Matthias RAINER

A Genetic Algorithm for Mixed-integer Multicriteria Optimization Problems and its Application to Engines in order to Optimize Fuel Consumption and Driving Performance

MASTER THESIS

written to obtain the academic degree of a Master of Science (MSc)

Master Programme Mathematical Computer Science

Graz University of Technology

Supervisor:
Ao.Univ.-Prof. Dr.techn. Bettina KLINZ

Institute of Mathematics B

Graz, February 2012

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....	....
date	(signature)

Acknowledgements

First, I want to thank AVL List GmbH. for providing me with the opportunity to write my master thesis on a very interesting topic. Special thanks are due to my colleagues at AST in Graz and Zagreb for their help when I struggled with problems.
Furthermore, I want to express my gratitude to Prof. Bettina Klinz from the TU Graz and Dr. Günter Offner from AVL for their great and extensive supervision of my work.
Finally, I want to thank my parents and Monika for their support and assistance.

Content

1 Introduction

The reduction of fuel consumption and thus CO2 emission of vehicles is an important topic from the environmental point of view. For this reason, for instance the European Union and California enforce regulations to limit the CO2 emission of vehicles since several years. In addition, customers increasingly request vehicles with low fuel consumption due to environmental reasons and increasing costs for fuel.
Therefore, fuel consumption and emissions of vehicles have to be minimized while the driving performance should stay at a high level. This can be achieved by optimizing the various subsystems of a vehicle, like the engine and the transmission. As there are many interdependent parameters to be considered, mathematical optimization is a powerful methodology to achieve the optimization targets within a short development time. This saves the effort of testing real vehicles and thus the resulting development costs.

"Optimization" means to find the best solution for a problem, such that defined constraints are satisfied. There are several ways to find an optimum. The least successful one is to guess a solution, as this way in general fails.
The best way would be to use an algorithm, which computes the exact optimum. For most practical optimization problems this is not possible due to unacceptable high computational effort. To overcome this obstacle, one applies heuristic approaches which often lead to very good, not yet optimal solutions.

The Genetic Algorithms, which are used in this thesis, are special meta-heuristics to find solutions close to the global optimum of a function with constraints, which have to be satisfied. In general, the result will not be the exact optimum in meta-heuristics, because the algorithm cannot guarantee, whether there exists a solution closer to the global optimum than the found solution. Another drawback is, that there exists no provable quality of the solution.
For example a provable quality for an algorithm, which attacks minimization problems containing only positive objective values, could be, that the solution of the algorithm is at most the optimal solution times a quality factor, which is greater than one. Compared to such approximation algorithms with provable quality, meta-heuristics are in general easier to develop, because there is no provable quality necessary. Due to the lack of such a proof, the solution of the algorithm can be quite far away from the optimal solution.
For more information about approximation algorithms and heuristics refer to Gonzalez [28].

A member of the class of Genetic Algorithms is the Non-dominated Sorting Genetic Algorithm II. This algorithm was implemented by Scheucher [55] in the optimization software AVL Design Explorer. The limitation of the implementation is, that the algorithm can handle only optimization problems containing continuous design variables. As the optimization problems performed with AVL Design Explorer need to consider integer design variables as well, it is necessary to extend the implemented algorithm to handle integer and mixed integer optimization problems within this thesis. To verify these extensions, optimization instances are generated using the vehicle simulation software AVL CRUISE. The obtained results with the extended mixed-integer genetic algorithm are promising and it can be concluded that it pays off to use integer variables where they are natural within the model.

1.1 Outline

The introduction to this thesis is done in Section 1 and the basic notions like single and multi-objective optimization problems from optimization theory, which are needed in the thesis, are introduced in Section 2. The genetic algorithms Non-dominated Sorting Genetic Algorithm I and II and a number of different crossover and mutation operators are presented in Section 3. Section 4 outlines the simulation software AVL CRUISE and the optimization software AVL Design Explorer. Section 5 deals with the implementation of the mixed-integer version of the Non-dominated Sorting Genetic Algorithm II within the framework of the AVL Design Explorer. Section 6 reports about the computational experiments which are performed with the new extended implementation. The test models and test instances used in these experiments are created with the simulation software AVL CRUISE. Finally, Section 7 concludes the thesis with a short summary.

2 Preliminaries about Optimization Problems and Algorithms

The basic notions from optimization theory that are needed in this thesis are introduced in this section. For further explanations, problem definitions, and algorithms in addition to the following subsections refer to Rao [51].

2.1 Optimization Problems with a Single Objective Function

This thesis mainly deals with optimization problems in the multi objective case with more than one objective function. Nevertheless, the special case of a single objective function is discussed in this section.
The problem consists of one objective function f and a set X of feasible solutions in the single objective case. Without loss of generality, the goal of the optimization problem is assumed to minimize the objective function f. This leads to the following generic problem.

Definition 2.1 (Generic Single Objective Optimization Problem)
The optimization problem

$$\begin{array}{lll} (GP) & \min\limits_{x} & f(x) \\ & s.t. & x \in X \end{array}$$

is referred to as ***generic single objective optimization problem*** *(GP).*
x is called the vector of ***decision*** *or* ***design variables****.*

It is assumed that the set X of feasible solutions is given by a set of inequality and equality constraints. This results in the following standard form for single objective optimization problems.

Definition 2.2 (Single objective optimization program in standard form)
The optimization problem

$$\begin{array}{llll} (SOP) & \min\limits_{x} & f(x) & \\ & s.t. & g_j(x) \geq 0, & j = 1, \ldots, k_s \\ & & h_i(x) = 0, & i = 1, \ldots, k_e \\ & & x^l \leq x \leq x^u & \end{array}$$

is referred to as ***single objective optimization program in standard form*** *(SOP). The functions f, g_j, $j = 1, \ldots, k_s$, and h_i, $i = 1, \ldots, k_e$ are given functions from $\mathbb{R}^n$ to $\mathbb{R}$ and $x^l \in \mathbb{R}^n$ and $x^u \in \mathbb{R}^n$ are given lower and upper bound vectors for the design variable vector x.*

The standard form of Definition 2.2 can be assumed without loss of generality for the following reason.

Note 2.1
A maximization problem can be transformed into minimization by multiplying the objective function by (-1). Likewise, inequalities of the type $g_j(x) \leq 0$ can be turned into $-g_j(x) \geq 0$.

The k_e equality constraints could be removed from the problem because two inequality constraints can replace one equality constraint:

$$h_i(x) = 0 \Leftrightarrow \begin{cases} h_i(x) \leq 0 & \text{and} \\ h_i(x) \geq 0 \end{cases}$$

This transformation is, however, not recommendable in practice.

The following definitions are standard for optimization problems.

Definition 2.3 (Feasibility)
A vector $x \in X$ is called ***feasible solution****. If $x \notin X$, it is referred to as* ***infeasible****.*
An optimization problem with $X = \emptyset$ is called infeasible, otherwise feasible.

The only way to convert an infeasible into a feasible problem is to modify one or more constraints to avoid the conflict between them.

Some of the feasible solutions in X can be better than others. This leads to the following definitions of optimality:

Definition 2.4 (Global Optimum)
A solution vector $x^1 \in X$ is a ***(global) minimum*** *for the generic single objective optimization problem, if*

$$f(x^1) \leq f(x) \quad \forall x \in X$$

Definition 2.5 (Local Optimum)
A solution vector $x^1 \in X$ is a ***local minimum*** *for the generic single objective optimization program, when there is no feasible solution $x^2 \in X$ in an ε-neighborhood $B_\varepsilon(x^1)$ of x^1 such that $f(x^2) < f(x^1)$.*
An open ball $B_\varepsilon(p)$ with the center $p \in \mathbb{R}^n$ and radius $\varepsilon \in \mathbb{R}$ is defined as

$$B_\varepsilon(p) = \{x \in X : d(x,p) < \varepsilon\}$$

where $d(x,p)$ is the distance between the point $x \in X$ and the center p of the ball.

Definition 2.6 (Local Search)
In ***local search algorithms*** *a neighborhood relation for the solutions in X has to be defined. The algorithm starts with an often randomly chosen candidate of X which is iteratively replaced by one of its neighbors. A solution for the next iteration will be chosen by taking the best solution in the neighborhood if it is better than the current one. The algorithm terminates, when there does not exist a better solution in the current neighborhood (the algorithm terminates at a* ***local minimum****) or after a predefined number of iterations.*

In some cases the obtained local minimum can by chance be the global one, but in general this kind of algorithm gets stuck in a local minimum.

Regarding Definition 2.2 sometimes the first two sets of constraints, the inequality and the equality constraints, are ignored and it is only dealt with the bounds $x^l \leq x \leq x^u$. Accordingly the following can be defined.

Definition 2.7 (Design Space)
*The set $X^d = \{x \in \mathbb{R}^n : x^l \leq x \leq x^u\}$ is called **design space**.*
*The set X of all feasible solutions from X^d is also referred to as **feasible design space**.*

Note that in this thesis the feasible design space X consists of the following set of solutions.

$$\begin{aligned} X = \{x \in \mathbb{R}^n : \; & g_j(x) \geq 0, \; j = 1, \ldots, k_s, \\ & h_i(x) = 0, \; i = 1, \ldots, k_e, \\ & x^l \leq x \leq x^u\} \end{aligned}$$

In the standard form (Definition 2.2) only the case of continuous variables was discussed. Some practical applications, however, require some or all of the design variables to take integer values. Thus, depending on the nature of the involved variables, one distinguishes the following classes of optimization problems.

Definition 2.8
The three different classes of optimization problems related to the required types of the design variables are the following:

- *In **continuous** optimization problems all design variables are continuous.*
- *In (pure) **integer** optimization problems all design variables are required to be integers.*
- *In **mixed integer** optimization problems some design variables are continuous and others are required to be integers.*

In this thesis the focus is on solving mixed integer optimization problems.

2.2 Optimization Problems with Multiple Objective Functions

In many practical applications there does not exist a single objective function to be optimized but several criteria that need to be taken into account.
For this reason, a multi-objective optimization problem can be derived from the single objective problem SOP in Definition 2.2 by replacing the single objective function by a vector of m objective functions

$$F(x) = [f_1(x), f_2(x), \ldots, f_m(x)]^T$$

where each $f_i(x)$ is a function from $\mathbb{R}^n$ to $\mathbb{R}$. This results in an objective function $F(x)$ from $\mathbb{R}^n$ to $\mathbb{R}^m$.

Definition 2.9 (Multi-objective Optimization Program in Standard Form)
The optimization problem

$$\begin{array}{llll} (MOP) & \text{“}\min\limits_{x}\text{”} & F(x) & \\ & s.t. & g_j(x) \geq 0, & j = 1, \dots, k_s \\ & & h_i(x) = 0, & i = 1, \dots, k_e \\ & & x^l \leq x \leq x^u & \end{array}$$

*is referred to as **standard form for multi-objective optimization problems** (MOP). Let again X denote the feasible design space as used in the single objective case. The meaning of “*min*” will be explained below.*

Constraints and bounds stay as they are in the single objective case. They are independent of the number of objective functions.

Definition 2.10 (Objective Function Vector)
*The **objective (function) vector** of $x^1 \in X$ in the single objective case is the **objective value** $f(x^1)$. In a multi-objective optimization problem the objective function vector of x^1 is given by $F(x^1) = [f_1(x^1), f_2(x^1), \dots, f_m(x^1)]^T$.*

In most cases there will not exist a single solution that is optimal for each of the m objective functions. This conflict is referred to as **objective conflict** by Eschenauer [20]. Due to this conflict a new concept of optimality is needed for the multi-objective case. In other words the minimum “ min ” in Definition 2.9 has to be defined.
The following definitions of Pareto-optimal points and a Pareto set (refer to Pareto [48]) is one possibility how to define “ min ”.

Definition 2.11 (Pareto Optimality)
*The objective vector $F(x^1) = [f_1(x^1), \dots, f_m(x^1)]^T$ of a solution $x^1 \in X$ is a **Pareto-optimal point** (a Pareto-optimal objective vector) for the problem MOP, when there is no vector $x^2 \in X$ with the following properties:*

- $f_i(x^2) \leq f_i(x^1)$ *for all* $i = 1, \dots, m$.
- *There exists at least one* $j \in \{1, \dots, m\}$ *with* $f_j(x^2) < f_j(x^1)$.

*The set of all solutions with Pareto-optimal objective vectors is called the **Pareto set**.*

Along the lines of the definition of the Pareto set for the best solutions, also a classification of all other solutions can be obtained. These solutions can be classified regarding mutual domination or non-domination. By means of the following approach, the solutions can be split up into sets from the set of the best to the set of the worst solutions.

Definition 2.12 (Non-domination)
When $x^1 \in X$ and $x^2 \in X$ are two solutions of problem MOP, then x^1 dominates x^2, in symbols $x^1 \succ x^2$, when the following properties hold:

1. *$f_i(x^1) \leq f_i(x^2)$ for all $i = 1, \ldots, m$.*
2. *There exists at least one $j \in \{1, \ldots, m\}$ with $f_j(x^1) < f_j(x^2)$.*

*A solution $x^1 \in X$ is **non-dominated**, when no solution $x^2 \in X$ dominates x^1. This set of non-dominated solutions is equivalent to the Pareto set.*

Two solutions $x^1 \in X$ and $x^2 \in X$ of an optimization problem have one of the following relations to each other:

- $x^1 \succ x^2$, x^1 dominates x^2.
- $x^2 \succ x^1$, x^2 dominates x^1.
- x^1 and x^2 are not comparable according to non-domination.

In the Genetic Algorithms introduced in Section 3 the idea of non-domination is used for the fitness assignment of the individuals. For the descriptions of these genetic multicriteria algorithms also the following definition of the "Pareto set" of a subset of X is needed.

Definition 2.13
*For a subset $A \subset X$ the solutions $x \in A$ which are not dominated by any other solution in A are called **A-efficient solutions**.*

Some examples for the size of Pareto sets are presented in the following.
The Pareto set of an optimization problem can be empty. That means that for each solution $x^1 \in X$ there exists another solution $x^2 \in X$ such that x^2 dominates x^1. One example for such a problem is given by Göpfert and Nehse [29]:

Example 2.1
The Pareto set of the optimization problem with the objective function $F(x) = [x_1, x_2]^T$ and the following feasible design space X is empty:

$$\begin{aligned} X = \{(x_1, x_2) \in \mathbb{R}^2 : \quad & -1 \leq x_1 \leq 1, & \\ -\sqrt{-x_1^2 + 1} < & x_2 \leq 0, & \text{if } -1 \leq x_1 \leq 0, \\ -\sqrt{-x_1^2 + 1} \leq & x_2 \leq 0, & \text{if } 0 < x_1 \leq 1\} \end{aligned}$$

This example can be easily changed to one with a non-empty Pareto set by appending an additional constraint and slightly modifying two others:

Example 2.2
The Pareto set of the new optimization problem with the objective function $F(x) = [x_1, x_2]^T$ and the following feasible design space X includes the two points (0,-1) and (-1,0):

$$
\begin{aligned}
X = \{(x_1, x_2) \in \mathbb{R}^2 : \quad & -1 \leq x_1 \leq 1, & \\
& x_2 = 0, & \text{if } x_1 = -1, \\
-\sqrt{-x_1^2 + 1} & < x_2 \leq 0, & \text{if } -1 < x_1 < 0, \\
-\sqrt{-x_1^2 + 1} & \leq x_2 \leq 0, & \text{if } 0 \leq x_1 \leq 1\}
\end{aligned}
$$

For a general theorem, whether the Pareto set of a specific optimization problem is non-empty, the definition of semi-continuous functions is needed (refer to Ehrgott [18]):

Definition 2.14 (Semi-continuous)
*A function F from $\mathbb{R}^n$ to $\mathbb{R}^m$ is $\mathbb{R}^m_{\geq 0}$-**semi-continuous** if*

$$F^{-1}(y - \mathbb{R}^m_{\geq 0}) = \{x \in \mathbb{R}^n : y - F(x) \in \mathbb{R}^m_{\geq 0}\}$$

*is **closed** for all $y \in \mathbb{R}^m_{\geq 0}$ where*

$$\mathbb{R}^m_{\geq 0} = \{y \in \mathbb{R}^m : \ y_i \geq 0, \ \forall\, i = 1, \ldots, m\}$$

Theorem 2.1 (Hartley [31])
When the feasible design space $\emptyset \neq X \subset \mathbb{R}^n$ of problem MOP is compact and F is $\mathbb{R}^m_{\geq 0}$-semi-continuous, then there exists a non-empty Pareto set.

Another important fact is that the Pareto set possibly consists of an exponential number of feasible solutions. The following optimization problem is an example for a problem with such a Pareto set (refer to Ehrgott [18]).

Example 2.3

$$
\begin{aligned}
\min_x \quad & [x, -x]^T \\
s.t. \quad & x_i \leq 1, \quad i = 1, \ldots, n \\
& -x_i \leq 1, \quad i = 1, \ldots, n
\end{aligned}
$$

The feasible design space of this problem is $X = [0, 1]^n$. One can see that all of the 2^n extreme points of the feasible design space are included in the Pareto set.

Due to the possibility for very large Pareto sets it is in general not possible to compute the whole set within polynomial time.

2.3 Exact Optimization Algorithms and Heuristics

The most ambitious goal for an optimization algorithm is to provide an exact optimal solution for all instances within a reasonable short time. For many practical applications the demand for efficient exact optimization algorithms is too high.
One way out of this dilemma is to be content with solutions that are reasonable good in most cases, but not necessarily optimal and possibly quite far away from the optimal solution. A class of algorithms with these properties is called **heuristic**.

For complex problems it is often not easy to develop problem specific heuristics. Such situations are the typical area of applications for so-called meta-heuristics like the Genetic Algorithms introduced in Section 2.5.

Definition 2.15 (Meta-heuristic)
Meta-heuristics *are not problem specific heuristic algorithms and can be applied to a huge number of problem classes.*

Most meta-heuristics include some stochastic part. Typically the idea behind using randomness is to overcome local optima because in general global and local optima do not coincide. Exceptions for that are linear problems, where each constraint and the objective function are linear. A more general class of optimization problems, where the local optimum coincides with the global optimum are convex optimization problems (refer to Boyd and Vandenberghe [5]), where all occurring functions (objective functions and constraints) are convex.

Definition 2.16 (Convex Functions)
A function $f(x)$ from $\mathbb{R}^n$ to $\mathbb{R}$ is called ***convex****, when the following is satisfied for all pairs of points $(x^1, x^2) \in \mathbb{R}^n \times \mathbb{R}^n$ and for all $\alpha \in [0,1]$:*

$$f(\alpha \cdot x^1 + (1-\alpha)\cdot x^2) \leq \alpha \cdot f(x^1) + (1-\alpha)\cdot f(x^2)$$

A function is called ***strictly convex*** *if*

$$f(\alpha \cdot x^1 + (1-\alpha)\cdot x^2) < \alpha \cdot f(x^1) + (1-\alpha)\cdot f(x^2)$$

A convex optimization problem has only one local minimum, which is coincidental the global optimum.

For optimization problems without this property, global optimization algorithms are needed. A well known global optimization algorithm for integer optimization problems is the Branch and Bound algorithm (refer to Land and Doig [39]). For more information about global optimization algorithms refer to Horst and Tuy [35].

