



# MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

## TRABAJO FIN DE MÁSTER SCENARIO GENERATION FOR THE COMPARISON OF AUTOMATED VEHICLE VARIANTS

Autor: Javier Martín Baroja

Director: Stefan Riedmaier

Madrid

Junio de 2019



**AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS**

**1. Declaration of authorship and accreditation thereof.**

The author Mr. /Ms. Javier Martín Baroja

**HEREBY DECLARES** that he/she owns the intellectual property rights regarding the piece of work: Scenario Generation for the Comparison of Automated Vehicle Variants that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

**2. Subject matter and purpose of this assignment.**

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

**3. Transfer and access terms**

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

- a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.
- b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.
- c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.
- d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.
- e) Assign these pieces of work a Creative Commons license by default.
- f) Assign these pieces of work a HANDLE (*persistent* URL). by default.

**4. Copyright.**

The author, as the owner of a piece of work, has the right to:

- a) Have his/her name clearly identified by the University as the author
- b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.
- c) Request that the work be withdrawn from the repository for just cause.
- d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

**5. Duties of the author.**

The author agrees to:

- a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

- b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
- c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
- d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

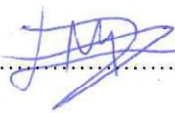
**6. Institutional Repository purposes and functioning.**

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

- a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
- b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
- c) The University shall adopt the necessary measures to safeguard the work in the future.
- d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on 07 of June....., 2019

**HEREBY ACCEPTS**

Signed.....

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

.....*Scenario generation for the Comparison of Automated*  
*Vehicle Variants*.....

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico *2018/19*..... es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es  
plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: *Javier Martín Baroja*

Fecha: *20 / 05 / 2019*

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: 

Fecha: *20 / 05 / 1993*





# MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

## TRABAJO FIN DE MÁSTER SCENARIO GENERATION FOR THE COMPARISON OF AUTOMATED VEHICLE VARIANTS

Autor: Javier Martín Baroja

Director: Stefan Riedmaier

Madrid

Junio de 2019





# SCENARIO GENERATION FOR THE COMPARISON OF AUTOMATED VEHICLE VARIANTS

**Autor: Martín Baroja, Javier.**

Director: Riedmaier, Stefan.

Entidad colaboradora: Technical University of Munich.

## RESUMEN DEL PROYECTO

### 1. Introducción

En los próximos años el sector de la automoción sufrirá varios cambios importantes. Uno de ellos es el uso de vehículos autónomos. Estos vehículos producirán mejoras remarcables en la forma en la que viajamos, ya que permitirán viajes más cómodos y seguros. Sin embargo, la industria automotriz necesita superar algunos desafíos hasta que esta tecnología esté lista para su uso generalizado. Uno de los principales retos es la validación de vehículos autónomos. Por validación se entiende el desarrollo de métodos y pruebas que aseguren que los vehículos sean lo suficientemente seguros para conducir en las carreteras. Hasta entonces los vehículos no pueden ser vendidos al público.

Los procesos de validación se dividen en dos grupos principales: pruebas físicas y simulaciones. Las pruebas físicas consisten en la conducción de un prototipo de un vehículo real en la carretera y la recopilación de datos para verificar que el comportamiento del vehículo es el correcto. Los principales problemas con estas pruebas son los altos costes, el gran consumo de tiempo y la dificultad para realizar maniobras muy específicas [1]. Una alternativa y complemento a las pruebas físicas es el uso de simulaciones. En las simulaciones se diseña un entorno virtual y se crea un modelo del vehículo autónomo para simular su movimiento en este entorno virtual. El uso de simulaciones con escenarios virtuales se ha utilizado para la validación de sistemas avanzados de asistencia al conductor (ADAS) o de vehículos automatizados [2]. Una alternativa a este método es usar una simulación en un entorno virtual para comparar dos sistemas. En este trabajo, el objetivo es desarrollar un método que genere escenarios virtuales específicos para comparar dos sistemas y descubrir en qué situaciones específicas se maximiza la diferencia de comportamiento entre los dos sistemas.

El proceso de generación de escenarios consiste en realizar variaciones en un escenario base con una determinada carretera y una determinada situación de tráfico. Antes de

comenzar el proceso se definen las especificaciones geométricas de la carretera. Los vehículos que conducen en esta carretera, así como sus movimientos y maniobras, también deben describirse. A continuación, algunos de los parámetros que definen el escenario se establecen como parámetros variables. Estos parámetros variables tienen valores diferentes en cada prueba concreta, lo que permite diferentes escenarios en los que el vehículo realiza pruebas de conducción.

Para el desarrollo del método, el primer paso es analizar el estado del arte existente. En los últimos años, muchos investigadores han utilizado técnicas de generación de escenarios para analizar sistemas avanzados de asistencia al conductor (ADAS) así como vehículos autónomos [3]-[4]. En las pruebas el propósito era encontrar las situaciones más críticas. Los trabajos utilizaron diferentes métodos para generar los casos de prueba específicos. Estos métodos se pueden dividir en dos grupos: métodos de cobertura y métodos de falsificación. El método de cobertura se basa en la selección aleatoria de los valores para los parámetros variables del escenario [5]. Por otro lado, el método de falsificación es un proceso iterativo [6]. En este último caso, los valores elegidos para los primeros escenarios se eligen aleatoriamente, pero después, los resultados de las simulaciones de los últimos escenarios simulados se utilizan para seleccionar el siguiente conjunto de valores que definen los siguientes escenarios a simular.

## 2. Metodología

La estructura del problema se muestra en **¡Error! No se encuentra el origen de la referencia.** En este método, el algoritmo selecciona valores para los parámetros variables del escenario. Con estos valores, se define un escenario específico en el que se ejecutan simulaciones para ambos sistemas. Después de la simulación, los resultados se envían de vuelta al algoritmo, que evalúa esa solución específica a través de la función de aptitud.

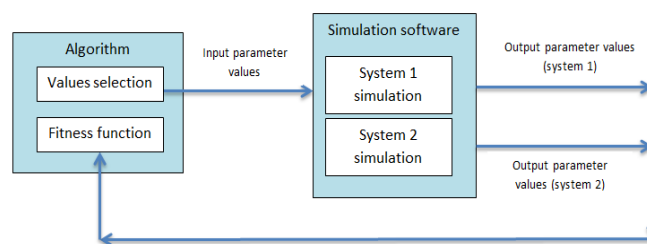


Ilustración 1: Estructura del problema

En primer lugar, se elige el algoritmo para la selección de valores y la función de aptitud. Los dos métodos para la selección de escenarios se introdujeron en el estado del arte: el de cobertura y el de falsificación. Entre estos dos métodos para seleccionar los valores que definen cada escenario específico, el método de falsificación es más complejo de implementar. Sin embargo, como es un proceso iterativo, enfoca la búsqueda alrededor de las áreas más prometedoras. La decisión es utilizar el método de falsificación como método base para el método a desarrollar. Después se evalúan varios algoritmos utilizados en los métodos de falsificación. Las opciones principales son los algoritmos evolutivos y el algoritmo de aprendizaje por refuerzo. La aplicación del aprendizaje por refuerzo es adecuada para casos donde una variable tiene muchos valores a lo largo de la simulación. Sin embargo, éste no es el caso para el proceso de generación de escenarios, donde se utilizan muchas variables y muchas de ellas tienen un valor fijo durante la simulación. Así, el algoritmo evolutivo se elige como el algoritmo encargado de seleccionar los valores de los parámetros variables.

El algoritmo evolutivo necesita una función de aptitud para saber la calidad de una solución determinada. La función de aptitud toma los resultados de una simulación en un escenario concreto y calcula un valor escalar que indica la calidad de esa solución específica. En el caso de estudio, en primer lugar, la simulación se ejecuta para un escenario específico (con ciertos valores para los parámetros variables). Los resultados de esta simulación se envían a la función de aptitud, que necesita calcular un valor para especificar cómo de diferentemente se comportan los dos sistemas para esa solución determinada (o conjunto de valores de parámetros variables). Un alto valor de aptitud indica que los dos sistemas se comportaron de manera muy diferente, que es el objetivo buscado. Por otro lado, un valor de aptitud cercano a cero significa que los dos sistemas hicieron prácticamente lo mismo. La fórmula específica aplicada para calcular la aptitud es la fórmula de error porcentual absoluto medio. Esta fórmula se selecciona porque la diferencia entre los resultados de los dos sistemas se mide como un porcentaje entre las diferencias de resultados en cada punto. Este valor es adimensional, por lo que varias magnitudes diferentes se pueden utilizar para calcular el valor de la aptitud, por ejemplo, la posición y la velocidad.

Antes de comenzar a ejecutar el método es necesario realizar algunas configuraciones. Los algoritmos evolutivos tienen cuatro parámetros principales: tamaño de la población, número de generaciones, tasa de recombinación y tasa de mutación. Dependiendo de sus

valores, el proceso de optimización puede ser más o menos eficiente. Pero los valores exactos varían de un problema a otro. Existe información sobre cómo los cambios en los valores de estos parámetros influyen en los resultados. El proceso para elegir los valores comienza con la selección de algunos valores típicos recomendados. A continuación, se prueban diferentes valores con el propósito de encontrar los valores que conducen a un mejor funcionamiento. Estas pruebas que cambian los valores de los parámetros del algoritmo evolutivo tienen lugar en el primer experimento.

La definición del escenario y la simulación se realizan mediante el uso del software de simulación. El algoritmo a cargo de seleccionar los valores para definir escenarios específicos y el código para controlar el software de simulación se implementa con el lenguaje de programación Python.

### 3. Resultados

Después de seleccionar el algoritmo, la función de aptitud, comprender los parámetros del algoritmo evolutivo e implementar todo, el método está listo para ejecutarse. Se realizan dos experimentos. El primer experimento es la prueba de concepto y sirve para seleccionar los valores para los parámetros del algoritmo evolutivo. En el segundo experimento, se comparan varios algoritmos para descubrir cuál puede encontrar un mejor resultado. El segundo experimento también sirve para comprobar que el método funciona en diferentes escenarios.

En el primer experimento se define un escenario consistente en una maniobra de adelantamiento. El proceso de optimización se ejecuta varias veces. En cada ronda de simulación, se evalúan 500 puntos. Eso significa que se simulan 500 escenarios específicos. Sin embargo, los valores de los parámetros del algoritmo evolutivo cambian de una ronda de simulación a otra. El algoritmo evolutivo específico utilizado es el algoritmo genético. Estos diferentes casos se ejecutan para averiguar qué conjunto de valores para los parámetros del algoritmo genético funcionan mejor.

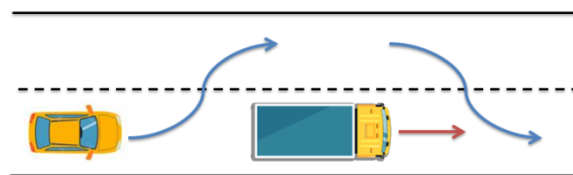
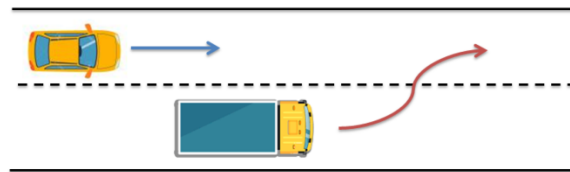


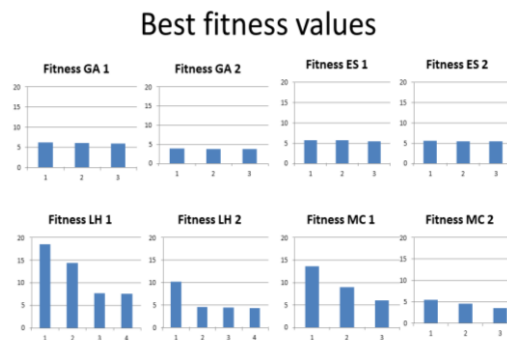
Ilustración 2: Maniobra de adelantamiento

En el segundo experimento, se define un nuevo escenario base. En este caso, se trata de una maniobra de cambio de carril de un camión hasta el carril donde conduce el vehículo autónomo. Los objetivos en este segundo experimento son diferentes. Uno de los objetivos es verificar si los valores seleccionados en el primer experimento para los parámetros del algoritmo genético también funcionan correctamente en un escenario distinto. El primer experimento es el conjunto de entrenamiento y el segundo experimento es el conjunto de prueba. El otro objetivo es la comparación entre diferentes algoritmos. El mismo escenario base se optimiza con cuatro algoritmos diferentes para comparar su rendimiento. Los cuatro algoritmos son dos algoritmos evolutivos (algoritmo genético y estrategias de evolución) y dos algoritmos de cobertura (muestreo de Monte Carlo y muestreo latino hipercúbico).



**Ilustración 3: Maniobra de cambio de carril**

Después de ejecutar estas simulaciones, los resultados muestran que los dos métodos de cobertura obtienen mejores resultados que los dos algoritmos evolutivos. Es decir, pueden encontrar escenarios específicos en los que la diferencia entre los dos sistemas a comparar es mayor. Esto se puede ver en la Ilustración 4, donde las mejores soluciones encontradas con muestreo latino hipercúbico (LH) y Monte Carlo (MC) tienen un valor de aptitud más alto. Hasta ahora, los dos métodos de cobertura son una opción más adecuada ya que son más fáciles de implementar y pueden encontrar mejores resultados.



**Ilustración 4: Resultados de aptitud para los 4 algoritmos**

Pero el experimento no termina aquí dado que se llevan a cabo más rondas de simulaciones. En las siguientes optimizaciones, el número de evaluaciones en el software de simulación aumenta de 500 a 2500. Este cambio se realiza para ver si el número anterior de simulaciones era demasiado bajo para los algoritmos evolutivos. Los dos algoritmos que se ejecutan con la mayor cantidad de simulaciones son el algoritmo genético y el muestreo latino hipercúbico. Los resultados obtenidos muestran que el algoritmo evolutivo mejora significativamente con este aumento en el número total de simulaciones en la búsqueda. Ese no es el caso del muestreo latino hipercúbico cuyos resultados son muy similares a los del caso anterior. Las últimas rondas de simulación se llevan a cabo con el uso de un método mixto que combina los métodos de cobertura y los algoritmos evolutivos. Los resultados indican que el método mixto produce una pequeña mejora en la solución en comparación con los otros métodos utilizados.

#### 4. Conclusión

Este experimento muestra que, para el escenario analizado, los métodos de cobertura son una opción más adecuada que los algoritmos evolutivos porque son más fáciles de implementar y pueden encontrar mejores soluciones. Eso significa que encuentran escenarios específicos donde la diferencia entre los dos sistemas es mayor. La otra opción adecuada es el método mixto, que es más complejo pero fue capaz de una pequeña mejora en los resultados.

Los experimentos realizados son útiles para verificar el funcionamiento correcto de este método en algunos escenarios y también sirven para comparar diferentes algoritmos para la selección del escenario. A partir de este punto, se pueden realizar mejoras futuras aumentando la cantidad de parámetros de entrada que configuran el escenario para tener situaciones más personalizadas. Otro desarrollo adicional sería utilizar el método en otras herramientas de simulación para verificar si el método también funciona correctamente.

Este método tiene algunas aplicaciones posibles en el desarrollo de vehículos automatizados. Una de estas posibles aplicaciones es comparar dos modelos virtuales. Por ejemplo, cuando se lanza una nueva versión y el objetivo es comparar su rendimiento en una carretera virtual en comparación con la versión anterior. Otra posible aplicación sería utilizar este método para comparar un sistema real con un modelo virtual del sistema. Eso

indicaría cómo de bien se ajusta el modelo al sistema real. Esto es importante porque si un modelo es muy realista, las simulaciones realizadas con ese modelo son más verídicas.

## 5. Bibliografía

- [1] N. Kalra, S. M. Paddock, „Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?“.
- [2] Q. Xia, J. Duan, F. Gao, T. Chen und C. Yang, „Automatic Generation Method of Test Scenario for ADAS Based on Complexity“ in *SAE Technical Paper Series*, 2017.
- [3] J. Zhou und L. d. Re, „Reduced Complexity Safety Testing for ADAS & ADF“, *IFAC-PapersOnLine*, Jg. 50, Nr. 1, S. 5985–5990, 2017.
- [4] L. Huang, Q. Xia, F. Xie, H.-L. Xiu und H. Shu, „Study on the Test Scenarios of Level 2 Automated Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018, S. 49–54.
- [5] S. Khastgir *et al.*, „Test Scenario Generation for Driving Simulators Using Constrained Randomization Technique“ in *SAE Technical Paper Series*, 2017.
- [6] M. Koren, S. Alsaif, R. Lee und M. J. Kochenderfer, „Adaptive Stress Testing for Autonomous Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 1–7.





# Scenario Generation for the Comparison of Automated Vehicle Variants

## 1. Introduction

In the next years the automotive sector will have several important changes. One of these is the use of autonomous vehicles. These vehicles will produce notorious improvements on how we travel as they will allow for more comfortable and safer journeys. Nevertheless, the automotive industry needs to overcome some challenges until this technology is ready for a widespread use. One of the main challenges is the validation of autonomous vehicles. By validation it is meant the development of methods and tests that ensure that the vehicles are safe enough to drive in the roads. Before that, the vehicles cannot be sold to the public.

The processes for validation are divided into two main groups: physical tests and simulations. Physical tests consist on driving a prototype of a real car in the road for a long distance and gather data to check that the vehicle's behavior is correct. The main problems with physical tests are the high costs, the high time consumption and the difficulty to perform very specific maneuvers [1]. An alternative and complement to the physical tests is the use of simulations. In the simulations a virtual environment is designed and a model of the autonomous vehicle is created for different tests in this virtual environment. The use of simulations with virtual scenarios has been used for validation methodology of specific driver assistance systems or automated vehicles [2]. A derivation of this idea is to use a simulation in a virtual environment to compare two systems. In the thesis the objective is to develop a method that generates specific virtual scenarios to compare two systems and find out in which specific situations the behavior difference between the two systems is maximized.

The scenario generation process consists on making variations to a base scenario with a certain road and traffic situation. So before starting the process some road geometry specifications are defined. The vehicles that are driving in this road as well as their maneuvers also need to be defined. Then some of the parameters to define the scenario are stated as variable parameters. These variable parameters will have different values from one test case to another, allowing for different scenario situations where the vehicle to test drives.

For the method development the first step is to analyze the existing literature. In the last years many researchers have used scenario generation techniques to analyze advanced driver assistance systems (ADAS) and autonomous vehicles [3]-[4]. In the tests the purpose was to find the most critical situations. The papers used different approaches to generate the specific test cases. These methodologies can be divided into two groups: coverage approach and falsification approach. The coverage approach is based on a random selection of the values for the variable parameters [5]. On the other hand, the falsification approach is an iterative process [6]. The values chosen for the first points are randomly chosen but after that the results from the simulations of the current points are used to select the next set of values to simulate.

## 2. Methodology

The problem structure is shown in Figure 1. In this method, the algorithm selects values for the variable parameters of the scenario. With these values, a specific scenario to test is defined. The simulator runs simulations in this specific scenario for both systems. After the simulation, the results are sent back to the algorithm, which evaluates that specific solution through the fitness function.

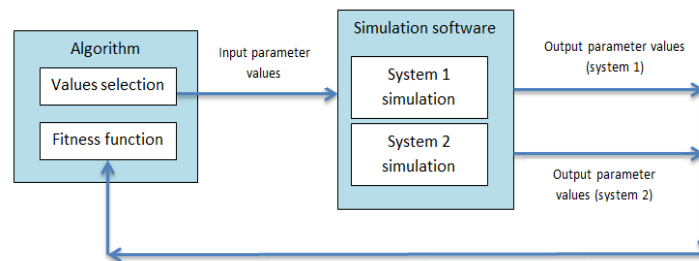


Figure 1: Method structure

First of all, the algorithm for values selection and the fitness function are chosen. The two approaches for the selection of scenarios were introduced in the state of the art. Between these two approaches to select the values that define each specific scenario, the falsification approach is more complex to implement. Nevertheless as it is an iterative process it focuses the search around the most promising areas. The decision is to use the falsification approach as the base method for the thesis. Then, several algorithms used in falsification approaches are evaluated. The main options are the evolutionary algorithms

and the reinforcement learning algorithm. The application of reinforcement learning is suitable for cases where a variable has many values along the simulation. However for the scenario generation process, many variables are used and many of them have a fixed value during the simulation. So the evolutionary algorithm is chosen as the algorithm in charge of selecting the variable parameter values.

The evolutionary algorithm needs a fitness function to know how good a certain solution is. The fitness function takes the results of a certain test case simulation and calculates a scalar value which indicates the quality of that specific solution. In the thesis case, firstly the simulation is run for a specific scenario (with certain values for the variable parameters). The results from this simulation are sent to the fitness function that needs to calculate one value to specify how different the two systems behaved for that certain solution (or set of variable parameter values). A big fitness value indicates that the two systems behaved very differently, which is the objective searched. On the other hand a fitness value close to zero means that the two systems did almost the same. The specific formula applied to calculate the fitness is the mean absolute percentage error (MAPE) formula. The formula is selected because the difference is measured as a percentage between the results differences at each point. This value is dimensionless so several different magnitudes can be summed into the same fitness value, for example position and speed.

Before starting to run the method it is necessary to make some configurations. The evolutionary algorithms have four main parameters: population size, number of generations, crossover rate and mutation rate. Depending on their values the optimization process can be more or less efficient. But the exact values vary from a problem to another. There is some literature on how the changes in these parameters values influence the results. The process to choose the values starts with selecting some recommended typical values. Then, different values are tested with the purpose of finding the values that lead to a better performance. These tests that change the values of the evolutionary algorithm parameters take place in the first experiment.

The scenario definition and simulation is done through the use of simulation software. The algorithm in charge of selecting the values to define specific scenarios and the code to control the simulation software is implemented with python programming language.

### 3. Results

After selecting the algorithm, the fitness function, understanding the evolutionary algorithm parameters and implementing everything, the method is ready to be run. Two experiments are carried out. The first experiment is the proof of concept and serves to select the values for the evolutionary algorithm parameters. In the second experiment, several algorithms are compared to find out which one can find a better result. The second experiment also serves to check that the method works in different scenarios.

In the first experiment an overtaking maneuver scenario is defined. The optimization process is run several times. On each simulation round, 500 points are evaluated. That means that 500 specific scenarios are simulated. However, the evolutionary algorithm parameter values change from one simulation round to another. The specific evolutionary algorithm used is the genetic algorithm. These different cases are run to find out which set of values for the genetic algorithm parameters work better.

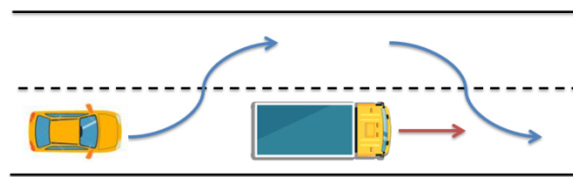


Figure 2: Overtaking maneuver (Scenario 1)

In the second experiment, a new base scenario is defined. In this case it is a cut-in maneuver from a truck in to the lane where the ego-vehicle is driving. The objectives in this second experiment are different. One of the objectives is to check if the values selected for the genetic algorithm parameters also perform well in a new scenario. The first experiment is the training set and the second experiment is the test set. The other objective is the comparison between different algorithms. The same base scenario is optimized with four different algorithms to compare their performance. The four algorithms are two evolutionary algorithms (genetic algorithm and evolution strategies) and two coverage approaches (Monte Carlo sampling and Latin Hypercube sampling).

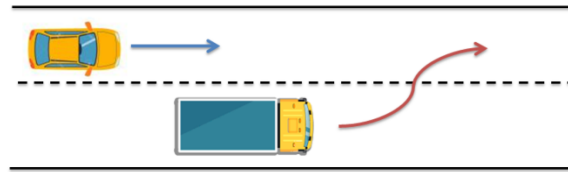


Figure 3: Cut-in maneuver (Scenario 2)

After running these simulations, the results show that the two coverage methods obtained better results than the two evolutionary algorithms. That is, they can find specific scenarios in which the difference between the two systems to compare was greater. That can be seen in Figure 4, where the best solutions found with Latin Hypercube Sampling (LH) and Monte Carlo (MC) have a higher fitness value. So far the two coverage approaches are a more suitable option as they are simpler to implement and they could find better results.



Figure 4: Fitness results for the 4 algorithms

But the experiment does not end here as more simulation rounds are run. In the following optimizations, the number of point evaluations in the simulation software is increased from 500 to 2500. This change is done to see if the previous number of simulations was too low for the evolutionary algorithms. The two algorithms to run with the increased number of simulations are the genetic algorithm and the Latin Hypercube sampling. The results obtained show that the evolutionary algorithm improves significantly with this increase in the total number of simulations in the search. That is not the case for the latin hypercube sampling which performs very similar to the case before. The last simulation rounds are carried out with the use of a mixed method that combines the coverage approaches and the evolutionary algorithms. Based on the results the mixed method produces a small improvement in the solution compared to the other methods used.

#### 4. Conclusion

This experiment shows that for the scenario tested the coverage approaches are a more suitable option than the evolutionary algorithms because they are easier to implement and they can find better solutions. That means that they find specific scenarios where the difference between the two systems is greater. The other suitable option is the mixed method, which is more complex but could improve the results.

The experiments carried out are useful to check the correct operation of this method in some scenarios and also served to compare different algorithms for the scenario selection. From this point, future improvements can be done by increasing the number of input parameters that configure the road in order to have more customized scenarios. Another further development would be to use the method in other simulations to check whether the method work also correctly.

This method has some possible applications in the development of automated vehicles. One of these possible applications is to compare two virtual models. For example, when a new version is released and the objective is to compare its performance on a virtual road compared to the previous version. Other possible application would be to use this method to compare a real system with a virtual model of the system. That would indicate how accurate the model and the real system are. So if a model is very realistic, the simulations made with that model are more useful for validation.

#### 5. References

- [1] N. Kalra, S. M. Paddock, „Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?“.
- [2] Q. Xia, J. Duan, F. Gao, T. Chen und C. Yang, „Automatic Generation Method of Test Scenario for ADAS Based on Complexity“ in *SAE Technical Paper Series*, 2017.
- [3] J. Zhou und L. d. Re, „Reduced Complexity Safety Testing for ADAS & ADF“, *IFAC-PapersOnLine*, Jg. 50, Nr. 1, S. 5985–5990, 2017.
- [4] L. Huang, Q. Xia, F. Xie, H.-L. Xiu und H. Shu, „Study on the Test Scenarios of Level 2 Automated Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018, S. 49–54.

- [5] S. Khastgir *et al.*, „Test Scenario Generation for Driving Simulators Using Constrained Randomization Technique“ in *SAE Technical Paper Series*, 2017.
- [6] M. Koren, S. Alsaif, R. Lee und M. J. Kochenderfer, „Adaptive Stress Testing for Autonomous Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 1–7.





