchi method*. Competitive manufacturing series. New York: Van Nostrand Reinhold. 247 pp. ISBN: 978-0-442-23729-5 (cit. on pp. 26–29, 31, 32, 35, 38–40).

Srinivas, M. and L.M. Patnaik (June 1994). "Genetic algorithms: a survey." In: *Computer* 27.6, 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. 3, 5, 6, 8–10, 23).

Taguchi, Genichi et al. (2005). *Taguchi's quality engineering handbook*. Hoboken, N.J. : Livonia, Mich: John Wiley & Sons ; ASI Consulting Group. 1662 pp. ISBN: 978-0-471-41334-9 (cit. on p. 32).

Whitley, Darrell (June 1, 1994). "A genetic algorithm tutorial." In: *Statistics and Computing* 4.2, 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. 23).

Xia, Xuemin et al. (Feb. 1, 2019). "Genetic algorithm hyper-parameter optimization using Taguchi design for groundwater pollution source identification." In: *Water Supply* 19.1, 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. 4).

Yang, Kai and Basem El-Haik (2009). *Design for six sigma: a roadmap for product development*. 2nd ed. London: McGraw-Hill [distributor]. 741 pp. ISBN: 978-0-07-154767-3 (cit. on pp. 25–32, 34, 37–39).