Due to the in general different local and global optima outside the class of linear or convex optimization problems it becomes important to be able to escape from a local optimum in which classical local search algorithms easily get stuck.

Prominent meta-heuristics are the following:

- Simulated Annealing (Kirkpatrick et al. [37])
- Tabu Search (Glover [24])
- Genetic Algorithms (Holland [34] and Deb [7])
- Other types of Evolutionary Algorithms (Dréo et al. [17])
- Ant Colony Algorithms (Dorigo et al. [15])

Genetic Algorithms play a special role in this thesis and are thus described in more detail in Section 2.5. A wide range of exact global optimization algorithms and further meta-heuristics were described by Pardalos and Romeijn [47] and Dréo et al. [17].

Due to the fact that a meta-heuristic does not know whether the true optimal solution has been found, a meta-heuristic needs stopping conditions. Two examples for stopping conditions are:

- Stop after a given maximum number of iterations.
- Stop after no or a too small improvement of the solutions in the last few iterations.

As mentioned before, due to these stopping conditions there is no reliability that the resulting solutions are close to the true optimal vectors when the algorithm terminates.

2.4 Approaches to Multi-objective Optimization Problems

There exist different approaches how to attack multi-objective optimization problems, refer to Ehrgott [18]. Approaches which do not treat multi-objective optimization problems as true multi-objective problems are briefly dealt with in this section. These approaches either turn the problem into a single objective one, like in the Weighted Sum Method and the ε-Constrained Method, or prescribe an order on the objective functions, as done in the Lexicographic Approach. Methods where multiple objective functions are turned into a single objective one are called scalarization methods. A second example for non-scalarization methods besides the Lexicographic Approach which is shown in this section is the Max-ordering optimization.
Approaches, where the multi-objective optimization problem keeps its multiple objectives, are introduced in Section 2.4.5 and 2.5.

2.4.1 Weighted Sum Method

A scalarization approach is the **Weighted Sum Method**, where each single objective function $f_i(x)$ is multiplied by a weighting factor ε_i. These terms are summed up leading to a single objective function $f(x)$.

$$f(x) = \varepsilon_1 \cdot f_1(x) + \cdots + \varepsilon_m \cdot f_m(x)$$

The weighting factors have to satisfy:

$$\sum_{i=1}^{m} \varepsilon_i = 1$$

A drawback of this method is that the resulting solution depends on the choice of the weights $\{\varepsilon_1, \ldots, \varepsilon_m\}$ (refer to Srinivas and Deb [59]). Another disadvantage of this approach is that the result consists of only one solution, while in multi-objective settings the decision maker usually prefers to be able to choose from a set of alternatives.
The advantage of such scalarization methods is the possibility to use the same algorithms as for single objective optimization problems.

2.4.2 ε-Constraint Method

The **ε-Constraint Method** designed by Haimes et al. [30] is another scalarization method. In this approach one of the objective functions stays as objective function and all the others become constraints. Let f_α be the chosen objective function that remains. Then the ε-constraint method considers the following problem.

$$\begin{array}{ll} \min\limits_{x} & f_\alpha(x) \\ \text{s.t.} & f_l(x) \leq \varepsilon_l, \quad l = 1, \ldots, \alpha-1, \alpha+1, \ldots, m \\ & x \in X \end{array}$$

where $\varepsilon_l \in \mathbb{R}$, $l = 1, \ldots, m$, and the original bounds of the problem are not changed by this approach. The obtained solution of course depends on the choice of $\varepsilon = \{\varepsilon_1, \ldots, \varepsilon_m\}$ and on α.

2.4.3 Lexicographic Approach

The idea behind the **Lexicographic Approach** (Ehrgott [18]) to multi-objective optimization problems is to sort the objective functions with respect to their importance and then to apply the classical concept of lexicographic optimality.

Definition 2.17 (Lexicographic Order)
A vector $a \in \mathbb{R}^n$ is ***lexicographically smaller*** *than $b \in \mathbb{R}^n$, $a <_{lex} b$, when $a \neq b$ and $a_k < b_k$ with $k = \min\{i : a_i \neq b_i, i \in \{1, \ldots, n\}\}$.*

Definition 2.18 (Lexicographic Optimality)
A solution vector $x^1 \in X$ of a multi-objective optimization problem is ***lexicographically optimal*** *if $F(x^1) <_{lex} F(x^2)$ for all feasible solutions $x^2 \in X \setminus \{x^1\}$ with $F(x^1) \neq F(x^2)$.*

Applying this approach, a multi-objective optimization problem can be attacked by computing the optimal solutions of the single objective problems beginning with the most important one, i.e. the one with the most important objective function. The feasible design space for the subsequent optimization problems consists of the optimal solutions of the current one. This is done until only one solution is left in the feasible design space for the subsequent objective function or when all functions were considered. The second stopping condition results in a set of lexicographically optimal solutions.

Algorithm 1 Lexicographic Optimization [18]

```
$X_1 = X$
**for all** $k = 1, \dots, m$ **do**
  Solve the single objective optimization problem $\min_{x \in X_k} f_k(x)$
  **if** $\min_{x \in X_k} f_k(x)$ results in one optimal solution $x^1$ **then**
    STOP, $x^1$ is the optimal solution of the instance.
  **if** $\min_{x \in X_k} f_k(x)$ is unbounded **then**
    STOP, the whole problem is unbounded.
  **if** $k == m$ **then**
    STOP, $\{\hat{x} \in X_m | f_m(\hat{x}) = \min_{x \in X_m} f_m(x)\}$ is the set of lexicographically optimal solutions.
  $X_{k+1} = \{\hat{x} \in X_k | f_k(\hat{x}) = \min_{x \in X_k} f_k(x)\}$
```

In the algorithm, $X_k \subset X$ is the feasible design space for the function $f_k(x)$ composed of all solutions of X_{k-1} which are optimal according to $f_{k-1}(x)$.
For the lexicographic optimization no ε has to be chosen as in the methods before, but it is often not possible to sort the objective functions according to their importance.

2.4.4 Max-ordering Optimization

Max-ordering is another non-scalarization method (refer to Ehrgott [18]). In this approach one considers the following optimization problem:

$$\begin{aligned} \min_{x} \max_{k \in \{1,\dots,m\}} \quad & f_k(x) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

That means that the optimal solutions are the solutions where the maximum objective value over all single objective functions is minimal.

2.4.5 True Multi-objective Optimization Algorithms

There exist also algorithms for multi-objective optimization problems which are really multi-objective ones. The goal of these algorithms is to treat the objectives as true multiple objectives. Examples are the meta-heuristics introduced in section 2.3. Important meta-heuristics, especially in this thesis, are Genetic Algorithms as described in Section 2.5.

Besides meta-heuristics there exist also deterministic algorithms for the computation of solutions of the Pareto set. Most of these algorithms work only for special cases of optimization algorithms. Some examples are:

- Multi-objective linear optimization: Multicriteria Simplex Algorithm (Ehrgott [18]).
- Multi-objective integer optimization: Several approaches (Ehrgott [18], Pardalos and Romeijn [47]).
- Multi-objective knapsack problem: Branch and bound (Ehrgott [18]).

For additional examples refer to Ehrgott [18]. Furthermore, in Shukla and Deb [58] and Giesy [23] some general approaches to solve multi-objective optimization problems are presented.

2.5 Genetic Algorithms

The main ideas behind Genetic Algorithms are briefly described in this section as they play a major role in this thesis. Genetic Algorithms go back to Holland [34] and form a subclass of evolutionary algorithms (Dréo et al. [17]). Evolutionary algorithms got their name from imitating the evolutionary principles of nature by recombining and mutating solutions to get new, maybe better ones. Other Evolutionary Algorithms are for example Evolution Strategy, Evolutionary Programming, and Genetic Programming. For details regarding these methods refer to Deb [7].

There exists a variety of different Genetic Algorithms. What they all have in common are that one deals with populations of individuals, which change in each generation, and a fitness value assigned to each individual. The fitness of an individual is computed with a fitness function.

In Genetic Algorithms several solutions (individuals) are computed in each optimization step (generation) instead of only one solution as in many classical optimization algorithms. This means that the output of the algorithm is also a population of solutions from which one can choose an appropriate one, typically one of the solutions with the best fitness value.

A generic Genetic Algorithm handles one population of solutions in each generation. A fitness value is assigned to all of the individuals of the current population. The population of the subsequent generation is computed by crossover and mutation of the individuals with the best fitness values. The crossover and mutation algorithms, which are part of each evolutionary algorithm, are used for the recombination of solutions to have a possibility for better individuals in the subsequent generation. A crossover algorithm recombines two or more solutions to a new one which is mutated afterwards using a mutation algorithm. For more information regarding recombination algorithms refer to Section 3.3.

Algorithm 2 Generic Genetic Algorithm [7]

```
$g = 0$
Generate the initial population $P_0$
Assign fitness values to the individuals of $P_0$
while Stopping conditions are not met do
    Choose fittest individuals of current generation for recombination
    Perform crossover
    Perform mutation
    Choose individuals for the new generation $P_{g+1}$ from $P_g$ and the recombined ones
    Assign fitness values to the individuals of $P_{g+1}$
    $g = g + 1$
```

The stopping conditions typically involve a maximum number of generations. There exist a lot of possible additional conditions like having found a solution which satisfies a minimum criterion or the computation time reached a maximum amount of time.
The generation counter in the algorithm is g and P_g is the population in generation g.

In Section 3.1 and Section 3.2 two Genetic Algorithms are described in more detail, namely the Non-dominated Sorting Genetic Algorithms I and II.

3 Non-dominated Sorting Genetic Algorithms

The main part of this section is devoted to introducing the Non-dominated Sorting Genetic Algorithms I and II. Furthermore, some possible hybridizations of a Genetic Algorithm with other optimization algorithms are described. At the end of the section some classes of crossover and mutation operators for Genetic Algorithms to solve mixed integer optimization problems are presented.
This section focuses on the theory of the used algorithms and operators. For details about the implementation and which of the operators and algorithms were chosen in the practical experiments refer to Section 5.2.

Rosenberg [52] suggested to use genetic search for a multi-objective optimization problem in 1967. In 1984 Schaffer [54] was the first who developed a multi-objective Genetic Algorithm, the Vector Evaluated Genetic Algorithm (VEGA). The problem of VEGA is that after a lot of iterations the individuals converge towards some solutions of the Pareto set. The objective vectors of these solutions can be in a specific region so there is no diversity in the solutions.

Goldberg [25] suggested to use the idea of non-domination and Pareto-optimality to avoid the convergence problem of VEGA. One of a few algorithms which are based on Goldberg's idea [25], the Non-dominated Sorting Genetic Algorithm I, is presented in Section 3.1. Another algorithm of this type is for instance MOGA (Multi-objective Genetic Algorithm), designed by Fonseca and Fleming [22].

3.1 Non-dominated Sorting Genetic Algorithm I

The **Non-dominated Sorting Genetic Algorithm (NSGA)** will be presented in this section. This algorithm is referred to as the Non-dominated Sorting Genetic Algorithm I (NSGA-I) in this thesis for better distinction from the Non-dominated Sorting Genetic Algorithm II (NSGA-II) which is presented in Section 3.2.

Srinivas and Deb [59] developed a Genetic Algorithm with non-dominated sorting to obtain a set of different solutions with objective vectors that are hopefully close to the Pareto-optimal points. Their algorithm is referred to as Non-dominated Sorting Genetic Algorithm I. In contrast to the algorithms introduced in Section 2.4 there is no dependence on weighting factors or sorting by the designer in this algorithm.

3.1.1 Non-dominated Sorting

Non-dominated sorting was developed to rank all found solutions from good to bad ones. The fitness assignment where each individual gets a fitness value, which is used for the selection of the best individuals for the subsequent population, is based on these ranks.

For the non-dominated sorting part there exist several possible approaches. The sorting algorithm step of the NSGA-I is based on a simple, but slow approach as presented below. For more examples refer to Deb [7] and Section 3.2.1.

The non-dominated sorting algorithm classifies the N solutions of a population P into the sets $\{F_1, \dots, F_r\}$. Each F_i is a set of solutions which do not dominate each other. This is achieved by first computing the set F_1 of P-efficient solutions. After that, in F_j the P'-efficient solutions, where P' is the population P without the solutions in the sets $F_1, \dots, F_{j-1}$ are stored.

Algorithm 3 non-dominated_sort(P) [7]

```
$i = 1$
$P' = P$
**while** $P' \neq \emptyset$ **do**
  $F_i = \emptyset$
  $d = 1$
  **for all** $x^1 \in P'$ **do**
    **for all** $x^2 \in P' \setminus \{x^1\}$ **do**
      **if** $x^2 \succ x^1$ **then**          // $x^2$ dominates $x^1$
        $d = 0$
        break
    **if** $d == 1$ **then**          // no solution in $P'$ dominates $x^1$
      $F_i = F_i \cup \{x^1\}$
  $P' = P' \setminus F_i$
  $i = i + 1$
```

In this algorithm d will be set to 0 if an individual x^2 dominates the current individual x^1, and will be set to 1, if x^1 is non-dominated. P is the current population and P' is the population P reduced by the elements in the resulting sets of the iterations before. The algorithm returns $F = \{F_1, \dots, F_r\}$, where each individual in the set F_j dominates all elements in the sets $F_{j+1}, \dots, F_r$ and all solutions in F_j are dominated by the elements in $F_1, \dots, F_{j-1}$. Furthermore, no individual in F_j dominates any of the others in the same set.

For the test whether x^1 is dominated by x^2, each of the m objective functions has to be checked and the two for-loops need $\mathcal{O}(N^2)$ steps where N is the number of solutions in population P. Due to the fact that the while-loop has to be called up to N times (when there exists only one element in each set F_i), the overall complexity for the non-dominated sorting algorithm non-dominated_sort(P) is $\mathcal{O}(m \cdot N^3)$.
A non-dominated sorting algorithm with a time complexity $\mathcal{O}(m \cdot N^2)$ will be presented in Section 3.2.1.

3.1.2 Fitness Assignment and Sharing

After the classification of the solutions into the sets $F_1, \ldots, F_r$, the individuals with objective vectors nearest to the Pareto-optimal points among the solutions in P are located in F_1. These elements will get the highest fitness value. The elements in F_i will get a smaller fitness value than those in F_{i-1} and a higher one than the individuals in F_{i+1}.

Another goal is to ensure higher fitness values for individuals in less crowded regions. The idea behind this is to attract the search procedure to these areas in order not to lose valuable candidates for good solutions. For this reason, the sharing function method (refer to Goldberg and Richardson [27]), where the distance to the other solutions is taken into account, is used for the fitness assignment.

For each pair of solutions $x^1 \in F_i$ and $x^2 \in F_i$, $x^1 \neq x^2$, the normalized Euclidean distance $d(x^1, x^2)$ between x^1 and x^2 is computed as follows.

$$d(x^1, x^2) = \sqrt{\sum_{i=1}^{n} \left(\frac{x_i^1 - x_i^2}{x_i^u - x_i^l} \right)^2}$$

Using this distance calculation a sharing function value between the individuals x^1 and x^2 is calculated as follows:

$$S(d(x^1, x^2)) = \begin{cases} 1 - \left(\frac{d(x^1,x^2)}{\sigma_{share}} \right)^2, & \text{if } d(x^1, x^2) < \sigma_{share} \\ 0, & \text{otherwise} \end{cases}$$

Here σ_{share} is the so-called sharing parameter.
Assume that $k < N$ good solution candidates are desired where N is the size of the population P. Deb and Goldberg [12] have suggested to compute the sharing parameter σ_{share} as follows:

$$\sigma_{share} = \frac{0.5}{\sqrt[n]{k}}$$

The last ingredient needed for the formulation of the fitness value assignment algorithm is the niche count $c(x^1)$ which is obtained as sum of the sharing function values between the solution $x^1 \in F_i$ and all other solutions in the same set:

$$c(x^1) = \sum_{x \in F_i} S(d(x^1, x))$$

Now the algorithm for the fitness value assignment to the individuals of population P in the algorithm NSGA-I can be formulated. To each solution $x^1 \in F_j$ a shared fitness value $G(x^1)$ is assigned using this algorithm. The shared fitness value of x^1 is computed as the fraction of a value which is slightly smaller than the minimum shared fitness value of the solutions in F_{j-1} and the niche count $c(x^1)$:

Algorithm 4 fitness_assignment(P, F) [7]

```
Compute $\sigma_{share}$
Choose small $\varepsilon > 0$
$G_{min} = N + \varepsilon$
**for all** $j = 1 \ldots r$ **do**
  **for all** $x^1 \in F_j$ **do**
    Fitness: $G'(x^1) = G_{min} - \varepsilon$
    $c(x^1) = \sum_{x \in F_j} S(d(x^1, x))$
    Shared fitness: $G(x^1) = \frac{G'(x^1)}{c(x^1)}$
  $G_{min} = \min_{x^1 \in F_j} G(x^1)$
```

After performing this routine each solution $x^1 \in P$ got its fitness value $G(x^1)$ assigned. These fitness values are used for the selection of the best individuals with the proportionate selection algorithm presented in the following section.

3.1.3 Proportionate Selection

After assigning a fitness value to each individual, individuals for the computation of the subsequent population are selected based on these fitness values. The selected solutions should be the best solutions of the population one generation before. In the case of the Non-dominated Sorting Genetic Algorithm I, a proportionate selection operator (Deb [7] and Goldberg and Deb [26]) is used, where each solution x^1 is selected with a probability according to its shared fitness value.

A multiset U of N individuals chosen from population P is created using the proportionate selection algorithm. The solutions stored in U will be used to compute new solutions to form the population of the next generation.
To choose N good solutions for U, a probability value $w(x^1)$ is computed for each solution $x^1 \in P$, depending on the shared fitness value of x^1. For each of the N elements in U solution $x^1 \in P$ is chosen with probability $w(x^1)$.

Since this selection of individuals is done randomly, a solution with a high fitness value might end up several times in U whereas a solution with a low fitness value might not be in U at all. Due to the different number of copies in U the recombination is done with individuals with higher fitness more likely than with those with lower fitness.

Algorithm 5 proportionate_selection(P) [7],[26]

```
$U = \emptyset$
$S = \sum_{x^1 \in P} G(x^1)$
for all $x^1 \in P$ do
    $w(x^1) = \frac{G(x^1)}{S}$        // A probability for each individual
while $|U| < N$ do
    Choose a random $x^2 \in P$ according to the probabilities $w(x^1)$, $x^1 \in P$
    $U = U \cup \{x^2\}$
```

$|U|$ is the number of elements in U. The random selection in the while-loop can be done for instance by computing N values $\{W_1, \ldots, W_N\}$ in the interval $[0, 1]$. The value W_j is the sum of the previously calculated probabilities $w(x^1), \ldots, w(x^j)$. After that a random number z between 0 and 1 is generated. If $z \in [0, W_1]$, x^1 will be the randomly chosen individual. Otherwise, if $z \in (W_j, W_{j+1}]$, $j \in \{1, \ldots, N-1\}$, the obtained solution will be x^{j+1}. W_N equals one according to the construction of W_j.