# Table of Content

- List of Abbreviations ..... III**
- Symbols..... V**
- 1 Introduction..... 1**
  - 1.1 Problem statement ..... 1**
  - 1.2 Thesis motivation ..... 2**
  - 1.3 Structure..... 2**
- 2 State of the Art..... 5**
  - 2.1 Autonomous vehicles and validation ..... 5**
  - 2.2 Scenario generation methods ..... 7**
    - 2.2.1 Coverage methods ..... 7
    - 2.2.2 Falsification method ..... 11
  - 2.3 Differential analysis..... 17**
  - 2.4 Optimization algorithms ..... 18**
    - 2.4.1 Evolutionary algorithms..... 18
    - 2.4.2 Genetic algorithm..... 20
    - 2.4.3 Continuous genetic algorithm ..... 22
    - 2.4.4 Evolution strategies..... 23
    - 2.4.5 Reinforcement learning..... 25
  - 2.5 Summary about the State of the Art ..... 27**
- 3 Methodology ..... 31**
  - 3.1 Process for Scenario definition ..... 32**
  - 3.2 Algorithm selection ..... 35**
    - 3.2.1 Falsification algorithm ..... 36
    - 3.2.2 Coverage algorithm ..... 37
  - 3.3 Fitness function..... 38**
  - 3.4 Evolutionary algorithms configuration ..... 42**
    - 3.4.1 Algorithm quality ..... 43
    - 3.4.2 End condition ..... 43

3.4.3	Influence of the evolutionary algorithm parameters .....	45
3.4.4	State of the Art for the selection of parameter values .....	46
<b>3.5</b>	<b>Implementation on Python and Simulation Software .....</b>	<b>48</b>
<b>4</b>	<b>Results.....</b>	<b>53</b>
<b>4.1</b>	<b>Evolutionary algorithm calibration .....</b>	<b>54</b>
4.1.1	Scenario definition .....	54
4.1.2	Genetic algorithm parameters .....	57
4.1.3	Simulations results.....	59
<b>4.2</b>	<b>Comparison between algorithms.....</b>	<b>65</b>
4.2.1	Scenario definition .....	65
4.2.2	Four algorithms for performance comparison .....	68
4.2.3	Simulation results for the four algorithms.....	69
4.2.4	Algorithm comparison summary .....	77
4.2.5	Solution improvements .....	77
<b>4.3</b>	<b>Simulations results summary and future challenges .....</b>	<b>81</b>
<b>5</b>	<b>Conclusion.....</b>	<b>85</b>
	<b>List of figures .....</b>	<b>i</b>
	<b>List of tables.....</b>	<b>iii</b>
	<b>Bibliography.....</b>	<b>v</b>
	<b>Appendix .....</b>	<b>ix</b>

# List of Abbreviations

ADAS	Advanced Driver Assistance Systems
ES	Evolution Strategies
GA	Genetic Algorithm
LHS	Latin Hypercube Sampling
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MC	Monte Carlo
MES	Mean Squared Error
RMES	Root Mean Squared Error



# Symbols

Symbols	Unit	Description
$S_1$	M	System 1 position
$V_1$	km/h	System 1 speed
$S_2$	m	System 2 position
$V_2$	Km/h	System 2 speed



# 1 Introduction

The autonomous car is a new technology in development that is expected to have a great impact in the future. It will change the way in which we see now transportation. Before the driverless cars are allowed to be sold to the public and allowed to circulate freely in the roads, it should be demonstrated that they are safe enough. The road and traffic situations possibilities are very large and the autonomous cars should be ready to overcome any of them without accidents.

## 1.1 Problem statement

The tests required to demonstrate the safety can be divided into real tests and simulations. The real tests consist of cars driving in real roads. But in order to guarantee the safety, the number of kilometers driven needs to be very high. Another problem is that the majority of scenarios are very simple for the car, but it is in some specific difficult scenarios where the attention needs to be focused in order to solve possible problems. So the physical tests are expensive, time-consuming and have more difficulty to test specific cases.

Another tool to test driverless cars safety complementary to the real test is the use of simulations. Thanks to these tools, a huge amount of tests can be made in short time and smaller cost. The focus of simulations is usually on finding the adequate scenarios to verify the car safety. With simulators thousands of scenarios can be created for the tests. Therefore, it needs to be some method to identify critical scenarios where the car does not drive safely and could have an accident.

Several studies have focused lately in this area with the objective of generating scenarios that lead to a critical situation. This specific area has been the subject of study in recent papers. However, other possible applications can be derived based on these methods. A promising application is the creation of a method that compares two systems. The objective would be to find specific scenarios in which the behavior of the two systems differ the most. A similar concept has been developed for aircraft applications. In the case of autonomous cars, this method could be applied to compare a real part with its virtual model and check how accurate the model is. If

a simulation model has some differences with the real system's behavior, then the simulation tests are not valid. It could be possible that the system performs successfully in certain scenario during a simulation but it does some mistakes in the same scenario in real life.

## 1.2 Thesis motivation

The thesis objective is the development of a generic method to compare two systems in a simulation environment. This method can be a useful help for the validation processes in autonomous vehicles. For the application of this method to a specific scenario, first the parameters that define the scenario are divided into fixed parameters and variable parameters. The fixed parameters are constant for all tests whereas the variable parameters can change from one test to another. In this situation two different vehicles or systems are tested. The method is responsible of trying different combinations of the variable parameters with the purpose of finding specific scenarios in which the difference between the two systems behavior is maximized.

The method is planned to be generic so that it can work in many different scenarios. The only difference is that before running the method, it is necessary to choose which ones are the variable parameters and define the constraints for the values of these variable parameters.

## 1.3 Structure

After the problem statement and the project objective definition, the next step is to explain how this new method was developed. The thesis structure is exposed next:

- The first part is the state of the art chapter. First of all, an introduction to autonomous vehicles and validation is found. Then the concept of scenario generation is introduced and some important research about this topic is analyzed. Besides, some optimization algorithms that can be applied to solve the problem are explained.
- The following chapter is the project methodology. In this chapter the different algorithm possibilities are evaluated. After that the method is developed. This includes the decisions about fitness function, algorithm quality measurement and choosing the parameters for the algorithm. The last part of the methodology chapter shows how the method was implemented with software tools. Simulation software was required for the



scenarios definition and simulation. The code for the optimization algorithm and to control the simulation was developed with Python.

- After that comes the results chapter. This chapter deals with the specific scenarios to simulate as well as their results. Two different scenarios are tested here. The first part simulates a scenario with the focus on calibrating the algorithm. A scenario is defined including its input and output parameters. The results are analyzed to understand which parameter values lead to better results. After that a second scenario is simulated. On the second one, several different algorithms are tested to find out which one is more suitable.
- Finally there is a conclusion chapter that includes a summary of the project, paying special attention to the most important points.



## 2 State of the Art

### 2.1 Autonomous vehicles and validation

Car transport is one of the main means of transport in today's society. Only in Europe there are over 250 million of passenger cars [1]. That is approximately one car for every two citizens. The reason for this large number is that these vehicles allow the owners a lot of autonomy for travelling. During the last decades there has been a huge development in the automotive sector. The improvements have been mainly motivated by economic, environmental, comfort and safety factors. The economic motivations are reducing the fuel consumption and keeping a competitive price on the vehicles. The environmental improvements include the reduction in the engines emissions. The automobiles comfort has been improved by the addition of driving assistance systems. Finally the safety development is concerned with avoiding accidents and reducing the damage in the case of an accident. Despite these important improvements, another important development will produce a big change in the automobile sector in the next years. This new development is the autonomous car.

The autonomous vehicles will allow for a higher safety in the road and a more comfortable travel for the car owner. Regarding the comfort, the driver will not have to pay attention to the road anymore and will be able to do other activities while travelling in a car. The second advantage is the improved safety. Nowadays one of the main problems with cars is the number of accidents and victims produced by them. Only in Germany 3180 people died in traffic accidents in 2017 [2]. That is a traffic-related death of about 4 people per 100.000 inhabitants per year. In 2000 about 7500 people died in vehicle accidents in Germany, more than double of the current numbers. Even after this notorious decrease, a big effort is established into improving the car safety. The estimation is that in 94% of car crashes, the critical reason that produced the accident can be assigned to the actions of the driver [3]. With the introduction and expansion of autonomous vehicles it is expected that the number of casualties produced by road accident will be close to zero in 2050 [4].

The economic impact is also expected to be huge. It is estimated that the automatic driving industry will increase its annual output value around 7 trillion dollars [5].

Nevertheless, the autonomous vehicles commercialization is more complex than other assistance systems. The autonomous car responsibility is much greater so the validation processes are far more complicated. In order to allow the sale of driverless cars, it should be demonstrated that they are safe enough. The validation is done through different vehicle tests. The tests required to demonstrate the safety can be divided into real tests and simulations. The real tests consist of vehicles driving on real roads. An example of road test is carried out by Broggi et al. [6]. But to demonstrate the safety, the cars need to drive many kilometers. According to Winner et al. [7] more than 100 million of kilometers need to be driven to demonstrate that autonomous vehicles are as safe as manually driven vehicles. In other papers the number is even greater. Kalra and Paddock [8] claim that 440 million of kilometers are necessary to demonstrate that autonomous vehicles can perform better than humans. Therefore it is very expensive and time-consuming to drive these big amounts of kilometers. Another problem associated to physical tests is the difficulty to analyze very specific scenarios.

The other tool to test driverless cars safety is the use of simulations. Autonomous cars simulators are of great importance for the development of this new technology. They can perform a huge amount of simulations that in the case of using a real car would be very time consuming and high costly. In order to perform simulations it is necessary to generate scenarios. That refers to the process of defining all the parameters for the scenario in which the software or hardware are tested. Scenario generation is usually focused on finding critical scenarios. When using car simulators, the number of possible scenarios that can be analyzed is huge. The majority of them are very simple for the autonomous car to drive without any problem. Therefore, it is important to have some method that can search for specific scenarios which lead to critical situations.

Another option is to use scenario generation to perform a differential analysis. That is a comparison between two systems. A differential analysis could be useful for example to compare two different autonomous car models. Another possible application is to compare a new update of some model with its previous version. Or it could compare a hardware part with its virtual model to check how accurate the model is. This is a promising research area as there is no specific work in differential analysis application for autonomous vehicles. In order to carry out differential analysis it is required to develop a method based on scenario generation.

## 2.2 Scenario generation methods

The term scenario generation is used for the setting of a road with a specific traffic situation on a simulator. The scenario generation includes the parameter setting for the road size and shape. It also includes the definition of other traffic vehicles that are driving on that road as well as their position and speed. Other objects such as pedestrians or bicycles can also be found. Some research on scenario generation for autonomous cars can be easily found. Many scientists have applied different methods in order to find critical scenarios on autonomous cars or driving assistance systems simulations.

On the scenario generation papers, the objective is often to find the most critical situations. In order to find these situations an algorithm makes variations in some of the parameters that configure the scenario. The different scenario generation methods used by many researchers can be divided into two groups depending on the criterion applied to determine how to variate the parameters. These two groups are coverage and falsification. Some of these studies on scenario generation are analyzed next. These studies show different applications of scenario generation for autonomous cars. The specific scenarios to test are different from one study to another. In addition, different algorithms are used for the validation process. The scenario generation process can run through simulators or can be based in mathematical models.

### 2.2.1 Coverage methods

By coverage method it is understood a process in which a set of points is randomly selected around the whole parameter space. The algorithm carries out one simulation with the parameter values corresponding to each point. But the points are chosen before starting the individual simulations so no feedback is given to know which area contains points that are having better results (in terms of objective function). The point selection is done in a way which tries to cover as homogeneously as possible the parameter space. Some scientists have studied scenario generation with the application of coverage methods. Some of these studies are commented next.

Scenario generation can be applied with the objective of computing the occupancy of an automated car on the road with the use of reachable sets [9]. In this study a model was created with the vehicle dynamics characteristics to calculate what area the car occupies after a certain maneuver. That is important to know if some traffic situations will lead to a collision or the vehicle would be able to avoid it. To do that, a set of possible initial states was established, under a set of

possible inputs and parameters. With these parameters, the ego-vehicle and other traffic participant possible states were calculated. The safety verification is based on mathematical models of how the vehicle behaves. Therefore, the result will only be good if the models are precise. The test maneuvers that were checked include evasive maneuver, moose test and cornering. The algorithm used by the authors is rapidly exploring random trees (RRTs). The objective followed with this algorithm was to cover the area around a reference trajectory. The verification procedure will calculate whether a planned trajectory in a scenario is safe. For that it calculates the occupancy of the ego-vehicle and checks that it does not intersect with the occupancy of other vehicles or obstacles.

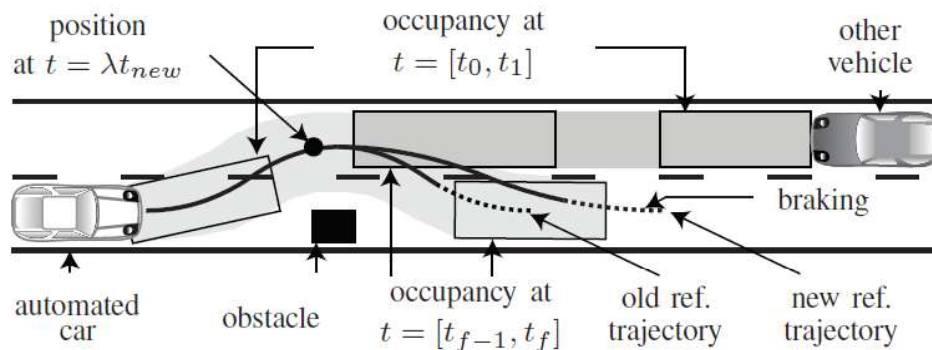


Figure 2.1: Vehicle occupancy calculation [9]

Another application of scenario generation can be done through a model based validation approach to validate an Advanced Driver Assistance System (ADAS) [10]. In this study, the first part consists in the system tuning. First the KPIs for the experiment are established, which are safety, fuel performance and driver comfort. Then, the researchers look for the parameters which have more influence have into these KPIs. In order to do that, a sensitivity analysis is carried out on 10 tuning parameters. The scenario for this sensitivity analysis consists of an ego-vehicle following a preceding vehicle. The sensitivity analysis used a robust neural network and found the 5 parameters that have a notorious influence on the system behavior. After tuning the system, the researchers proceed to the model validation. Here a parameter space coverage is determined using Design of Experiments (DOE). The scenario selection process for the validation consists of a vehicle cutting in from the right. New KPIs are selected, which include minimum clearance distance and minimum headway time. They define a criterion for critical situation and run the experiment.

Huang et al. [11] analyze some features that are included in level 2 autonomous cars through test scenarios. Level 2 functions include assistance systems such as adaptive cruise control (ACC), active lane changing control and active lane keeping control. The test scenarios used for validation include different combination of vehicles surrounding the ego-vehicle. For each vehicles position combination, test scenarios are defined by setting different motions and maneuvers on the ego-vehicle and the other vehicles. The scenario to test was formed by a three lanes road. For the cases definition, the area around the ego-vehicle was divided into eight quadrants. The cases were defined depending on how many vehicles were around the ego-vehicle and in which of these quadrants they were located. So they paid special attention to the relative position of the other traffic vehicles respect to the autonomous car.

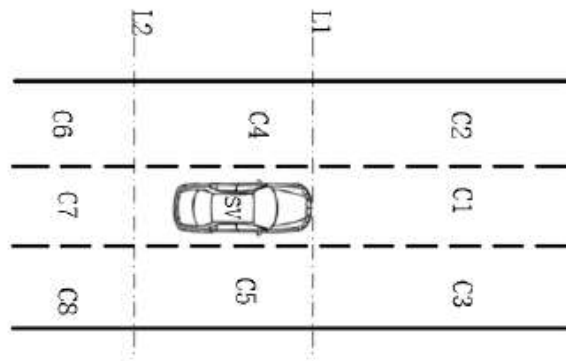


Figure 2.2: Possible positions for traffic vehicles [11]

The ego-vehicle could perform five different driving maneuvers: going straight, left and right lane change and moving laterally left and right. These different movements for the ego-vehicle and other participant vehicles are combined to generate test scenarios.

Another study is concerned about the huge sample space of inputs for the verification and validation of Advanced Driver Assistance Systems (ADAS) [12]. To solve this problem the proposal includes the use of randomization techniques to generate scenarios. There are two levels of randomization process: test scenario randomization and test case randomization. A test scenario can have many test cases, each of those includes some variation but they are included in the same test scenario if the environmental conditions are kept the same way. The randomization method is applied in a driving simulation environment. The specific system to be tested in the scenarios was an automatic emergency braking system (AEB). The AEB system was implemented in a real hardware unit whereas the rest was done in a driving simulator environment.

Most of the cases were focused on creating different tests cases but maintaining the same road for all cases. Another application of scenario generation can be the creation of roads. Kim et al. [13] develop an automatic road network generator. The paper is focused on building a road environment for simulations. The generator will receive the information about the number of curves, distance between curves and the curvatures. With this information it will create virtual roads in 3D. The framework should be able to generate several road structures based on the curve coverage criteria. The coverage criterion refers to the curves information mentioned before. But instead of having fixed values about the road characteristics, what is given is a range of values for each parameter. The scenario generated parameters should fulfill all these criteria. These scenarios can be used to test assistance systems such as adaptive cruise control and lane keeping assistance.

Xia et al. [14] propose another method for testing driver assistance systems that automatically generates test scenarios. The researchers are concerned about developing a bench test that is more efficient than previous testing methods and can cover more scenarios. They also developed an index to represent how complex a certain test case is. First the influence factors of the driver assistance system are listed. The factors can be continuous or discrete. Only the discrete factors are used for the test to reduce the number of test cases. But some of the continuous factors are discretized. For the test, the real camera is used but the environment is virtually generated. The test cases are generated by combining different values of the factors previously selected with an N-wise combinatorial testing method. In the test, the assistance system needs to recognize the environment (signs, lane marks) in a road.

In another paper about advanced driver assistance systems and automated driving functions (ADAS and ADF) the objective is to reduce the testing complexity [15]. One proposal is the use of a test case catalogue to cover the main critical situations. The scenarios tested take place in highways. In the basic test cases, the ego-vehicle can only perform one kind of maneuver. First the critical maneuver corresponds to longitudinal movement. The test cases include front-end and rear-end collision between ego-vehicle and other vehicle driving in the road. On the second set of tests the focus is on the lateral movement. The test cases check the possible front-end and rear-end collision when the ego-vehicle changes the lane. After the definition of the basic test cases, complex test cases can be created by combining several basic cases.



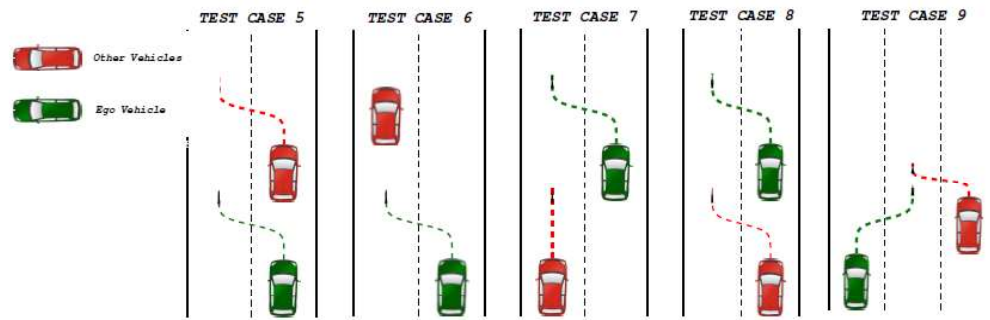


Figure 2.3: Test cases for vehicle testing [15]

The last coverage approach is based on ontologies. That is the use of expert knowledge to identify the scenarios which are more difficult for the car to handle. The scenario creation process is more creative than systematic. Knowledge-based systems can be used for scenario generation in the development and validation of automated driving functions [16]. In this case the expert knowledge is applied to computer models. For example the design of a traffic situation and the interaction between all the traffic participants is based on this knowledge. The traffic knowledge can be separated into several layers and the interaction between layers should also be modelled.

The studies commented before were focused on creating scenarios for autonomous car testing. But the particularity of them is that they used a coverage method during the analysis step. That means that they were establishing different points to cover the search space during the simulation.

### 2.2.2 Falsification method

Other test scenario studies use another approach that can be named as falsification. There is a main difference between the falsification approaches and coverage approaches. The coverage approaches tried to establish a lot of points with the parameters values to cover the search space homogeneously. The falsification approach uses a different method. It establishes first a few points on the search space. After evaluating the results on those points with an objective function, it uses that information to choose the next points to analyze. So it has some feedback in order to have a better guidance on how to select the following points. For example, if the objective is to maximize the objective function value, then the points which gave a higher value after

a simulation round will be selected. Based on these points, new points are selected and a new simulation is done. This process is repeated iteratively.

The first falsification method tries to generate automatically traffic situations for which it is hard to obtain a safe motion plan [17]. That refers to situations with a small solution space that is enough to avoid a collision. Some situations recorded from real traffic are taken and the approach is applied to increase the criticality. In order to do that the initial states of all traffic participants (position, speed, size...) are optimized until the desired size of drivable area is obtained. The value of the desired drivable area is decided manually depending on what level of criticality is searched in the test. For that the drivable area of the ego-vehicle is computed. If the scenario includes other traffic participants then their drivable area is also computed and this area is removed from the allowed space for the ego-vehicle. One scenario example is a road with obstacles on both sides. As they increase the size of an obstacle the drivable area gets reduced until it reaches a specific value previously fixed. The drivable area was modeled mathematically and the problem was formulated as a Quadratic Programming Problem. Three scenarios are tested with this method: one is about obstacle evasion. The second one is an urban road situation with two lanes. All the vehicles drive there in the same direction. The last scenario is a rural road. It also has two lanes but one for each direction.

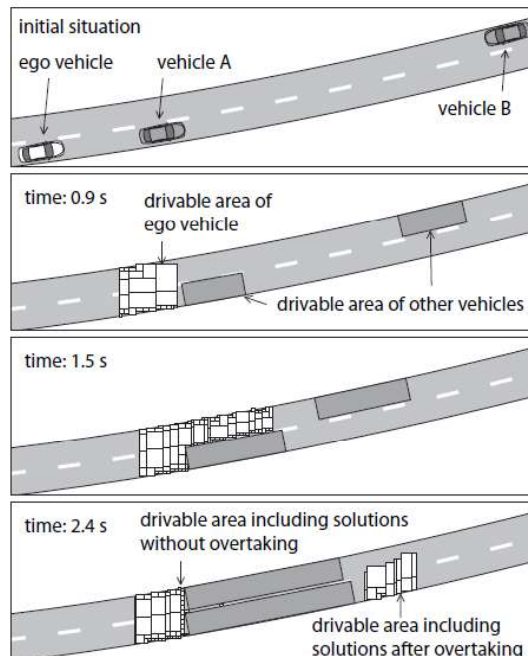


Figure 2.4: Drivable area calculation [17]

In another study the objective is to reduce the testing effort for driverless vehicles [18]. In this paper a stochastic optimization method is used in order to minimize a cost function and find faulty behavior regions. The method is an iterative approach that will lead the tests cases to critical regions in the parameter space. Instead of running the optimization algorithm directly on the system, a surrogate model of the system was designed. This surrogate model is less computationally expensive. The algorithm can be used in black box problems. At the beginning, the algorithm receives a search space and makes an initial sampling. After each iteration the algorithm will zoom in and reduce the size of the new sampling set. For the optimization task the algorithms used were the Differential Evolution genetic optimization algorithm and Particle Swarm optimization. The testing scenario was an emergency brake maneuver for a passenger car. In this scenario the car was driving on the highway and encounters an obstacle. The brake system needs to brake to avoid the car collision against the obstacle.

Another interesting research develops a method to identify critical scenarios based on simulations [19]. This method is formed by a simulation-based toolchain. The purpose is the identification of scenarios that are critical for the current level of development of the autonomous vehicle. The simulation framework is formed by the vehicle dynamics simulation, a traffic simulation as environment and a cooperation simulation. A scenario is defined by a set of parameters which are constrained between some maximum and minimum values. But inside this parameter space the number of possible scenarios that can be created is huge. The selection of the parameters values can be made through opinions, catalogues and recorded data. But the scenarios defined by these metrics can neglect critical scenarios. In this method, the toolchain creates specific scenarios by selecting different values for the parameters of the logical scenario automatically. The objective is to identify critical scenarios over the entire parameter range. One of the scenarios analyzed is a highway with three lanes and three vehicles driving. The ego-vehicle is driving on the right lane and the other two vehicles are on the middle lane. In this scenario the ego-vehicle performs a lane change so it is positioned in the gap between the other two vehicles.

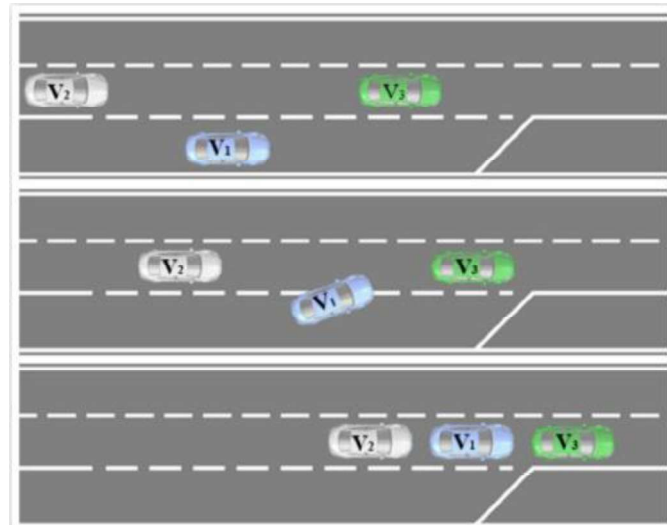


Figure 2.5: Highway scenario for testing [19]

Another testing option for autonomous vehicles is through adaptive stress testing [20]. The approach is based on changing elements in the environment to produce a collision of the vehicle. With that the decision making of the autonomous vehicles is tested. The novelty of the study is that the problem is formulated as a Markov decision process. The problem is solved with the application of a deep reinforcement learning algorithm to find probable failure cases. The deep reinforcement learning algorithm is also compared with Monte Carlo Tree Search (MCTS) to check which option performs better. The scenario where the simulation takes place is a road with two lanes and a pedestrian crosswalk. In the road there is a single autonomous car approaching this pedestrian crosswalk. One or more pedestrians cross the road depending on the case. The different test cases are defined by giving different values to the pedestrians position and speed. The algorithm is the one in charge of selecting the values for the position and speed. The first case had only one pedestrian but more were added to later tests. The results found were that the reinforcement learner was able to find collisions with a smaller number of calls than the Monte Carlo method.

Lindlar [21] proposes the application of evolutionary testing to automate tests. By evolutionary testing it is understood a method to solve testing problems with evolutionary algorithms. The evolutionary algorithm is in charge of selecting suitable test cases. The main application of the proposed method is for embedded software testing. These tests include complicated data sequences. The scenario chosen for the testing process is formed by two vehicles driving in the same lane. The vehicle on the back is the vehicle where a system is tested whereas the vehicle

on the front is a traffic vehicle. The front vehicle speed has many variations during the test and the vehicle behind needs to avoid getting too close to the other vehicle.

Another case of automatic test generation for autonomous cars is based on the use of S-TaLiRo toolbox [22]. This approach generates automatically test cases to test motion controllers in autonomous vehicles. The tests are performed in simulation environments. Initial states and inputs are updated by stochastic optimization methods with the objective of achieving small robustness values. The robustness metric tells how far a system is from failing to meet the safety requirements. In this method the toolbox uses a global optimization method to minimize robustness. At the beginning a sample space is created. The initial states and input functions values are sampled. The output trajectory is obtained and supplied to the function that evaluates the robustness. The evaluation function returns one value that is a measure of how close the simulation is to reach an unsafe state. The scenario is a straight road with two lanes. The inputs are the target speed for the vehicle to test and the speed and lateral position for the other vehicle. In this scenario two vehicles under test are driving on the right lane. A third vehicle drives on the left lane and suddenly starts a lane change maneuver to the right lane.

A similar study to the previous ones is the automatic test case generation with gradient descent optimization [23]. The concern is about high-fidelity models because the test cases generation requires long time. To improve that they use low-fidelity models to drive the test generation process. The gradient descent method is based on computing the gradients of the system dynamics to search for a minimizer. The function that refers to the system performance is the one that they try to minimize. As the system dynamics function can be nonlinear, an alternative is to use a low-fidelity model derived from the complex model to guide the search. The system tested was a Full Range Adaptive Cruise Control (FRACC). The scenario can be defined as stop and go. The vehicle to test follows another vehicle that is increasing and decreasing its speed during the whole simulation.

Scenario generation is used for many applications as explained in the previous papers. The typical systems and scenarios to test are quite varied. Besides, the selection method for the test cases changes from one study to another. In order to have a better comprehension, the papers are summarized in Table 2.1: Scenario generation papers summaryTable 2.1.

Table 2.1: Scenario generation papers summary

Optimization method	Scenarios tested	Reference
Rapidly exploring random trees	Evasive maneuver, moose test & cornering	[9]
DOE	Cut in	[10]
Manually selected test cases	3 lanes road with vehicles around ego-vehicle	[11]
Randomization techniques	Emergency braking system	[12]
Coverage criteria	Road generation	[13]
N-wise combinatorial testing	Environment recognition in a road	[14]
Test case catalogue	Front-end rear-end collision	[15]
Ontologies	Lane change	[16]
Quadratic programming problem & Binary search	Obstacle evasion & highway with two lanes	[17]
Differential evolution, GA & PSO	Emergency braking maneuver	[18]
Parameter change through toolchain	Lane change	[19]
MCTS and reinforcement learning	Pedestrian crosswalk in a road	[20]
Evolutionary algorithms	ACC	[21]
Optimization method from S-TaLiRo	Cut in	[22]
Gradient descent method	ACC	[23]

## 2.3 Differential analysis

In all the papers mentioned before, the simulations were focused on autonomous vehicles scenarios. In those simulations the objective was to find critical scenarios for the autonomous cars. Nevertheless, the objective searched in differential analysis is different. Scenario generation for differential analysis refers to the comparison of two systems in variable scenario to find the situations where the differences in behavior are maximized. There is no research on scenario generation for differential analysis applied to autonomous cars. However there exists a study about differential analysis in the aircraft sector.