Algorithm 6 random_element(P) [7]

```
Fix the order of the individuals: $P = \{x^1, \ldots, x^N\}$
for all $j = 1, \ldots, N$ do
    $W_j = \sum_{i=1}^{j} w(x^i)$
Generate random number $z \in [0, 1]$
if $z \in [0, W_1]$ then
    STOP, return $x^1$ as the randomly chosen solutions
for all $j = 1, \ldots, N-1$ do
    if $z \in (W_j, W_{j+1}]$ then
        STOP, return $x^{j+1}$ as the randomly chosen solutions
```

3.1.4 Main Algorithm

All subroutines needed for the Non-dominated Sorting Genetic Algorithm I were introduced in the previous sections. Finally all these components will be put together to obtain the NSGA-I algorithm.

The starting population P_0 is randomly chosen. The population P_g of generation g is sorted by the non-dominated sorting algorithm (Algorithm 3). According to the resulting sets $F_1, \ldots, F_r$ the fitness is assigned (Algorithm 4) to each solution of the population. Algorithm 5 chooses and stores the solutions for the recombination process in U. The last step is the recombination with crossover and mutation algorithms (refer to Section 3.3). The recombination is applied to U to get the population P_{g+1} for the subsequent generation $g + 1$.

Algorithm 7 Non-dominated Sorting Genetic Algorithm I [59]

```
Generate random start population $P_0$
while $g \leq g_{max}$ do
    $F$ = non-dominated_sort($P_g$)        // $F = \{F_1, \ldots, F_r\}$
    fitness_assignment($P, F$)
    $U$ = proportionate_selection($P_g$)
    $P_{g+1}$ = new_population($U$)
    $g = g + 1$
```

The Non-dominated Sorting Genetic Algorithm I has a time complexity of $\mathcal{O}(m \cdot N^3)$ for each iteration of the algorithm due to the routine non-dominated_sort(P) which has a higher complexity than any other component of the algorithm. The Non-dominated Sorting Genetic Algorithm II, a successor of the NSGA-I algorithm, with an improved time complexity of $\mathcal{O}(m \cdot N^2)$ per iteration will be presented in the next section.

3.2 Non-dominated Sorting Genetic Algorithm II

The Non-dominated Sorting Genetic Algorithm II (**NSGA-II**) was designed by Deb et al. [13].

Deb et al. [13] designed the NSGA-II algorithm due to the following reasons:

- The non-dominated sorting algorithm of NSGA-I has a high time complexity ($\mathcal{O}(m \cdot N^3)$).
- Elitism, as used in NSGA-II, can speed up evolutionary algorithms (refer to Zitzler et al. [65]).
- The sharing parameter σ_{share} needed in NSGA-I is difficult to specify.

Elitism means that the best individuals of the previous generation are copied in the subsequent population such that good solutions do not get lost. This means that not all solutions of the new generation are recombinations of individuals. An elitist Genetic Algorithm chooses the best individuals among the individuals from the current generation and from the recombined solutions.

To avoid the sharing parameter, the Crowded Comparison Operator is used in the algorithm NSGA-II. For more details regarding this algorithm refer to Section 3.2.3.

3.2.1 Fast Non-dominated Sorting Algorithm

The **Fast Non-dominated Sorting Algorithm** is an improvement of the sorting algorithm used in the first version of the Non-dominated Sorting Genetic Algorithm (NSGA-I, refer to Section 3.1, Srinivas and Deb [59], and Deb [7]).

The time complexity of the Fast Non-dominated Sorting Algorithm is $\mathcal{O}(m \cdot N^2)$ compared to $\mathcal{O}(m \cdot N^3)$ of the sorting algorithm used in the NSGA-I. The space complexity of the new faster algorithm increases, however, from $\mathcal{O}(N)$ to $\mathcal{O}(N^2)$.

In the new algorithm for each solution x^1 of the current population P the number of solutions dominating x^1 and the set of solutions dominated by x^1 are stored. For this reason, the needed storage space increases, but the time complexity decreases.

The Fast Non-dominated Sorting Algorithm for a population P can be formulated as follows:

Algorithm 8 fast_non-dominated_sort(P) [13]

```
for all $x^1 \in P$ do
  for all $x^2 \in P$ do
    if $x^1 \succ x^2$ then        // $x^1$ dominates $x^2$
      $S(x^1) = S(x^1) \cup \{x^2\}$
    else if $x^2 \succ x^1$ then        // $x^2$ dominates $x^1$
      $n(x^1) = n(x^1) + 1$
  if $n(x^1) == 0$ then        // no solution dominates $x^1$
    $F_1 = F_1 \cup \{x^1\}$
$i = 1$
while $F_i \neq \emptyset$ do
  $H = \emptyset$
  for all $x^1 \in F_i$ do
    for all $x^2 \in S(x^1)$ do
      $n(x^2) = n(x^2) - 1$
      if $n(x^2) == 0$ then
        $H = H \cup \{x^2\}$
  $i = i + 1$
  $F_i = H$
```

$n(x^1)$ is the number of solutions dominating x^1 and $S(x^1)$ is the set of solutions in P dominated by x^1.
The set F_i is the i-th set according to the non-dominated sorting algorithm. The first set F_1 is the set of solutions not dominated by any other solution in population P, F_{j+1} is the set of solutions not dominated by any other solution besides the elements in $\{F_1, \dots, F_j\}$. When $x^1 \in P$ is an element of F_j, then the non-dominated rank $r(x^1)$ of solution x^1 equals to j.

The double for-loop for the computation of $S(x^1)$ and $n(x^1)$, for all $x^1 \in P$, has a computational complexity of $\mathcal{O}(m \cdot N^2)$ and the second part, the computation of the sets F_i, has

a complexity of $\mathcal{O}(N^2)$.
This results in $\mathcal{O}(m \cdot N^2)$ for the overall time complexity of the Fast Non-dominated Sorting Algorithm.

3.2.2 Crowding Distance

After sorting the solutions according to non-domination, the elements in the same set F_i do not dominate each other. For this reason, a comparison operator is needed to decide, which of the solutions are favored.
To that end, Deb et al. [13] have estimated the density of the solutions by using the crowding distance. This distance is used such that solutions in less crowded regions are favored. The crowding distance of solutions situated in such regions will be greater than elsewhere.

Definition 3.1 (Crowding Distance)
*The **crowding distance** $d_c(x^1)$ of a point (a solution) $x^1 \in P$ to its neighbors with respect to each of the objectives is a measure for the size of the largest cuboid containing x^1 and no other solution of the population.*

The calculation of the crowding distance for each point x^1 in a set $H \subset \mathbb{R}^n$ is done as follows. The routine sort(H, j) sorts the points in H according to their objective values with respect to the j-th objective function $f_j(x)$. This is done for each function. So a point gets two neighbors with respect to each objective function. The crowding distance of x^1 is computed by summing up the differences between the objective values of the left and the right neighbors with respect to each of the objective functions.

Algorithm 9 crowding_distance(H) [13]

```
$k = |H|$
for all $i = 1, \dots, k$ do
  $H[i]_{dist} = 0$
for all $j$: $f_j$ objective do
  $H = \text{sort}(H, j)$        // sort $H$ with respect to the objective values
  $H[1]_{dist} = H[k]_{dist} = \infty$        // Solutions at the boundary are always favored
  for all $i = 2, \dots, (k-1)$ do
    $H[i]_{dist} = H[i]_{dist} + (H[i+1].j - H[i-1].j)$
```

$H[i]$ is the i-th element in H and $H[i].j$ is the value of objective f_j of the i-th element in H. The resulting value $H[i]_{dist}$ is the crowding distance of the current i-th element in H. The crowding distances $d_c(x^l)$, $l = 1, \dots, k$, are the distances $H[i]_{dist}$ as computed by this algorithm. The set H is one of the sets F_i generated by the non-dominated sorting algorithm in the Non-dominated Sorting Genetic Algorithm II.

The computational complexity depends mainly on the complexity of the sorting algorithm

($\mathcal{O}(k \cdot \log(k))$ for sorting k elements). This results in an overall complexity of $\mathcal{O}(m \cdot k \cdot \log(k))$ when the set H has k elements. The factor m results from the loop over all objective functions.

3.2.3 Crowded Comparison Operator

Using the non-dominated rank and the crowding distance for each solution in the population, the **crowded comparison operator** (Deb et al. [13]) for comparing two solutions $x^1 \in X$ and $x^2 \in X$ can be defined:

Definition 3.2 (Crowded comparison operator $\geq_c$)

$$x^1 \geq_c x^2 \iff ((r(x^1) < r(x^2)) \text{ or } ((r(x^1) = r(x^2)) \text{ and } (d_c(x^1) > d_c(x^2))))$$

This means that lower rank is preferred and when both solutions have the same rank, the greater crowding distance is the favored one.

3.2.4 Main Algorithm

Finally the non-dominated sorting Genetic Algorithm NSGA-II is formulated in this section. To that end, an operator for the selection of good individuals for the recombination to new ones is defined.

Definition 3.3 (Binary Tournament Selection)
Selecting a solution $x^2 \in X$ from the elements in P with the ***binary tournament selection*** *means to do the following:*

1. *Choose (randomly) two solutions of P.*
2. *x^2 is the "better" of these two solutions.*

The "better" solution could be the one with the better objective value in a chosen objective function, the greater solution with respect to the Crowded Comparison Operator (refer to Section 3.2.3) or the solution with smaller rank.

In the following algorithm P_g and Q_g represent the parent population P and the child population Q in generation $g \in \{0, \ldots, g_{max}\}$. The first parent population P_0 is created randomly and the first child population Q_0 is computed by binary tournament selection according to the rank of the solutions, mutation, and crossover from P_0. The algorithm can be performed with an arbitrary crossover and mutation operator. Crossover and mutation methods are used to recombine two selected individuals to new ones. Some examples for such operators are given in Section 3.3 and in Deb [7].
The populations P_g and Q_g both have size N and sort$(P_g, \geq_c)$ sorts the solutions in P_g according to the crowded comparison operator.
The following algorithm results:

Algorithm 10 Non-dominated Sorting Genetic Algorithm II [13]

```
$g = 0$
while $g \leq g_{max}$ do
  $R_g = P_g \cup Q_g$
  $F = \text{fast\_non-dominated\_sort}(R_g)$        // $F = \{F_1, \ldots, F_r\}$
  $i = 1$
  while $|P_{g+1}| < N$ do
    $\text{crowding\_distance}(F_i)$
    $P_{g+1} = P_{g+1} \cup F_i$
    $i = i + 1$
  $\text{sort}(P_{g+1}, \geq_c)$        // Sort $P_{g+1}$ with respect to the crowded comparison operator
  $P_{g+1} = P_{g+1}[1 \ldots N]$        // Take the first $N$ elements of $P_{g+1}$
  $Q_{g+1} = \text{new\_population}(P_{g+1})$
  $g = g + 1$
```

R_g is the combined population with size $2 \cdot N$. In the algorithm the populations P_g and Q_g are joined to the population R_g. The new P_{g+1} consists of the first sets in $F = \{F_1, \ldots, F_r\}$ until P_{g+1} has at least N elements. The resulting P_{g+1} with at least N elements is sorted with respect to the crowded comparison operator. After sorting, the new population P_{g+1} consists of the first N elements of this set. That means that from the last added set only the elements with the greatest crowding distance are included in the new population P_{g+1}. The new child population Q_{g+1} is computed with the binary tournament selection approach according to the Crowded Comparison Operator, mutation, and crossover from P_{g+1}.

The routine fast_non-dominated_sort(R_g) has a computational complexity of $\mathcal{O}(m \cdot N^2)$. As seen in the sections before, all the other parts of the algorithm have a smaller complexity. This implies that the overall time complexity of the algorithm for one iteration (for one generation) is $\mathcal{O}(m \cdot N^2)$.

3.2.5 Controlled Approach

A variant of the Non-dominated Sorting Genetic Algorithm II is the Controlled NSGA-II (refer to Deb and Goel [11]). In this variant the selection of the solutions from the sets in F, computed by the non-dominated sorting algorithm, is modified.

In the controlled approach, not the first sets in F, but solutions from more sets are chosen. This is done by restricting the number of solutions which are taken from each set F_j. For this purpose a geometric distribution is used. The maximum number of solutions taken from F_j, n_j, is computed from n_{j-1} using a user defined reduction parameter $\alpha < 1$. Due to this parameter the maximum number of chosen solutions decreases for higher j. Two consecutive maximum numbers of solutions have the following relationship:

$$n_j = \alpha \cdot n_{j-1}$$

In this approach, as in the common NSGA-II, the combined population R is computed and sorted according to non-domination with the resulting set of sets $F = \{F_1, \ldots, F_r\}$.
The maximum number of solutions taken from set F_j is calculated as follows:

$$n_j = N \cdot \frac{1-\alpha}{1-\alpha^r} \cdot \alpha^{j-1}$$

When n_j is greater than the number of solutions in F_j, all elements of F_j are chosen and the difference $n_j - |F_j|$ is added to n_{j+1}. Otherwise, when n_j is smaller than the number of solutions in F_j the solutions are chosen according to tournament selection using the crowded comparison operator.
This process is stopped when there are N solutions in the new population. When the number of chosen solutions is smaller than N after considering the last set F_r, the missing solutions are chosen from the remaining ones starting in the first sets, stopping when the new population consists of N solutions.

According to Deb and Goel [11] the controlled approach increases diversity. High diversity is often a goal because otherwise the obtained solutions could get stuck in a specific region and thus possibly better solutions in other regions are not considered.

3.3 Recombination

The selection operator chooses good solutions, but does not produce new solutions, so recreation operators are needed. In the case of Genetic Algorithms this work is done by recombination operators which combine selected solutions to new ones. Most algorithms use two more or less randomly chosen solutions for the recombination. These parent individuals generate a new child solution which is mutated afterwards.
This principle of combining two parents and mutating the resulting child is an imitation of the recombination process in nature, where a child is a crossover of its father and its mother with some additional mutation.

In the following sections crossover and mutation operators as they can be used in Genetic Algorithms or other evolutionary algorithms are described.

3.3.1 Crossover

The crossover operator generates a new child solution by combining two parent individuals. Often only a few solutions of the new population are recombined by crossover, because crossover is only performed with a given probability, the crossover probability. Otherwise, when there is no crossover, only one of the parent solutions is copied and maybe mutated for the new population. There exist several possible crossover operators to create the offspring (refer to Deb [7]).
Some examples for crossover operators handling continuous design variables are given below. In the following, x_i^1 and x_i^2 for $i \in \{1, \ldots, n\}$ denote the design variables of the parent

solutions $x^1 \in \mathbb{R}^n$ and $x^2 \in \mathbb{R}^n$. The resultant children are $y^1 \in \mathbb{R}^n$ and $y^2 \in \mathbb{R}^n$, when the operator generates two children, or y^1, when there exists only one offspring.

The crossover operators presented below are only suitable for optimization problems with continuous design variables. For crossover operators for integer or mixed integer optimization algorithms refer to Section 3.5.1.

- One-point Crossover

In the One-point Crossover a random integer $w \in \{1, \dots, n\}$ is chosen as the index where left of it the child gets the design variable values of one parent and on the right of the index of the other one.

Algorithm 11 one_point_X(x^1, x^2)

```
Choose $w \in \{1, \dots, n\}$ randomly
for all $i = 1, \dots, w$ do
    $y_i^1 = x_i^1$
    $y_i^2 = x_i^2$
for all $i = w + 1, \dots, n$ do
    $y_i^1 = x_i^2$
    $y_i^2 = x_i^1$
```

This One-point Crossover can be extended to an S-point Crossover where S random indices $\{i_1, \dots, i_S\}$ are chosen. Additionally, i_0 is set to 0 and i_{S+1} is set to N. The first child gets the values of parent one in the segments from i_{j-1} to i_j when $j \in \{1, \dots, S+1\}$ is odd and of parent two when j is even. For the second child it is the other way around.
As per Deb [7] the One-point Crossover operator is not very successful in generating children with good objective vectors.

- Blend Crossover, BLX

Assuming $x_i^1 < x_i^2$, the child value y_i^1 is computed with the Blend Crossover as a random number out of the interval

$$[x_i^1 - \beta \cdot (x_i^2 - x_i^1), x_i^2 + \beta \cdot (x_i^2 - x_i^1)]$$

where β is the BLX parameter. As per Eshelman and Schaffer [21] $\beta = 0.5$ is a recommendable choice for β.
The child value can be computed with the BLX operator using

$$y_i^1 = (1 - \gamma_i) \cdot x_i^1 + \gamma_i \cdot x_i^2$$

where $\gamma_i = (1 + 2\beta) \cdot u_i - \beta$. Here u_i is a random number between 0 and 1.

From the formula above the following can be concluded:

$$(y_i^1 - x_i^1) = \gamma_i \cdot (x_i^2 - x_i^1)$$

Obviously there is a direct dependence between the difference of the parents and the difference of one parent and the offspring. This is a useful property because when the optimum is close to the current solutions, the differences between the solutions will get small and the child solution should also be not far away.

During the first iterations of the algorithm, when the optimal solution is far away in general, the differences between a parent and the child will be higher. Due to the dependence above, the BLX operator uses a wider search space during the first generations because at this time also the parents are not very close to each other due to their more or less random distribution.

- Unfair Average Crossover

Nomura and Miyoshi [46] presented the Unfair Average Crossover operator where the value of one parent is more important for the child than the value of the other. Two children of the two parents are computed as follows:

$$y_i^1 = \begin{cases} (1+\alpha) \cdot x_i^1 - \alpha \cdot x_i^2, & \text{for} \quad i = 1, \ldots, j \\ -\alpha \cdot x_i^1 + (1+\alpha) \cdot x_i^2, & \text{for} \quad i = j+1, \ldots, n \end{cases}$$
$$y_i^2 = \begin{cases} (1-\alpha) \cdot x_i^1 + \alpha \cdot x_i^2, & \text{for} \quad i = 1, \ldots, j \\ \alpha \cdot x_i^1 + (1-\alpha) \cdot x_i^1, & \text{for} \quad i = j+1, \ldots, n \end{cases}$$

In the formulas the parameter j is a random integer between 1 and n and the parameter α is chosen from the interval $[0, 0.5]$.
Since with the Unfair Average Crossover the child is more influenced by one of its parents, the mean of the population changes. As per Deb [7], the change of the mean of the population in the direction of one parent would be hard to motivate, even if this parent were the better of these two solutions.

Beyer and Deb [3] postulate that the following two properties should hold after using a crossover operator:

1. The mean of the population should not change.
2. The higher the generation number is, the higher the variance of the population should be.

For more information regarding the two postulated properties refer to Beyer and Deb [3].

- Simulated Binary Crossover, SBX

Another crossover operator is the Simulated Binary Crossover operator (SBX, refer to Deb and Agrawal [8] and Deb and Agrawal [9]).
In the routine associated with this operator, as above x_i^1 and x_i^2 are the values of the i-th design variable of the parent solutions $x^1 \in \mathbb{R}^n$ and $x^2 \in \mathbb{R}^n$, and y_i^1 is the value of the child's i-th design variable. The value η_C is a distribution index. When η_C is high, the child value will be not as far away from the parents than when η_C is low ("near-parent" solution).

The SBX operator is applied, when a random number between 0 and 1, generated by randomperc(), is smaller than or equal to a given crossover probability. Otherwise the child solution is equal to one of the parent solutions. The number of design variables is n.