Lee et al. [24] developed a differential analysis to compare aircraft anti-collision systems. A new version had been developed for an anti-collision system and the purpose was to compare this new version with the previous version. What they wanted to check is whether the new version was able to suffer any possible accident that did not happen on the previous version. For that they drove two simulated systems with the objective of maximizing the differences between their outcomes. The process to compare the two systems formulated the test as a sequential decision process and optimized it with the use of reinforcement learning algorithms.

To define the problem, two instances of the simulator were created. One of the instances contained the new system while the second instance contained the base system. The rest of the instances were the same for both cases, including the environmental conditions.

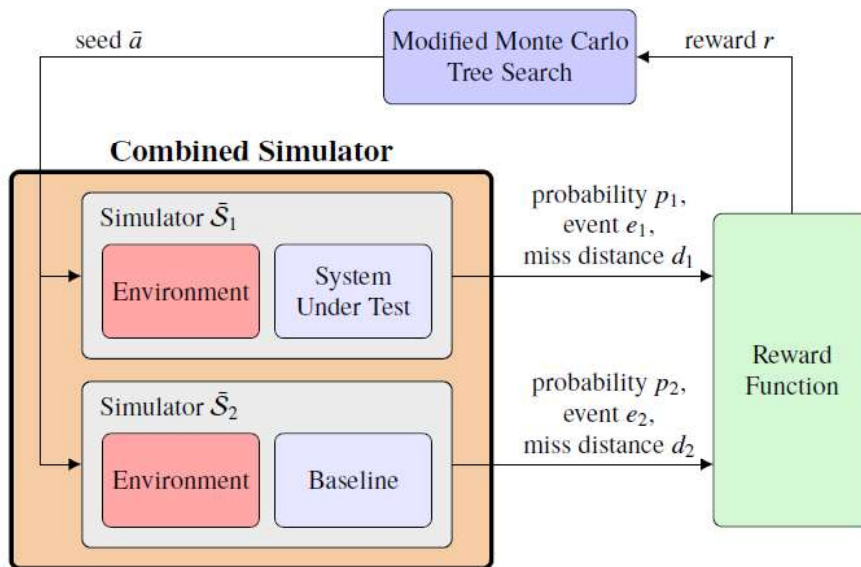


Figure 2.6: Differential analysis with Reinforcement Learning structure

The comparison between the two systems is made through two output variables. One of them is a variable that indicates whether a crash happened and the other one is the missed distance in case that a crash did not happen. These variables are included in the reward function of the reinforcement learner and will be used to guide the search into the scenarios with a greater difference between the systems behavior.

## 2.4 Optimization algorithms

As it was stated, the thesis objective is to find the specific parameters on a scenario that maximize the difference between two systems driving in the same scenario. A scenario is defined with some fixed parameters and some variable parameters. In order to obtain these specific values that maximize the difference some method or algorithm needs to be designed. This algorithm will be in charge of selecting different value combinations or parameters that define the scenario. The two targets will be to find the parameters that maximize the difference between the two systems and to have a method that does not consume a lot computationally.

In the problem stated there are no formulas that calculate the output based on the inputs. What needs to be analyzed is a black box problem where the simulation software receives some input values and after being executed it gives some output values. There is no function that gives the output values as a combination of input values. So approaches based on function derivations are not valid. Some interesting algorithms than can be applied to solve the problem were used on the scenario generation papers previously mentioned. Two of them are explained next. These methods are evolutionary algorithms and reinforcement learning algorithms.

### 2.4.1 Evolutionary algorithms

One interesting approach to solve the problem is by the use of evolutionary algorithms. Evolutionary algorithms include a group of several algorithms which are based on natural selection and biological evolution processes. Some of the evolutionary algorithms are listed next:

- Genetic algorithm
- Genetic programming
- Evolutionary programming



- Gene expression programming
- Evolution strategy
- Differential evolution
- Neuroevolution
- Learning classifier system

In these algorithms each set of values to be evaluated is called individual and a group of individuals is called population. The evolutionary algorithms work iteratively. In each iteration there is a set of possible solutions (or individuals). This set of individuals at a specific iteration is called generation. Then the individuals in the generation are evaluated. Based on each individual's result and depending on the specific algorithm selected, a new generation is created. There are five steps in an evolutionary algorithm, which are explained next:

- **Initial population:** The initial population is defined by the creation of random individuals.
- **Evaluation:** For each individual or set of solutions the objective value is computed. That means that the information in each individual is processed through some method or software. The information obtained from this process is what is called objective value.
- **Fitness Assignment:** Each individual's fitness value is calculated as a function of the objective values previously obtained. This function is called fitness function and it is crucial for the algorithm performance.
- **Selection.** Some individuals are chosen for reproduction based on their fitness value.
- **Reproduction.** This process involves the creation of new individuals for the new generation by means of crossover and mutation processes.

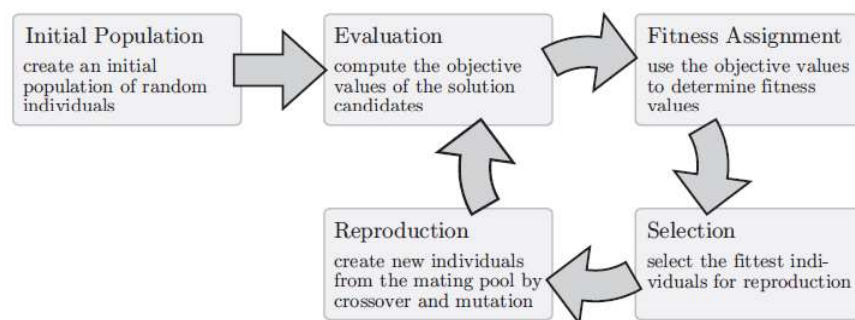


Figure 2.7: Evolutionary algorithms structure [25]

### 2.4.2 Genetic algorithm

Genetic algorithms are a subclass of evolutionary algorithms. They are used in optimization problems and like the other evolutionary algorithms they are based on the theory of natural selection and evolutionary biology. These algorithms are useful for searching through large and complex data sets. They are able to find reasonable solutions to complicated problems as they can solve unconstrained and constrained optimization issues. Genetic algorithms are usually applied to binary set of solutions where the individuals are sets of binary numbers.

The steps for all evolutionary algorithms were explained before (crossover, selection and mutation). However depending on the specific algorithm, the order of these steps applications is different. Besides, each algorithm has peculiarities concerning the selection, crossover and mutation processes. At the beginning of the algorithm an initial population is randomly created. After that the algorithm runs iteratively.

The selection, crossover and mutation operators for genetic algorithms are presented next [26]:

- Selection. This step is what makes it possible to focus on the solutions that are performing better and discard the solutions with bad results. To use the selection step, the fitness value for each individual at the current generation should be previously calculated. Based on this fitness value the selection operator chooses individuals with a probability proportional to their fitness.

$$Prob [b_i \text{ is selected}] = \frac{fitness(b_i)}{\sum_{j=1}^m fitness(b_j)} \quad (2.1)$$

This method is also known as the roulette wheel. That is because the graphical representation of this process looks like the roulette wheel game. The wheel is divided in as many sectors as the population size. One sector corresponds to each individual. The sector sizes are different and are bigger the higher the fitness value for that individual is.

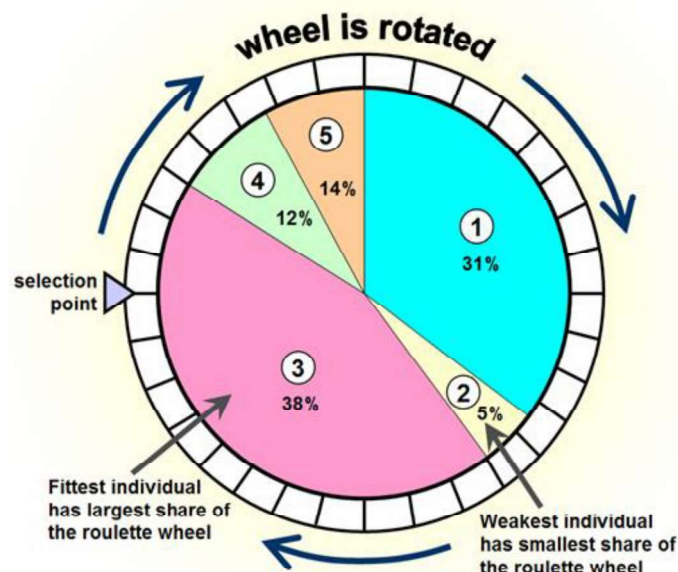


Figure 2.8: Roulette wheel selection process

- Crossover. In this step, new individuals (offspring) are created for the new generation as a combination of two individuals of the current generation (parents). The crossover can be one-point crossover, N-point crossover and uniform crossover. The one-point crossover is done by cutting the two strings that are the parents at a random position. Then the two tails are swapped. The N-point is like the one-point but instead of cutting the strings at one position, it cuts the strings at N different positions and swaps some parts of the string. Uniform crossover consists on going one by one through every position and at every position randomly choose to copy the number from one parent or the other.
- Mutation. This operator is in charge of changing randomly some bits in the individuals. This random change is done to improve the generation variety and explore more search space. The problem with a high mutation is that some good solutions can be lost from one generation to the next one.

Some of the problem that can be found in genetic algorithms is that the algorithm can find good solutions in one of the iterations but this solution can be altered through crossover and mutation and this solution would get lost. In order to avoid losing some of the best solutions found at a certain point in the optimization the concept of elitism is introduced [27]. Elitism takes some of the best values in a population and sends them to the new generation without alteration. With that the best solutions found at that moment are preserved.

### 2.4.3 Continuous genetic algorithm

The binary genetic algorithm is able to solve many different problems. The binary GA is the one in which each individual is formed by a string of binary values. Nevertheless in some problem the variables are real values. Transforming each real value into a binary number requires a lot of bits, depending on the number size and the decimals precision. Another option is to represent this real number with floating numbers [28]. This alternative needs less storage than the binary genetic algorithm as one single floating-point number contains the number information instead of several bits of integers. The continuous genetic algorithm can also be called real-valued genetic algorithm. The structure used for the continuous genetic algorithm is the same than for the common genetic algorithm. The main difference is that variables are represented by floating-point numbers. The rest of the algorithm works the same way.

The process is summarized next. As extracted from [29], at the beginning an initial population is randomly created. In continuous genetic algorithm each individual is an array of several real numbers. Each real number refers to a certain parameter. For the random initialization it is necessary to define the boundaries for these real numbers. That is made according to the problem definition. For example the first number on each individual could have a real value between 10 and 40. If the problem does not have a limitation for some parameter then it is recommended to define artificially a boundary around a promising area to search.

After random initialization for the first population, the rest of the steps are the same as for common genetic algorithm. The selection process will choose some individuals based on their fitness value. The crossover operator will create new individuals and the mutation will perform some random alterations to these new individuals. The mutation operator has also some value limitations, the same way as the initial population. In binary genetic algorithm the mutation operator will select some elements and change the value from 0 to 1 or from 1 to 0. In continuous genetic algorithm when the operator selects an element to alter, the new number can be constrained or not. Constraining the new number is the common option. That is made that way in order to search a new point around the current point, which is more probable to give good solutions. The constraints for the mutation operator can be done establishing a range around the current number in which the new random number needs to be. The constraint can also be set by choosing the new number with a normal function with a mean equal to the current value and some standard deviation.

As a summary, some of the characteristics and reasons to use genetic algorithms are the next:

- Genetic algorithm is an iterative optimization process

- It is based on three operators: crossover, selection and mutation
- Genetic algorithm work with probabilistic selection rules instead of deterministic selection rules.
- Its search does not proceed from a single point but from a population of points. So it can avoid getting stuck in a local optimum solution.

#### **2.4.4 Evolution strategies**

Evolution strategies is another algorithm in the group of evolutionary algorithms. It is also based on natural selection processes. This algorithm is a global optimization method and is widely used for real value solution spaces. It is commonly used for black box optimization processes. Those are the ones where no functional expression is given so the derivatives cannot be calculated.

The algorithm also uses the three mechanisms that are common for all evolutionary algorithms: recombination, mutation and selection. Nevertheless the use is different than for the genetic algorithms.

A summary of how evolution strategies work is explained next as extracted from [30]- [31]:

First the recombination (or crossover) operator is applied. It selects some parents and combines them to create new solutions. The number of new solutions is defined by the parameter  $\lambda$ . Then the new solutions are subjected to some changes through the mutation step and the fitness is calculated. After that some individuals are selected for the new generation. The number of selected individuals is defined by parameter  $\mu$ . This process is repeated through several iterations until the end condition is met. Termination condition is typically defined by reaching a fitness value or by limiting the number of iteration.

The three operators, recombination, mutation and selection, work very similar than for the case of the genetic algorithm. The main differences are in the selection process. For the genetic algorithm the selection process is done at the moment of selecting the parents for the crossover operation. What the selection did was to choose parents with a probability proportional to their fitness. In evolution strategies the parents are selected randomly for the crossover. Then the mutation process is applied the same way as for the genetic algorithm. After that is when the selection operator appears. It is in charge of selecting the best individuals and these ones are going to be the next generation. The selection process takes the candidate individuals for the

next generation ordered from best fitness value to worst. From this group of individuals it selects the  $\mu$  best individuals. Next the three operators are explained in detail.

Biologically recombination consists on mixing the genetic material of two parents. In evolution strategies recombination is used to combine the information on two or more individuals in order to create a new solution. Recombination operators for two parents are more common but it is also possible to use more than two. The number of parents which will be used on each recombination process is defined as  $\rho$ . There are two common recombination operators: dominant recombination and intermediate recombination. Dominant recombination combines the genes of all parents. As the individuals are arrays of real numbers, the recombination operator goes one by one through all the positions in the array. For each position it chooses one real number from that same position but from one of the parents. With this process it is expected that the real numbers on each position in the array that lead to better fitness values (good genes) are spread over the rest of the population. The other recombination operator is intermediate recombination. Here the new individual is created by making the average of each component for all parents.

Mutation is the second operator on evolution strategies. It randomly changes some number in the individuals in order to have more variety and explore more areas. In individuals which are formed by binary numbers, as in genetic algorithms, mutation consists on changing some of these numbers from 0 to 1 or the opposite. However when working with real value numbers, mutation is different. When changing some number the possibilities are too many, so what is usually made is to delimit the new random value. One option to delimit this value is to choose it by summing to the current value another number obtained from a normal distribution with 0 mean and  $\sigma$  standard deviation.

The selection operator is in charge of choosing the fittest individuals. Selection can be employed in two different ways: When selecting parents for the crossover (mating selection) or by selecting the best individuals out of a set of solutions (survival selection). Evolution strategies usually have survival selection. That is that it selects the  $\mu$  best individuals to form the new generation. The survival selection can be comma selection and plus selection. The plus selection selects the best individuals out of a group formed by the parents and the offspring in that iteration. On the other hand the comma selection selects the fittest individuals only out of the offspring individuals.

### 2.4.5 Reinforcement learning

Reinforcement learning is one of the three machine learning categories. Machine learning algorithms are usually divided into three groups: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning is a method that learns to classify data from a set of training examples. Once it has adjusted some parameters it can predict the results from new data. Some examples of supervised learning algorithms are linear regression and logistic regression.

The second group in machine learning is unsupervised learning. Here the algorithms receive some information and the objective is to classify these data points into groups or clusters. Some of the common unsupervised learning algorithms are k-means and anomaly detection.

The third category is reinforcement learning and it is the one interesting for the thesis. Reinforcement learning is defined as a problem where the agent learns behavior through trial and error interactions with a dynamic environment. Reinforcement Learning: A Survey. An example of reinforcement learning algorithm is Q-learning.

Some terms need to be defined to correctly understand how reinforcement learning works:

- Reward function: This function is in charge to value how good or bad a certain selected action in a specific state is.
- Action (A). Possible moves than an agent can make when it is at a certain state
- State (S). Specific situation or configuration where the agent is at a specific moment.
- Agent. It is the subject that takes actions.
- Environment. The world where the agent moves and interacts. The environment receives the state and the action of the agent and it returns a reward.
- Policy. The policy is responsible to select an action at each state.
- Value. It is the expected long-term return of current state under current policy.

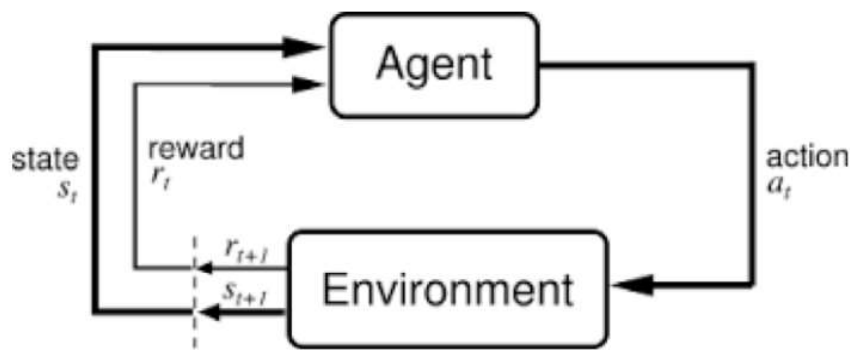


Figure 2.9: Reinforcement Learning structure

To sum up in reinforcement learning an agent goes through different steps. On each step the agent receives an input and the current state ( $s$ ) of the environment. After that the agent chooses an action ( $a$ ) that will generate an output. This action changes the environment state and the value (or reward) of the state transition is sent to the agent. The agent's policy should choose the actions that will lead in the long run to maximize the sum of rewards [33].

Reinforcement learning has many differences compared to supervised learning problems. In reinforcement learning there is no data provided with some inputs and its outputs. Instead after choosing an action it receives a reward but does not know which action would lead to a maximum reward in the long run. For that the agent needs to collect data about actions, states, transitions and rewards.

Reinforcement learning is often modeled as a Markov Decision Process (MDP). MDP is a mathematical framework used to model decision making in situations where outcomes are partly random and partly under the control of a decision maker [34].

One main difference between reinforcement learning and supervised learning is that the first one needs to explore the environment, whereas supervised learning algorithms are purely exploitative. Reinforcement learning algorithms work with the balance of exploration and exploitation. Exploration is based on trying different things and comparing them to the ones before to check if they are better or not. Exploitation is using the things that have worked better in the previous steps.

The possible applications of reinforcement learning algorithm nowadays include a wide variety of options. One of the applications is for resource management in computers [35]. In this case the algorithm needs to allocate limited resources to several tasks. Another current application is to control traffic lights in the cities [36]. The reinforcement learner was used to solve the traffic



congestion problem that many big cities suffer. Through this technique the coordination between several traffic lights can be done more efficiently. Reinforcement learning can also be applied to robotics [37] or to the chemistry industry, where this algorithm was used to optimize chemical reactions [38].

## 2.5 Summary about the State of the Art

The topics covered in the chapter are divided into three groups: an introduction to autonomous vehicles validation, a discussion about scenario generation papers and an overview about some optimization algorithms.

On the first part, the current methods to ensure that an autonomous vehicle is safe were discussed. As a car can produce severe accidents and the autonomous software will have a lot of responsibility, current effort is put into creating validation methods that can check whether an autonomous vehicle is safe or not. After the vehicles are validated they should be allowed to be sold to the public and drive next to other thousands of vehicles. The tests to check driving assistance systems and autonomous vehicles are divided into physical tests and simulations. The physical tests are carried out by driving a real car in the roads. As the possible cases that a car may find in the road are huge, the requirements for validation include a very big number of kilometers driven with the autonomous vehicle. In addition, physical test are quite expensive to elaborate.

Another option for testing that can complement the physical tests is simulation testing. In these cases the systems are tested through virtual environments. If it is only one part to test, the test can be formed by the real part and a simulated environment. In other occasions the vehicle dynamics is virtually modeled and the whole test is carried out in a computer. Some vehicle simulators are able to reproduce a real car behavior very closely.

After the introduction to validation, some of the latest research on scenario generation was analyzed. These studies were divided into two groups depending on the way of selecting the test cases. Coverage and falsification approaches are the two ways of making decisions for scenario generation. Coverage methods are based on searching a lot of random points in the scenario variables search space. In falsification approaches the objective is to use the information from the previously analyzed scenarios cases to choose the next test cases.

The scenario generation studies have important differences between them. That is because the objectives, simulation tools, optimization techniques and scenarios to test differ from one paper to another. Some of these differences are summarized.

The general purpose in scenario generation is to develop a method that can choose different test cases in certain scenarios to check some system functionality. But the specific objective set for the scenario testing was not always the same. Some of the scenarios are focused on testing an autonomous vehicle by finding critical situations. Other studies are focused on calculating the drivable area of the ego-vehicle in different traffic situations. In addition, some studies tested a specific driving assistance system whereas in other cases the whole autonomous vehicles were tested.

Concerning the vehicle modelling, in some papers a mathematical model of the vehicle dynamics was developed. On the other hand other studies used already developed vehicle simulation software. Besides, the scenario options analyzed have a lot of variety. A common option is a vehicle performing a braking maneuver to avoid a collision against an obstacle or another vehicle. Highways are a common scenario where ACC system can be tested as well as lane change maneuvers. Another scenario that was used is a crosswalk on a road. In most cases the road geometry was kept constant for all the test cases but there is also a case for automatic generation of roads.

The algorithm that selects the variable values is also different. Some of the options include reinforcement learning and evolutionary algorithms. In other papers specific toolboxes are used that use their own optimization method. The random selection of values is also a valid option.

Another interesting study was focused on differential analysis. In this case the scenario to test was not formed by an autonomous vehicle but by an aircraft. The test wanted to compare a current version of an anti-collision system with a base system. With this comparison, the purpose was to find some cases in which the new version suffered an accident but not the previous version.

From these papers, important knowledge could be gathered to solve the problem stated in the thesis. The optimization methods will be discussed to find which one is more suitable. Besides, the method developed needs to be generic. So it can work in many scenarios. After the method implementation at least one scenario needs to be chosen to check that the method is working. Some of the scenarios present in the papers will probably be chosen for the test.

The last part of the chapter summarizes some of the optimization methods that can be used to solve the problem of selecting and optimizing the variables values for a scenario. The algorithms explained were evolutionary algorithm and reinforcement learning algorithms. The algorithms evaluation and selection process are explained in the methodology chapter.



### 3 Methodology

Some papers were presented which contained methods for scenario generation. That previous work should be a base to decide how to solve the problem stated in the thesis. Given a driving scenario with some fixed parameters and some variable parameters, the thesis's objective was to find the specific variable parameters values which maximize the difference between two systems (two cars) when they drive at that scenario. The scenario definition, car specifications and driving simulation was done with the help of simulation software. After the problem statement some decisions needed to be taken about how to develop this problem. These decisions included scenario definition, method used for optimization and the configuration that needed to be done for this method to work correctly.

The scenario definition means choosing interesting scenarios where the simulation takes place and selecting the fixed and the variable parameters. Then a method needed to be chosen for the maximization problem. The method was in charge of selecting the variable parameters values to define the road scenario that was sent to the simulation software to carry out the simulation. This method needed to select the parameters with the purpose of finding the optimal results. Once the algorithm was selected, the next step was to configure this algorithm and to define the remaining parameters for the problem.

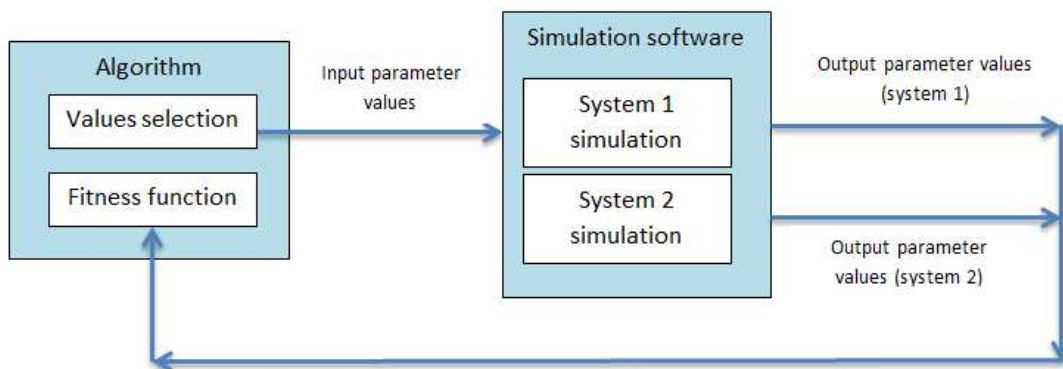


Figure 3.1: Problem structure

Figure 3.1 shows the general problem structure. This chapter is first focused on scenario generation and algorithm selection. After that all decisions concerning the algorithm definition are explained. Finally the implementation is carried out. In the implementation step, the simulation software is explained and all the code needed for the algorithm and simulation control is written. With the implementation finished the method is ready to work.

### 3.1 Process for Scenario definition

In simulations, scenarios represent the roads and traffic situations that a vehicle can find when it is driving. Several features need to be defined to create a scenario. The first part is formed by the physical road. The parameters to define the road give information about the road shape and size. These parameters are the road geometry, the lane width and the number of lanes. Other parameters can include the pavement friction and the height of each stretch of the road. Road definition also includes driving restrictions such as speed limits, traffic lights as well as driving directions on each lane. The second part on scenario definition is related to the participants on the road. These participants are the vehicle to simulate as well as other cars and trucks that are also driving. Participants can also be bicycles and pedestrians for the case of urban scenarios. Finally a scenario is not complete until the maneuvers are defined. The maneuvers indicate what is going to be the behavior of the ego-vehicle during the simulation. The maneuver commands will tell the car the changes concerning acceleration, braking and steering. The maneuver definitions are also applied to other traffic vehicles, not only the ego-vehicle.

One example of a simulation scenario is shown in Figure 3.2. This scenario is formed by a highway with three lanes. The ego-vehicle is the yellow vehicle in the central lane and a truck drives in the right lane. To complete the scenario definition it would be necessary to define the maneuvers for both vehicles.

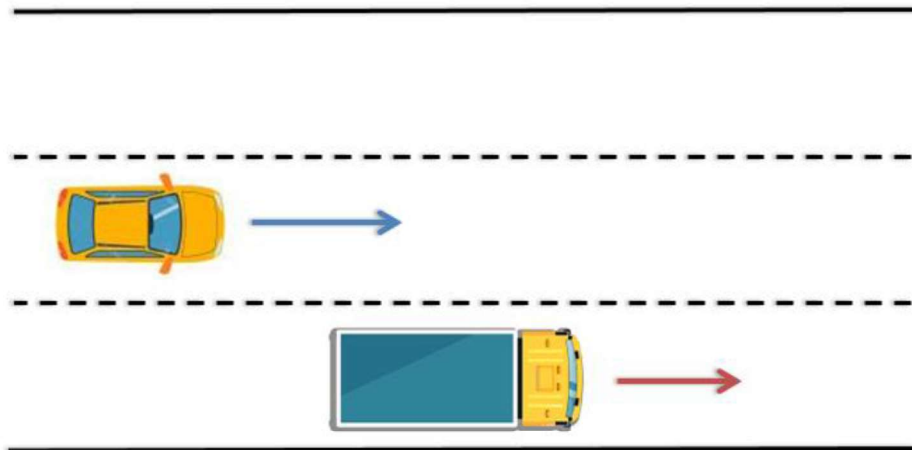


Figure 3.2: Driving scenario example

After defining how a simulation scenario is defined, the focus is on deciding what kinds of scenarios are interesting to analyze for the master thesis case. The main idea is to have a scenario with some fixed parameters and some variable parameters. Given this scenario the objective is to change the variable parameters in order to maximize some criterion. In most scenario generation papers the objective was to find the most critical situations. However, for this case the developed algorithm changes the variable parameters to maximize the difference between two cars (two systems) driving on the same scenario.