In the algorithm, at first a random number z between zero and one is generated. The value β_s is derived by Deb and Agrawal [8] such that the area under the curve from 0 to β_s of a certain probability function equals to z. The probability function is chosen by Deb and Agrawal [8] such that the the Simulated Binary Crossover has a similar search power as the Single-point Crossover operator (refer to Section 3.5) for integer design variables. For more information about the SBX operator refer to Deb and Agrawal [8] and Deb and Beyer [10].

Checking the upper and the lower bounds as done in this routine is needed in each of the presented crossover operators because if the bounds were not satisfied, the solution would be infeasible. Now the algorithm of the SBX operator can be formulated.

Algorithm 12 SBX(x^1, x^2) [9]

```
if randomperc() $\leq$ $crossoverProbability$ then          // random value
  for all $i = 1, \ldots, n$ do          // for all design variables
    if randomperc() $> 0.5$ then
      $y_i^1 = x_i^2$
    else
      if $|x_i^1 - x_i^2| == 0$ then
        $y_i^1 = x_i^1$
      else
        if $x_i^1 < x_i^2$ then
          $x_1 = x_i^1$, $x_2 = x_i^2$
        else
          $x_1 = x_i^2$, $x_2 = x_i^1$          // $x_1 < x_2$
        $x_l = x_i^l$
        $x_u = x_i^u$          // Bounds of the design variable $i$
        $z$ = randomperc()
        $\beta = 1 + \left(2 \cdot \frac{x_1 - x_l}{x_2 - x_1}\right)$
        $\alpha = 2 - \beta^{-(\eta_C + 1)}$
        $\beta_s = \begin{cases} (z \cdot \alpha)^{\frac{1}{\eta_C+1}} & \text{if } z \leq \frac{1}{\alpha} \\ \left(\frac{1}{2 - z \cdot \alpha}\right)^{\frac{1}{\eta_C+1}} & \text{otherwise} \end{cases}$
        $c_1 = \frac{1}{2} \cdot (x_1 + x_2 - \beta_s \cdot (x_2 - x_1))$
        $\beta = 1 + \left(2 \cdot \frac{x_u - x_2}{x_2 - x_1}\right)$
        $\alpha = 2 - \beta^{-(\eta_C + 1)}$
        $\beta_s = \begin{cases} (z \cdot \alpha)^{\frac{1}{\eta_C+1}} & \text{if } z \leq \frac{1}{\alpha} \\ \left(\frac{1}{2 - z \cdot \alpha}\right)^{\frac{1}{\eta_C+1}} & \text{otherwise} \end{cases}$
        $c_2 = \frac{1}{2} \cdot (x_1 + x_2 + \beta_s \cdot (x_2 - x_1))$
        if $c_1 < x_l$ then $c_1 = x_l$
        if $c_2 < x_l$ then $c_2 = x_l$
        if $c_1 > x_u$ then $c_1 = x_u$
        if $c_2 > x_u$ then $c_2 = x_u$
        if randomperc() $\leq 0.5$ then
          $y_i^1 = c_2$
        else
          $y_i^1 = c_1$
else
  for all $i = 1, \ldots, n$ do          // Take parent1 as a child
    $y_i^1 = x_i^1$
```

3.3.2 Mutation

After computing a child solution with a crossover operator, this solution is going to be mutated. The mutation operator is like the crossover operator responsible for searching for new solutions which should be better than the solutions in the current generation. Since both, crossover and mutation, have randomness included, this cannot be ensured.
When the used crossover operator produces two children, one of these has to be chosen. The design variables of the parent are x_i and of the child y_i with $i \in \{1, \ldots, n\}$.

There exist many different mutation operators, some are described in Deb [7]. For example, the following operators can be used for the mutation of continuous design variables in a Genetic Algorithm. For mutation operators for mixed integer or integer optimization problems refer to Section 3.5.2.

Similarly to the crossover probability there is also a mutation probability. For this reason, not all solutions of the new generation are modified by a mutation operator. So there are four possibilities for a solution of the new population:

- No recombination has been done.
- Only the crossover operator was used.
- Only the mutation operator was used.
- Both, crossover and mutation, were performed.

In the cases when mutation is not performed, the result of the crossover operator is used as the new solution. This can be a crossover of two parents or a copy of one of the parents. In the following paragraphs several mutation operators are presented.

- Random Mutation Operators

The Random Mutation I is a simple method, where the child y_i is chosen randomly from the interval $[x_i^l, x_i^u]$ (refer to Michalewicz [42]):

$$y_i = x_i^l + u_i \cdot (x_i^u - x_i^l)$$

u_i is a continuous random number between 0 and 1 and x_i^l and x_i^u are the lower and the upper bound of the design variable x_i. In contrast to the other mutation operators listed in this thesis, the child is completely independent of the parent solution in this method. This mutation operator is equivalent to the creation of the initial generation where each solution is randomly chosen within the bounds of the different design variables.

As per Deb [7] there exists another random mutation operator. In the Random Mutation

II the child is not independent of the parent solution. In this operator the new solution is chosen randomly within an interval around the parent solution:

$$y_i = x_i + (r_i - 0.5) \cdot \Delta_i$$

In this formula Δ_i is the predefined length of the interval where x_i is in the center and the new solution y_i is a random number within this interval. The parameter r_i is a random number between 0 and 1. In contrast to the Random Mutation I the bounds have to be checked after this calculation because x_i can be close to its bounds such that the randomly chosen value is outside the bounds.

- Non-uniform Mutation

Other than in the Random Mutation I, the probability to be closer to the parent solution in the Non-uniform Mutation is higher than to be far away. An increasing number of generations results in an increasing probability to be nearby the parent, whereas in the Random Mutation II the interval, from which the random child value is chosen, is always as big as in the first generation. In the Random Mutation I the probability is the same for a solution nearby the parent individual or far away.
Michalewicz [42] proposed to use the following formula:

$$y_i = x_i + \tau \cdot (x_i^u - x_i^l) \cdot (1 - r_i^{\left(1-\frac{g}{g_{max}}\right)^b})$$

g_{max} is the maximal number of generations and r_i is a random number between 0 and 1. The parameter τ is one of the values -1 or 1, each with a probability of 50%. Michalewicz [42] proposes to set paramter b equal to five.

- Normally Distributed Mutation

In the Normally Distributed Mutation operator (refer to Deb [7]) a Gaussian probability distribution with mean 0 is used:

$$y_i = x_i + N(0, \sigma_i)$$

The parameter σ_i is fixed by the user or can be adapted for each generation.
The same mutation operator is used in the evolutionary algorithm Evolution Strategy, for more information regarding this strategy refer to Deb [7].

- Parameter-based Mutation Operator

The mutation operator which was implemented for the Genetic Algorithm for continuous design variables is a parameter-based mutation which is shown in the following (refer to Deb and Agrawal [9]).
A given mutation probability has to be greater than a randomly chosen number between

0 and 1 (generated by randomperc()), otherwise no mutation for the value of this design variable is performed. The parameter η_M is a predefined mutation parameter.

Algorithm 13 mutation(p)

```
for all $i = 1, \dots, n$ do        // for all design variables
  if randomperc() $\leq$ *mutationProbability* then
    $x = x_i$        // Value of the $i$-th design variable
    $x_l = x_i^l$
    $x_u = x_i^u$
    $\delta_1 = \frac{x - x_l}{x_u - x_l}$
    $\delta_2 = \frac{x_u - x}{x_u - x_l}$
    $a$ = randomperc()        // Random number $a \in [0, 1]$
    $w = \frac{1}{\eta_M + 1}$
    if $a \leq 0.5$ then
      $z = 1 - \delta_1$
      $v = 2 \cdot a + (1 - 2 \cdot a) \cdot (z^{\eta_M + 1})$
      $\delta_s = v^w - 1$
    else
      $z = 1 - \delta_2$
      $v = 2 \cdot (1 - a) + (a - 0.5) \cdot (z^{\eta_M + 1})$
      $\delta_s = v^w - 1$
    $x = x + \delta_s \cdot (x_u - x_l)$
    if $x < x_l$ then $x = x_l$
    if $x > x_u$ then $x = x_u$
    $y_i = x$
```

In this routine x_i, $i \in \{1, \dots, n\}$ are the values of the design variables of the parent solution, and x_i^l and x_i^u are the lower and upper bounds. The resulting solution is y_i.

- Particle Swarm Mutation

A mutation operator based on the Particle Swarm Optimization is introduced in Section 3.4.1.

3.4 Hybrid Genetic Algorithms

Combining a Genetic Algorithm with other local or global optimization algorithms can improve both the quality of the solution and the efficiency to find it. For studies regarding the improvement using a Genetic Algorithm in combination with local optimization refer to Deb et al. [14], Bilchev and Parmee [4], and Krasnogor and Smith [38].
It can be an advantage to search for the best local solution in the current region around a solution found by the Genetic Algorithm (GA) by using a local optimization algorithm,

because a GA can rapidly find the region of the optimal solution, but for this kind of algorithm it is hard to find the exact optimal solution.

In the following sections, some possible hybridizations of a Genetic Algorithm with other (global or local) optimization algorithms are presented. For more possibilities how to combine Genetic Algorithms with other algorithms refer to the review of El-Mihoub et al. [19].

3.4.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) developed by Kennedy and Eberhart [36] is also a population based optimization algorithm like the Genetic Algorithms, i.e., populations change to new, hopefully better ones to compute the best solution.

Hassan et al. [32] compared a Genetic Algorithm with a particle swarm optimization algorithm using statistical tests. In these tests the PSO seemed to have a better computational efficiency than Genetic Algorithms. As per Hassan et al. [32] the difference between the efficiency of PSO and GA is highly dependent on the instances solved.

In the Particle Swarm Optimization algorithm, a generation consists of a number of particles. Each particle is one particular solution of the optimization problem.
The i-th particle p_i^{g+1} of the subsequent generation $g+1$ is computed using the values of the design variables of three different particles:

- The particle p_i of generation g.
- The best known solution for the current particle p_i.
- The best known solution for any particle.

The calculation of the velocity (change pattern) of the i-th particle to compute the generation $g+1$ is according to dos Santos Coelho [16]

$$v_i^{g+1} = \omega \cdot v_i^g + c_1 \cdot r_1 \cdot (b_i - p_i^g) + c_2 \cdot r_2 \cdot (b - p_i^g)$$

The i-th particle of generation $g+1$ is calculated as the sum of the value of the particle in generation g and its velocity:

$$p_i^{g+1} = p_i^g + v_i^{g+1}$$

In a generation with N particles where each particle consists of n design variables denote $v_i^g \in \mathbb{R}^n$ the velocity and $p_i^g \in \mathbb{R}^n$ the position in the search space of the i-th particle in generation g. The currently best solution for the particle p_i is b_i and the best solution of all particles is stored in b. c_1 and c_2, the cognitive and the social learning rate (Shi and Eberhart [57]), are predefined positive constants and r_1 and r_2 are random values between 0 and 1.

Example values for c_1 and c_2 are $c_1 = c_2 = 2$ from Shi and Eberhart [57] and $c_1 = c_2 = 2.05$ used by dos Santos Coelho [16].
$\omega \in [0, 1]$ is called the inertia weight. According to Shi and Eberhart [57] a large inertia weight has a trend towards global optimization whereas a small ω has a trend towards local optimization.
Shi and Eberhart [57] propose to start with a high ω and decrease ω linearly in each optimization step until a given small end value is reached. That means that the value of ω for the current generation g is calculated as follows:

$$\omega_g = (\omega_0 - \omega_{g_{max}}) \cdot \frac{g_{max} - g}{g_{max}} + \omega_{g_{max}}$$

In this formula, ω_0 is the value for ω in generation 0 and $\omega_{g_{max}}$ the value for ω in the last generation.
Experiments of Shi and Eberhart [57] showed that a value of 0.9 for ω_0 and a value of 0.4 for $\omega_{g_{max}}$ are good choices.

Now the PSO algorithm reads:

Algorithm 14 Particle Swarm Optimization [36]

```
for all $i = 1, \ldots, N$ do        // $N$ is the number of particles
  Compute random initial particle $p_i$
  $b_i = p_i$
  if $f(p_i) < f(b)$ then
    $b = p_i$
  Initialize randomly the particle's velocity $v_i$
while Termination criterion is not met do
  for all $i = 1, \ldots, N$ do
    Choose $r_1, r_2 \in [0, 1]$ randomly
    Calculate new velocity $v_i = \omega \cdot v_i + c_1 \cdot r_1 \cdot (b_i - p_i) + c_2 \cdot r_2 \cdot (b - p_i)$
    Calculate new particle $p_i = p_i + v_i$
    if $f(p_i) < f(b_i)$ then
      $b_i = p_i$
      if $f(b_i) < f(b)$ then
        $b = b_i$
```

After termination of the algorithm, b is the best known solution. The initial particles have large influence on the process of updating the velocity, so the initial generation is very important.

Two possibilities how to combine a Genetic Algorithm as introduced in Section 2.5 and the Particle Swarm Optimization algorithm are mentioned below.

Sequential Hybridization

Because of the influence of the initial particles one possibility for a hybrid Genetic Algorithm with Particle Swarm Optimization is to use a Genetic Algorithm for computing the initial generation for the Particle Swarm Optimization algorithm. Using this approach the initial generation is no longer a random one, but the computational effort for the initial generation computed by a Genetic Algorithm is much higher.
This sequential hybridization of a Genetic Algorithm was tested for a reactive power optimization problem by Lu et al. [40].

Parallel Hybridization - Mutation Operator

Besides the sequential hybridization it is also possible to use parallel hybridization where parts of the Particle Swarm Optimization Algorithm are used for instance in the recombination part of the Genetic Algorithms as described in Zhang et al. [64]. A mutation operator based on the formula of the Particle Swarm Optimization Algorithm as given above is presented in this section.

In the new mutation operator the formulas of the Particle Swarm Optimization are slightly modified for their use for the mutation operator. The new mutated child is calculated with the following formula:

$$p_i^{new} = \omega \cdot p_i + c_1 \cdot r_1 \cdot (p_i^u - p_i) + c_2 \cdot r_2 \cdot (b - p_i)$$

In the formula p_i^u is the upper bound vector of the i-th particle and b is the individual with the best objective vector from the previous generation. ω, c_1 and c_2 are positive constants and r_1 and r_2 are two random numbers between 0 and 1 as defined before.
Bai and Gan [2] recommended the values c_1 and c_2 to be set to 1.49445. The inertia weight ω is proposed to be set to 0.7298.

3.4.2 Adaptive Local Search

Genetic Algorithms are quite good as global optimization goals are concerned, but there is no local optimization included. Therefore it can be an advantage to combine a GA with local search methods. Yun [63] tested hybrid Genetic Algorithms with two different adaptive local search schemes.

The local search method is applied after the mutation step to compute the local optimal solution before the insertion of the individual in the new generation. This method is only started when the local search scheme after the previous iteration of the Genetic Algorithm decides that the local search is needed. Two different possibilities how to decide whether the search technique is started or not are shown below.

Local Search Technique - Hill Climbing Method

In the Hill Climbing Method (refer to Russell and Norvig [53]), N random individuals are computed in the neighborhood of the given individual. Among all of these individuals the one with the best objective value is chosen.

Yun [63] presented two local search schemes to decide whether a local search technique like the Hill Climbing Method is called or not. In the following two paragraphs these local search schemes will be described.

Local Search Scheme 1

One possibility for making the decision whether the local search is called, is to compute the ratio $V(g)$ for the current generation g using $v(g)$, the average objective value of all solutions in generation g:

$$V(g) = \frac{v(g)}{v(g-1)}$$

The iterative hill climbing method is called if $V(g) > 1$ (because minimum problems are considered), otherwise only the Genetic Algorithm is performed. This means that the local search is used when there has been no improvement of the average objective values within the last two generations.

Local Search Scheme 2

One problem of Genetic Algorithms is that when the individuals converge to the optimum, they are getting more and more similar to each other. To avoid this the following local search scheme can be used to increase the diversity.

At the beginning of the algorithm, the similarity coefficients $SC_{i,j}$ are calculated for each pair (x^i,x^j) of individuals as follows

$$SC_{i,j} = \frac{\sum\limits_{k=1}^{n} \delta(x_k^i, x_k^j)}{n}$$

where

$$\delta(x_k^i, x_k^j) = \begin{cases} 1 & \text{if } |x_k^i - x_k^j| \leq \alpha \\ 0 & \text{otherwise} \end{cases}$$

x_k^i denotes the value of the k-th design variable of individual x^i and α (e.g. $\alpha = 0.5$) is a constant to decide how similar two individuals are.

The value $\overline{SC}$ which is used for the decision whether local optimization is performed or not is the average value of all $SC_{i,j}$:

$$\overline{SC} = \frac{\sum\limits_{i=1}^{N} \sum\limits_{j=i+1}^{N} SC_{i,j}}{s}$$

In this formula is $s = \frac{N \cdot (N-1)}{2}$ the number of similarity coefficients $SC_{i,j}$ which have to be calculated.
The average similarity coefficient is a number between 0 and 1. When it is near to 1, there is a high similarity and the local search method will be performed.
For the decision whether the similarity coefficient is regarded as near to 1 there has to be a given threshold β, e.g. $\beta = 0.9$. When $\overline{SC}$ is greater than β, the local search is called, otherwise not.

3.5 Recombination in Mixed Integer Genetic Algorithms

In the previous sections only continuous design variables were taken into account. The main step for handling integer and mixed integer optimization problems with Genetic Algorithms is to modify the recombination operators such that they are able to handle integer design variables too.
The operators presented in the following sections are used when integer design variables are going to be recombined. When a continuous design variable is recombined in a mixed integer problem, the same operators as in the continuous optimization algorithm are used. It can be an advantage to choose different crossover and/or mutation probabilities for integer and continuous variables.

Some of the operators in the following sections need a rounding operator to transform the continuous output of the function into an integer variable. This can be done for instance with one of these two methods:

- Round off to the nearest integer.
- Round up or down with a probability of 50%.

The risk to get equal values is much smaller using the second method because the same continuous values are possibly rounded to different integers.

3.5.1 Integer Crossover

Some possible crossover operators for integer design variables are presented in this section. Only a few crossover operators listed below do not need a rounding operator afterwards.

- Simulated Binary Crossover for Integers

The Simulated Binary Crossover (SBX, Algorithm 12) uses continuous design variables. It is also possible to handle integer variables with this operator by computing an integer from the continuous solution with rounding.

- S-point Crossover

When the S-point Crossover is applied to two parent solutions, S random indices of the design variables are chosen. Between these indices the values of parent one and parent two are chosen alternately. This means for example that from the first variable to index one the values of parent one are taken, then for the next variables to index two the values from parent two etc. In this routine no rounding function is needed because the values of the parents have to be integers.

- Scattered Crossover

In the Scattered Crossover a random binary array with the number of design variables n as length is generated. Then for each design variable the value of parent one is taken, when there is a 0 in the array at that position, otherwise the value of parent two is chosen. This operator is a generalization of the S-point Crossover operator from above. The binary array ensures that the values of the child solution is taken randomly of parent one or parent two. When this method is used, there is no rounding function needed with the same reason as when using the S-point Crossover.