After this, it is necessary to decide which scenarios are useful for the simulations and which parameters are going to be fixed and which are going to be variable. Interesting scenarios for simulation need to have sudden changes on other traffic participants or difficult situations for the ego-vehicle. If the scenario is too simple, the simulation is not going to provide new information. For example if the car is driving alone on a straight highway, it does not need to make any maneuver. More complex scenarios are created instead so the ego-vehicle needs to make adjustments in its speed and direction. Some scenarios that are used for vehicle testing were analyzed in the state of the art chapter. This summary is shown in Table 2.1. Some of the typical test scenarios are roads with two lanes. Sometimes the driving direction of the two lanes is the same and other times it is the opposite. About the maneuver it is common that they involve changing lanes. One of this lane change maneuvers is an overtake maneuver. Another option is a lane change of the ego-vehicle into a lane where another vehicle is already driving. A cut in scenario can also be tested where a vehicle changes lane and positions itself in front of the ego-vehicle. Other common scenarios include braking maneuvers produced because the vehicle

finds other vehicles, pedestrians or obstacles in the middle of the road. Active cruise control systems are commonly tested with scenario generation methodology. In these cases a vehicle drives on the road with variable speed and the ego-vehicle adapts its speed to the speed of a preceding vehicle.

The scenarios are divided in three levels depending on the abstraction: functional, logical and concrete scenarios [39]. Functional scenarios include a general description of the elements that compose it. By logical scenarios it is understood the scenarios in which the range for its parameters is defined. Finally a concrete scenario includes the exact values that all of its parameters take. When testing autonomous vehicles, the common procedure is to set a logical scenario for the study. This scenario has some fixed parameters and some variables that can have values between a specific range. Then, for each test case, a set of values is chosen for the variables. Therefore, each test case is formed by a concrete scenario.

After choosing the scenario, some of the parameters are set as fixed parameters and others as variable parameters. These variable parameters are the ones that the algorithm is allowed to change. The scenario geometry is in most cases fixed for the scenario generation papers. An alternative is explained by Kim et al. [13], where a method generates automatically a road network. However, other parameters which are involved with the vehicles movement and maneuver may have more impact into having different vehicle behavior. So the road geometry parameters are set as fixed parameters. The variable parameters are usually the initial traffic situation and the maneuvers. The initial traffic situation means the position and speed of every vehicle at the beginning of the simulation. The parameters related to maneuvers control the changes in speed and position of the vehicles during the simulation as well as the timing of these maneuvers. For example, a scenario could be defined as a straight road that is 500 meter long and has three lanes. These are considered as part of the geometry so they do not change from one simulation to another. The scenario could have another vehicle apart from the ego-vehicle. The variable parameters could be the two vehicles initial position and speed. Besides if the ego-vehicle performs a maneuver in the simulation, a variable parameter could be the exact moment when this maneuver happens or other parameters that control the maneuver characteristics.



## 3.2 Algorithm selection

The thesis objective is the maximization of the difference between two systems driving at the same road scenario. The road is defined in a simulation software with some fixed parameters and some variable parameters. The purpose is to find which of these variable parameters make a bigger difference between the behavior of one system and the other. As the simulation software works like a black box, the procedure is to send different combinations of these variable parameters to the simulation software to find a good solution. The algorithm selection is concerned with finding a suitable method to select these variable parameters to send to the simulator. The perfect method maximizes the probability of finding good solutions in a scenario.

Scenario generation methods can be divided in two different groups according to the way of approaching the problem. These two groups are coverage and falsification. These two groups include different methods that can be used to choose the parameters that are sent to the simulation software. After defining a scenario to analyze with some fixed parameters and variable parameters, the algorithms select the variable parameters for each simulation. These two approaches were analyzed on the state of the art chapter and their main characteristics are summarized next:

The coverage approach is focused on choosing many points homogeneously distributed around the search space. For the scenario case, the variable parameters are constrained between some values. The search space will have as many dimensions as number of parameters and there will be a maximum number of points to analyze. Then a coverage method will divide this search space in as many areas as maximum number of points and these areas are homogeneously distributed. After this division of the search space into different areas the method will choose one point from each area. All the selected points will be sent to the simulation software.

The other approach is falsification. In the coverage approach all points are selected before starting the simulations. On the contrary the falsification approach chooses the points based on the information obtained from previously simulated points. The process is divided in several iterations. On each generation this method uses the information from the previous simulation results to select the new iteration points. The advantage of this method is that it focuses more in the most promising areas and barely selects points from areas with bad results. The inconvenient of this method is the possibility of getting stuck in some local optima.

Both methods can have some advantages and some problems. Falsification approaches have the advantage of focusing the search around the best values found at the previous evaluations. With

that they avoid losing time searching in areas not interesting and allow for a better search on promising areas. On the other hand falsification approaches get easily stuck on local optima and need more complex algorithms compared to coverage approaches. Based on this evaluation, the decision is to develop some falsification method with certain complexity but that could lead to better results. After optimizing a scenario with a falsification method, some coverage method was also used to compare the results performance. The coverage methods are much easier so it did not take so long to configure and could alert from problems when falsification methods get stuck in local maximums.

*Table 3.1: Three methods evaluation for scenario generation case*

	Coverage	Falsification
Complexity	Low	High
Problems	Dispersity at the search	Can get stuck in a local maximum

#### 3.2.1 Falsification algorithm

The first objective is developing some falsification approach to make the differential analysis between two systems. The algorithm selects parameters to send to the simulation software but it sees the simulation software as a black box. Some promising approaches that use the falsification approach and could be used for black box optimization problems are evolutionary algorithms and reinforcement learning algorithms. These algorithm were also applied in some papers related to scenario generation commented on the state of the art section. So it is clear that they could be useful for this kind of applications.

The suitability of each kind of algorithm depends on what kind of scenarios are going to be analyzed. Reinforcement learning is suitable for scenarios where some of the subjects, for example another vehicle, need to make several decisions during the simulation time. In this case the reinforcement learner controls the steering and the acceleration and braking of the other vehicle. The reinforcement learner tries different combination of accelerating and steering maneuver and it searches for the situations with better rewards (or fitness values). On the other hand when the scenario has parameters that are going to have fix values during each simulation, it does not make sense to use the reinforcement learning approach. Instead an evolutionary algorithm can

be in charge of selecting the parameters. For example if the initial and constant speed of a vehicle is a parameter, then it can be described with one value for the whole length of that specific simulation. In terms of complexity the reinforcement learning is more complex and requires more computational effort to run than evolutionary algorithms.

Therefore, if the parameters to optimize in the scenario are a sequential process a reinforcement learning algorithm looks like a promising approach. For example if the variable parameter is the steering wheel position along one simulation and the learner can make several changes of the steering wheel position. The different positions of the steering wheel are the sequential process. An evolutionary algorithm can also be used for sequential decisions processes if the number of decisions is not very high or if the reinforcement learning looks too complex to develop.

On the other hand, if the parameters do not change as a sequential process a better approach is the use of evolutionary algorithms. That is when the parameters have a value that is going to be constant in the simulation. For example one parameter is the constant speed of one of the traffic vehicles. This speed does not change during one simulation so it is not a sequential decision process.

The final decision is to use evolutionary algorithms because a lot of scenarios have many variable parameters which keep a constant value during each simulation. These parameters are mainly values for different positions and velocities for the vehicles in the scenario. Evolutionary algorithm is a category that includes several algorithms that have a common structure. The specific evolutionary algorithms that are adopted are the genetic algorithm and evolution strategies.

### **3.2.2 Coverage algorithm**

This kind of algorithms selects points distributed around all the search space. So for the differentiation process a coverage algorithm would choose first all the points to simulate in the software and after the selection of all the points the simulation starts. So the algorithm does not receive feedback on how the points are performing. These approaches are not iterative. From the point of view of the exploration-exploitation trade-off they are full exploration.

The simplest algorithm is to select random values for the input parameters. This method is called Monte Carlo method. Given a problem with three input parameters constrained between some maximum and minimum values and with 500 points to simulate on the simulation software, the algorithm would select 500 points. Each of these points would have the 3 input

parameters chosen randomly between the correspondent maximum and minimum values for each parameter.

Monte Carlo method is a suitable option for the optimization problem. Nevertheless, other coverage approaches try to cover the parameter space more homogeneously. That is the case of Latin Hypercube Sampling (LHS) algorithm [40]. This algorithm divides the search space in rows and columns and it takes points so that two points do not coincide in the same row and column. For example in a 2D search space with 100 points, Latin Hypercube algorithm would divide the space into 100 rows and 100 columns. If the first point is on row number three and column number four, then the rest of the points can be neither in row three nor in column four.

Another possible algorithm is orthogonal sampling. It works the same way as the Latin Hypercube with the addition of a new element. Apart from rows and columns, it also divides the search space into cells. So when looking for 100 points, this algorithm would divide the search space into 100 rows, 100 columns and 100 cells. And only one point can be selected in each row, each column and each cell.

### 3.3 Fitness function

One important element in evolutionary algorithms is the fitness function. The fitness function is in charge of giving one value that explains how good or bad a certain solution is. It needs the results obtained at the simulation of a certain set of solutions (or individual) and transforms this output information into a scalar value. The evolutionary algorithm calculates the fitness value of each individual and in each generation. The selection step needs these values in order to compute the next generation. Given a generation where each of its individuals has already been simulated, the selection operator chooses the individuals with a higher fitness value and uses them for the following generations. Besides, the fitness value at the end of the simulation indicates the best solutions found in that simulation. This value is necessary to analyze the algorithm quality. After trying different parameters to calibrate an evolutionary algorithm the decision on the algorithm performance is based on the best solutions found. That is the solutions with higher fitness value. In addition the fitness value can also be used to compare the performance of different algorithms. Therefore the fitness function is of critical importance for the algorithm's performance. A well-defined fitness function improves the solution search and makes it possible to find the desired result with fewer generations.

The fitness function is different for each problem and depends on the specific requirements. In the case of this thesis the objective that is searched concerns the difference maximization between two systems driving at the same scenario. For each individual the simulation software receives some inputs about the scenario characteristics. After the simulation, the software provides the algorithm two sets of outputs. One set of outputs for each system. The output values contain information from the car gathered during the simulation. This information can include physic parameters like position, speed and acceleration but also information about sensors. In order to make the explanation in this subchapter easier two output parameters are chosen: longitudinal position and speed.

The method used for the fitness function needs to transform the output variables information of two systems into a single scalar value that expresses how different these two systems behave. Some of the possible methods to obtain this value are common methods used for statistics. They include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE). These methods mentioned are commonly used to compare estimations and real results. They receive a set of points and for each point a real value and a predicted value is included. Then the mathematical formulas of these methods are applied to obtain one value that expresses how big the error of the whole set of points is. For the case of the thesis instead of comparing estimations with predictions, the comparison will be between the results on system one and system two. That is between the different positions and velocities obtained after simulating the two systems.

On the following these methods are analyzed to see if they are suitable for the fitness function and which one can perform in a better way.

Mean Squared Error (MSE) measures the average of the square of errors. At each point it takes the difference between the two systems (A and B) and elevates it to the power of two. As the difference is squared this method gives special importance to the values that are very different. If the difference is duplicated then the MSE value is multiplied by four.

$$MSE = \frac{1}{n} \sum_{i=1}^n (A_i - B_i)^2 \quad (3.1)$$

Root Mean Squared Error is very similar to MSE. The only difference is that after calculating the value, it makes the square root of this value. So if we want to make a ranking with several individuals and a fitness value assigned to each individual, the order is going to be the same

when using MSE and RMSE. The difference would be that the fitness value for RMSE is in the order of magnitude of the position and speed differences.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (A_i - B_i)^2} \quad (3.2)$$

A third possible method is the Mean Absolute Error (MAE). This metric measures the error as a difference in absolute value for all points and makes the average. Whereas MSE and RMSE penalized more the big errors in specific points (because it adds the square value of the error), MAE penalizes all differences the same way. So MAE should be used when all the errors need to be penalized the same way and RMSE in the case where it is preferred to give more importance to big errors compared to small errors.

$$MAE = \frac{1}{n} \sum_{i=1}^n |A_i - B_i| \quad (3.3)$$

Another option to measure differences is with the use of Mean Average Percentage Error (MAPE). This method is similar to MAE. It calculates the absolute value of the difference at each point, but after that it divides this difference between the first value. By doing that, the result obtained is the error as a percentage of the point value.

$$MAPE = \frac{100\%}{n} \sum_{i=1}^n \left| \frac{A_i - B_i}{A_i} \right| \quad (3.4)$$

To better understand the difference of MAPE a small example is introduced. Supposed that two points in the simulation give the following results and the objective is to compare the speed differences between two vehicles:

Point 1: Speed of vehicle 1: 10km/h. Speed difference between vehicle 1 and 2: 5km/h

Point 2: Speed of vehicle 1: 50km/h. Speed difference between vehicle 1 and 2: 5km/h

At point 1: MAPE = 0.5

At point 2: MAPE = 0.1

So having 2 points with the same difference (5km/h), RMSE and MAE would give the same error to point 1 and to point 2 because they only take into account the speed difference. On the other hand MAPE gives different results because it takes into account the percentage of that difference compared to the speed value.

Another advantage of MAPE is that its value is a percentage error so several magnitudes can be calculated together and expressed as a scalar value. For example calculating differences between position, speed and acceleration and having a unique fitness value that expresses how much the difference is. With RMSE and MAE that would be more complicated because position and speed have different measure units so summing into one final fitness value them is somehow arbitrary.

Let's say that the position difference and velocity difference wants to be calculated and expressed into a single fitness value. If the speed difference is 10km/h and the position difference is 60m (using MAE), then the fitness value could be calculated by summing this two quantities. Fitness value = 60 + 10 = 70. But the value of 70 wouldn't mean anything as it is a sum between meters and kilometers per hour. And why not summing speed difference in kilometers per hour and position difference in kilometers instead of meters.

MAPE solves this problem. There is no problem of summing different units as all the differences are stated as percentage values. Besides, the final value is the mean of percentages errors so some information can be obtained just by looking at the number. If for example the fitness obtained is 5 using MAPE, then we can know that the average between the differences at the points observed is 5%. So the position and speed differences between the two systems will be 5% as average for the every point.

If another method such as RMSE or MAE was used, a final fitness value would be difficult to interpret. If the fitness value is 5, we couldn't intuit what that means as it is a sum between different magnitudes (position and speed).

Based on the previous analysis the most suitable method to use for the fitness function is the Mean Average Percentage Error (MAPE). The evolutionary algorithm code has one function where it calculated the fitness value of a specific solution. The algorithm receives the results obtained in the simulator with a specific solution and applies the MAPE formula to the speed and positions values measured in the simulation for the two systems. With this formula it obtains one scalar value. This scalar value is the fitness value for that specific solution. The fitness value is needed for the selection step on the evolutionary algorithms. The fitness value gives

also information about how good the final solution is. Therefore, it is used to compare several algorithms performance. It can also serve as a comparison between different variations of the same algorithm.

*Table 3.2: Possibilities for Error estimation to use on fitness function*

	MSE	RMSE	MAE	MAPE
Error measurement	Quadratic to the difference	Quadratic to the difference	Proportional to the difference	As a percentage of the difference
Units of measurement	With units (km/h or m)	With units (km/h or m)	With units (km/h or m)	No units as it is a percentage

### 3.4 Evolutionary algorithms configuration

Once the evolutionary algorithm has been selected and its code developed, it still remains an important step. This was the parameter selection for the algorithms. Parameter selection consists on the customization of an algorithm with the objective of obtaining a better performance. These algorithm parameters can be divided into two groups: structural parameters and numerical parameters [41]. Structural parameters are the main factors that affect the algorithm performance. The code needs to be changed in order to alter these parameters. These parameters include the coding scheme, operator types and stopping criterion. On the other hand numerical parameters are formed by the population size at each generation, the number of generations, the mutation rate and the crossover rate. Once the code is finished these numerical parameters are easily changed. Some other parameters can also be adjusted depending on the specific algorithm used. An evolutionary algorithm with the right parameters can give results orders of magnitude better than the same algorithm with bad parameters definition [42]. Some quality measurement method needs to be defined in order to evaluate how good or bad certain solution is. The solution quality depends on the parameters selected for the algorithm. The other part that needs to be defined and which is related to the algorithm quality is the end condition.



### 3.4.1 Algorithm quality

The next step is to define how to measure an evolutionary algorithm quality. The two common ways to measure an evolutionary algorithm performance are solution quality and algorithm speed [42]. For solution quality method a specific computational limit is established. This limit can be for the project case setting a maximum number of simulations in the simulation software (for example one thousand simulations). After this limit has been reached the best solutions of each algorithm are compared. This is done by using the fitness value obtained. With this information the best algorithms will be the ones that obtain a better fitness value. The second method to measure the algorithm's performance is called algorithm speed. This procedure consists on stopping the algorithm only when a specified fitness value has been reached. Then the best algorithms are the ones that require less time to reach this value. The problem with that is that it is difficult to estimate how long it will take for the algorithm to reach that certain value. So the algorithm can be running for days without reaching that value.

### 3.4.2 End condition

The condition for the algorithm's end is defined depending on the performance method. There are three possible options for the end condition. The first one is by limiting the computational effort. That is by establishing a maximum number of simulations. The advantage is that the total computation time is limited but the problem is that the algorithm can stop before reaching a solution good enough. This maximum number of simulations for the evolutionary algorithms is the result of the population size multiplied by the number of generations. This end condition allows to compare several algorithms by comparing the final fitness value. That is by measuring the performance as solution quality.

The second end condition is to set a certain objective value that we want to obtain. Then the algorithm will run until this value is reached. This method is more suitable for the cases where it is known before the value that is needed. The problem with that is that it is difficult to estimate how long it will take for the algorithm to reach that certain value. So the algorithm can be running for days without reaching that value. The algorithm performance would be measured by analyzing the computation time needed to reach that value.

A third option for final condition definition is to compare the results in a generation with the results on the generation before and finishing when this difference is smaller than certain threshold.

Table 3.3: End condition and Algorithm Quality

	End condition	Algorithm Quality measurement
Option 1	Limited number of simulations (computational limit)	Compare final fitness value
Option 2	Reach certain fitness value	Compare number of simulations needed to reach this value (System speed)
Option 3	Improvement between generations is smaller than a threshold	Check both final fitness value and num- ber of simulations needed

From these three options to define the end condition, the one with the limited number of computations looks better as it is easy to predict the total simulation time and it allows for an easy comparison between different algorithms efficiency. The problem with the second option is that the fitness value that is going to be obtained is not known, not even its order of magnitude. So if the fitness value set as end condition is too high, the algorithm will never reach this point. And the disadvantage of option 3 is that it can make the algorithm stop too early or too late. If the population size is too large, the algorithm needs a lot of generations to converge so the computational load can be huge. On the other hand, an evolutionary algorithm can have a constant fitness for a few generations and after those continue to improve the solution. But with the 3<sup>rd</sup> end condition the algorithm would stop in these few generations with stagnation, even if they are between the first generations. So the simulation would be very short.

As explained before, by establishing a number of computations limitation, the parameters of population size and number of generations depend on the value of each other. That is because the maximum number of computations is calculated as the population size multiplied by the number of generations. But it is still necessary to decide whether to have a bigger population size or a greater number of generations.

### 3.4.3 Influence of the evolutionary algorithm parameters

Different genetic algorithm problems can need very different parameters values for a successful performance. The parameters that configure a genetic algorithm are population size, number of generations, mutation rate and crossover rate. It is not clear which specific parameter values should be used for a random genetic algorithm problem [43]. The procedure to choose the best parameters is complicated. One way to do it is by conventions [42]. But this method is very inaccurate. It can serve as estimation on how to start but will not lead to the best results. A better option is trying to understand the interaction between these parameters before choosing the values.

This interaction is influenced by what is called the exploration-exploitation trade-off. To understand this exploration-exploitation trade-off first both terms are defined. Basically exploration is concerned with trying a lot of different values and covering a big search area. On the other hand exploitation is defined as deepen the search around some specific values or region in the search space.

In general when changing some of the evolutionary algorithm parameters we can improve one of these two terms (exploration or exploitation) but decrease the other one. Having a huge exploration would mean an algorithm with a single generation but a lot of points to search on that generation. The problem with that is that the algorithm cannot focus on the most promising areas to search for better values. Each new point to evaluate is randomly selected. On the other hand an algorithm which maximizes exploitation but has minimum exploration would have only one value per generation and will always look other points around that one. This will probably lead to a point stuck in a local optimum being unable to reach the global optimum.

How does each parameter affect to the exploration and exploitation? First of all the population size and the number of generations are analyzed. These two parameters need to be explained at the same time because they are dependent of each other. That is because the evolutionary algorithm end condition was set as the algorithm reaching a maximum number of simulations previously defined. The population size and the number of generations will be set so that the product between these two parameters is equal to this maximum number of simulations. So an increase in one of these two parameters means a decrease in the other parameter. Population size is linked to exploration. As a reminder in evolutionary algorithms the population size is the number of individuals (or solutions) which are computed at each generation. So the larger this number is, the more search space is covered in one generation. On the other hand the number of generations is closely related to exploitation. In an algorithm with a large number of generations,

it deepens the search around some specific areas. That happens because the evolutionary algorithm looks for new solutions near the points that had previously given good fitness values. As a summary a high population size will lead to a big search space whereas a high number of generations will conduce to focus the search on specific areas. The equilibrium between these two variables is not so easy to find and it depends on the problem. In a problem with a lot of local optima, a greater population size (exploration) may help to find a better solution. However in a problem with few local optima an approach with more generations (exploitation) will have better expectations.

The other factors that influence this exploration-exploitation trade-off are the mutation rate and the crossover rate. The mutation rate states the probability that some element inside an individual is randomly changed to a new one. These random changes are responsible to improve the variability so that the algorithm can search new points around the current solution. Therefore, the bigger the mutation rate is, the greater exploration the algorithm has. The problem with a high mutation rate is that it can eliminate current good solutions reached after several generations. Then, a big mutation rate means low exploitation.

The last parameter is crossover rate. This parameter indicates the probability that an offspring (or new individual) is created as a combination between two parents (two individuals from the previous generation). The other possibility for a new offspring is that the new individual is a copy of only one parent. An algorithm with 0% crossover rate would be pure exploitation. It would take the solutions from the first generation and never combine them. The algorithm would only make mutations from each individual solution. On the other hand a 100% crossover rate means that every new offspring is a combination of two parent individuals. So a low crossover rate produces high exploitation. But a high crossover rate does not mean high exploration as the new individuals contain information from the previous solutions. Therefore, high crossover rate produces equilibrium between exploration and exploitation. Crossover rate has usually high values but never reaches 100%. That is done to bring some of the best solutions from one generation to the next one without modification. It is called elitism. So with a crossover rate around 70-80% most of the new individuals would be combinations of two parents. And the rest of the new individuals would be a copy of some of the parents which had a high fitness value.

#### **3.4.4 State of the Art for the selection of parameter values**

Once this interaction is understood, some research is analyzed where different values are tried to find the best parameters combination. Deb and Agrawal [43] try different GA parameters on

typical functions to analyze which parameters lead to better results. They set a maximum number of function evaluations and change the three parameters stated before: population size, crossover rate and mutation rate. The researchers test the parameters effect on different functions. Some of these functions were unimodal, four-peaked and multimodal functions. In unimodal function the maximum number of function evaluations is set to 500 and the best performance is found for a population size close to 10. That value is quite small compared to the number of generations as the maximum number of generations was  $500/10 = 50$ . For the analysis on the four-peaked function the computational load is increased from 500 to 9000. The results show that the optimal population size has a value between 100 and 200. For this case the population size and the number of generations have values in the same order of magnitude. The last case is the one about multimodal functions. Because of the complexity of this function the computational load is increased from nine thousand to forty five thousand. For this function the population size needed to reach good results was much greater than for the case of other functions. The best performance is obtained for a population size of about one thousand.

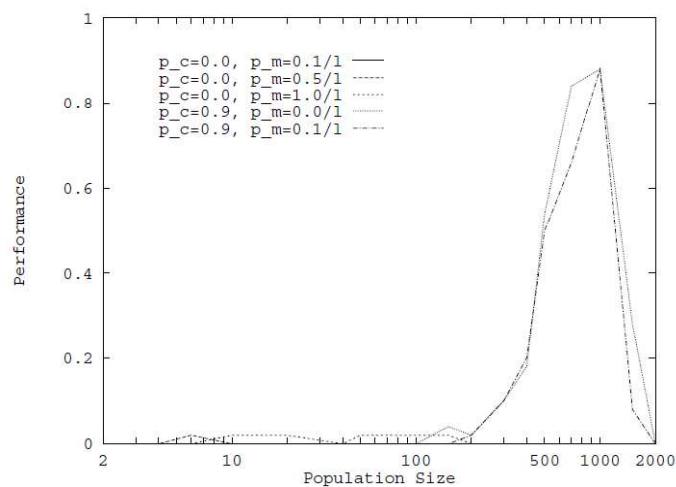


Figure 3.3: Genetic algorithm performance for different parameters [43]

In this case the population size is much bigger as the number of generations, which has a value about 50. In addition, it is observed that in multimodal functions the crossover operator is important for a good result. To sum up, the functions which have more peaks or local optima need a higher number of function evaluations to find the optimal solution. Besides, the population size compared to the number of generations increases when the function has more peaks.

Other research is focused about the optimal population size [44]. On this paper there is not a limit of computations as it was established for this case. Instead the algorithm runs until it con-

verges. The results obtained inform about how the solution improves with the population size until it reaches a point where the population is too big and the convergence is quite difficult.

When analyzing the impact of these parameters, Boyabatli and Sabuncuoglu [41] conclude that the effect of crossover rate is insignificant for their studied case. Besides they also analyze the impact of mutation rate and state that a mutation probability of 0.4 is the most convenient value for their case of study.

The parameters selected for an evolutionary algorithm can be constant during the optimization process but can also vary. This parameter change is called parameter control [42]. For some advanced problems the use of variable parameters can be a promising option. Laoufi et al. [45] change the crossover and mutation rate depending on the fitness value that the current points have. High values of mutation and crossover rate mean higher exploration and less exploitation. What the adaptive algorithm does is to increase the value of these parameters when the population is close to getting stuck in a local optimum and reduce their values when the population is too scattered. But the development of a control strategy for these parameters is quite complex.

Some papers mention typical values for the evolutionary algorithms. Some research suggests a crossover rate between 0.5 and 0.7, a population size of 100 and a mutation rate between 0.01 – 0.1 [42]. Other possible values are a crossover rate between 0.65 and 1 and a mutation rate =  $1/n$  [41].

For the case of study the process starts with typical parameter values. Several simulations runs are carried out changing some of the algorithm parameters from one run to the next one in order to improve the algorithm performance. Besides, the results are analyzed. The new parameters set to test are decided based on these results and the knowledge about parameters interaction.

## **3.5 Implementation on Python and Simulation Software**

After choosing the algorithm, its parameters and the scenarios to simulate, the next step is to implement it all. The scenarios definition and simulations are carried out with vehicle simulation software. This is software to test vehicles. This software allows the creation of customized scenarios as well as a precise definition of the vehicles to simulate.

For the algorithm design Python programming language is used. The script also includes the code for the control of the simulation software and for the information exchange. The sequence is the following: First of all the code containing the optimization algorithm chooses some parameters to simulate. Then the simulation software scenario is updated with this information. After that, the simulation start order is given. After the simulation is finished, the results are read. These results are sent to the algorithm. The algorithm calculates some fitness value with the information from the results. Based on the fitness value, the algorithm chooses the points for the next generation.