- Arithmetic Crossover

The Arithmetic Crossover operator is another crossover operator for continuous design variables where a rounding operator is needed to handle integer or mixed integer optimization problems. The value of the child's design variable y_i^1 is calculated by the following formula:

$$y_i^1 = a \cdot x_i^1 + (1 - a) \cdot x_i^2$$

a is a continuous random number between 0 and 1 and x_i^1 and x_i^2 are the values of the i-th design variable of the parents x^1 and x^2.

- Blend Crossover, BLX

The Blend Crossover operator of Section 3.3 can be used for integer or mixed integer optimization problems by using a rounding operator after calculating the design variable of the child solution with the original continuous operator.

- Unfair Average Crossover

The Unfair Average Crossover (refer to Section 3.3), originally designed for continuous design variables, can be used for problems with integer design variables when the results are rounded.

3.5.2 Integer Mutation

After the crossover operator the mutation operator has also to be modified for integer optimization. Some possibilities are presented below. Most of the operators are continuous mutation operators where the solution is rounded by a rounding operator like the two mentioned at the beginning of Section 3.5.

- Parameter-based Mutation Operator

For the mutation it is also possible to use the existing Parameter-based Mutation Operator as described in Section 3.3 and to round the solution afterwards.

- Random Change Mutation

The Random Change Mutation can be used without a rounding function. When the design variable is going to be mutated according to the mutation probability, the new value is chosen randomly out of the possible values for this integer variable. The possible values are all integer values between the lower and the upper bound of this variable.

- Particle Swarm Mutation

As described in Section 3.4.1 a modified formula of the Particle Swarm Optimization can be used in the recombination step. For an integer mutation the result of the mutation operator based on PSO has to be rounded. This mutation operator combined with a rounding operator was used by dos Santos Coelho [16].

- Non-uniform Mutation

The Non-uniform Mutation Operator, as shown in Section 3.3, can be used for integer variables when the solution of the operator is rounded for example with one of the two rounding operators as listed at the beginning of this subsection.

- Normally Distributed Mutation

For using the normally distributed mutation operator for integer variables the result of the formula in Section 3.3 has to be rounded.

4 A brief introduction to AVL Design Explorer and AVL CRUISE

The aim of this thesis is to extend an optimization package implemented in the optimization software AVL Design Explorer such that integer and mixed integer multi-objective optimization problems can be handled.
First, AVL Design Explorer is briefly introduced at the beginning of this section. Afterwards, the vehicle simulation software AVL CRUISE is described. AVL CRUISE is called by the AVL Design Explorer for the computation of the objective values to generate the results which are shown in Section 6. Calling AVL CRUISE for each objective function evaluation causes the duration of one simulation in AVL CRUISE to be the main factor for the running time of the AVL Design Explorer.

4.1 Optimization Software AVL Design Explorer

AVL Design Explorer [1] is a software to run a Design of Experiments (DoE) or an Optimization algorithm. The objective values are computed by simulation programs of AVL. One example for such a simulation program is the vehicle simulation software AVL CRUISE, which will be introduced in Section 4.2. This software is also used as objective function evaluator for generating the results in Section 6.

The two components of the AVL Design Explorer, Optimization and Design of Experiments, respectively, and their associated algorithms will be briefly introduced in the following section. For more details refer to the AVL Design Explorer User Guide [1].

4.1.1 Optimization

The focus in this thesis is put on the optimization part of the AVL Design Explorer. Optimization enables the user to possibly increase the quality of a model, previously defined for instance in AVL CRUISE. The procedure is started by defining the parameters of the model which are used as design variables for the optimization problem. After that, the AVL Design Explorer is started. The AVL Design Explorer calls the simulation software AVL CRUISE to evaluate the objective functions. AVL CRUISE runs a simulation for each individual of the current population which has not been considered in any population before to obtain the corresponding objective value.

When using AVL CRUISE, the selectable objective functions in AVL Design Explorer differ with the calculation tasks which are activated in the basis model for the optimization. Some examples for possible calculation tasks are described in Section 4.2. Each selected objective function can independently be defined to be minimized or maximized. A target which has to be reached by an objective function can also be specified. The user chooses the optimization algorithm and provides values for the size of the population and for the number of generations. By multiplying these two numbers and adding the number of individuals in the two initial populations the number of simulation runs is calculated.

When the user chooses to perform optimization, one of the following three optimization algorithms can be selected regardless of whether a single or a multi-objective optimization problem is considered.

- **NLPQL** (Nonlinear Programming Quadratic Linesearch) (refer to Schittkowski [56]).
- **Nelder-Mead** (refer to Nelder and Mead [45]).
- **Genetic Algorithm** as introduced in Section 3.2.

When more than one objective function is selected, these three algorithms can be used as single objective optimization algorithms by applying the ε-Constraint or the Weighted Sum approach as described in Section 2.4.2 and Section 2.4.1. The only possibility to optimize without scalarization is to select the Genetic Algorithm. By using the Genetic Algorithm it is possible to choose between the two scalarization approaches and true multi-objective optimization.
In any case the Non-dominated Sorting Genetic Algorithm II (NSGA-II) is used when the Genetic Algorithm is selected, regardless of whether scalarization is chosen or not. The algorithm NSGA-II is described in Section 3.2 and its implementation in Section 5.1.

4.1.2 Design of Experiments

The other component provided by the AVL Design Explorer is Design of Experiments (DoE). DoE can be used to analyze the influence of the selected design variables on the output values of the simulated model or to search for a good starting point for an optimization process which can be performed afterwards. When using Design of Experiments a number of different design points is generated and the output of the called simulation software is analyzed.

Note 4.1 *In the theory of Design of Experiments a design variable is called a **factor** and the value of a factor is referred to as a **level**.*

AVL Design Explorer provides the following algorithms which can be chosen for the Design of Experiments.

- **Full Factorial Design**:
  - The Full Factorial Design discretizes the interval between the bounds of the factors in equally spaced levels. Commonly two levels are used, where all vertices of the design space (a hypercube) are evaluated (refer to Moen et al. [43]).
- **Orthogonal Array**:

  - The approach Orthogonal Array is a fractional factorial design. The number of designs of the Full Factorial Design is reduced such that the number of evaluations decreases. This reduces the amount of time needed for the DoE but also some information is lost due to the lower number of designs. For more information about Orthogonal Arrays refer to Hedayat et al. [33].

- **Latin Hypercube**:
  - The design space of each factor is uniformly divided to get s levels. A column in a table is assigned to each factor and the entries of this column are a random permutation of the s levels. The design points, which are evaluated for the DoE, are the rows of the table (refer to Press et al. [50]).

- **Sobol Sequence**:
  - In this method the idea of quasirandom Sobol sequences is used for the generation of the design points (refer to Press et al. [50]).

For time expensive simulations surrogate models can be used which try to imitate the original simulation. The two selectable regression methodologies to create surrogate models are Support Vector Machines (refer to Press et al. [50]) and Relevance Vector Machines (refer to Tipping [60]).

4.2 Vehicle Simulation Software AVL CRUISE

The vehicle simulation software AVL CRUISE [6] is briefly introduced in this section. For detailed information regarding AVL CRUISE refer to Pfau [49] and the AVL CRUISE User Guide [6].

Considering the behavior of engines or even whole vehicles using simulation software is very important in the automotive industry. Without such software, companies would have to build much more prototypes of their products resulting in an extension of the construction process. The simulation software AVL CRUISE is one of those programs which simulate vehicle dynamics to calculate fuel consumption, emissions, and driving performance. Besides the simulation of cars it is also possible to simulate motorcycles or trucks using AVL CRUISE. In addition to cars including a conventional combustion engine, there exists the possibility to generate models of hybrid and electric vehicles.

The generation of a model, some possible calculation tasks, and the selectable calculation modes are presented in the following paragraphs.

- Simulation Model

A user builds up a simulation model in AVL CRUISE like a real vehicle including different modules like tire, brake, engine, clutch, and many more. A schematic representation of a

vehicle as build up in AVL CRUISE is shown in Figure 4.1. This figure shows a manual car with front-wheel drive. For a detailed explanation refer to Section 6.1.1.
Between the modules forces and signals can operate. Almost any connection between the modules is possible, some connections are even necessary for an authentic model. An example is the gas pedal in the cockpit which has to be connected to the load signal of the engine.
Each module is constructed with defining data and some input and output variables. These are transferred from and to other modules via connections between them.

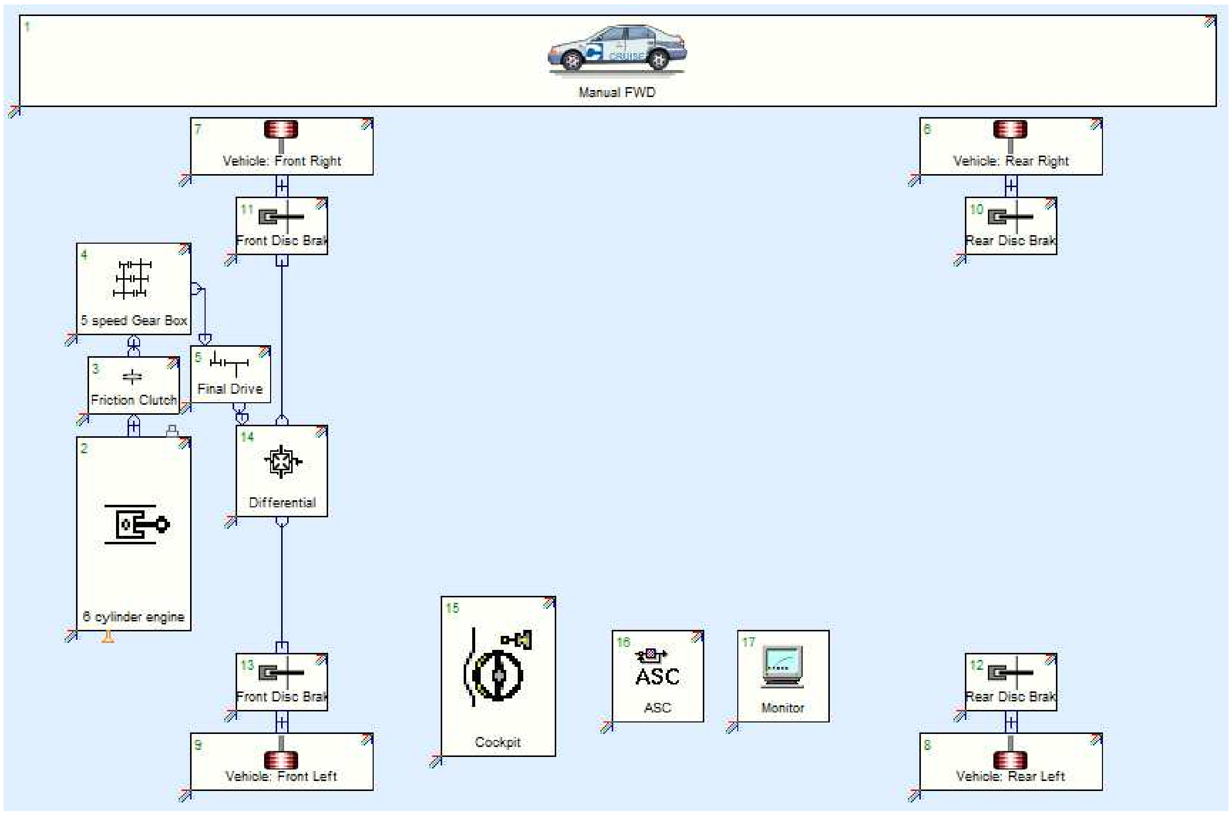


Figure 4.1: Schematic representation of an AVL CRUISE Model - Manual Front-Wheel Drive

Besides the modules in Figure 4.1 which are included in most of the vehicles there exist also components for user defined functions written in C or Fortran. In addition, interfaces can be included in a model to call external programs like MATLAB [41] during a calculation. Such external programs can be used for the definition of control sequences and modules by the user without the necessity of a new compilation of AVL CRUISE.

- Calculation Tasks

After creating a simulation model the required calculation tasks have to be chosen. Seven calculation tasks for the simulation of a vehicle are possible, for instance Cycle Run for the determination of fuel consumption and emissions and Full Load Acceleration for the calculation of the acceleration behavior. Different tasks result in different output values and curves.

In the task **Cycle Run** the emissions and the fuel consumption of a vehicle are computed according to a specific vehicle velocity profile. This profile is given as a table of the desired velocity for the vehicle model over time. Standardized driving cycles can be used to compare the emissions and the fuel consumption of vehicles for instance in Europe. The comparability of emissions is used for the definition of emission standards like the current standard Euro 5 (refer to [62]) which has to be reached for new cars which are sold in the European Union.

Driving cycles which are used for testing the modifications in AVL Design Explorer are UDC (Urban Driving Cycle) and NEDC (New European Driving Cycle). NEDC consists of four consecutive UDCs and an additional fifth part where higher vehicle velocity and a higher gear are desired. For detailed information to both driving cycles refer to the council directive of the European Union [61].

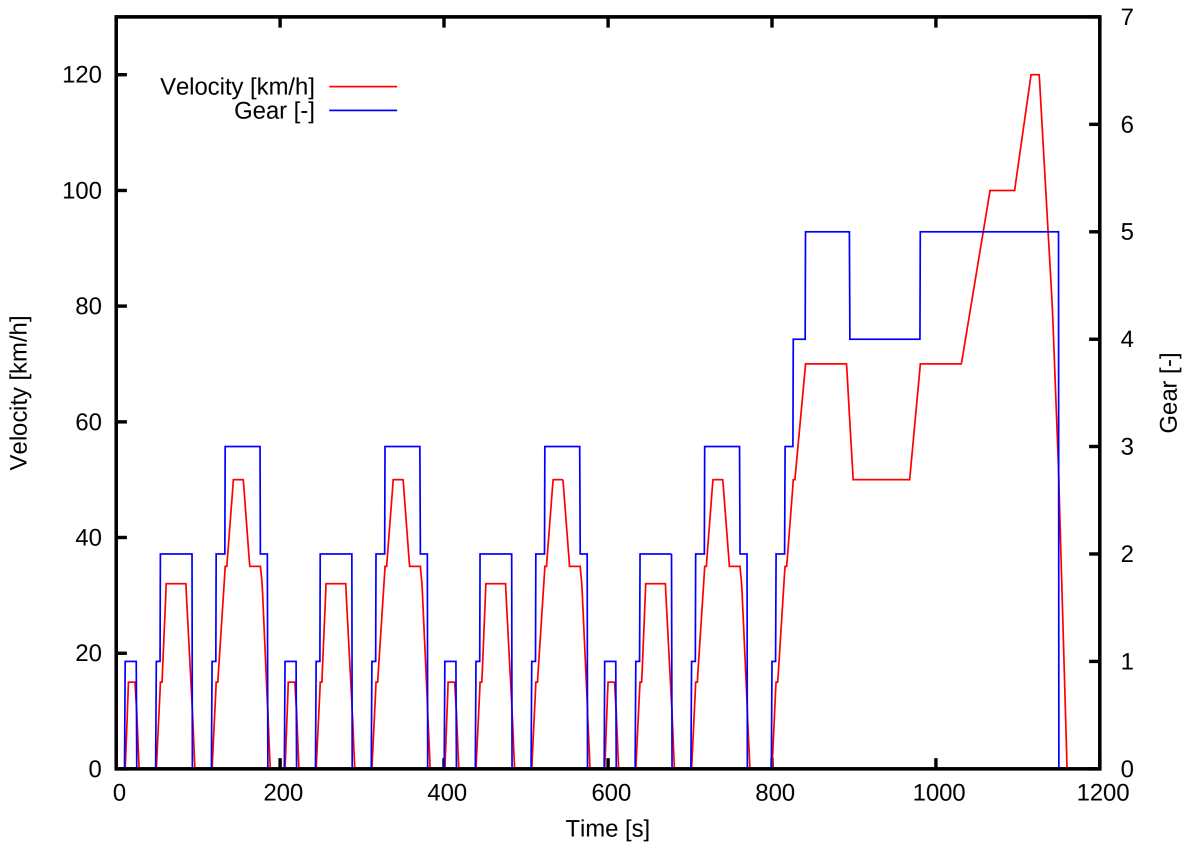


Figure 4.2: New European Driving Cycle

Euro 5 is in force for the approval of vehicles from September 1st, 2009 and for the registration and sale of new cars from January 1st, 2011. The successor is Euro 6 which will be in force from September 1st, 2014 for the approval and from September 1st, 2015 for registration and sale of new cars. These standards are defined using the NEDC. For more details refer to the regulations of the European Union [62].

The second calculation task, mentioned in this thesis, is **Full Load Acceleration**. Three subtasks exist for this task:

- **Maximum Acceleration in all Gears**:
  - For each gear, acceleration and velocity are calculated at full throttle.
- **Shifting Gears from Standstill**:
  - The acceleration from standstill to the maximal possible velocity is calculated using this task.
- **Elasticity**:

  - The acceleration for instance from 60 km/h to 120 km/h is calculated with or without gear shifting. In addition, the acceleration between two engine speeds without shifting can be calculated.

For more information about these tasks and other selectable tasks refer to Pfau [49] and the AVL CRUISE User Guide [6].

- Calculation Modes

A user can choose one of the following calculation techniques for each of the selected calculation tasks.

- **Stationary**:
  - The result values for the model are calculated for each engine speed of the vehicle assuming that no acceleration of the vehicle affects the behavior of the model. Stationary means that velocity is taken into account for the calculation, but no acceleration.
- **Quasi-stationary**:
  - The result values for the model are calculated similarly as in the stationary computation, but a fixed value for the acceleration of the vehicle is allowed. Other than in the Stationary calculation, time points can be defined for which the calculation has to be performed. This enables for example the calculation of the task Cycle Run.
- **Simulation**:
  - The behavior of the vehicle model is simulated including a driver which controls the velocity and the acceleration of the vehicle. A module named Driver acts as this controller.

Due to the more restricted calculation of the Stationary and the Quasi-stationary mode, using a fixed vehicle acceleration, the calculation time for the Simulation mode is much higher than for the other ones.

The choice of the calculation technique depends on the chosen task, the required results, and the desired time for the calculation. For instance, the previously mentioned tasks Full Load Acceleration and Cycle Run can be calculated either with quasi-stationary or simulation whereas the task Constant Drive can only be calculated with stationary computation. The task Constant Drive computes emissions and fuel consumption while driving with constant speed so there is no acceleration of the vehicle which has to be taken into account. Additionally, the maximum achievable velocity can be obtained when the calculation task Constant Drive is activated.

5 Implementation

This section is devoted to the implementation part of the thesis. At the beginning the implementation of the Genetic Algorithm for optimization problems with continuous design variables is described. Afterwards, the changes in the code to enable the AVL Design Explorer to handle integer design variables, which was the goal of this thesis, are presented.

5.1 Optimization with Continuous Design Variables

Parts of the source code of the AVL Design Explorer, mainly regarding the Genetic Algorithm for single and multi-objective optimization problems, are described in this section. Prior to this thesis only the continuous optimization algorithm Non-dominated Sorting Genetic Algorithm II (NSGA-II, refer to Section 3.2 and Deb et al. [13]) was included in the AVL Design Explorer.
Besides this algorithm the AVL Design Explorer provides other optimization algorithms and the possibility to do Design of Experiments. For more details regarding the optimization software AVL Design Explorer refer to section 4.1 and to the AVL Design Explorer User Guide [1].

Since AVL Design Explorer is implemented in Python, the main part for handling the optimization is programmed in Python. The optimization algorithms, which are called from the Python routines, are implemented in Python, C++, and Fortran, depending on the algorithm.

As shown in Figure 5.1, the NSGA-II for **continuous design variables** consists of three steps, where the first one is the same for all optimization algorithms. This structure was not changed during the modifications for the handling of integer and mixed integer optimization problems. The three steps are:

1. A **Python Script** manages the communication between the optimization routines and one of AVL's simulation programs, for instance AVL CRUISE.

2. The **Optimization Interface** translates the input data of the current optimization instance, defined by the Python script, into a C++ data structure.

3. The **Optimization Routine** NSGA-II (Section 3.2) is implemented in the C++ library MOMHLib++ [44] (Multiple Objective Metaheuristics Library in C++).

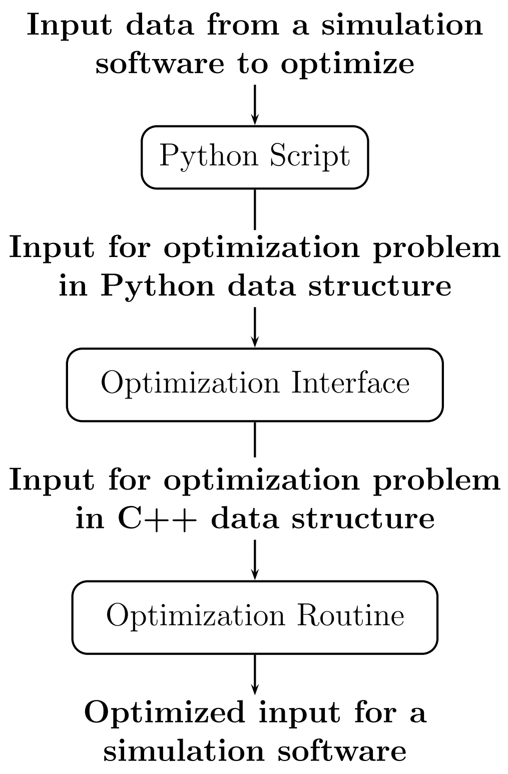

Input data from a simulation software to optimize
Python Script
Input for optimization problem in Python data structure
Optimization Interface
Input for optimization problem in C++ data structure
Optimization Routine
Optimized input for a simulation software

Figure 5.1: The main steps of the program using the NSGA-II and its information flow

The second and the third step are different for the Nelder-Mead method and the Nonlinear Programming Quadratic Linesearch (NLPQL). For the Nelder-Mead method only the Python Script is needed because it is implemented directly in Python. The input data for the NLPQL is parsed to C++ using the Optimization Interface, but a Fortran routine and no C++ library is called to perform optimization.

These three main parts of the program are described in detail in the following subsections, mainly regarding the implementation of the Non-dominated Sorting Genetic Algorithm II for optimization problems with continuous design variables.

5.1.1 Python Script

The base file to run the optimization process in the AVL Design Explorer is the Python file *Optimization_Plan.py*, where the different algorithms are called and the optimization problem is defined. This file is mainly an interface to the different optimization algorithms. The equivalent file for Design of Experiments is the Python file *DOE_Plan.py*.

The main class of *Optimization_Plan.py* is `Optplan`. This is the base class for the derived classes for all single and multi-objective optimization algorithms implemented in the AVL Design Explorer (refer to Section 4.1):

- NSGA-II: `class Opt_GENETIC(Optplan)`
- NLPQL: `class Opt_NLPQL(Optplan)`
- Nelder-Mead method: `class Opt_Simplex(Optplan)`

These derived classes include similar methods which differ according to the different purposes for the three optimization algorithms.

Some important methods of the base class and the derived classes are listed below. The methods of the base class and the class implementing Nelder-Mead which are not in the following list are mainly used for the evaluation of the results.

- `class Optplan`:
  - In `setProblemDefinition` the optimization problem with its design variables, constraints, and objectives is defined. `setProblem`, which is implemented in the wrapper to C++, is called to set up the C++ optimization instance, when NSGA-II or NLPQL is the selected optimization algorithm.
  - The method `runOptimization` calls `setProblemDefinition`. In addition, the C++ wrapper is called by `runOptimization` to perform the optimization when NLPQL or NSGA-II is the required algorithm.
- `class Opt_GENETIC(Optplan)`:
  - In `setMethodParameters` the parameters for the Genetic Algorithm are set, for instance mutation and crossover probability and the maximum number of generations.
  - In `runOptimizationAlgorithm`, called by `runOptimization` in `Optplan`, the optimization algorithm of the C++ library is called by a wrapper.
- `class Opt_NLPQL(Optplan)`:
  - In `setMethodParameters` the parameters for the NLPQL are set, for instance the maximum number of iterations and the desired final accuracy.
  - In `runOptimizationAlgorithm`, called by `runOptimization` of the base class, the Fortran routine of the NLPQL optimization algorithm is called by a C++ wrapper.
- `class Opt_Simplex(Optplan)`:
  - In `setMethodParameters` the parameters for Nelder-Mead are set, for instance the maximum number of iterations and two tolerance parameters.

  - This class has its own method `runOptimization` which handles the call of the optimization algorithm because Nelder-Mead is implemented directly in this class without calling an external library.
  - In `runOptimizationAlgorithm`, called by `runOptimization`, the optimization is performed.

From these Python classes, methods in C++ and a Fortran routine are called by a C++ wrapper. For this purpose the C++ wrapper *MOMH_wrapper.cpp* is described in the following subsection.

5.1.2 Optimization Interface

In *MOMH_wrapper.cpp* the instance of an optimization problem, defined by the Python script, is parsed into a C++ data structure and the desired optimization algorithm is called. This can be either the Non-dominated Sorting Genetic Algorithm II in C++ or the Nonlinear Programming Quadratic Linesearch in Fortran. The focus here is on the NSGA-II.
Important methods in *MOMH_wrapper.cpp*, called by *Optimization_Plan.py*, are the following:

- `set_Problem` parses common parameters of the NLPQL or NSGA-II instance coming from Python to C++ parameters and calls the C++ method `setProblem` to set up the C++ instance. Examples for such parameters are:
  - Number of design variables: `int iNumDV`
  - Lower bounds: `vector<double> dLowerB`
  - Upper bounds: `vector<double> dUpperB`
  - Objective types: `vector<int> iObjType`
    (these can be either a minimum, maximum, or a target value)
  - Constraint types: `vector<int> iConstrType`
    (0: equality, 1: inequality (lower bound only), 2: inequality (upper bound only))
- `set_ParameterGA` parses the specific parameters for NSGA-II and calls the C++ method `setParameterGA` to set the parameters in the C++ instance. Examples for the NSGA-II specific parameters are:
  - Mutation probability: `double dMutationProbability`
  - Crossover probability: `double dCrossoverProbability`
  - Number of generations: `int iNumGeneration`
  - Size of the populations: `int iNumPopulation`
  - Reduction parameter: `double dPopulationGeometricalFactor`
    (for the controlled approach as described in Section 3.2.5)

- In `set_ParameterNLPQL` the parameters for the NLPQL algorithm are parsed and the method `setParameterNLPQL` is called to set the C++ parameters for NLPQL.
- The method `run_Algorithm` differs, when the value `typeAlgorithm` is set to 0 or 1:
  - NSGA-II will be called if `typeAlgorithm` equals 0, otherwise NLPQL will be the desired algorithm.
    - If NSGA-II is the desired algorithm, `Run()` will be called which is implemented in the C++ file *constrainednsgaiic.cpp*.
    - To perform NLPQL optimization, `nlpql_optimize` is called. This method is a Fortran routine, implemented by Scheucher [55] in *nlpql_optimize.f90*.
  - `set_ParameterGA` and `set_ParameterNLPQL` are also called according to the value `typeAlgorithm`.

The methods above set all parameters and build up the complete instance of the optimization problem.

5.1.3 Optimization Routine

The optimization is performed in the third step. When NSGA-II is the used optimization algorithm, the desired algorithm is implemented, as described by Deb et al. [13] and Deb [7], in the C++ file *constrainednsgaiic.cpp*. A brief overview of the implemented algorithm (for more details refer to Section 3.2) is presented below.

Besides the main C++ file there exists another major file for the algorithm. The recombination, using crossover and mutation operators, is implemented in *newsolution.cpp*. This is also the file containing the main modifications of the Genetic Algorithm for handling integer design variables.
The implementation of the NSGA-II in the file *constrainednsgaiic.cpp* is described in the following.

During the optimization process, the three different populations P, Q, and R for each generation are handled using the following variables in the code:

- The parent population `P` with size `m_iPopulationSize`.
- The child population `Q` with size `m_iPopulationSize`.
- The merged population `R`: parent|child with size `m_iDoubledPopulationSize`.

After starting the optimization process the initial population `P` is generated as a set of `m_iPopulationSize` random solutions.
By means of the Fast Non-dominated Sorting Algorithm as described in Section 3.2.1 and implemented in *tmotool.cpp*, `P` is divided into the set of sets `iF`. The resultant `iF` is used for the fitness value assignment to each individual in `P` using the non-dominated rank.

The child population `Q` of the initial generation consists of individuals where each is computed by the recombination of two solutions. Each of these two solutions is one with minimum non-dominated rank of two randomly chosen solutions in `P`. This selection operator is called binary tournament selection (refer to Section 3.2.4) and is implemented in *tmotool.cpp*.

- To perform recombination, crossover and/or mutation is used (Section 3.3) depending on the crossover and mutation probabilities. The routines of the crossover and the mutation operators are implemented in *newsolution.cpp*.

The following steps are executed until a stopping criterion like the maximum number of generations is reached:

1. After merging `P` and `Q` to one population `R`, this double population is sorted according to non-domination resulting in the set of sets `iF`. The first set in `iF` is the set of the `R`-efficient solutions.

2. According to `iF` each solution in `R` gets a fitness value, the non-dominated rank. In addition, for each solution in `R` the crowding distance is calculated as shown in Section 3.2.2.

    - **Recall 5.1** *The crowding distance is a measure for the largest region around a solution without any other solution.*

3. The parent population `P` for the subsequent generation consists of half of the solutions in `R`. These solutions are chosen by the controlled approach as described in Section 3.2.5 using the predefined reduction parameter.

4. From this parent population `P`, a child population `Q` is computed. For this reason, two solutions of `P` are recombined to one solution for `Q`. Each of these two solutions is selected by choosing randomly two individuals of `P` and taking the maximum with respect to the crowded comparison operator (binary tournament selection).

    - **Recall 5.2** *When the crowded comparison operator is used, then a solution is better, when its non-dominated rank (fitness) is smaller or if it has the same rank and a bigger crowding distance (refer to Section 3.2.3).*

5. Stop, when the maximum number of generations is reached.

5.2 Optimization with Continuous and Integer Design Variables

The modifications in the source code of the AVL Design Explorer to enable optimization of integer design variables using a Genetic Algorithm are presented in this section.

For enabling the Genetic Algorithm NSGA-II to handle integer and mixed integer optimization, operators for the recombination of integer variables are necessary. For this reason, some of the mutation and crossover operators which are presented in Section 3.5 are implemented in the file *newsolution.cpp*. The recombination operators of the NSGA-II for optimization problems with continuous design variables are also implemented in this C++ file.

The implemented operators for crossover and mutation are listed below. For details about these methods refer to Section 3.3 and Section 3.5

Implemented crossover operators:

- Simulated Binary Crossover
- Scattered Crossover
- Arithmetic Crossover

Implemented mutation operators:

- Parameter-based Mutation
- Random Change Mutation
- Particle Swarm Mutation

The desired operator is determined by four integer variables which have to be set equal to the corresponding number:

- `croAlgInt` chooses the crossover operator for integer design variables.
- `croAlgCont` chooses the crossover operator for continuous design variables.
- `muAlgInt` chooses the mutation operator for integer design variables.
- `muAlgCont` chooses the mutation operator for continuous design variables.

Due to the results of the tests which are shown in Section 6 the crossover operator Scattered Crossover is chosen as the default crossover operator for the integer design variables. Initial test runs also showed that the Parameter-based Mutation operator is the best choice of these three as the mutation operator for the integer design variables. The continuous design variables are still handled by the operators Simulated Binary Crossover and the Parameter-based Mutation.

Simulated Binary Crossover and the Parameter-based Mutation were implemented for continuous design variables prior to this thesis. Only a rounding function had to be added to use them also for integer design variables. The following rounding functions are implemented and one of those is used according to an integer variable which has to be set directly in the code:

- Round up and down with a probability of 50%, when `roundAlg` equals 0.
- Round off to the nearest integer, when `roundAlg` equals 1.

These rounding functions are used both for modifying the operators that have been implemented for the continuous case as well as in the Arithmetic Crossover and the Particle Swarm Mutation Operator, when the current design variable has to be an integer. The Scattered Crossover does not need a rounding function because the elements of the parents are already integers and the random numbers in the Random Change Mutation are chosen from the integer values between the bounds, so there is no rounding function necessary.

For the modification and the extension of the code some additional methods are necessary:

- `double Round` includes the two rounding functions which are called according to the value of `roundAlg`.
- `double Omega` is necessary for the Particle Swarm Optimization Mutation Operator.
- `double CheckBounds` calls the selected rounding operator of `Round`, if necessary, and checks, whether the bounds of all design variables are satisfied. If a bound is not satisfied, the variable will be set to its violated bound. The bound of an integer variable can be continuous such that this design variable is not feasible, when it is set to one of its bounds. For this reason, an integer design variable which is set to its bound is rounded. When the violated bound is the upper bound, the variable is rounded down, otherwise it is rounded up.

For the Particle Swarm Optimization Mutation Operator a solution stored in the variable `pBest` is introduced in *constrainednsgaiic.cpp*. This solution is randomly chosen out of the individuals with the best fitness value of the generation before. `pBest` is an additional transfer parameter to the recombination method in *newsolution.cpp*.

The first part of the implementation was the transfer of the necessary information about the type of the design variables from AVL CRUISE to *newsolution.cpp*. This is necessary before starting with the implementation of the recombination operators. For the transfer of the information an additional property `pType`, which is equal to one of the types `int` and `float`, is introduced for all parameters in Python. In the implementation of NSGA-II and the C++ wrapper, an additional binary array `dvType` of size `iNumDV` is introduced. Each entry of this array is either 0 or 1, depending on whether the corresponding design variable is continuous or required to be an integer. For the array `dvType` the C++ wrapper has to be extended to parse the types of the design variables to C++ and the array is added to the method `bool setProblem` as an additional transfer parameter.

Another part of the implementation work is the correction of the outputs of the AVL Design Explorer. The target is that an integer looks like an integer without a comma in the output files. For this reason, in the parts of the code where outputs are written, an additional `if` condition according to the property `pType` or the boolean array `dvType` is included.

For one output file another array `intParList` has to be included in the Python part. `intParList` stores the names of all integer design variables. This is necessary in a part of the code where only the name and the value of the design variable, which is currently written into a file, is accessible. The additional array `intParList` is needed because the variables are in general stored as doubles and not as integers. The conversion to integer values is done, if necessary.
The visual output, which is directly shown in the AVL Design Explorer, is not modified to integers by now, only the output into files.

6 Computational Experiments

The goal of this section is to apply the algorithms, which have been implemented in this thesis, to instances of mixed-integer multi-objective optimization problems, which arise in the context of optimizing vehicle models at AVL. The basic test models are described in Section 6.1. The following three sections present the obtained results.

6.1 AVL CRUISE Models

In the following the two basic test models, namely the Manual Front-wheel Drive (Section 6.1.1) and the Range Extender (Section 6.1.2) will be described.

6.1.1 Basic Test Model - Manual Front-wheel Drive

In most of the tests performed, a variation of the standard model Man_FWD (Manual Front-wheel Drive), which is distributed with AVL CRUISE, is used. This standard model has the following schematic representation in the AVL CRUISE user interface version v2011:

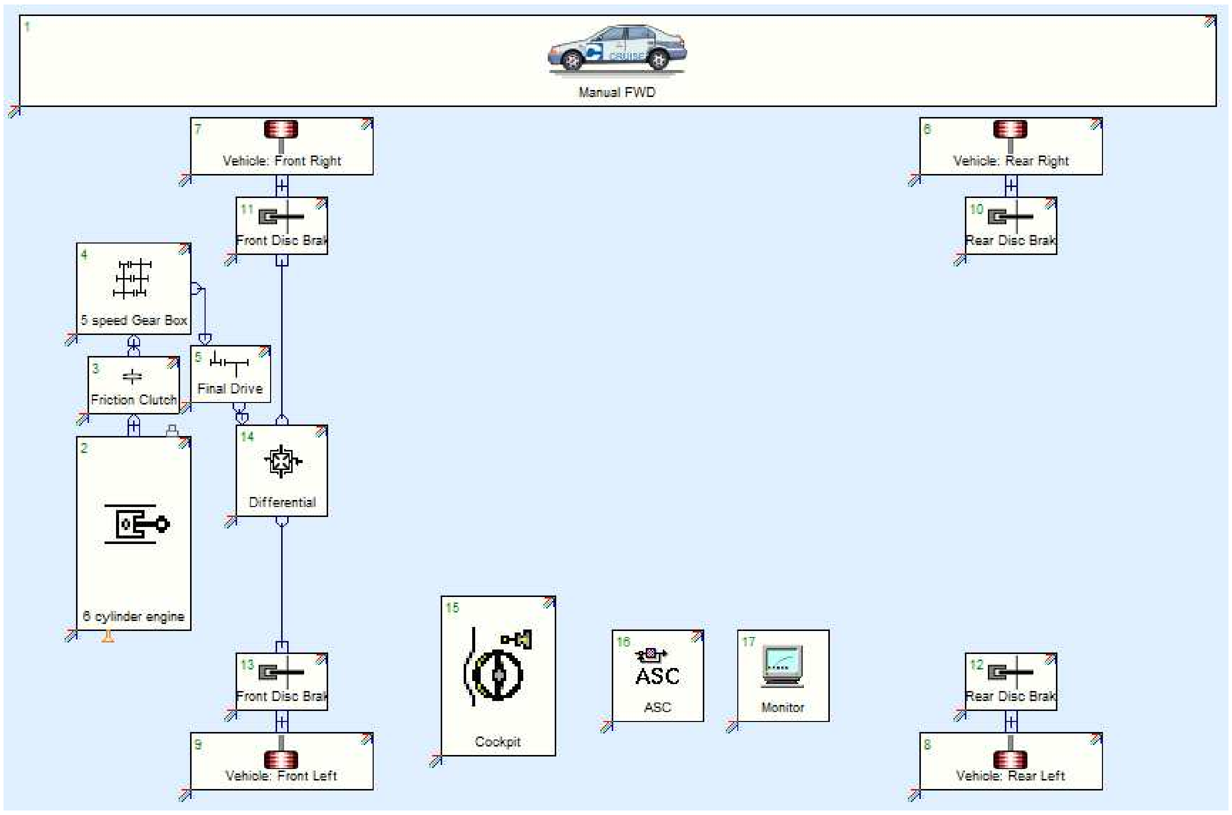
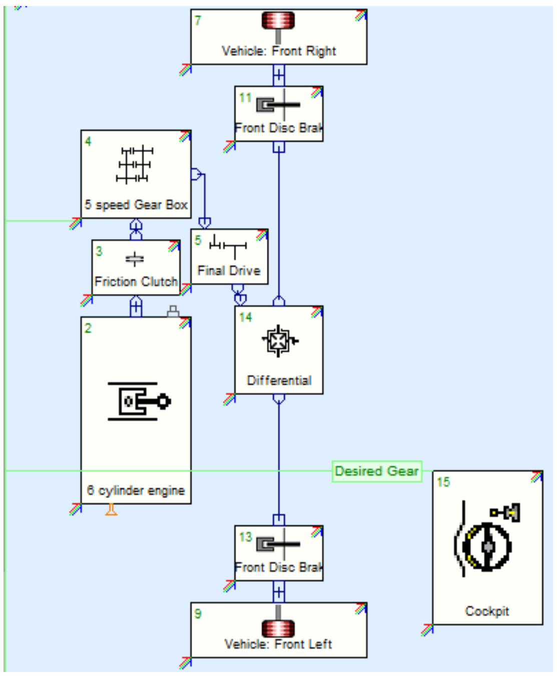
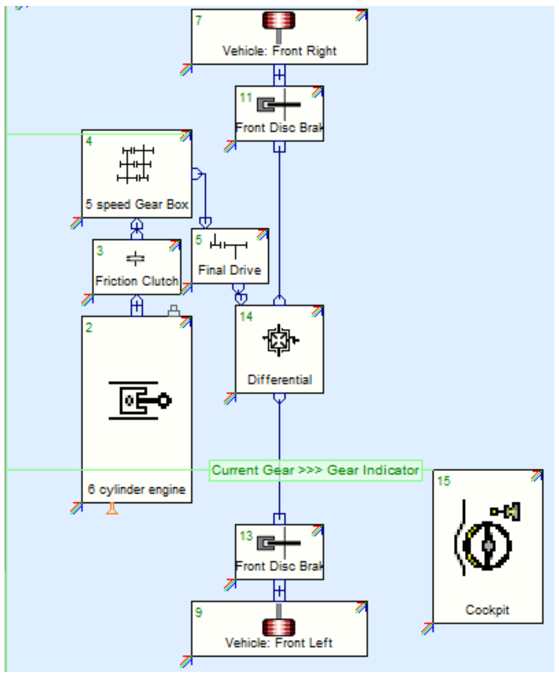


Figure 6.1: Schematic representation of the AVL CRUISE Model Man_FWD - Manual Front-wheel Drive

Via the connections which are drawn between the modules in Figure 6.1 forces are transported. The signal transportation, i.e. input and output values of the modules, can be seen by clicking on the small colored arrows which are attached to the lower left and the upper right corner of each module in the AVL CRUISE user interface.
The arrow on the lower left corner of the module corresponds to the input and the one on the upper right to the output values. For an example refer to the module Gear Box in Figure 6.2. The desired gear is the input value coming from the module Cockpit and the output value of the module Gear Box is the current gear which is transferred to the module Cockpit.