The general structure of the whole Python code is shown in Figure 3.4:

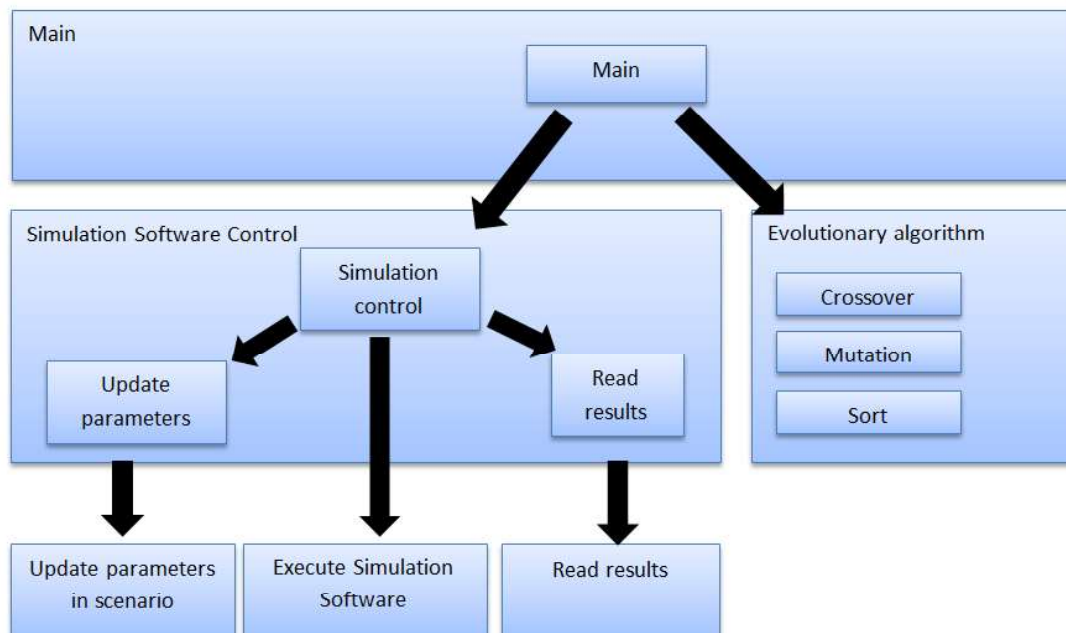


Figure 3.4: Code structure

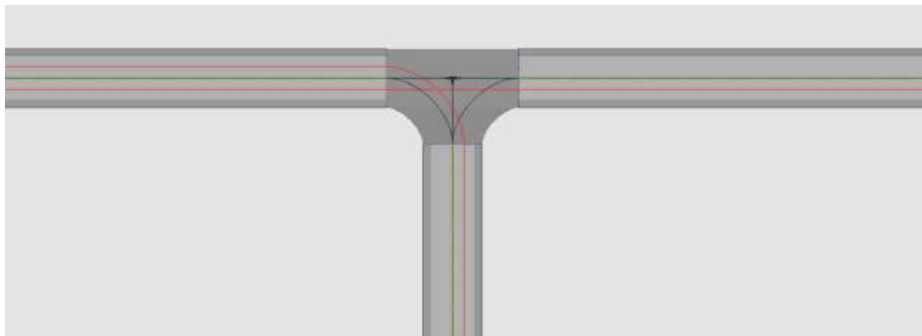
- Main block. This block contains the main function that is in charge of calling the important functions in the rest of the code. These functions can be divided into two groups: the ones related to the optimization algorithm and the ones to control the simulation software.
- In the main function, the instances for the simulation software control file are initialized here with the scenario file names and the path. The instance for the evolutionary algorithm is also initialized here and including the parameters (population size, crossover rate and mutation rate).

- After defining the instances, the population is initialized and the code enters a loop. Inside the loop the evolutionary algorithm functions will be called iteration after iteration.
- The main file also calls the function that saves the final results and plots the figures.
- The evolutionary algorithm block is another file that contains all the evolutionary algorithm functions. The function to generate the first population is defined here. The evolutionary algorithm operators (selection, crossover and mutation) functions are also defined in this file. This block also contains the fitness calculation and the sorting function. Finally some functions in this block are in charge of saving the results and drawing some graphs for a better interpretation of the results. This code is different when using the coverage approaches as they do not need the crossover, mutation and selection steps.
- The blocks on the left in Figure 3.4 have all the code related to open and control the simulation Software. The block denominated Simulation Software control is in charge of giving the order to run the simulation. In order to do that, this file controls the functions in the three small blocks: update parameters, execute Simulation Software and read results.
  - After receiving a group of individuals to simulate in the simulation software it will first call the update function. Before starting the optimization process a scenario needs to be beforehand defined. But some changes are produced to this scenario along the simulation. The update parameters file will update these changes just before every simulation.
  - The second sub-block is in charge of executing the simulation software. When the scenario has been updated, the Simulation Software control file calls the function in the Execute Simulation Software block. This function sends the starting order and receives a warning when the simulation has concluded.
  - The third sub-block reads the results from the simulation just made. It has access to a file that is created after every simulation. With this information it interprets and gets the output variables that are necessary for the algorithm.



The other part in implementation is concerned with the simulation software. This is advanced software for vehicles simulation. To carry out a simulation it is necessary to first define some parameters. These parameters include the road, the vehicle to simulate, the other vehicles (traffic on the road) and the maneuver for all the vehicles.

The road definition lets the user freely decide the shape of the roads, the number of intersections and lanes number on each road. The length and width of each road can also be customized. It also allows setting speed limits that can be different on each stretch and including traffic lights to control the traffic flow. In the case of one or more road intersections, the routes that the ego-vehicle and other vehicles are going to follow are also established here. Some decoration elements can also be added. For the simulation in the thesis the scenarios are quite simple and are formed by a straight road with two lanes.



*Figure 3.5: Road definition in the simulation software*

The vehicle needs to be chosen after defining the road. The software allows for a precise customization on the vehicle in many areas such as dimensions, engine, suspension, tires and aerodynamics. It is also possible to configure the sensors that the vehicle includes. The software offers a list of vehicles already configured. As the focus was on changing the scenario situation, most of the vehicles parameters were left unchanged and only some small adjustments were made. Apart from the ego-vehicle to simulate, one can also include other traffic vehicles in the scenario to have a more realistic situation.

Finally the maneuvers need to be created. The maneuvers give information about how the car needs to behave during the simulation. The maneuver indicates the position, speed or acceleration that the car needs to have at different points in time during the simulation. The other vehicles that are considered traffic can also have maneuvers defined.

After deciding the scenarios needed for the experiments, they are implemented into this simulation software. The road geometry, all the vehicles participating and their respective maneuvers

are created into a base file that is updated with the variable parameter values before each simulation.

To sum up this chapter, firstly the process of scenario generation is explained. After that comes the evaluation about the suitable algorithms for the problem. Some of the options are analyzed and finally the evolutionary algorithm is selected. Other simpler algorithms like Monte Carlo method are explained as they can be used to compare the performance of the evolutionary algorithm. Then, the concept of fitness function is introduced and a specific formula to calculate it is decided. The fitness function calculates how good a certain solution is. Later, the focus is on the configuration of the evolutionary algorithm. It includes selecting the end condition and understanding how to choose the parameters for the population size, number of generations, crossover rate and mutation rate. Finally the implementation is explained. The vehicle simulation software is presented and the code structure is developed with Python programming language. The code part includes the optimization algorithm and the instructions to control the simulation software.

## 4 Results

In the previous chapter the methodology for the optimization problem is described. The algorithms to use are selected as well as some decisions concerning the implementation into the code. After the algorithm implementation, the method is ready to be used to generate scenarios in simulations. The remaining steps before the simulations can be run are divided into two groups: scenario definition and algorithm parameter selection.

Scenario definition includes deciding the specific scenario to simulate and choosing which are going to be the input and the output parameters. A scenario is created by defining the road and traffic conditions in the simulation software. The input parameters are the variable parameters from the scenario. As a reminder the objective of the method is to find a specific scenario where the difference between two systems is maximized. The scenario is given by some fixed parameters and some variable parameters. The variable parameters are the ones that are going to change from one simulation to another in order to find the specific values of these variable parameters that maximize the systems differences. These variable parameters can also be called input parameters as they are the parameters that the simulation software is going to receive as inputs from the main code. The output parameters are the results that after each simulation are sent back from the simulation software to the main code. The fitness value is calculated with the information from the output parameters through the application of the corresponding fitness function.

By algorithm parameter selection it is meant the selection of some values for the evolutionary algorithm parameters. These parameters are the population size, the number of generations, the mutation rate and the crossover rate. There is not a perfect value for these parameters as every problem is different.

On the first part of the results chapter some simulations are carried out. This serves as a proof of concept to check that the algorithm and simulation software are working fine. The other objective of the first round of simulations is to calibrate the evolutionary algorithm. That means that some simulations will be run with different algorithm parameters to find out which values are more appropriate for this kind of problem.

The second round of simulations is formed by the testing of different algorithms. Several simulations are carried out on the same scenario. However, the algorithm in charge of selecting the parameters is different in each round of simulations. With the simulations results, the different algorithm are evaluated to check which one has a greater performance.

The fact of having two different scenarios to test also serves as a way for checking that the genetic algorithm with some specific parameter values can solve several different scenarios. The first scenario can be seen as a training set for the algorithm parameters. Then the second scenario is the test set and it can be found out whether the parameter value selection was done correctly.

### **4.1 Evolutionary algorithm calibration**

First of all some tests are carried out to check that the implementation was done correctly and to calibrate the parameters that control the evolutionary algorithm. The idea is to find some values for the genetic algorithm parameters that make the search process more efficient. These values for the genetic algorithm parameters are not only useful for this specific scenario. As explained in the previous chapter, different problem require different genetic algorithm performance and there are not standard values that always perform well. However, for the case of study, even if the scenarios are different, there are still a lot of common elements in the problem structure. Therefore, the values obtained for these specific scenarios can also be used for other scenarios.

#### **4.1.1 Scenario definition**

In order to start the simulations a scenario needs to be defined. The scenario has to be a situation that a car can find when driving on a road. A scenario with a single car driving on a straight road at constant speed is very simple and does not require any change in the car speed or steering. Therefore, it is recommended that the scenario includes some maneuvers. There is no specific scenario that is required to test as the method should work for any scenario. Several different scenarios are used in other scenario generation papers. All these scenarios are shown in Table 2.1. From the mentioned scenarios, the interesting ones are the ones which contain both steering and speed changes along the simulations. If the maneuvers are more complex, the possible behavior differences between the two vehicles should also be larger. Based on this, the scenario selected for the first simulation is an overtaking maneuver. This scenario happens in a

highway with two lanes. At the beginning the ego-vehicle is driving on the right lane after a truck. The ego-vehicle speed is greater than the truck's speed. When the ego-vehicle gets close to the truck it starts the overtaking maneuver. It changes to the left lane and after overtaking the truck it drives back to the right lane.

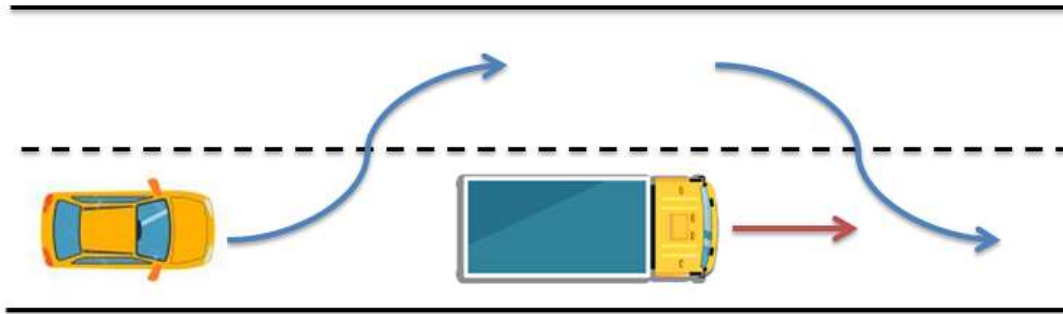


Figure 4.1: Scenario to simulate: Overtaking maneuver

This scenario contains many fixed parameters during the simulations but some other parameters are selected to be variable. The value of the variable parameters is different from one simulation to another. These variable parameters are the ones that changed in order to maximize the difference between the two systems. For this case three variable parameters are selected. These three parameters and its constraints are the following:

- Parameter 1: Ego-vehicle speed. Between 20 – 50 km/h
- Parameter 2: Truck speed. Between 0 – 10 km/h
- Parameter 3: Distance between the ego-vehicle and the truck at the beginning of the simulation. Between 60 – 140 m

These three parameters can also be called input parameters. They are the parameters that the algorithm chooses and sends to the simulation software. So they are the inputs that the simulation software receives before each simulation. It is possible to select more input parameters for a better scenario customization. However, each extra input parameter adds an extra dimension to the search space. So the problem would require a higher computational effort to find good solutions. Therefore, these three parameters are selected as they are the main parameters that control the vehicle maneuvers. The three input parameters cannot have any value as they are constrained between a maximum and a minimum. The constraints are necessary to ensure that the ego-vehicle speed is always greater than the truck speed. If that was not the case, then the overtake maneuver would never happen. The constraint for the distance is defined to avoid the distance being too large or too small. If it is too large, the vehicle needs a lot of time to reach the

truck. If it is too small, the ego-vehicle starts the simulation too close to the truck and may have problems to start the overtaking maneuver.

The output parameters also need to be selected. To understand how the output parameters are chosen it is important to understand how the maximization method works. The process for the optimization is the following: First the algorithm chooses one set of solutions. That means selecting one value for each of the three input parameters (ego-vehicle speed, second vehicle speed and distance between the ego-vehicle and the other vehicle at the beginning of the simulation). The information about the three inputs of that specific solution is sent to the simulation software. The scenario is updated and two simulations are done in the same scenario, one simulation for each of the two systems. The two systems are two different car models for the ego-vehicle. The simulation results from the two simulations are saved. The simulation results parameters that are selected as output parameters are sent back to the algorithm. With the output parameters values for that solution, the algorithm calculates a fitness value.

Therefore, the output parameter selection depends on the specific fitness function selected. For the first simulations, the fitness function calculates the difference between the final position and final speed of the two systems. These two variables are selected because they are the ones which should have a greater difference at the end of the simulations. So the final position of the first car at the end of the simulation is compared with the final position of the second car at the end of its simulation. The same happens for the final speed. This is a simple version of the fitness function. After the first set of simulations the fitness function is updated in order to obtain a more accurate fitness value. The formula for the fitness function used on the first set of maximization processes is shown next:

$$Fitness = |S_{f_1} - S_{f_2}| + |V_{f_1} - V_{f_2}| \quad (4.1)$$

Therefore, the output parameters are:

- $S_{f1}$ : Position of the ego-vehicle at the end of the first simulation (system number one) in meters.
- $V_{f1}$ : Velocity of the ego-vehicle at the end of the first simulation (system number one) in kilometers per hour.
- $S_{f2}$ : Position of the ego-vehicle at the end of the second simulation (system number two) in meters.

- $V_{f2}$ : Velocity of the ego-vehicle at the end of the second simulation (system number two) in kilometers per hour.

With that the scenario and its input and output parameters are defined. The next step is selecting which ones are going to be the two systems that are going to be compared. The two systems are the two vehicles to be simulated and compared. There needs to be some differences between the two vehicles to be able to compare them. Some examples of possible differences could be achieved by the use of different vehicles, by changing some of the mechanical or physical parameters of the vehicles, by changing sensors or by using different vehicle software. For the first case of study these two systems are two different car models available at the simulation software. As they are different models, they have different engines, mass, suspension, etc. So they behave differently when driving in the simulation.

#### **4.1.2 Genetic algorithm parameters**

The scenario is defined with its input and output parameters. Besides, the two systems are specified. After that comes the algorithm parameter selection. The algorithm in charge of selecting the values for the input parameters was a genetic algorithm. The general structure of the genetic algorithm was explained in the previous chapter. But some genetic algorithm parameters have to be decided before starting the simulations. In genetic algorithms there are no standard values for these parameters that are valid for all problems. Some values may work better in some cases and worse in other. These genetic algorithm parameters are:

- Population size
- Number of generations
- Mutation rate
- Crossover rate

In this first part the objective is to try different combinations of these parameters to find out which values work better. The idea is making several simulation rounds. In all of these simulation rounds the scenario, input and output parameters as well as the two cars are exactly the same. The only difference is the genetic algorithm parameters.

It is a difficult task to select the right genetic algorithm parameters. Some studies were explained in the previous chapter for a better understanding of how each parameter influences the

result. These studies serve to select some of the initial values for these parameters and also to be able to analyze the results. The process to calibrate is the following: First a computational load is defined. This is the number of solutions that are simulated in each test run. A solution (a set of input parameters values) is what defines a specific test case in a scenario. The end condition for the algorithm is set as reaching a certain number of simulations. That is the computational load. The limit set is 500 simulations for all the simulation rounds. The limit could be much larger. But as the simulations require some time, having thousands of simulations would mean to have each round of simulations running for days. Because of the limited computational resources, this limit was set to a value of 500. The crossover rate is also given a fixed value of 80% inside the range suggested by Boyabatli and Sabuncuoglu [41]. The crossover rate is kept fixed because it doesn't have a great impact changing this parameter. The population size and the number of generations are dependent of each other as its product should be the computational load.

$$\text{Computational load} = \text{Population size} \times \text{Generations number} \quad (4.2)$$

Some combinations of these two values are tried. The population size values were between 10 and 25 and the number of generations between 20 and 50.

The last parameter to define is the mutation rate. The value for this parameter was more complicated to be defined correctly. Some suggestions were to use a value about  $1/n = 1/3 = 0.33$  (where  $n$  is the number of input parameters). In the experiment, many different values were tried starting from 10% up until 50%. The following table shows the different parameters combinations that are tried in the simulations:

Table 4.1: Genetic algorithm parameters for test cases

	1	2	3	4	5	6	7
Crossover rate (%)	80	80	80	80	80	80	80
Mutation rate (%)	10	20	20	30	30	40	50
Population size	10	10	20	10	20	20	25
Number of generations	50	50	25	50	25	25	20
Computational load	500	500	500	500	500	500	500



### 4.1.3 Simulations results

After the genetic algorithm parameter selection, the simulations are run.

Several topics need to be explained regarding which genetic algorithm parameters give better results, which scenario solutions are the ones that maximize the difference between the two cars and how the iteration process works.

*Table 4.2: Best solution found at each case's simulation*

Case	1	2	3	4	5	6	7
Car speed (input 1)	33,6	23,3	32,2	20,9	31,2	30,4	29,4
Truck speed (input 2)	4,2	3,4	0,7	5,1	4,4	9,9	3,9
Initial distance (input 3)	42,3	45,4	67,3	94,4	42	43,3	45,8
Fitness	34,6	30,1	30,7	16,7	40,6	35,1	30

First in most cases the best solution has a fitness value between 40 and 50. Nevertheless the solution point at which the best values for each case were found is quite different. That means that this optimization problem has a lot of local optima. That is a sign that this kind of problem is what is called a multimodal problem [46]. Some of the input parameters tend to have similar values independently from the case. For example the input parameter number three (initial distance) has a value between 40 and 45 for all cases but two. And in one of these exceptions, in case number four, the best solutions obtained have a very bad fitness value. That is because in case four the search gets stuck in a bad local optimum. Parameter number two (truck speed) seems quite disperse so it may not have a great impact on the fitness value. Input parameter two is constrained between 0 and 10 and the solutions have values all over this range. In input parameter number one (car speed) some common values can be found as in all cases this value is between 20 and 35 (and it is constrained between 25 and 50).

After a general overview of the results it is the time to analyze the optimization process of each case in more detail.

Some problems found when the number of generations is much bigger than the population size are that they got stuck easily in local optima with bad values. That is what happened in cases 1, 2 and 4. In these cases the population size is 10 and the number of generations is 50. In Figure 4.2 it is shown the fitness evolution for case number two. The figure shows at each generation (or iteration) one fitness value. This is the fitness value obtained by the best point in that generation. It is possible to see that after the tenth generation the solution does not improve. The algorithm is looking all the time around one local optimum and is not able to escape and search new solutions. That search is very inefficient as most of the simulations are being carried out approximately using the same solutions.

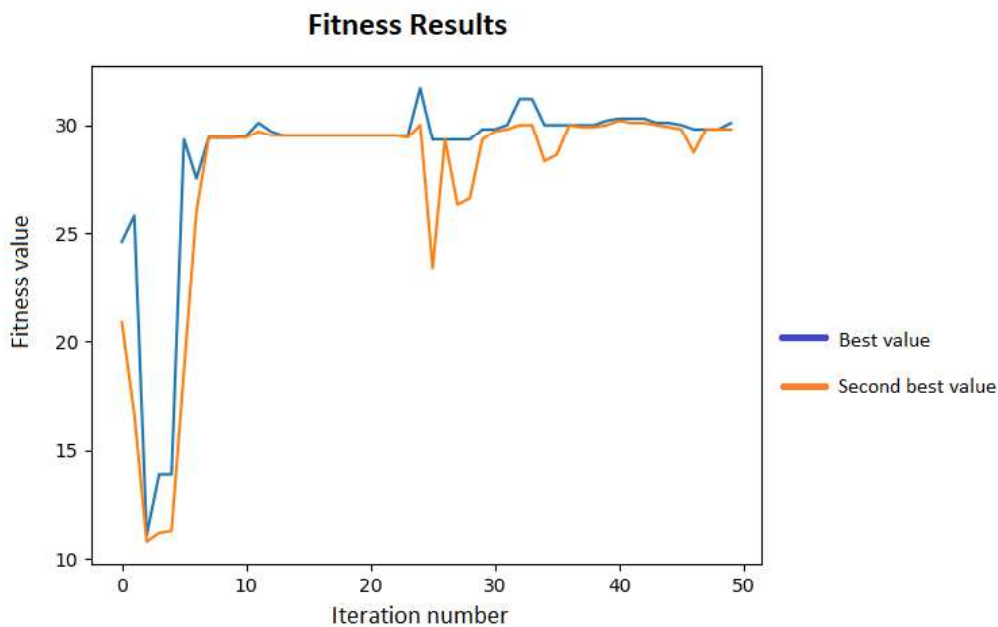


Figure 4.2: Fitness results for Test case 2

Figure 4.2 is showing the results for case 2 but the same happened on case 4. Some box plots are drawn for a better results analysis. In the next box plots for case number 4 (Figure 4.3, Figure 4.4 and Figure 4.5) the values in the iterative process during the simulations are displayed. There are three box plots, each of them shows information about one of the input parameters. The box plots show what values of that specific input parameter were used in each generation. On the first generation the values are very disperse as the initialization is done randomly. But after a few generations the values chosen for each input parameter are very concentrated into some small range. Converging into some values does not need to be bad. The problem with this case is that the converging process is very quick. In just 4 generations the range gets very small so the algorithm covers a very small part of the search space.

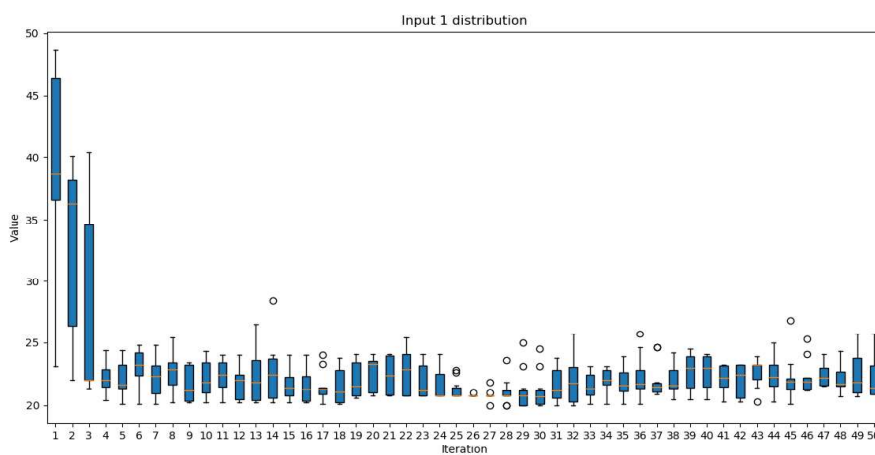


Figure 4.3: Input 1 values distribution for Test case 2

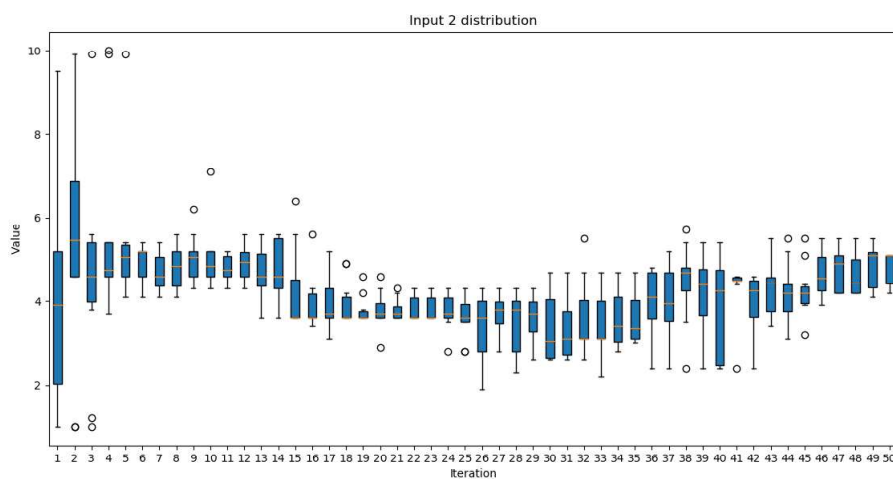


Figure 4.4: Input 2 values distribution for Test case 2

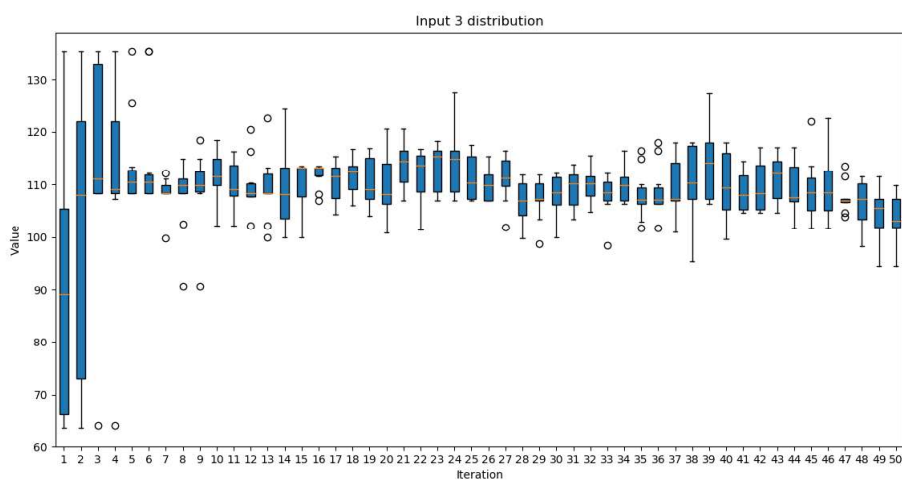


Figure 4.5: Input 3 values distribution for Test case 2

In contraposition to those previous cases, case number 5 is analyzed next. This is the case that gave the best fitness result. The genetic algorithm parameters used are a bigger population size (20) and a smaller number of generations (25). The mutation rate has a greater value (30%) in order to have more exploration. The optimization process looks better as the algorithm does not converge so quickly as before. That is a good sign because it means that the algorithm covered better the search space. That can be seen in Figure 4.7, Figure 4.8 and Figure 4.9. Furthermore the algorithm keeps improving the solution after many generations in opposition to the other case where the best fitness value got stuck after few generations.

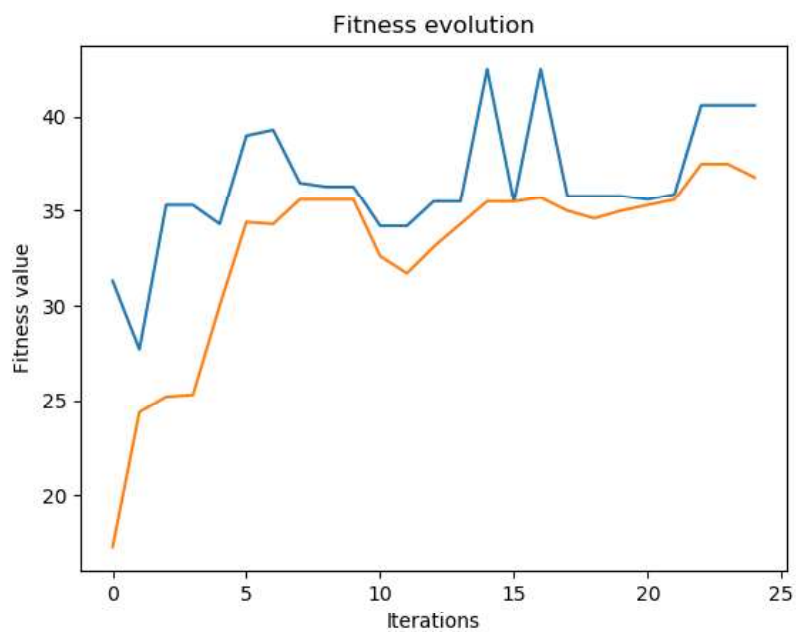


Figure 4.6: Fitness results for test case 5

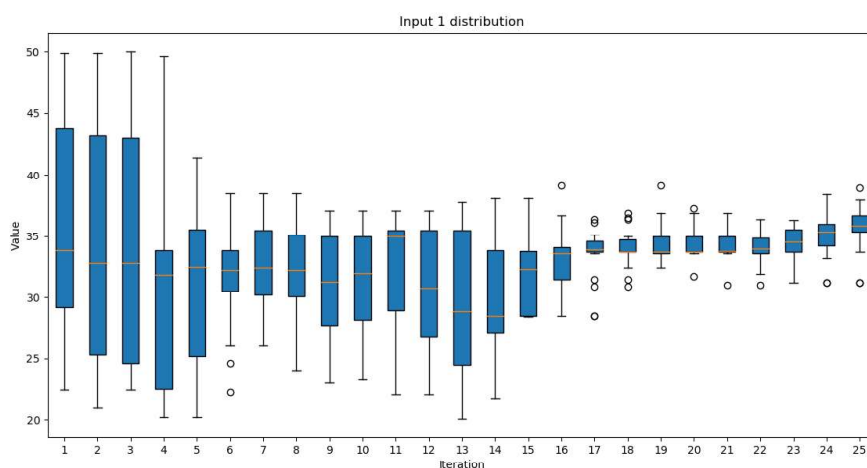


Figure 4.7: Input 1 values distribution for test case 5

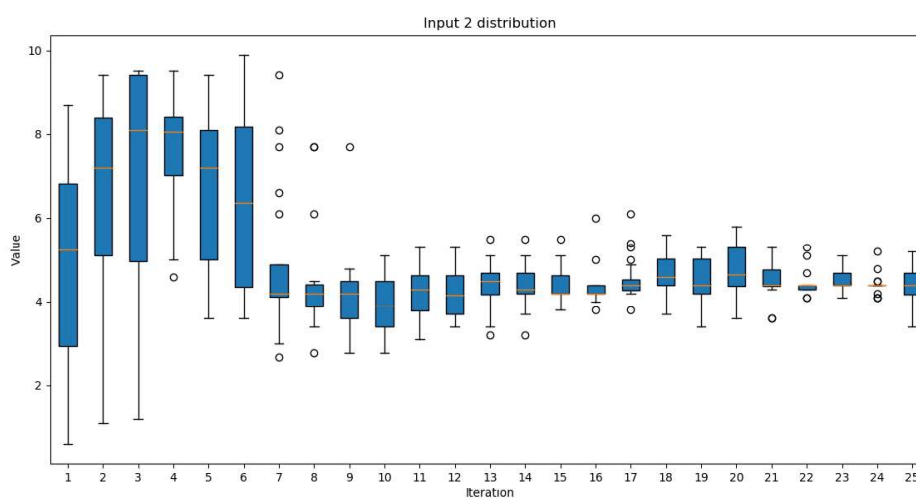


Figure 4.8: Input 2 values distribution for test case 5

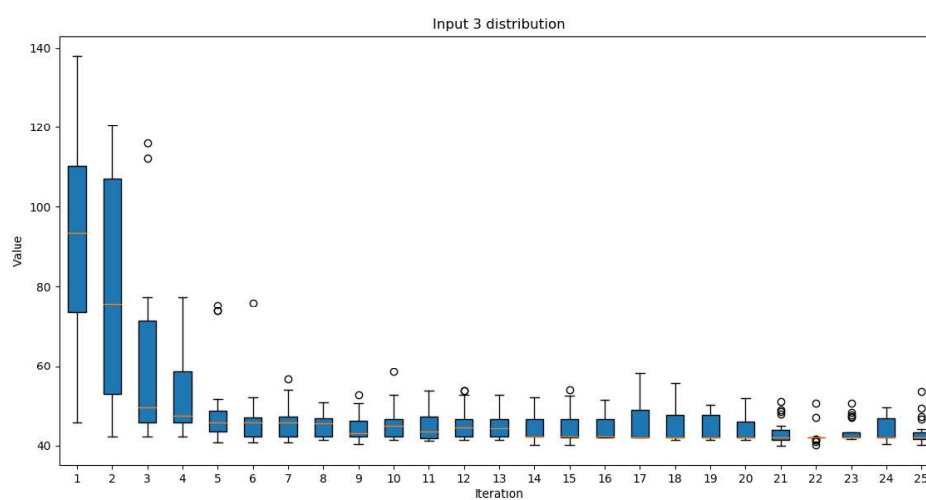


Figure 4.9: Input 3 values distribution for test case 5

The last case to analyze is case number 7. Here the population size is even greater (25) than before and the mutation rate is also increased (50%). Regarding the solution convergence it can be seen that the input parameters values have quite a big range during all the simulations. That can have some advantages by searching in more different areas but this dispersion is also caused by the big mutation rate. Having a mutation rate so high was problematic because the solutions change too much from one generation to the next one. With that some good solutions can get lost.

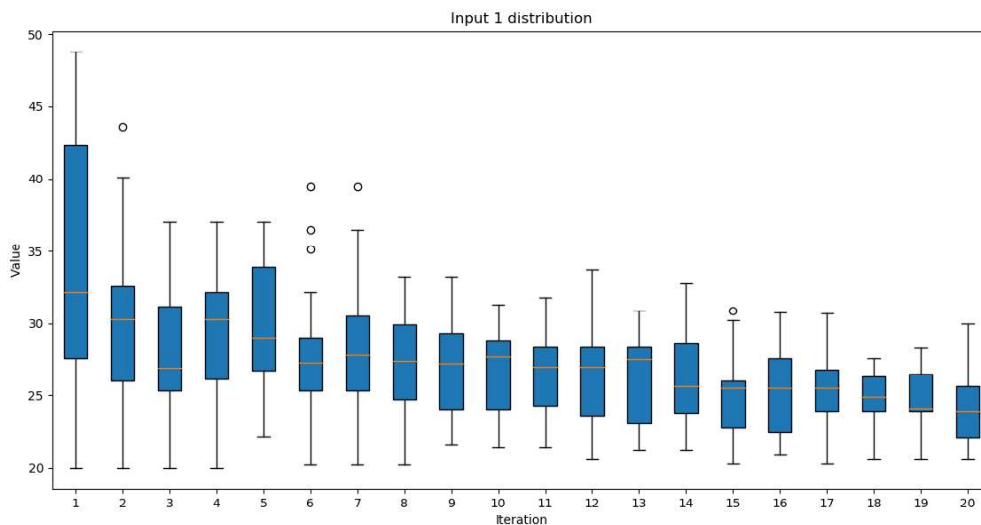


Figure 4.10: Input 1 values distribution for test case 7

Several conclusions can be obtained from the previous results. First of all the trade-off between population size and number of generations can be analyzed. If the number of generations is quite big and the population size is too small, the optimization process does not perform very well. A bigger population size will improve the search by having a greater coverage. It can improve the exploration while decreasing the exploitation. Of course the computational load can also be increased to increase either the population size or the number of generations. A good set of values for this case with 500 maximum simulations is to have a population size between 20 and 25 and the number of generations between 20 and 25. Secondly a mutation rate around 30% looks like a good option. As each solution is formed by three input parameters, a value of  $1/n = 1/3 = 0.33$  should work well. So a mutation rate between 30% and 40% is chosen. And the crossover rate is set to 80%. After finding these values for the genetic algorithm parameters, the idea is to use them also in other scenarios. If the maximum number of simulations is kept at 500 and the number of input parameters is kept at 3, then the same parameter values should work

fine to test other scenarios. Changing the number of input parameters would mean a change in the mutation rate according to the rule of  $1/n$ . Besides, if the total number of simulations is increased, the population size and the number of generations would also increase.

## 4.2 Comparison between algorithms

The first round of simulations is useful to see which evolutionary algorithm parameters work better for the difference maximization problem that is covered in the thesis. The population size and number of generations trade-off can be better understood by analyzing the algorithm evolution along the iterations. Also the crossover and mutation rate are set to appropriate values.

Many changes are presented in this second round of simulations. Firstly the new objective is the comparison between different optimization algorithms. In the previous subchapter the optimization process was always done with the use of a genetic algorithm. In each set of simulations the difference was the genetic algorithm parameters. However, in the new simulations, several algorithms are used to optimize the scenario. And the algorithms performance is compared to find which option is more suitable. The scenario to analyze is also different. The method developed to maximize the differences between two vehicles needs to be applicable not only in one case but in many different scenarios. Having a new scenario is necessary to check that the method can also work correctly in other cases.

### 4.2.1 Scenario definition

As stated before, the second experiment will be carried out in a different scenario. This second round of simulations serve as a test set to check if the algorithm from the first experiment also performs correctly in this second case. The scenario to simulate for this case is a cut-in maneuver. This is another of the typical maneuvers that are used for autonomous systems validation as shown in Table 2.1. The scenario happens also in a highway with two lanes. At the beginning of this scenario, the ego-vehicle is driving on the left lane and a truck drives on the right lane. The truck's velocity is smaller and its initial position is some meters ahead of the ego-vehicle. After some time driving, the truck changes to the left lane until it is situated just in front of the other vehicle. The ego-vehicle, which is equipped with an active cruise control, reduces its speed if the truck is too close.

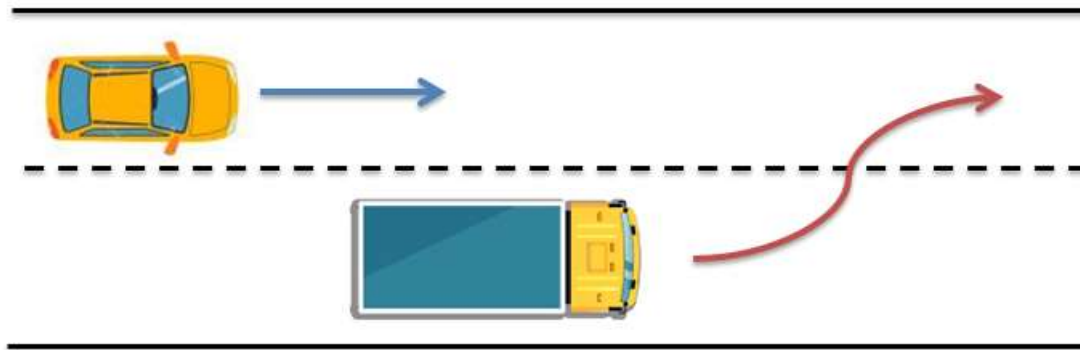


Figure 4.11: Second scenario to simulate - Cut in maneuver

The same way as in the previous scenario, it is necessary to define some input and output parameters. The fitness function and the two systems to compare also need to be selected. The input parameters correspond to the parameters from the scenario that change from one simulation to the next. The input parameters selected for the cut-in scenario were the same than for the first case. The scenario is expected to simulate a situation in a highway so the vehicles speed is higher than in the previous case. The input parameters and their constraints are:

- Input parameter 1: Ego-vehicle speed. Between 80 – 130 km/h
- Input parameter 2: Truck speed. Between 80 – 130 km/h
- Input parameter 3: Distance between the ego-vehicle and the truck at the beginning of the simulation. Between 40 – 200 m

The fitness function is defined differently than before. The new version is more complex but the fitness value calculated is far more precise. This is achieved in two ways: first of all, several values of the ego-vehicle's position and speed during the simulation are measured. Instead of measuring only the values at the end of the simulation, the simulation's length is divided into several points. At each of these time points, the output variables are measured. This allows having a more detailed knowledge about these variables evolution along the simulation. In addition, the formula used to transform the position and speed values into one scalar value is different. The metric used is called Mean Average Percentage Error (MAPE). This metric's selection was explained in detail in the methodology chapter. As a summary at each point in time, this method takes the position or speed between the two vehicles (two systems) and calculates its difference as a percentage.



$$MAPE = \frac{100\%}{n} \sum_{i=1}^n \left| \frac{A_i - B_i}{A_i} \right| \quad (4.3)$$

The output parameters are the ego-vehicle position and speed at different points in time. Now that the fitness functions takes into account the output variable values at several points during the simulation, having the vehicle position as the only output variable could be possible and would give an accurate fitness value. But in other scenario tests, the fitness function may require several output variables. As it should be possible to use this method in different scenario tests, the use of at least two output variables is more interesting. With that it can be demonstrated that the method can work with several output variables. And the MAPE formula result is dimensionless so it is possible to utilize variables that have different units. So the two output variables are the vehicle position and speed. Moreover, having more output variables leads to a fitness value that reflects better the differences between the systems.

Therefore, the output parameters are:

- S1: Array with the ego-vehicle's position at several moments in time on the first simulation (system number one) in meters.
- V1: Array with the ego-vehicle's speed at several moments in time on the first simulation (system number one) in kilometers per hour.
- S2: Array with the ego-vehicle's position at several moments in time on the second simulation (system number two) in meters.
- V2: Array with the ego-vehicle's speed at several moments in time on the second simulation (system number two) in kilometers per hour.

The remaining decision is about the two systems. In the first case the two systems were two different vehicle models. That is also the case on the second case but with some differences. The ACC sensor is updated so it was not the same for the two vehicles. In one of the vehicles the range of the ACC sensor is left to its standard value of 150 meters whereas on the other vehicle it is reduced to 100 meters. The motivation for this change in the sensor is to have some difference between the two systems not only in the dynamics behavior but also in the assistance systems.

### 4.2.2 Four algorithms for performance comparison

For the maximization process of the scenario, several algorithms are used. The results are compared to check which algorithm can have a great performance in these cases and why. There are four different algorithms selected:

- Genetic algorithm (GA)
- Evolution Strategies (ES)
- Monte Carlo method (MC)
- Latin Hypercube sampling (LHS)

The first one is the genetic algorithm. It was previously explained and properly calibrated by having some simulations. The second algorithm is another evolutionary algorithm called evolution strategies. This algorithm is quite similar to the genetic algorithm so its implementation was easily done. The motivation for the use of this algorithm is that it is commonly used for real number optimization processes, as it is the case in the thesis. The other two algorithms to use were Monte Carlo method and Latin Hypercube sampling. These two methods are coverage methods. That means that they do not divide the optimization problem into several iterations and use the results from the previous iterations to help the selection of possible solutions. They select points randomly or almost randomly around the search space and send them to the car simulator.

The comparison is made by checking which method can find solutions with a greater quality. In order to do that, the computational load is the same for all the methods and the fitness value of the best individuals found by each method is compared. The fitness value is at first used for the comparison of different individuals during the evolutionary algorithm optimization. Nevertheless, the fitness value of the final result is also a good way to measure the algorithm quality to find a good solution. So the fitness value is established as the metric to compare several algorithms. To have the same computational load, a limit in the number of simulations is set at the same value for the four algorithms. The number of simulations was set to 500. Each of these 500 simulations includes one simulation for system 1 and one simulation for system 2 with the same input parameters. After finishing the 500 simulations, the best solutions found with each method are selected and its fitness value compared.

The two coverage methods, Monte Carlo method and Latin Hypercube sampling, do not have any algorithm parameter to set. On the other hand, the two evolutionary algorithms need some values for its parameters. The values for these parameters are chosen based on the results of the

previous scenario simulations where different values for these parameters were tested. Although the previous experiments to find the evolutionary parameter values were only made for the genetic algorithm, the same parameters are used for the evolution strategies. As the genetic algorithm and the evolution strategies structure is very similar, the assumption is that the same parameter values are valid for both algorithms. The parameters are and the values selected are:

- Population size: 20-25
- Number of generations: 20-25
- Mutation rate: 30%
- Crossover rate: 80%
- $\lambda$  (only for evolution strategies): 20
- $\mu$  (only for evolution strategies): 50- 60

With that the four algorithms are ready and the simulations can be carried out.

### **4.2.3 Simulation results for the four algorithms**

The results analysis is divided in several steps. First it is focused on analyzing which algorithm found the best solutions. So the highest fitness values reached by each algorithm are compared. After that the analysis is focused on the input parameters that lead to those good solutions and find out the relation between the different solutions. Finally the good scenarios will be analyzed in order to understand why those specific input parameters values lead to a great difference between the two vehicles (high fitness value).

Let's start with the fitness value. Figure 4.12 includes the fitness results for several simulations. What the figure shows are eight different column graphs, one for each simulation run. On each simulation run, the three (or four) best points are selected. Each column graph contains the fitness value obtained for these best points.

## Best fitness values

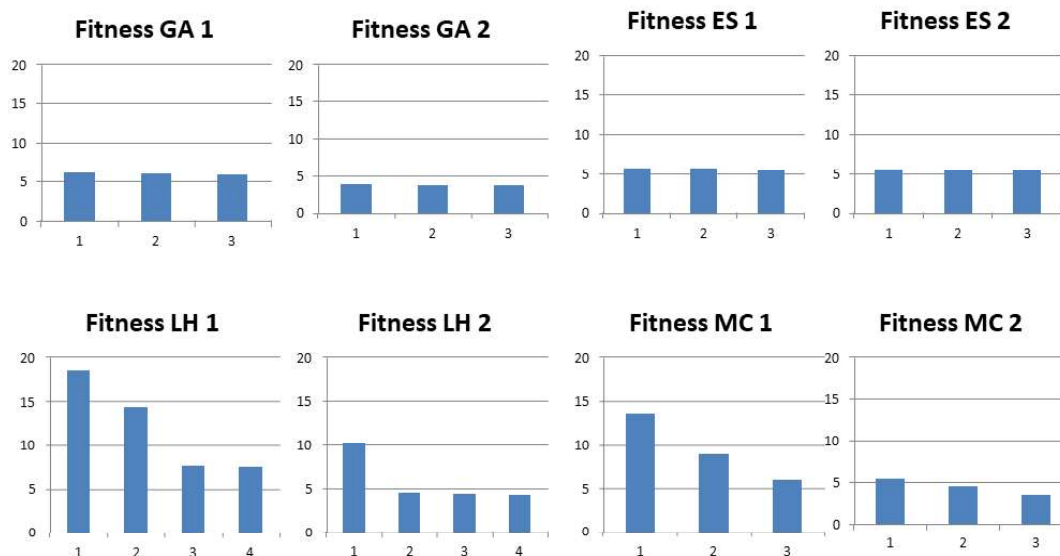


Figure 4.12: Fitness values for the best solutions in each simulation run

Moreover, there are eight column graphs and only four algorithms because each algorithm is run twice. This is done because the simulations are not deterministic. For the Monte Carlo method and the Latin Hypercube sampling, the points chosen to analyze are randomly selected. In evolutionary algorithms, the first generation is also randomly selected. And the crossover and mutation operators work with some randomness. Therefore, when several simulations are done through an evolutionary algorithm, the points used along the simulation process are different, even if the algorithm parameters are kept constant. This randomness could lead to results that are exceptionally good or bad because of good or bad luck at the search. By running each algorithm several times it is expected to reduce possible outliers in the results. The second simulation with each algorithm was necessary to check how big this variation corresponding to good luck or bad luck in the search is.

Based on these fitness results the coverage approaches, which are Latin hypercube sampling and Monte Carlo method, obtain better solutions than the evolutionary algorithms. They find higher fitness points even though in the two coverage approaches the algorithm implementation is simpler. The evolutionary algorithms are not able to find any solution with a fitness value greater than 7. Their best points have a fitness value around 5 and 6. On the other hand the coverage approaches find several solutions with a fitness value greater than 7. Other remarks can be made by looking more into detail at the Latin hypercube sampling and Monte Carlo method. In

Figure 4.12 the three or four best points are shown. But if the rest of the points are checked the conclusion is that only about the 10 best values out of the 500 in the whole simulation give medium or good quality values (with a fitness value equal or greater than 4). In the coverage approaches the differences between the best solutions and the worst is quite big. The results in the evolutionary algorithms are different. These algorithms provided several solutions with a fitness value very close to each other. The reason for this is due to the optimization process that constitutes evolutionary algorithms. As they are iterative processes, though there is a trend to convergence as more generations are created. Therefore, the final generation contains individuals which have a lot of similarities between each other. That is why the fitness value is also quite similar for many of the individuals that belong to the last generation in evolutionary algorithms. To sum up the two coverage methods can find better solutions. But out of the 500 points chosen to simulate, most of them are bad options and only a few of them obtain a great fitness value. On the other hand, evolutionary algorithms cannot find such good solutions. But it is observed some similarities between the individuals in the last generation due to the algorithm convergence.

The second part of the results interpretation is focused on the input parameter values obtained. The values of the ego-vehicle speed (first input parameter) at the best points found in the simulations are shown in Figure 4.13. The figure is similar to the one before but instead of showing the fitness value of the best points, the values for the first input parameter are displayed.

## Input 1: Ego vehicle speed

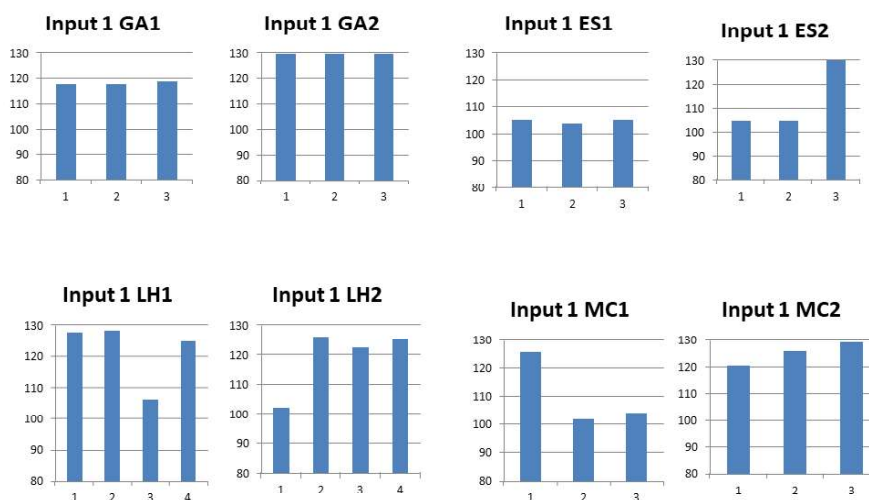


Figure 4.13: Input 1 values for the best solutions in each simulation run

What can be observed generally is that the values are quite dispersed. There are no points in the range 80 – 100 but in the range 100-130 the dispersion is big. The same behavior is observed for the truck speed (second input parameter). There is a lot of variety on the values where the best fitness was obtained. The case is different for the truck initial position (third input parameter). In this specific parameter it is observed a convergence into a range of values between 40 – 60 meters for most of the good solutions.

### Input 2: Truck speed

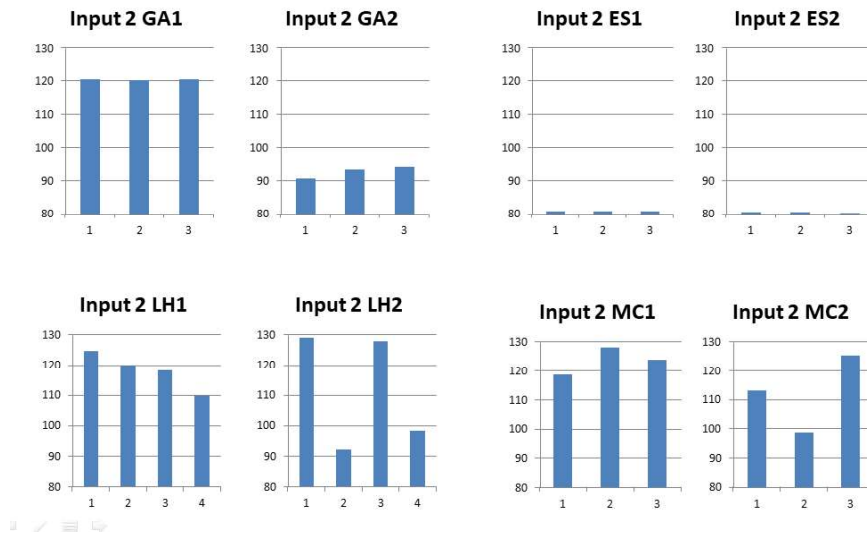


Figure 4.14: Input 2 values for the best solutions in each simulation run

### Input 3: Truck initial position

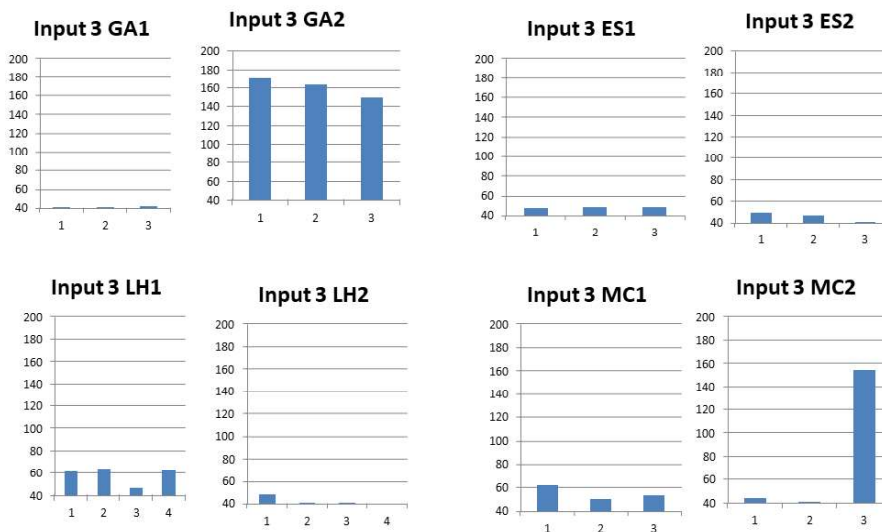


Figure 4.15: Input 3 values for the best solutions in each simulation run

Two conclusions can be obtained from the input parameters values information. First of all, the value of the third input (truck initial position) is quite important in order to obtain a good solution. In most of the cases this parameters value is constrained between 40 and 60 meters. There are a few exceptions where this parameters value is over 140 meters but in those cases the fitness value obtained was not so good compared to the best points in all of the simulations. So a low value in the third parameter is critical to find a good solution. On the first and second input parameter, a convergence around some values is not observed. In the first input parameter, the values observed are usually high but scattered distributed. In the second input parameter the values are even more disperse. So it can be inferred that the third input is the more critical value to find a good solution and the second input is the one which has smaller effect into the results. This dispersion in the second parameter and convergence around some value for the third parameter can be observed in the evolutionary algorithms optimization process in Figure 4.16 and Figure 4.17.

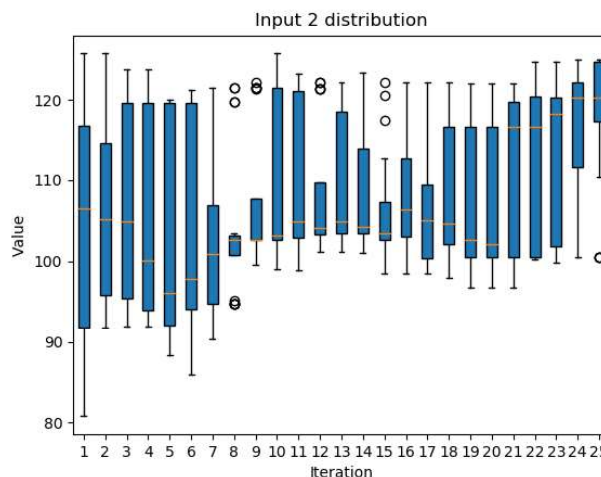


Figure 4.16: Input 2 values distribution for Genetic Algorithm

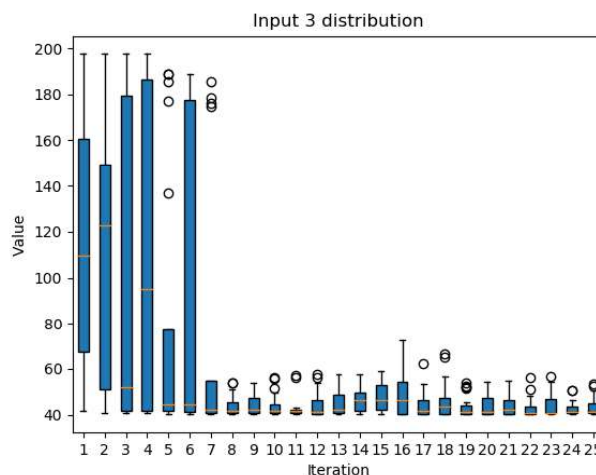


Figure 4.17: Input 3 values distribution for Genetic Algorithm

The box plot shows the values for the second input and third input that were used during one of the genetic algorithm optimization process. Each box corresponds to all the values used in one generation. This algorithm selects the new input parameters values based on which values lead to higher fitness in the generation before. The box plots show this high convergence for the third input parameter and the dispersion for the second input parameter. That is because a low value for the third input parameter was a clear sign for the genetic algorithm that the results were better.

The second conclusion deals with the dispersion. The fact of having so much dispersion in the input parameters is an indicator that the problem has a lot of local optima around the search space. The problem function is not defined but works as a multimodal function. This kind of function has a lot of maximum or minimum points around the parameters space. The next figure shows an example of multimodal function for a two dimensional problem.