(a) Input: desired gear from Cockpit



(b) Output: current gear to Cockpit

Figure 6.2: Examples for an input and an output signal of the Gear Box

After the brief overview of the transfer of signals and forces via connections in a model, different parts of the model, which represent specific parts of a car, are described.
The values of the variables used in this model are shown in the third column of the tables below. The descriptions of the variables in the second column are taken from the AVL CRUISE User Guide [6] and the variable names match with the names used in AVL CRUISE. Additionally, the unit of the variable is provided in the fourth column.

- Vehicle:

One of the main components of all models in AVL CRUISE is the module Vehicle. It contains the general information like the volume of the gas tank or the weight of the vehicle.

Variable Name	Description	Value	Unit
Gas Tank Volume	Fuel tank volume of the vehicle.	0.075	[m^3]
Distance from Hitch to Front Axle	The hitch is the coupling point for the trailer.	3300	[mm]
Wheel Base	Distance between the front and the rear axle.	2650	[mm]
Height of Support Point at Bench Test	Height of the point where the vehicle is fixed to the wall while driving on a chassis dynamometer.	100	[mm]
Curb Weight	Weight of the empty vehicle.	1450	[kg]
Gross Weight	Maximum admissible weight.	1930	[kg]
Frontal Area	Cross section area of the vehicle.	1.88	[m^2]
Drag Coefficient	Factor of the air resistance, depending on the shape of the vehicle.	0.32	[-]
Lift Coefficient Front Axle	Considers the influence of the vehicle velocity on the front wheel loads.	0.032	[-]
Lift Coefficient Rear Axle	Considers the influence of the vehicle velocity on the rear wheel loads.	0.01	[-]
Tire Inflation Pressure Front Axle		2.0	[bar]
Tire Inflation Pressure Rear Axle		2.2	[bar]

- Tire:

The number of tires (or wheels) depends on the kind of the vehicle (motorcycle, car, ...), which is modeled. The standard model Man_FWD is a car, so there exist four tires which link the vehicle to the road. Each of these four wheels could have different defining data. In the test model Man_FWD all wheels are identical.

Variable Name	Description	Value	Unit
Static Rolling Radius	The radius of the loaded, not moving vehicle.	305	[mm]
Dynamic Rolling Radius	Distance between the center of the wheel and the road surface for the loaded, moving vehicle.	312	[mm]
Inertia Moment		0.51	[kg m^2]

- Brake:

The number of brakes coincides in general with the number of tires. It is possible to choose between drum or disc brakes. The four brakes in the test model are disc brakes.
As for the tires it is possible to define all brakes using different data. In contrast to the tires, there is a difference between the data of the two front and the two rear brakes, which is shown in the table below.

Variable Name	Description	Value	Unit
Inertia Moment front	The inertia moments of the front and the rear brakes.	0.02	[kg m^2]
Inertia Moment rear		0.015	[kg m^2]
Brake Piston Surface front	The areas of the hydraulic cylinders.	1800	[mm^2]
Brake Piston Surface rear		1500	[mm^2]
Effective Friction Radius front	The radius where the braking force applies.	130	[mm]
Effective Friction Radius rear		110	[mm]
Friction Coefficient front	Friction coefficient between brake drum and the brake shoes.	0.25	[-]
Friction Coefficient rear		0.25	[-]
Efficiency front	Considers the effects of the conversion of the hydraulic into the mechanical part of the brake.	0.99	[-]
Efficiency rear		0.99	[-]

- Combustion Engine:

For the module Combustion Engine it is possible to choose between a diesel and a gasoline engine. The engine is defined by characteristic curves, defining for instance the fuel consumption and the emissions, and a number of other variables.

The curves are not shown in this work, only the description of the variables is provided.

Variable Name	Description	Value	Unit
Engine Type	Engine Type is one of diesel and gasoline.	Gasoline	[-]
Engine Displacement	Displacement of all cylinders together.	2478	[cm^3]
Number of Cylinders		6	[-]
Number of Strokes		4	[-]
Idle Speed	Idle Speed can also be defined as a function of engine temperature.	750	[1/min]
Maximum Speed	The maximum reachable engine speed.	6000	[1/min]
Inertia Moment	Contains all parts of the engine, e.g. crankshaft and flywheel.	0.134	[kg m^2]
Fuel Density		0.76	[kg/l]

- Clutch:

When a change of the current gear is necessary, the module Clutch disconnects the engine and the drive train. After the procedure of changing the gear, the clutch reconnects the engine and the drive train.
The component Driver controls the clutch via the module Cockpit.

Variable Name	Description	Value	Unit
Inertia Moment In	The inertia moment at the drive side.	0.01	[kg m^2]

Inertia Moment Out	The inertia moment at the power take-off side.	0.01	[kg m^2]
Maximum Transferable Torque	Between drive and power take-off side.	350	[Nm]

- Gear Box:

The module Gear Box is defined by a table which contains a row for each gear, including the idle gear (gear 0) and all forward gears. The idle gear has always transmission ratio 1.0.

A variant of a gear box with five gears is part of this model, but any number of gears can be chosen. It is possible to use an automatic or a manual gear box. In the manual variant which is used in the standard model, the component Driver, which is described below, shifts the gears. An additional module Gear Box Control or Gear Box Program controls the gear shifting of an automatic model. The module Gear Box of the manual model Man_FWD, which is used in this work, uses the values listed in the table below.

The Inertia Moments In and Out, which are the inertia moments at the drive side and at the power take-off side, are the same for each gear in this model.
It is possible to define the transmission ratios of the gears in two ways. Either the value of the ratio is inserted directly or the number of teeth input and output can be chosen. Then the transmission ratio is calculated by

$$\text{Transmission Ratio} = \frac{\text{Number of Teeth Out}}{\text{Number of Teeth In}} \tag{1}$$

where Out and In abbreviate Output and Input.
When only the transmission ratio is defined, the numbers of teeth are chosen by AVL CRUISE such that the given transmission ratio and the ratio of the two chosen numbers coincide. This is eased by slightly correcting the transmission ratio to keep the numbers of teeth under a certain bound.

Gear [-]	**Transmission Ratio** [-]	**Number of Teeth In** [-]	**Number of Teeth Out** [-]	**Inertia Moment In [kg m^2]**	**Inertia Moment Out [kg m^2]**
0	1.00	10	10	0.0015	0.005
1	3.62	50	181	0.0015	0.005
2	2.22	50	111	0.0015	0.005

3	1.51	100	151	0.0015	0.005
4	1.08	25	27	0.0015	0.005
5	0.85	20	17	0.0015	0.005

- Final Drive:

The module Final Drive is a Single Ratio Transmission (SRT), which is a gear step with a fixed transmission ratio.

Variable Name	Description	Value	Unit
Inertia Moment In	The inertia moment at the drive side.	0.008	[kg m^2]
Inertia Moment Out	The inertia moment at the power take-off side.	0.015	[kg m^2]
Transmission Ratio	Define the transmission ratio either as a value or by the numbers of teeth in and out using Formula (1).	3.0	[-]
Number of Teeth In		10	[-]
Number of Teeth Out		30	[-]

- Differential:

Due to the module Differential inner and outer wheels rotate with different speed during cornering. This is necessary because the inner wheels have to cover less distance than the outer ones. Additionally, wheels at different axles can rotate with different speed in a vehicle with four-wheel drive.

Variable Name	Description	Value	Unit
Differential Lock	In an unlocked differential the two power take-off torques are the same and the speeds can be different.	unlocked	[-]

Torque Split Factor	Factor one means, that there is the same torque on both outgoing sides.	1.0	[-]
Inertia Moment In	The inertia moment at the drive side.	0.015	[kg m^2]
Inertia Moment Out 1	The inertia moments at the power take-off sides.	0.015	[kg m^2]
Inertia Moment Out 2		0.015	[kg m^2]

In addition to the modules above, there are also controls and special modules included in the model. These are presented below.

- Cockpit:

The module Cockpit is the link between the component Driver and the vehicle. Connections of this module to other ones are only transferring signals and no forces. Examples for these signals are the vehicle velocity and the vehicle acceleration, which are sent to the component Driver.

Variable Name	Description	Value	Unit
Shift Mode	The Shift Mode is either manual or automatic.	manual	[-]
Number of Gears Forward	These variables define the numbers of gears.	5	[-]
Number of Gears Reverse		1	[-]
Maximum Brake Force	The maximum brake pedal force.	100	[N]

- Anti Slip Control (ASC):

The module Anti Slip Control checks, whether the force, which should be transmitted by all wheels, does not exceed the maximum transmittable force. This module is not used for calculations in this model. If it were used for the calculation, the load signal or the clutch release would be changed when the force, which has to be transmitted, is higher than the maximum transmittable force.

- Monitor:

The module Monitor enables AVL CRUISE to show results while the calculation is running. The user can specify data, he wants to see during the calculation, such as for instance:

Variable	Unit
Vehicle Acceleration	[m/s^2]
Vehicle Velocity	[km/h]
Engine Speed	[1/min]
Engine Load Signal	[-]

There exist also components which are not contained in Figure 6.1 but are parts of the Calculation Tasks (for an introduction to AVL CRUISE and its Calculation Tasks refer to Section 4.2).
One of such additional components is the Driver which has been mentioned in the description of the modules above several times.

- Driver:

The component Driver imitates a human driver. A good imitation is achieved by input data for different purposes:

- Shifting behavior
- Starting behavior
- Driving behavior

For the starting and the driving behavior the user can define data in the expert mode, but by default this is done by AVL CRUISE using default values.

The **Shifting behavior** can be chosen to be according to vehicle velocity, engine speed or engine speed in the next gear. In the standard model, which is used for the generation of the results, the shifting is done in the task Cycle Run “according to profile”. This means, that the desired gear for each time step of the driving cycle is given by the cycle.
In the task Constant Drive, which is used for obtaining the maximum velocity, the shifting is done according to velocity. The following table is used for the shifting process according to velocity:

Gear [-]	Upshifting [km/h]	Downshifting [km/h]
1	25.744	23.0
2	40.225	38.0
3	64.360	60.0
4	74.014	70.0

This table defines at which vehicle velocity the driver has to shift up to the next gear and at which velocity back down to this gear.

The **Starting behavior** and the **Driving behavior** are set to the default values in the standard model.

6.1.2 Model of a Hybrid Vehicle - Range Extender

The second test model is a Range Extender which is also distributed with AVL CRUISE [6]. Figure 6.3 shows the schematic representation of this model in the AVL CRUISE user interface version v2011. A Range Extender is a plug-in hybrid electric vehicle (PHEV). In contrast to hybrid vehicles it is also possible to recharge the battery using a plug to an external power source. The plug-in hybrid concept combines the advantages of a hybrid and an electric vehicle. When the battery charge decreases, the combustion engine recharges the battery, so it is possible to drive long distances with a Range Extender.

In addition, for short distances the emissions (especially when using renewable energy to recharge the battery) are less than when driving a car which contains only a conventional combustion engine. The operating costs also decrease due to lower energy costs. On the other side, the acquisition costs are higher than when buying a conventional vehicle.

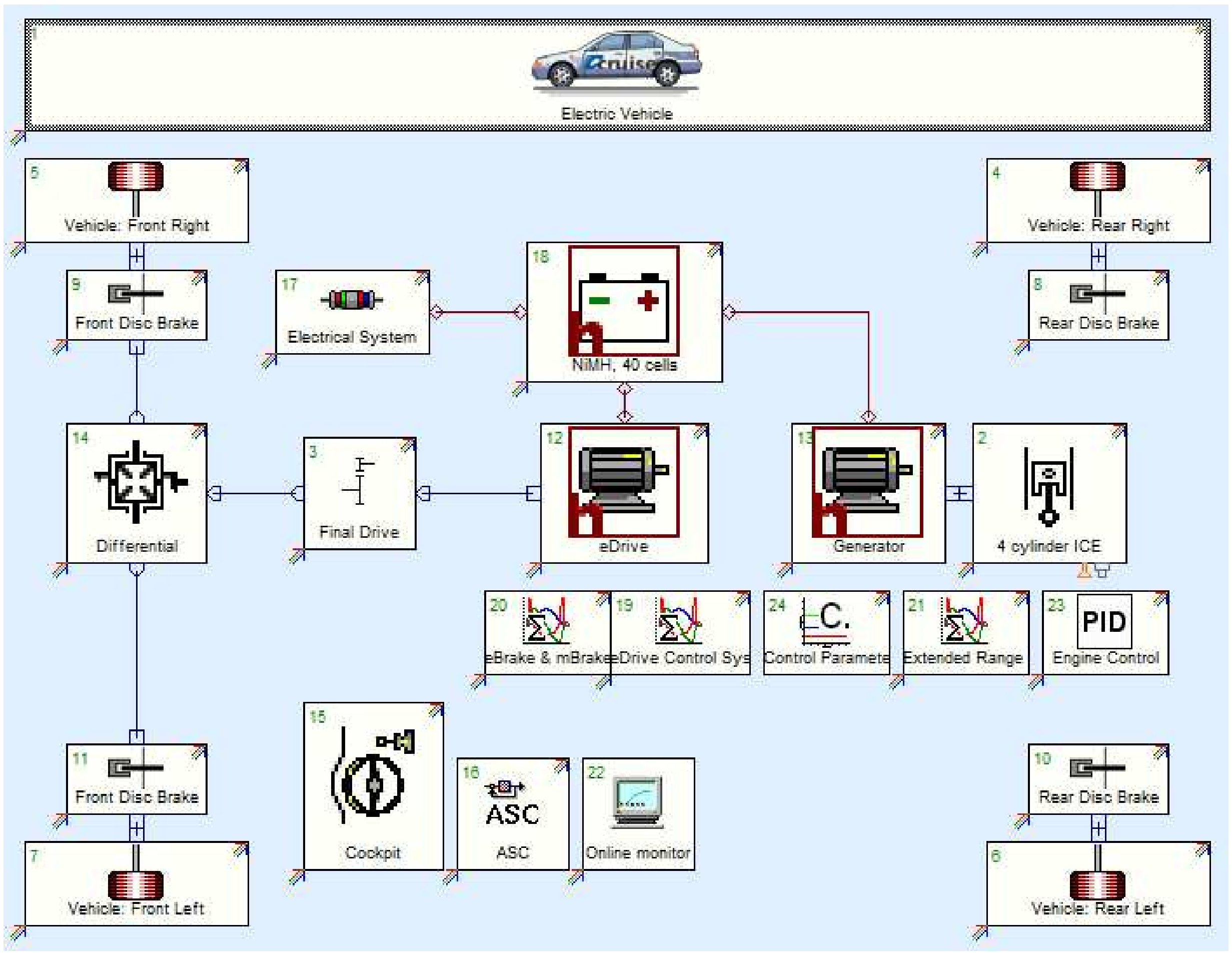


Figure 6.3: Schematic representation of the AVL CRUISE Model of a Range Extender

The modules which are used to build the model of the Range Extender are presented in the following paragraphs. The descriptions are taken from the AVL CRUISE User Guide [6].
The following modules are also included in the first test model Man_FWD. For more details about these modules refer to Section 6.1.1.

- Vehicle:

The module Vehicle of the Range Extender defines an electric vehicle without a gas tank.

Variable Name	Description	Value	Unit
Gas Tank Volume	Fuel tank volume of the vehicle.	0.0	[m^3]
Distance from Hitch to Front Axle	The hitch is the coupling point for the trailer.	2467	[mm]
Wheel Base	Distance between the front and the rear axle.	2467	[mm]
Height of Support Point at Bench Test	Height of the point where the vehicle is fixed to the wall while driving on a chassis dynamometer.	500	[mm]
Curb Weight	Weight of the empty vehicle.	1200	[kg]
Gross Weight	Maximum admissible weight.	1580	[kg]
Frontal Area	Cross section area of the vehicle.	1.97	[m^2]
Tire Inflation Pressure Front Axle		2.4	[bar]
Tire Inflation Pressure Rear Axle		2.4	[bar]
Resistance Function:	The driving resistance is computed by a function without a reference vehicle.		
Constant Part		143.06	[N]
Linear Part		0.0	$\left[\frac{N}{km/h}\right]$
Square Part		0.03399	$\left[\frac{N}{(km/h)^2}\right]$

- Tire:

The Range Extender model models a car, so there exist four tires containing identical data which link the vehicle to the road.