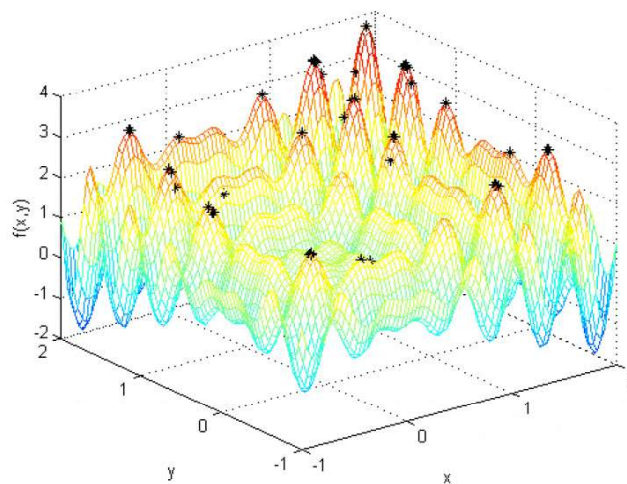


Figure 4.18: Multimodal function [47]

That makes the maximization process more complicated. The evolutionary algorithms try to exploit the search into an area but they get stuck very easily in some local optima. So after several generations they are not able to increase considerably the solution. This can be graphically seen in Figure 4.19 and Figure 4.20. The horizontal axis shows the number of the generation (or iteration) and the vertical axis is the fitness value obtained at that specific generation. The different lines show the four best solutions at each generation. The blue line is the best solution at each generation, the orange line is the second best solution, the green line the third and the red line the fourth.



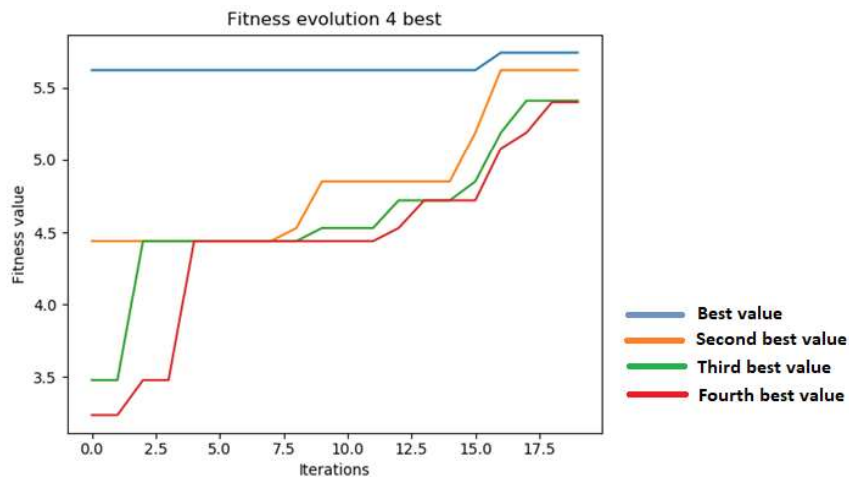


Figure 4.19: Fitness value evolution through several generations for the Evolution Strategies

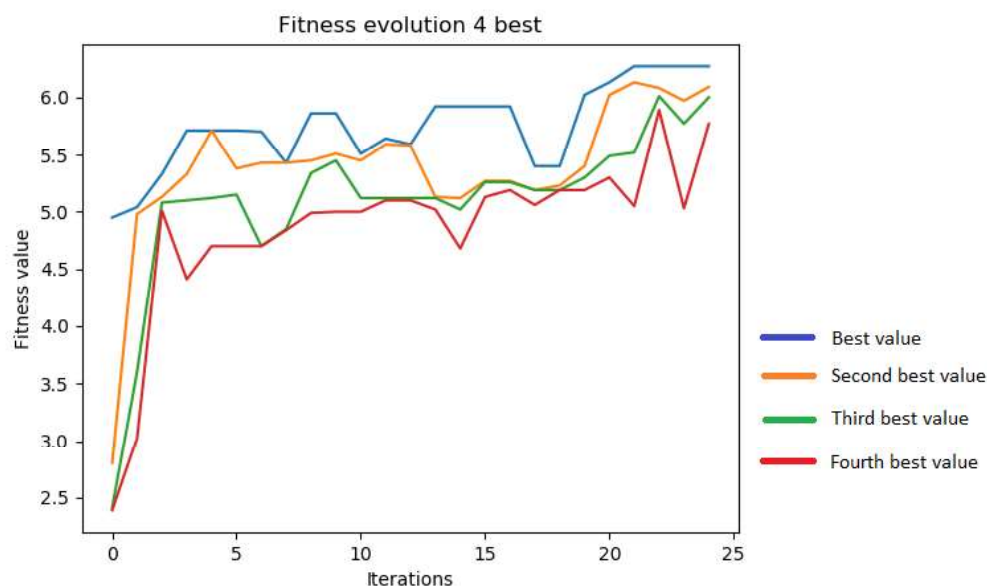


Figure 4.20: Fitness value evolution through several generations for genetic algorithm

It is common to observe that the best fitness value do not improve significantly from one generation to the next one. The best point found in the first generation had a fitness value around 5 or 6. But after many generations this value's improvement is quite small. The optimization process could be improved by increasing the exploration and decreasing exploitation. That could be done by changing the evolutionary algorithm parameters. Increasing the population size would mean a higher exploration. That would imply that the number of generations is lowered in order to keep the total number of simulations constant. A lower number of generations means a smaller exploitation.

## 4 Results

Finally the interpretation of the input parameters values in the scenario comes. The motivation is to understand why some specific input parameters values lead to a great difference between the two cars behavior in the same scenario. The result on the best point found is shown next:

Table 4.3: Results for the best point found

Parameter	Value	Minimum	Maximum
Ego- vehicle speed (input 1)	127,7	80 km/h	130 km/h
Truck speed (input 2)	104,5	80 km/h	130 km/h
Initial distance (input 3)	61,8	40 m	200 m

In this specific scenario the ego-vehicle starts the simulation driving at a about 127,7 km/h whereas the truck drives at 104,5 km/h and is situated 60 meters ahead of the car. After a few seconds the truck starts a maneuver to change from the right lane to the left lane. So what is obtained is a higher speed of the ego-vehicle respect to the truck. Besides, the vehicle speed value is quite high. As a reminder the speed value was limited between 80 and 130 km/h. So the speed value is very close to the maximum possible. Moreover, the initial distance between the car and the truck is close to the minimum possible (the range was from 40 to 200 m). A graph is provided where the ego-vehicle speed values are plotted. The simulations take place in the same test case but one line is for the system 1 and the other line for the system 2.

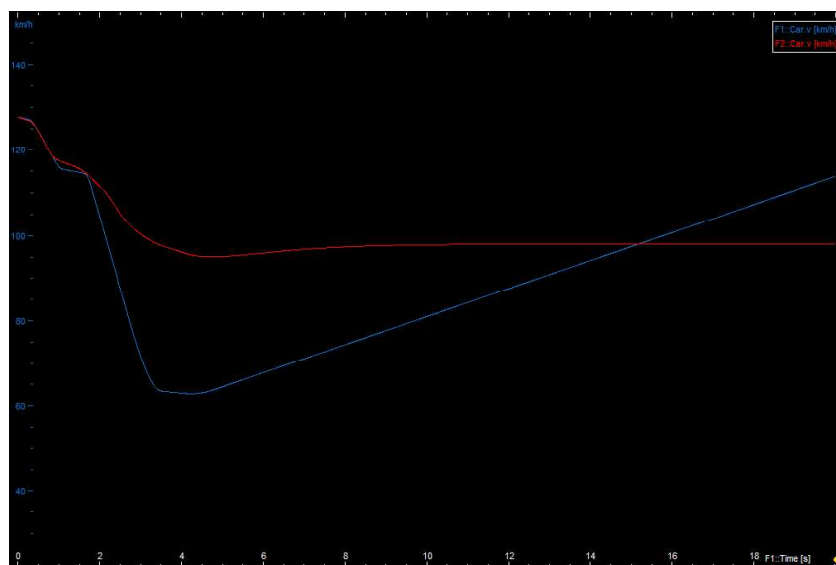


Figure 4.21: Ego-vehicle speed profile during the simulations of both systems

With the input parameters values explained and the information from the graph it should be possible to have an idea about why this specific scenario leads to a big difference between the two cars behavior (the two systems to test). The fact of the truck being close to the car produces a big braking maneuver when the truck changes lane and is situated in front of the ego-vehicle. In general big accelerations and big decelerations are prone to increase the differences if two different cars are compared. The aggressive maneuvers lead to the vehicle being put to the limit of its performance. If a vehicle has a faster dynamic response than the other one, then the differences at these maneuvers will be wide. Besides, the ACC range was set to different values. That means that after the truck cuts in, one of the car tries to get more separated from the truck than the other car. So one of the cars needs to reduce more the speed or keep it at a lower value for longer.

#### **4.2.4 Algorithm comparison summary**

The second round of simulations consisted of a cut-in scenario where a truck was changing from one lane to the other just in front of the ego-vehicle. The objective for this second scenario is to compare the performance of different algorithms. The total number of simulations allowed is set to a constant value of 500 for every algorithm and the best results obtained are compared. The four algorithms are divided into two groups: the evolutionary algorithm and the coverage methods. The best results are obtained in most cases for the coverage methods: Latin hypercube sampling and Monte Carlo method. It is also observed that the problem had a lot of local optima. That could be the reason why the coverage approaches performed better. As there are many local optima, it is easy for the evolutionary algorithms to get stuck in some of those local optima. So they cannot take full advantage of the iterative process. On the other hand the Latin Hypercube and Monte Carlo method have a great exploration so they can perform better.

#### **4.2.5 Solution improvements**

One of the problems that the evolutionary algorithms find is that the problem had a lot of local optima. The evolutionary algorithms try to improve the solution by searching in the area of previous good solutions thanks to the mutation operator. But if the problem has many local optima, it is difficult to find the global optimum. The solution for that would be to increase the coverage. Previously the simulations limit was set to 500. This value could be increased in order to have

algorithms with higher population size and same number of generations. The higher population size means that the algorithm can cover a larger number of points in each generation.

The first objective for the solution improvement is to check if finding a good value on the first generation was very helpful for the algorithm. The experiment carried out consists on using the same genetic algorithm for the cut-in scenario optimization. But in this case the initial population contains one of the good solutions found in the previous optimizations with the Latin hypercube sampling. Only that individual is manually included and the rest of the initial population are randomly initialized like the cases before. Figure 4.22 shows the fitness corresponding to the 4 best points found at each generation. The blue line shows the best individual's fitness and the other colors shows the second best, third best and fourth best solutions fitness.

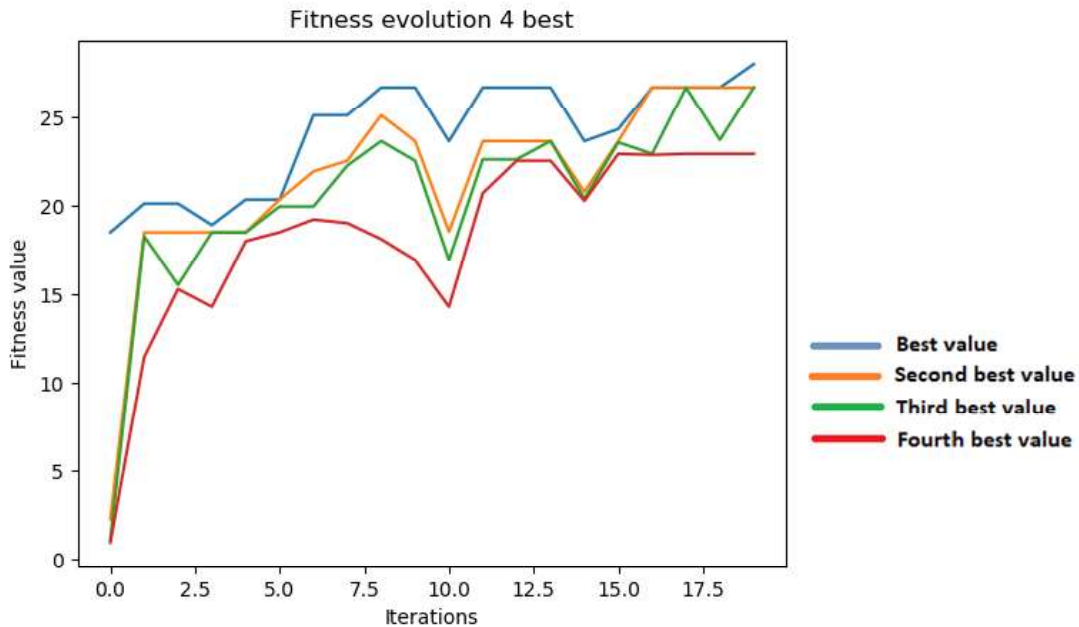


Figure 4.22: Fitness evolution for GA with a manually introduced point in the initial population

What the figure shows is that the algorithm is still able to improve the solution. So finding a good value in the first population is critical for the evolutionary algorithms in order to find a good final result. Increasing the population size improves the possibilities of finding a good individual in the first generation. Therefore, increasing the population size is a suitable option to find better points. It is not the same for the number of generations. Some of the evolutionary algorithms optimization runs previously studied had already converged before the 20<sup>th</sup> generation. So improving only the number of generations does not look promising.

Therefore, the conclusion obtained is that keeping the same number of generations and increasing the population size could lead to better results. For that it would be necessary to increase the total number of simulations. This value was previously set to 500 and for the following simulations is increased to 2500. The algorithms to test with the new number of simulations are the genetic algorithm and the Latin hypercube sampling. Two simulations are run with each algorithm and the results are compared with the previous cases where only 500 simulations were allowed.

## Best fitness values

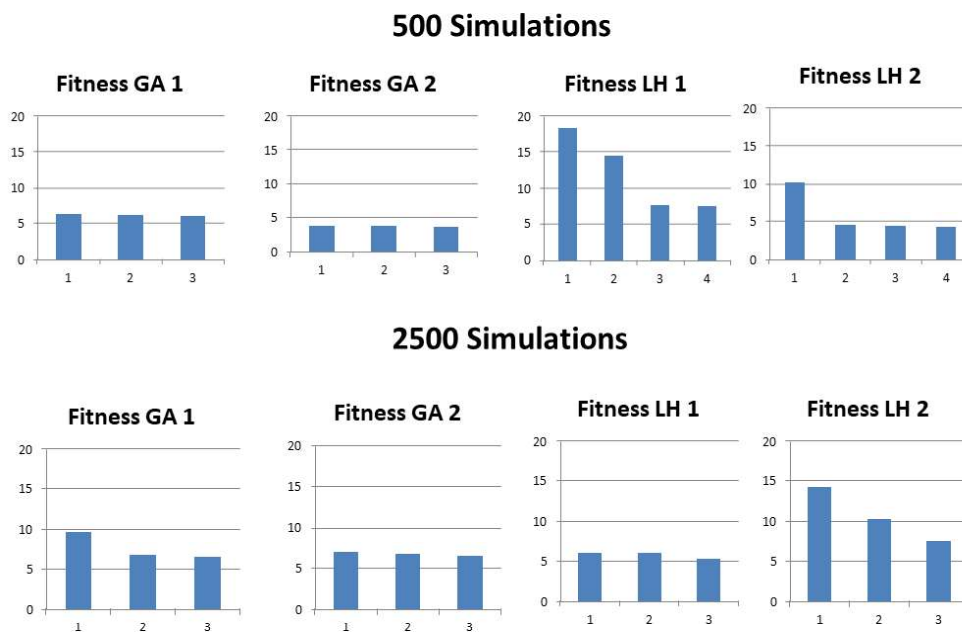


Figure 4.23: Fitness results comparison between algorithms with 500 and 2500 simulations

Figure 4.23 shows the new fitness value results and its comparison with the previous cases. Each bar graph is a simulation run. The four bar graphs at the top show the fitness results on the previous simulations and the four bar graphs at the bottom show the new simulations results. Each bar graph includes the fitness value corresponding to the three best solutions in that simulation. In the case of the genetic algorithm, the additional simulations are helpful to find solutions with higher fitness value. In the two new simulation runs they achieve a better result than the previous cases. The best individuals found result in a fitness value around 7 and in one individual it reached 9,7. In the previous cases, the best individuals had a fitness value between 4 and 6. That can be a sign that the population size in the previous cases was too small. In this new simulation the population size is increased from around 25 to 125 and the number of gen-

erations is kept at the same value. This higher population size increases the exploration and lead to a better search.

On the other hand the Latin Hypercube sampling does not seem to improve with the number of simulations. In fact, for these cases the best solutions found are worse than before. But that should be because of bad luck in that specific search. As the points selection is done randomly, the more points that are evaluated, the higher are the probabilities of finding a good solution. But for the Latin Hypercube case, the improvement on the total number of simulations from 500 to 2500 does not seem to be a help for the solution improvement.

When comparing the algorithms for the new case with 2500 simulations, it is not so clear which algorithm performs better. One of the Latin Hypercube did found a better point but that was not the case for the other simulation.

One last experiment is performed. In this new simulation round the objective is to improve the search algorithm based on all the information from the previous experiments. As the problem has a lot of local optima, a high coverage is critical to find a good solution. But the iterative process from the evolutionary algorithms can also be applied to improve the solutions. The new concept was a mix of the two kinds of algorithms: coverage and evolutionary algorithms. The new algorithm creates a big initial population and from this initial population the best individuals are selected and an iterative process is run. The iterative process will improve the solution through the mutation and selection operators from the evolutionary algorithms. So it is a combination between Monte Carlo method and evolutionary algorithms.

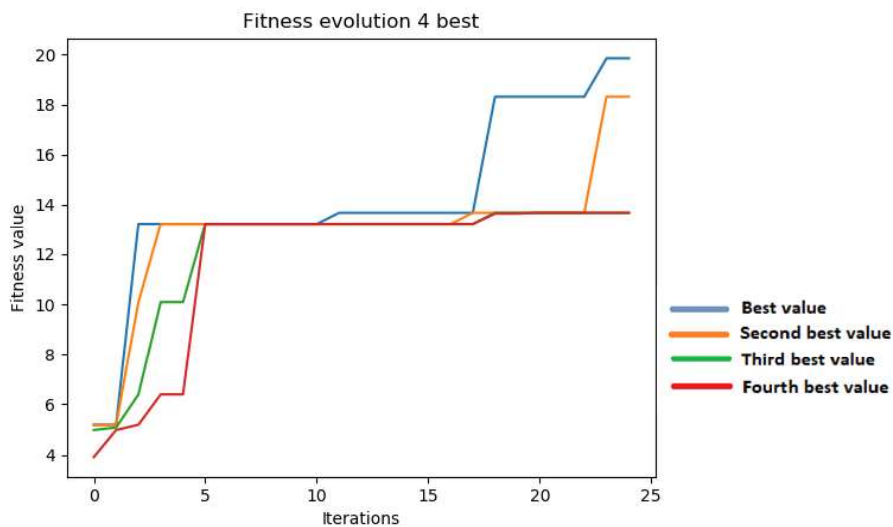


Figure 4.24: Fitness values evolution for mixed algorithm

---

This mixed algorithm is run with an initial population of 500 and another 500 simulations with the evolutionary algorithm. The results obtained outperform other results obtained with only evolutionary algorithms. In fact the best solution obtained from all simulation runs is obtained with this method. The fitness value was almost 20.

### 4.3 Simulations results summary and future challenges

The results chapter contains several simulations with different algorithms and scenarios. The most important results and analysis are summarized next.

On the first part of the chapter, some simulations are run with a genetic algorithm. These simulations serve as a first proof of concept and to calibrate the genetic algorithm parameters. The simulations take place in a scenario about an overtaking maneuver. It is a highway where the ego-vehicle performs an overtake maneuver to a truck. The ego-vehicle speed, truck speed and initial truck position are the input parameters. This scenario is run looking for the specific cases where the difference between the two cars tested is bigger. The algorithm applied is the genetic algorithm for all test cases but with different algorithm parameters in each case. The parameters modified are the four important parameters for genetic algorithm tuning: population size, number of generations, crossover rate and mutation rate. These simulations results are analyzed through some graphs and by observing the iterative process.

The second part of the chapter includes several simulations where different methods are used for the input parameter value selection and optimization process. Moreover, the scenario for the tests is new. In this case the scenario is a cut-in maneuver in a highway. A truck drives on the right lane and at certain moment it changes to the left lane where the ego-vehicle drives. The three input parameters are the same than for the previous scenario: Ego-vehicle speed, truck speed and truck initial position. Instead of generating the test cases with the genetic algorithm, four different methods are applied. The results are compared to find out which method performs better. The four methods are divided into two groups: evolutionary algorithms and random methods. The two evolutionary algorithms are genetic algorithm and evolution strategies. The other two coverage methods are Monte Carlo method and Latin hypercube sampling. All the methods run the same number of simulations and with the same input parameters.

The results show that the two coverage methods can find better solutions. From the best solutions analysis it is inferred that the optimization problem has many local optima. That is the reason why the evolutionary algorithms do not perform well. Some improvements options are later proposed. They are mainly focusing in increasing the evolutionary algorithms coverage. For that, the maximum number of simulations is increased so that the population size can be increased maintaining the number of generations constant. These changes allow an improvement in the results obtained by the genetic algorithm.

The final simulation run is based on a mixed method which combines random sampling with evolutionary algorithms. This method has a big initial population where the points are randomly chosen and from this initial population it uses the mutation and selection operators to find new solutions. This mixed method achieved a performance improvement compared to the other evolutionary algorithms.

To sum up, the simulations results show that the coverage approaches based on random point selection have a good performance in the scenario test. The scenario having a lot of local optima makes it difficult for the evolutionary algorithm to search the best solutions as they would require a very big population size. The only method that could reach or improve the solution was a mixed method with based on a big initial population randomly selected and then use an iterative process based on the evolutionary algorithms.

Regarding further development in this project, it would be interesting to analyze new scenarios to see if the same patterns that were found in the thesis occur also for other cases. The scenario possibilities for vehicle testing are huge. The method could be applied to some of these scenarios to find behaviors difference between two systems.

Another interesting update would be to increase the number of input parameters from the current 3 used in these tests. That can allow for a much more customized scenario. The problem with a high number of input parameters is that the number of possible combinations is higher. Therefore, the computational load to find the best points would increase significantly. The new input parameters can be about the scenario environment. But an interesting application would be that some of the new input parameters were related to the vehicles maneuvers. Instead of having vehicles with constant speed, that can allow to have a variable speed profile for the different vehicles in the scenario.

Finally, the two systems to compare during the simulations were two different cars modeled inside the simulator. A possible alternative in this is the use of a real system and compare it with



the virtual version of that system. An example of this is the comparison between a camera and its virtual version. The real camera records a scenario virtually generated and the virtual camera receives the same scenario. Then the recognition systems can be compared, such as sign recognition or other vehicles recognition. The combination between a real system and a virtual system also opens the door to other experiments such as comparing data collected with a real vehicle driving with data from virtual tests.



## 5 Conclusion

The development of autonomous vehicles will produce many novelties and improvements in the future way of travelling. But it is also an important challenge to the automotive industry as a lot of changes will happen in the following years. One of these challenges is involved with the validation of autonomous cars. That is, demonstrating that their driving abilities are good enough to be allowed to drive autonomously together with other traditional vehicles.

The validation processes are based on physical tests with real cars and on simulations. The use of simulations is widely extended because they allow for a cheaper and faster way to test specific driving situations. It is in this area of using simulations for the validation of autonomous cars where the master thesis idea of comparing systems with simulations appears. The thesis is focused on developing a method that is able to generate specific scenarios to compare two systems and find out in which specific situations the behavior difference between the two systems is amplified. So a logical scenario describing a driving situation is given with some of its parameters set to fixed values and other parameters set to variable values. The method tries different combinations of the variable parameter values searching for a specific solution which maximizes the difference between the two systems.

In order to develop this method, the first part includes the analysis of several scenario generation papers. These papers test advanced driving assistance systems and autonomous vehicles to find critical situations. Based on this previous research, the different approaches to select test cases are evaluated. One of the main questions is whether to use a random selection approach (coverage) or an iterative process (falsification). The falsification approach is chosen as the base method because it focuses the search in the most promising areas, even if it is more complex to develop. Between the possible falsification approaches another evaluation is carried out to compare the evolutionary algorithm and the reinforcement learning algorithm. In this case, the evolutionary algorithm is selected as the algorithm in charge of selecting the parameter values in the comparison process. The reason for this is the parameters that are later chosen as variable parameters in the scenario. Reinforcement learning is useful for parameters that have different values during the simulation and where the algorithm makes decisions. On the other hand, for

variable parameters with constant values during a simulation, the evolutionary algorithm can perform better.

After the algorithm decision, several steps come to configure the fitness function and algorithm parameters. The fitness function is defined through the mean absolute percentage error (MAPE) formula. This formula serves to compare two sets of values and it gives a scalar value that indicates how different the two systems compared are. This specific formula was chosen because the difference is measured as a percentage between the value differences at each point and the current value. This value is dimensionless so several different magnitudes can be summed into the same fitness value.

The next part of the method development is the algorithm parameters configuration. The four main parameters are population size, number of generations, crossover rate and mutation rate. The problem is that each problem requires different genetic algorithm parameter values in order to run with a high performance. The solution for this is divided in two parts: The first part includes the research to understand the interaction between the parameters to what is called the exploration-exploitation trade-off. The research part also serves to find some standard values that can be used as first values. The second part is related to test different parameter values to find out which values perform better. These tests are run in the first experiment with the simulation software.

In this first experiment a logical scenario is defined, which is an overtaking maneuver. But instead of running simulations with a fixed set of values for the genetic algorithm parameters, several test cases are defined. In each test case a specific combination of genetic algorithm parameter values is set. These cases are run to check which value combination works better.

After the first experiment, a second experiment with a different scenario is carried out. The objectives in this case are different. One objective is to check whether the genetic algorithm with the previously selected parameter values can also work correctly in other scenarios. So the first experiment would work as a training set and this second experiment as a test set. In the second experiment another objective is the comparison between different algorithms in the same scenario. Simulations controlled by four algorithms are run to find out which algorithm finds a better solution. The four algorithms are two evolutionary algorithms (genetic algorithm and evolution strategies) and two coverage methods (Monte Carlo sampling and Latin Hypercube sampling). This second scenario is a cut-in maneuver from a truck in front of the ego-vehicle. The first results show that the coverage methods can find better solutions than the evolutionary algorithms. So in this first simulation rounds the coverage approaches are a better option as they

are easier to implement and able to find good solutions. More simulations are run in this second scenario. In the next ones the limit of the number of simulations is increased from 500 to 2500 to check if the evolutionary algorithm can improve their performance with a wider search. The algorithms run are the genetic algorithm and the Latin Hypercube sampling. In this case the genetic algorithm can find better solutions than the same algorithm before whereas the Latin Hypercube sampling does not have any significant change. The final experiment is performed with a mixed method with the combination of coverage and evolutionary algorithms. This mixed method is able to improve a little the solution compared to the previous methods.

To sum up, for the scenario tested the use of coverage approaches is a better option than the evolutionary algorithms as they are easier to implement and the solutions that they find are similar or better than the ones found by the other algorithms. The other possibility is a mixed method which starts as a coverage approach and continues as an evolutionary algorithm. This mixed method is more complex to develop and configure but their results are something better than the ones obtained with other methods.

After the method development, comes the question about the future improvements and the possible applications that it may have. Regarding the improvements it would be interesting to increase the number of input parameters. This would allow having a customized and very detailed scenario.

This method can also have several applications in the world of automated vehicles and validation. One option is the comparison of two virtual models. For example, it could be applied to a vehicle that is modelled with two different techniques giving two different models. Then with the comparison test, the specific scenarios where the two virtual models behave more differently can be found. And it can be checked why the difference is so big in that specific case. The two virtual models to compare can also be a new model update and its previous version.

Another application area is related to the comparison between a virtual model and the real system that the model wants to represent. In order to do this comparison, some concrete scenarios should be specified. Then a physical test with the real vehicle or system can be carried out in these specific scenarios and the model can be simulated in the scenario virtually defined. With the results it would be possible to check in which specific cases the differences are greater. Then, these specific cases can be analyzed in detail to understand the reason of this difference. Therefore, the model can be improved and become more realistic.



# List of figures

Figure 2.1: Vehicle occupancy calculation [9].....	8
Figure 2.2: Possible positions for traffic vehicles [11] .....	9
Figure 2.3: Test cases for vehicle testing [15] .....	11
Figure 2.4: Drivable area calculation [17] .....	12
Figure 2.5: Highway scenario for testing [19].....	14
Figure 2.6: Differential analysis with Reinforcement Learning structure .....	17
Figure 2.8: Evolutionary algorithms structure [25] .....	19
Figure 2.8: Roulette wheel selection process .....	21
Figure 2.9: Reinforcement Learning structure .....	26
Figure 3.1: Problem structure .....	31
Figure 3.2: Driving scenario example .....	33
Figure 3.3: Genetic algorithm performance for different parameters [43] .....	47
Figure 3.4: Code structure .....	49
Figure 3.5: Road definition in the simulation software .....	51
Figure 4.1: Scenario to simulate: Overtaking maneuver.....	55
Figure 4.2: Fitness results for Test case 2.....	60
Figure 4.3: Input 1 values distribution for Test case 2 .....	61
Figure 4.4: Input 2 values distribution for Test case 2 .....	61
Figure 4.5: Input 3 values distribution for Test case 2 .....	61
Figure 4.6: Fitness results for test case 5.....	62
Figure 4.7: Input 1 values distribution for test case 5.....	63
Figure 4.8: Input 2 values distribution for test case 5.....	63
Figure 4.9: Input 3 values distribution for test case 5.....	63
Figure 4.10: Input 1 values distribution for test case 7.....	64
Figure 4.11: Second scenario to simulate - Cut in maneuver .....	66
Figure 4.12: Fitness values for the best solutions in each simulation run.....	70
Figure 4.13: Input 1 values for the best solutions in each simulation run .....	71
Figure 4.14: Input 2 values for the best solutions in each simulation run .....	72

Figure 4.15: Input 3 values for the best solutions in each simulation run ..... 72

Figure 4.16: Input 2 values distribution for Genetic Algorithm ..... 73

Figure 4.17: Input 3 values distribution for Genetic Algorithm ..... 73

Figure 4.18: Multimodal function [47]..... 74

Figure 4.19: Fitness value evolution through several generations for the Evolution Strategies 75

Figure 4.20: Fitness value evolution through several generations for genetic algorithm ..... 75

Figure 4.21: Ego-vehicle speed profile during the simulations of both systems ..... 76

Figure 4.22: Fitness evolution for GA with a manually introduced point in the initial population  
..... 78

Figure 4.23: Fitness results comparison between algorithms with 500 and 2500 simulations . 79

Figure 4.24: Fitness values evolution for mixed algorithm ..... 80



# List of tables

Table 2.1: Scenario generation papers summary ..... 16

Table 3.1: Three methods evaluation for scenario generation case ..... 36

Table 3.2: Possibilities for Error estimation to use on fitness function ..... 42

Table 3.3: End condition and Algorithm Quality ..... 44

Table 4.1: Genetic algorithm parameters for test cases ..... 58

Table 4.2: Best solution found at each case’s simulation ..... 59

Table 4.3: Results for the best point found ..... 76



# Bibliography

## Bibliography list

- [1] ACEA Report, „Vehicles in use Europe 2018“, 2018.
- [2] Bundesanstalt für Straßenwesen, „Traffic and Accident Data“, 2018.
- [3] National Highway Traffic Safety Administration and U.S. Department of Transportation, „Crash Stats: Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey“, 2015.
- [4] J. Dokic, B. Müller, G. Meyer, „European Roadmap Smart System for Automated Driving“, *EPoSS*, 2015.
- [5] Roger Lancot, „Accelerating the Future: The Economic Impact of the Emerging Passenger Economy“, 2017.
- [6] A. Broggi, P. Cerri, S. Debattisti, M. C. Laghi, P. Medici, D. Molinari, M. Panciroli, A. Prioletti, „PROUD—Public Road Urban Driverless-Car Test“, *IEEE Trans. Intell. Transport. Syst.*, Jg. 16, Nr. 6, S. 3508–3519, 2015.
- [7] H. Winner, W. Wachenfeld, „Absicherung automatischen Fahrens“, *TU Darmstadt*, 2013.
- [8] N. Kalra, S. M. Paddock, „Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?“.
- [9] M. Althoff und J. M. Dolan, „Reachability computation of low-order models for the safety verification of high-order road vehicle models“ in *2012 American Control Conference (ACC)*, Montreal, QC, Jun. 2012 - Jun. 2012, S. 3559–3566.
- [10] H. Beglerovic *et al.*, „Model-based safety validation of the automated driving function highway pilot“ in *Proceedings, 8th International Munich Chassis Symposium 2017*, P. P. E. Pfeffer, Hg., Wiesbaden: Springer Fachmedien Wiesbaden, 2017, S. 309–329.

- [11] L. Huang, Q. Xia, F. Xie, H.-L. Xiu und H. Shu, „Study on the Test Scenarios of Level 2 Automated Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 49–54.
- [12] S. Khastgir *et al.*, „Test Scenario Generation for Driving Simulators Using Constrained Randomization Technique“ in *SAE Technical Paper Series*, 2017.
- [13] B. Kim, A. Jarandikar, J. Shum, S. Shiraishi und M. Yamaura, „The SMT-based automatic road network generation in vehicle simulation environment“ in *Proceedings of the 13th International Conference on Embedded Software - EMSOFT '16*, Pittsburgh, Pennsylvania, 2016, S. 1–10.
- [14] Q. Xia, J. Duan, F. Gao, T. Chen und C. Yang, „Automatic Generation Method of Test Scenario for ADAS Based on Complexity“ in *SAE Technical Paper Series*, 2017.
- [15] J. Zhou und L. d. Re, „Reduced Complexity Safety Testing for ADAS & ADF“, *IFAC-PapersOnLine*, Jg. 50, Nr. 1, S. 5985–5990, 2017.
- [16] G. Bagschik, T. Menzel und M. Maurer, „Ontology based Scene Creation for the Development of Automated Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 1813–1820.
- [17] M. Althoff und S. Lutz, „Automatic Generation of Safety-Critical Test Scenarios for Collision Avoidance of Road Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 1326–1333.
- [18] H. Beglerovic, M. Stolz und M. Horn, „Testing of autonomous vehicles using surrogate models and stochastic optimization“ in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Okt. 2017 - Okt. 2017, S. 1–6.
- [19] S. Hallerbach, Y. Xia, U. Eberle und F. Koester, „Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles“, *SAE Intl. J CAV*, Jg. 1, Nr. 2, S. 93–106, 2018.
- [20] M. Koren, S. Alsaif, R. Lee und M. J. Kochenderfer, „Adaptive Stress Testing for Autonomous Vehicles“ in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Jun. 2018 - Jun. 2018, S. 1–7.
- [21] F. Lindlar, „Modellbasierter evolutionärer Funktionstest“, 2012.
- [22] C. E. Tuncali, T. P. Pavlic und G. Fainekos, „Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles“, Brazil, 2016, S. 1470–1475.

- [23] C. E. Tuncali, S. Yaghoubi, T. P. Pavlic und G. Fainekos, „Functional gradient descent optimization for automatic test case generation for vehicle controllers“ in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an, Aug. 2017 - Aug. 2017, S. 1059–1064.
- [24] R. Lee *et al.*, „Differential Adaptive Stress Testing of Airborne Collision Avoidance Systems“ in *2018 AIAA Modeling and Simulation Technologies Conference*, Kissimmee, Florida, 01082018, S. 277.
- [25] T. Weise, „Global Optimization Algorithms - Theory and Application“, S. 95–99, 2009.
- [26] U. Bodenhofer, „Genetic Algorithms: Theory and Applications“, S. 18–23, 2000.
- [27] C. R. Reeves, „Genetic Algorithms“ in *International Series in Operations Research & Management Science, Handbook of Metaheuristics*, M. Gendreau und J.-Y. Potvin, Hg., Boston, MA: Springer US, 2010, S. 109–139.
- [28] C. Z. Janikowc und Z. Michalewicz, „An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms“, *University of North Carolina*, 1991.
- [29] „The Continuous Genetic Algorithm“ in *Practical Genetic Algorithms*, R. L. Haupt und S. E. Haupt, Hg., Hoboken, NJ, USA: John Wiley & Sons, Inc, 2003, S. 51–66.
- [30] O. Kramer, „Evolution Strategies“ in *SpringerBriefs in Applied Sciences and Technology, A Brief Introduction to Continuous Evolutionary Optimization*, O. Kramer, Hg., Cham: Springer International Publishing, 2014, S. 15–26.
- [31] *A Visual Guide to Evolution Strategies*. [Online] Verfügbar unter: <http://blog.otoro.net/2017/10/29/visual-evolution-strategies/>. Zugriff am: Mai. 15 2019.
- [32] *A Beginner's Guide to Deep Reinforcement Learning*. [Online] Verfügbar unter: <https://skymind.ai/wiki/deep-reinforcement-learning>. Zugriff am: Mai. 15 2019.
- [33] L. P. Kaelbling, M. L. Littman und A. W. Moore, „Reinforcement Learning: A Survey“, *jair*, Jg. 4, S. 237–285, 1996.
- [34] F. S. Hillier, E. A. Feinberg und A. Shwartz, *Handbook of Markov Decision Processes: Methods and Applications*. Boston: Springer US, 2002, p. 1-14.
- [35] H. Mao, M. Alizadeh, I. Menache und S. Kandula, „Resource Management with Deep Reinforcement Learning“ in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks - HotNets '16*, Atlanta, GA, USA, 2016, S. 50–56.

- [36] I. Arel, C. Liu, T. Urbanik und A. G. Kohls, „Reinforcement learning-based multi-agent system for network traffic signal control“, *IET Intell. Transp. Syst.*, Jg. 4, Nr. 2, S. 128, 2010.
- [37] J. M. Pierre, „End-to-End Deep Learning for Robotic Following“ in *Proceedings of the 2018 2nd International Conference on Mechatronics Systems and Control Engineering - ICMSCE 2018*, Amsterdam, Netherlands, 2018, S. 77–85.
- [38] Z. Zhou, X. Li und R. N. Zare, „Optimizing Chemical Reactions with Deep Reinforcement Learning“ (eng), *ACS central science*, Jg. 3, Nr. 12, S. 1337–1344, 2017.
- [39] [www.pegasusprojekt.de](http://www.pegasusprojekt.de), „Scenario Description: Requirements & Conditions – Stand 4“.
- [40] M. D. McKay, R. J. Beckman und W. J. Conover, „A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code“, *Technometrics*, Jg. 21, Nr. 2, S. 239, 1979.
- [41] O. Boyabatli and I. Sabuncuoglu, „Parameter Selection in Genetic Algorithms“, *Institutional Knowledge at Singapore Management University*, 2004.
- [42] A. E. Eiben und S. K. Smit, „Evolutionary Algorithm Parameters and Methods to Tune Them“ in *Autonomous Search*, Y. Hamadi, E. Monfroy und F. Saubion, Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 15–36.
- [43] K. Deb and S. Agrawal, „Understanding Interactions among Genetic Algorithm Parameters“, *Kanpur Genetic Algorithms Laboratory (KanGAL)*, 1999.
- [44] S. Gotshal and B. Rylander, „Optimal Population Size and the Genetic Algorithm“.
- [45] A. Laoufi, S. Hadjeri, A. Hazzab, „Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms for power economic dispatch“, *International Journal of Applied Engineering Research*, 2006.
- [46] M. Preuss, *Multimodal optimization by means of evolutionary algorithms*. Cham: Springer, 2015, p. 11-12.
- [47] A. Zeblah, *Multimodal function*. [Online] Verfügbar unter: [https://www.researchgate.net/figure/Multimodal-function-see-online-version-for-colours\\_fig2\\_223708478](https://www.researchgate.net/figure/Multimodal-function-see-online-version-for-colours_fig2_223708478). Zugriff am: Mai. 15 2019.

# Appendix

Appendix A	Result plots for overtake scenario .....	xi
Appendix B	Result plots for cut in scenario .....	xvii





# Appendix A Result plots for over-take scenario

Some of the results obtained in the first scenario simulation are shown next:

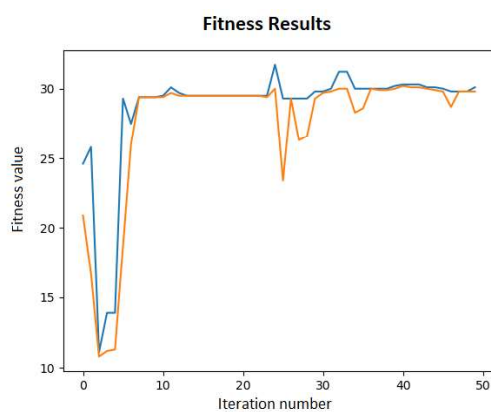


Figure A 1: Fitness for test case 2

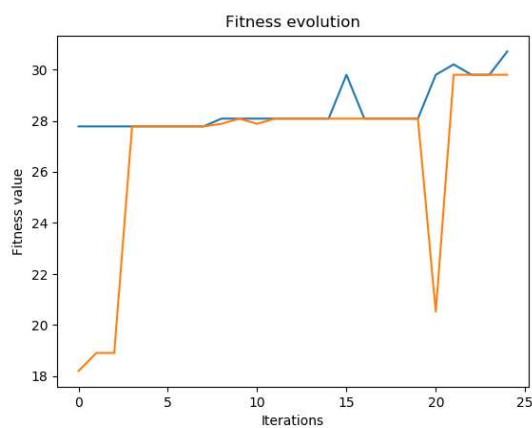


Figure A 2: Fitness for test case 3

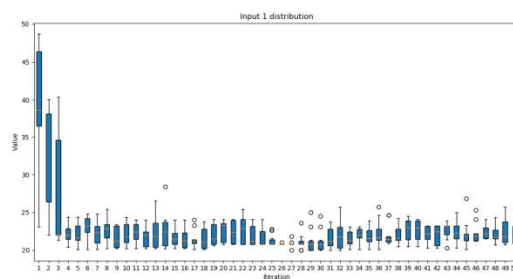


Figure A 3: Input 1 distribution for test case 4

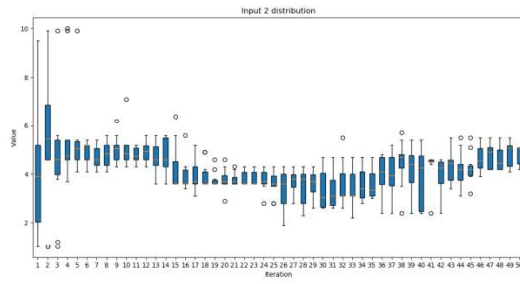


Figure A 4: Input 2 distribution for test case 4

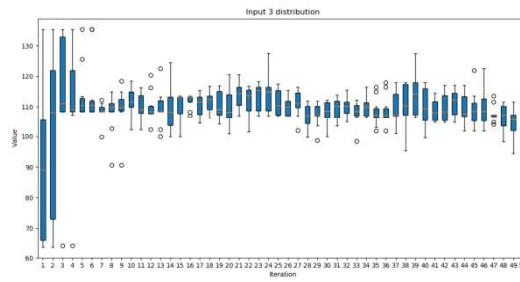


Figure A 5: Input 3 distribution for test case 4

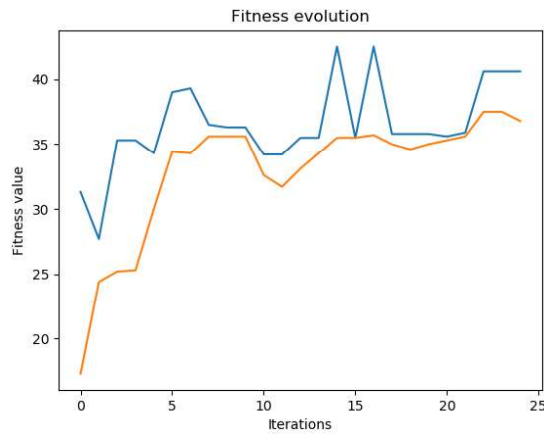


Figure A 6: Fitness for test case 5

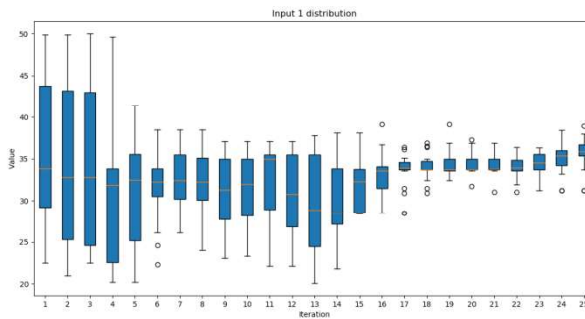


Figure A 7: Input 1 distribution for test case 5

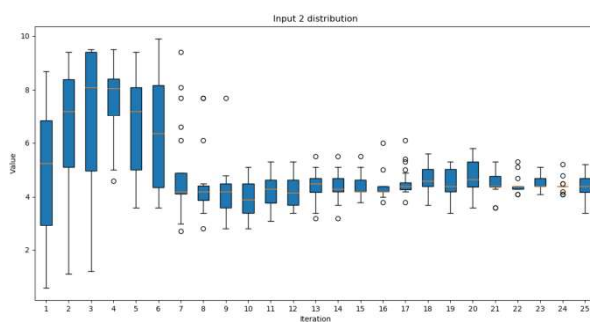


Figure A 8: Input 2 distribution for test case 5

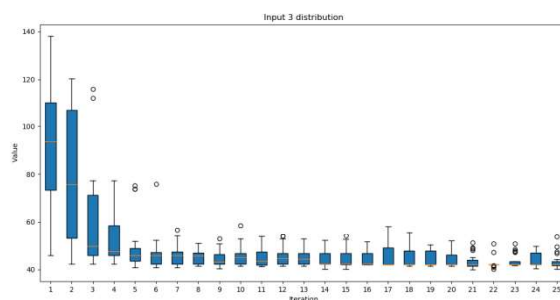


Figure A 9: Input 3 distribution for test case 5

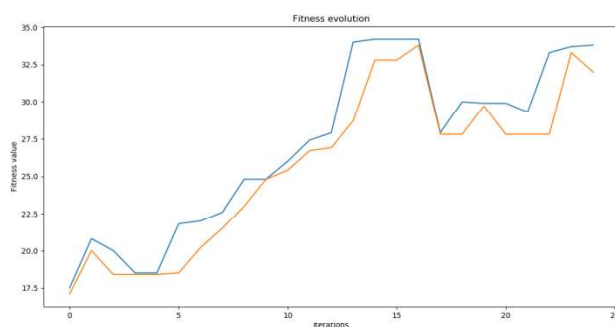


Figure A 10: Fitness for case 6

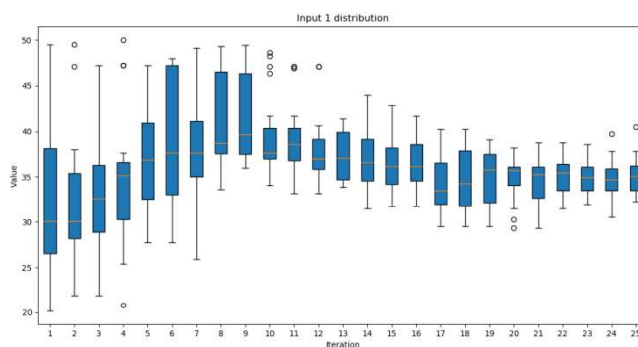


Figure A 11: Input 1 distribution for case 6

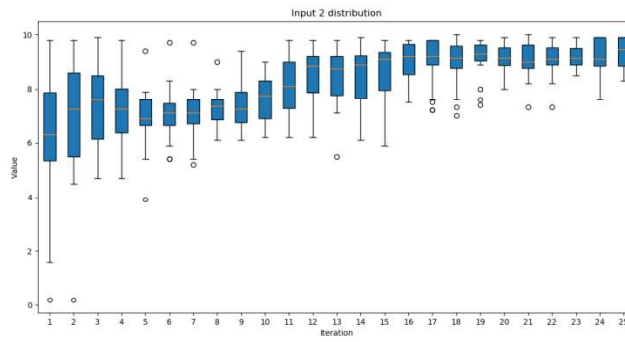


Figure A 12: Input 2 distribution for case 6

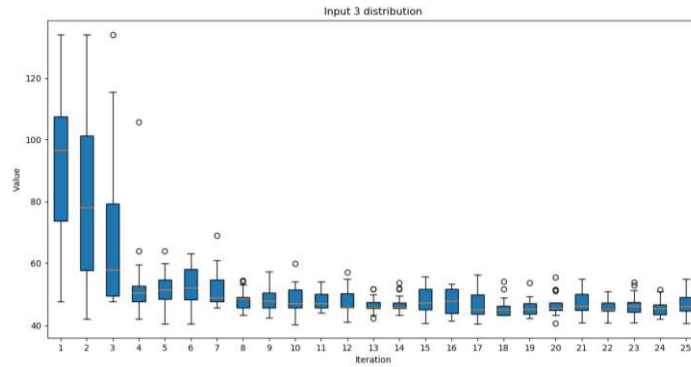


Figure A 13: Input 3 distribution for case 6

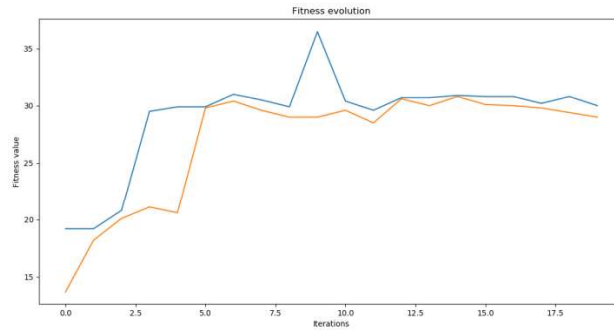


Figure A 14: Fitness evolution for case 7

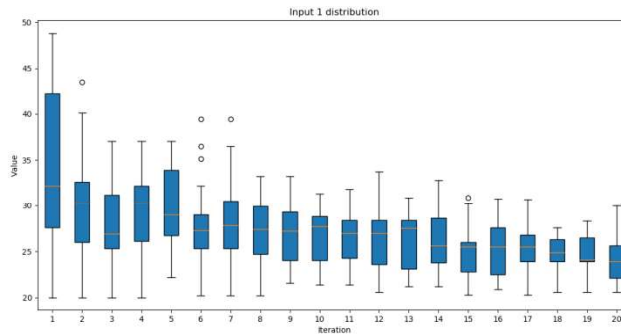


Figure A 15: Input 1 distribution for case 7

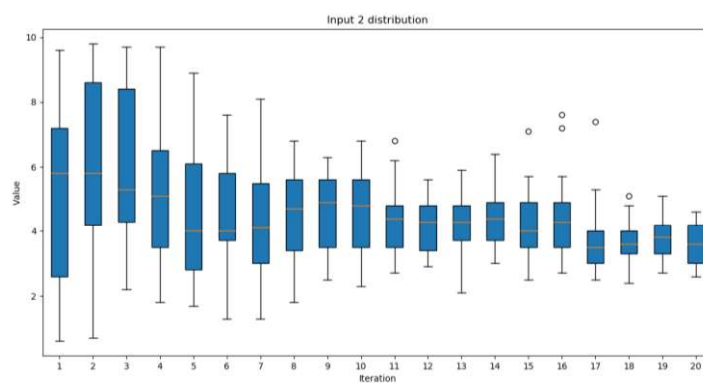


Figure A 16: Input 2 distribution for case 7

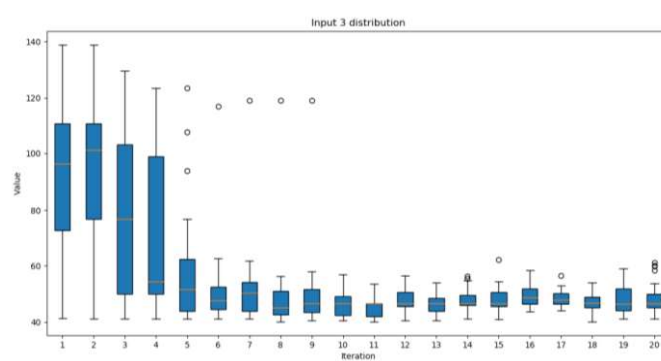


Figure A 17: Input 3 distribution for case 7



## Appendix B Result plots for cut in scenario

This subchapter contains graphs obtained during the simulations of the second scenario.

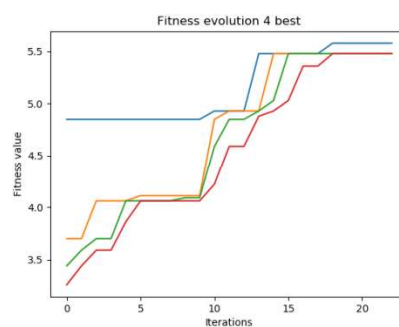


Figure B. 1: Fitness evolution with ES

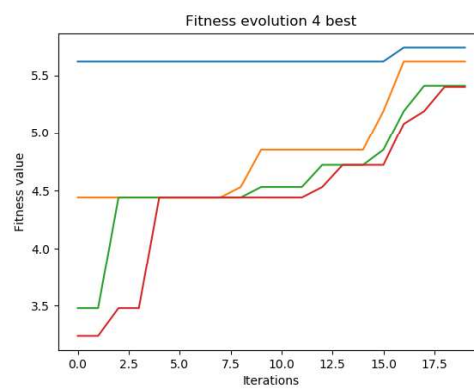


Figure B. 2: Fitness evolution with ES (2)

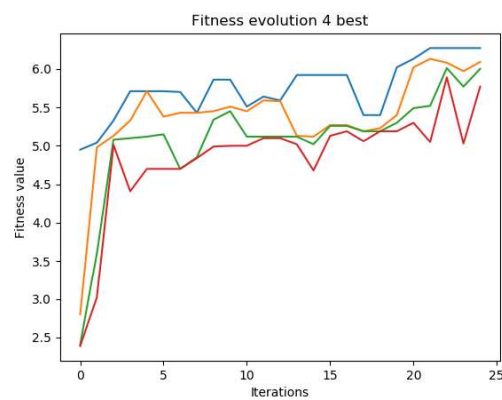


Figure B. 3: Fitness evolution with GA

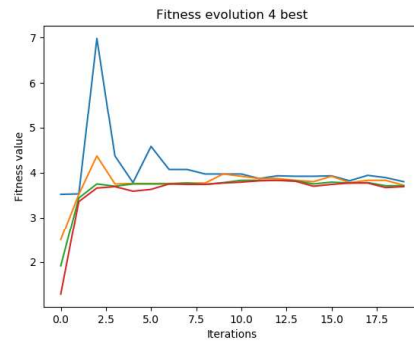


Figure B. 4: Fitness evolution with GA (2)

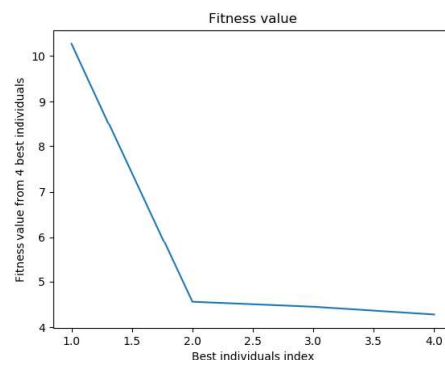


Figure B. 5: Fitness evolution with LHS

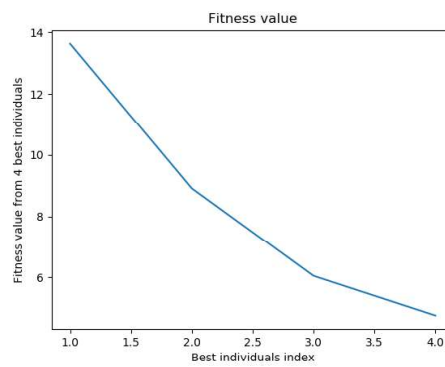


Figure B. 6: Fitness evolution with MC

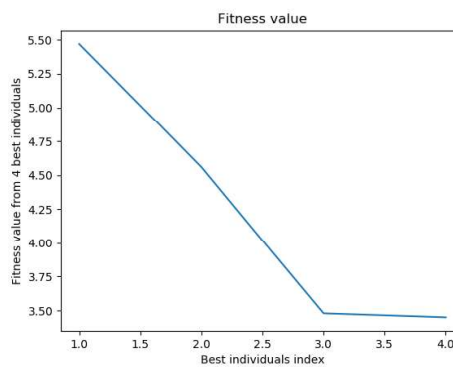


Figure B. 7: Fitness evolution with MC (2)