Variable Name	Description	Value	Unit
Static Rolling Radius	The radius of the loaded, not moving vehicle.	287	[mm]

Dynamic Rolling Radius	Distance between the center of the wheel and the road surface for the loaded, moving vehicle.	301	[mm]
Inertia Moment		0.1431	[kg m^2]

- Brake:

The four brakes in the Range Extender are disc brakes.
Analogous to the model Man_FWD there is a difference between the data of the two front and the two rear brakes, which is shown in the table below.

Variable Name	Description	Value	Unit
Inertia Moment front	The inertia moments of the front and the rear brakes.	0.02	[kg m^2]
Inertia Moment rear		0.015	[kg m^2]
Brake Piston Surface front	The areas of the hydraulic cylinders.	1800	[mm^2]
Brake Piston Surface rear		1500	[mm^2]
Effective Friction Radius front	The radius where the braking force applies.	130	[mm]
Effective Friction Radius rear		110	[mm]
Friction Coefficient front	Friction coefficient between brake drum and the brake shoes.	0.25	[-]
Friction Coefficient rear		0.25	[-]
Efficiency front	Considers the effects of the conversion of the hydraulic into the mechanical part of the brake.	0.99	[-]
Efficiency rear		0.99	[-]

- Combustion Engine:

The Range Extender uses the conventional combustion engine for recharging the battery. For the module Combustion Engine it is possible to choose between a diesel and a gasoline engine.

Variable Name	Description	Value	Unit
Engine Type	Engine Type is one of diesel and gasoline.	Gasoline	[-]
Engine Displacement	Displacement of all cylinders together.	2000	[cm^3]
Number of Cylinders		4	[-]
Number of Strokes		4	[-]
Idle Speed	Idle Speed can also be defined as a function of engine temperature.	800	[1/min]
Maximum Speed	The maximum achievable engine speed.	6000	[1/min]
Inertia Moment	Contains all parts of the engine, e.g. crankshaft and flywheel.	0.134	[kg m^2]
Fuel Density		0.76	[kg/l]

- Final Drive:

The module Final Drive is a Single Ratio Transmission (SRT), which is a gear step with a fixed transmission ratio.

Variable Name	Description	Value	Unit
Inertia Moment In	The inertia moment on drive side.	0.01	[kg m^2]
Inertia Moment Out	The inertia moment on power take-off side.	0.015	[kg m^2]
Transmission Ratio	Define the Transmission Ratio either as a value or by the numbers of teeth in and out using Formula (1).	3.737	[-]
Number of Teeth In		19	[-]
Number of Teeth Out		71	[-]
Efficiency		0.96	[-]

- Differential:

Due to the module Differential inner and outer wheels rotate with different speed during cornering.

Variable Name	Description	Value	Unit
Differential Lock	In an unlocked differential the two power take-off torques are the same and the speeds can be different.	unlocked	[-]
Torque Split Factor	Factor one means, that there is the same torque on both outgoing sides.	1.0	[-]
Inertia Moment In	The inertia moment on drive side.	0.015	[kg m^2]
Inertia Moment Out 1	The inertia moments at the power take-off sides.	0.015	[kg m^2]
Inertia Moment Out 2		0.015	[kg m^2]
Efficiency		0.96	[-]

- Cockpit:

The module Cockpit is the link between the component Driver and the vehicle.

Variable Name	Description	Value	Unit
Shift Mode	The Shift Mode is either manual or automatic.	manual	[-]
Number of Gears Forward	These variables define the numbers of gears.	1	[-]
Number of Gears Reverse		1	[-]
Maximum Brake Force	Maximum brake pedal force.	100	[N]

- Anti Slip Control (ASC):

The module Anti Slip Control of the Range Extender is not used for calculations.

- Monitor:

The module Monitor enables AVL CRUISE to show results while the calculation is running. The user can specify data, he wants to see during the calculation, such as for instance:

Variable	Unit
Velocity	[km/h]
SOC	[%]
Engine Speed	[1/min]

SOC is the current state of charge of the battery, given as a percentage of the maximum charge, which is defined in the module Battery H.
The following modules are only part of the model Range Extender.

- Battery H:

The Range Extender contains a NiMH (Nickel-metal Hydride) battery. Battery H is the module, which simulates the battery for the hybrid model. For example, the number of battery cells and the maximum charge of the battery are defined in this module.

Variable Name	Description	Value	Unit
Maximum Charge	Maximum charge of each cell.	10.0	[Ah]
Initial Charge	Charge of the cell at the beginning of the calculation defined as a percentage of the maximum charge.	60.0	[%]
Nominal Voltage	This voltage depends on the used material.	7.2	[V]
Maximum Voltage		9.0	[V]
Minimum Voltage		6.0	[V]
Number of Cells per Cell-Row		40	[-]
Number of Cell-Rows		2	[-]
Internal Resistance Charge	Internal resistance of a cell in charge mode.	0.0197	[Ohm]
Internal Resistance Discharge	Internal resistance of a cell in discharge mode.	0.0269	[Ohm]

- Generator:

The module Generator is simulated by an electric machine. This electric machine supplies the battery with current by means of the module Combustion Engine when a recharge of the battery is necessary.

Variable Name	Description	Value	Unit
Type of Machine	An asynchronous motor (ASM) and a permanent magnetic synchronous motor (PSM) are selectable.	ASM	[-]
Nominal Voltage	The nominal voltage has to be the same as in the onboard network.	320.0	[V]
Inertia Moment	The inertia moment of the generator.	0.0001	[kg m^2]
Maximum Speed	The maximum angular velocity the electric machine can run at.	10000.0	[1/min]
Drag Torque at Maximum Speed		0.0	[Nm]
Initial Temperature	Temperature at the start of the calculation.	20.0	[C]

- eDrive:

The module eDrive is also simulated by an electric machine. In contrast to the module Generator, the module eDrive does not recharge the battery, but it uses the current of the battery to move the vehicle. The variables in this module are set to the same values as the variables of the generator, but the module eDrive contains additional characteristic maps which are not shown in this thesis.

- Electrical System:

The module Electrical System is an electric consumer. This is represented by Ohmic resistors in the onboard network. These resistors simulate the loss of electric current. An electric consumer can for instance be the air conditioning in a vehicle.

Variable Name	Description	Value	Unit
Nominal Voltage	This defines the voltage in the onboard network.	320.0	[V]
Threshold Value		0.5	[-]
Direction	Positive direction means that the switch is turned on, if and only if the input is above the threshold value.	positive	[-]
Reference	Absolute means that the resistance is switched on and off according to the switch.	absolute	[-]
Exceeding Value Range	Admissible means that the resistance will be extrapolated if the engine speed is outside the defined range.	admissible	[-]

- PID Control:

The module PID Control (Proportional-Integral-Derivative Control) computes the load signal for the engine from the two input values desired engine speed and actual engine speed.

Variable Name	Description	Value	Unit
Proportional Parameter		20.0	[-]
Integral Parameter		0.0002	[1/s]
Derivative Parameter		0.0025	[s]
Output Value Limitation:	The output value (load signal) can be between 0.0 and 1.0.		
Minimum		0.0	[-]
Maximum		1.0	[-]

- Constants:

Some constants are defined in this module which are used during the simulation. The state of charge (SOC) is given as a percentage of the maximum charge which is defined in the module Battery H:

Variable Name	Value	Unit
Maximum Brake Pressure	50	[bar]
SOC_Min	50	[%]
SOC_Max	60	[%]

- Functions:

The following C-functions are used in the Range Extender model:

Function Name	Description
eBrake & mBrake Unit	Conversion of eDrive Torque to Brake Pressure.
eDrive Control System	Transition from driving to braking.
Extended Range Control	Starting and shutting down engine and charging the battery.

- Driver:

The component Driver of the Range Extender is the same as the one defined in Section 6.1.1 for Man_FWD.

6.2 Test Problem 1: Optimization of Gear Ratios

The first test problem concerns optimizing the gear ratios. The underlying test model Man_FWD has been introduced in Section 6.1.1. The optimization of the gear ratios is chosen as the first test problem, because it is possible to optimize the gear ratios using the previously existing continuous optimization algorithm and thus to compare the results of the continuous approach to the results of the modified algorithm, which optimizes the integer version of this problem.

To optimize the gear ratios is a realistic example for an integer optimization problem because the optimization of the transmission ratio can result in an infeasible or very costly number of teeth. For this reason, it is better to optimize the teeth numbers directly and to apply an optimization routine that can deal with integer variables.

As shown in Section 6.1.1, the module Gear Box is defined by a table with one row for each gear. The design variables are the transmission ratios of the gears without the idle gear. There is no possibility to use a variable, which is stored in tables in AVL CRUISE modules as a design variable. For this reason, the following modification of the standard model of Figure 6.1 is done, as shown by the schematic representation of the modified model in Figure 6.4.

The module Gear Box, which contains the data for the five different forward gears and the idle gear, is divided into five separate boxes. Each of these boxes is defined by data in a table which consists of two rows. The first row contains the data of the idle gear and the second one the data of the gear which is defined in this box. The boxes are activated by switches which choose the box of the currently desired gear. The information of the desired gear is an input information for each of the five switches coming from the module Cockpit. The current gear is transferred from the function Actual Gear back to the module Cockpit.

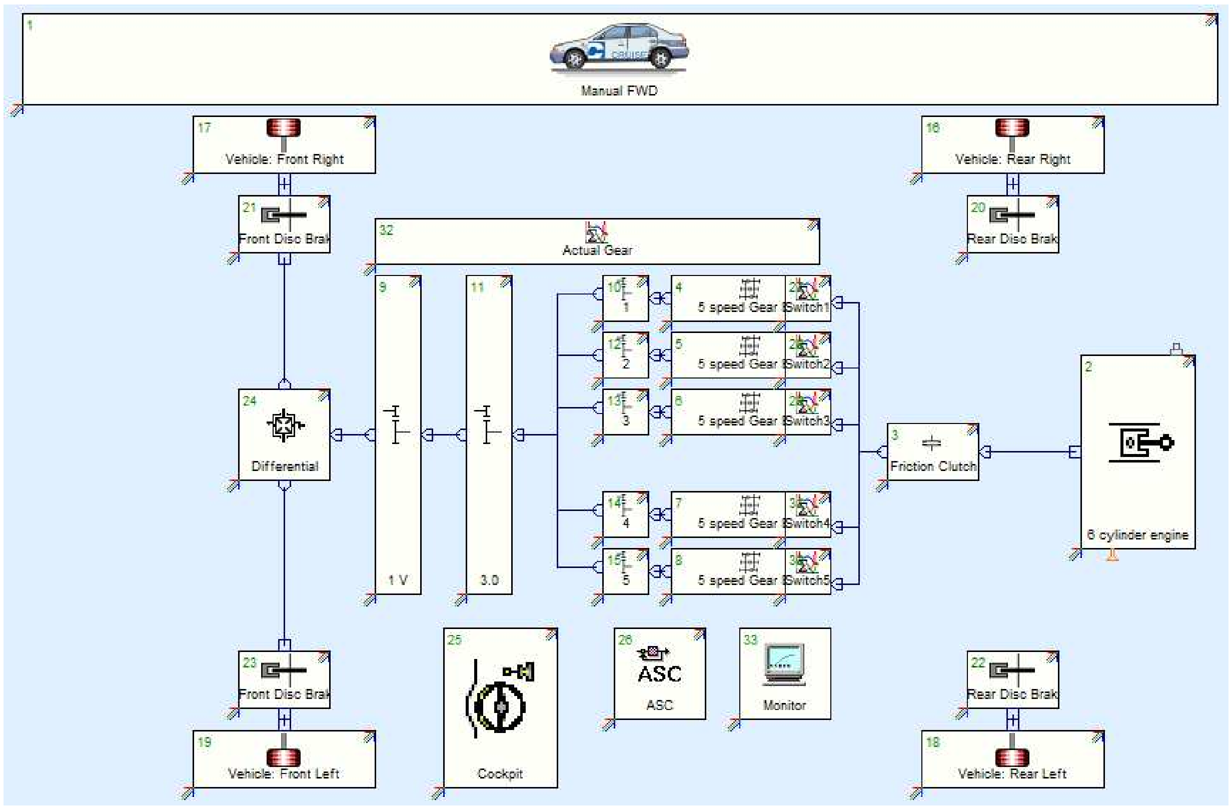


Figure 6.4: Schematic representation of the modified AVL CRUISE Model

6.2.1 Optimization Instance

The separation into five different boxes is done such that after each box an additional Single Ratio Transmission (SRT) can be placed. The Single Ratio Transmissions have a transmission ratio as a single variable and the numbers of teeth as two single variables without a table in the module definition. So it is possible to use the variables of the SRT as design variables.

The transmission ratios of the SRTs are the base values for the design variables in the optimization problem and thus set to 1.0. The base value can be defined as the continuous transmission ratio or as a pair of two numbers of teeth in the integer version. In the continuous version, the continuous variable is the design variable for this gear. When the integer version is chosen, the number of teeth output is the used design variable. The continuous transmission ratio is computed internally by AVL CRUISE using the numbers of teeth via Formula (1).

Due to this calculation the base values for the numbers of teeth have to be the same for input and output in each module Single Ratio Transmission such that the transmission ratio equals to one.

By making use of the construction, which uses the modules Gear Box and Single Ratio Transmission for each gear, the gear ratio of each gear is the product of the transmission ratios in both modules. This means, that the transmission ratio in the SRT is a percentage for the change of the ratio in the module Gear Box.

The two considered objective functions of the first test problem are:

- Minimal Fuel Consumption
- Minimal CO2 Emission

In addition to tests with these two objective functions, a number of test runs have also been performed with the Maximum Velocity as a third objective function. The reason for the third objective function is, that a too small gear ratio for the fifth gear can decrease the maximum velocity (refer to Pfau [49]). When this objective function is added, the constraints, that the resulting gear ratios have to decrease with higher gears, have to be included into the problem. This is necessary because otherwise the gear ratio of the fifth gear can be set to a higher value than the gear ratios of the lower gears.

As mentioned above, there are different design variables in the integer and the continuous case, the numbers of teeth output and the transmission ratios of the Single Ratio Transmissions of the gears one to five.

The following table contains the lower and the upper bounds of the design variables of the integer optimization problem, the numbers of teeth of the factors for the gear ratios of gear one to gear five:

Variable Name	Lower Bound	Base Value	Upper Bound	Unit
NoT1out	300	600	720	[-]
NoT2out	300	600	720	[-]
NoT3out	300	600	720	[-]
NoT4out	300	600	720	[-]
NoT5out	300	600	720	[-]

A base value of 600 is chosen to get results, which are similar to the results of the continuous optimization problem. If a smaller base value were used, there would exist fewer possible obtainable values for the optimization algorithm.

The lower and the upper bounds of the design variables of the continuous optimization problem, the factors for the gear ratios of gear one to gear five, are shown in the following table:

Variable Name	Lower Bound	Base Value	Upper Bound	Unit
TR1	0.5	1.0	1.2	[-]
TR2	0.5	1.0	1.2	[-]
TR3	0.5	1.0	1.2	[-]
TR4	0.5	1.0	1.2	[-]
TR5	0.5	1.0	1.2	[-]

For the calculation of the lower and the upper bounds in the tables above, 50% and 120% of the base values are used.

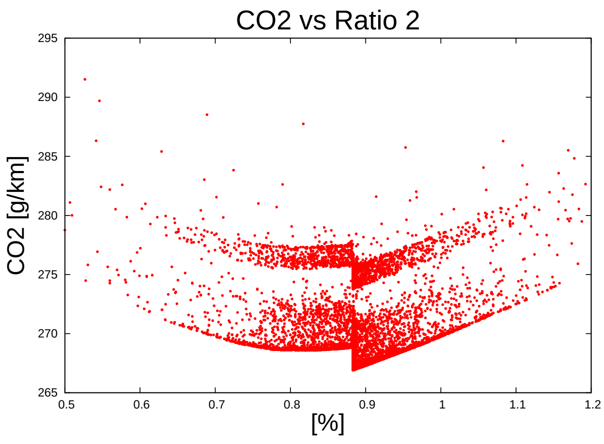
There is no constraint used in this instance with two objective functions, even though the constraints, that the gear ratios should decrease for a higher gear, would be necessary. These constraints would not change anything on the results because the obtained solutions satisfy the constraints anyway. An exception for this is the optimization problem with the maximization of the Maximum Velocity as a third objective function. In this case these constraints have to be included in the problem definition because otherwise the ratio for the fifth gear could be higher than the one of the fourth gear to achieve a higher maximum velocity.

As mentioned at the beginning of this section the goal of using this example is the comparison of results obtained with continuous optimization with ones computed by integer optimization. In addition to the comparison of continuous and integer optimization, different crossover operators are considered for the integer optimization in relation to the results of the continuous optimization. The results are shown in the following section.
The mutation and crossover operators are selected from the ones in Section 5.2 according to initial test runs.

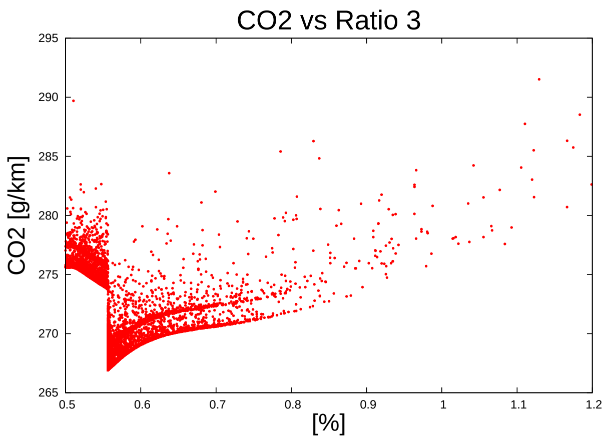
6.2.2 Results

For the comparison between the continuous and integer optimization the best obtainable solution for the continuous variant is computed. This is done by running the algorithm for 150 generations with populations of size 80. Including the two start populations this results in 12160 design points. Some of them are duplicates, but the majority differs from each other. In the integer variant of the problem there are more duplicates due to the finite number of values.

The following figures show the values of the objective functions CO2 Emission and Fuel Consumption over the factor for the gear ratio of the gears two and three as these are the most interesting ones. In the chosen Urban Driving Cycle (UDC), gear four and five are never used and gear one is rarely used because UDC simulates a drive in a city with a maximum vehicle velocity of 50 km/h. The pictures of the objective functions Fuel Consumption and CO2 Emission are similar because in general the two values are related to each other.
The bounds of the design variables are set to 0.5 for the lower bound and 1.2 for the upper bound.

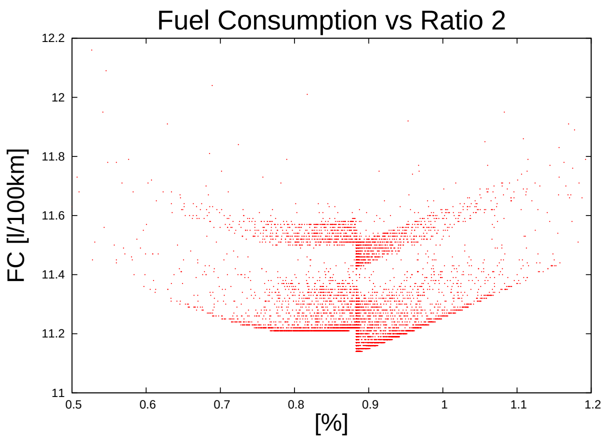


(a) Factor for the gear ratio of gear 2

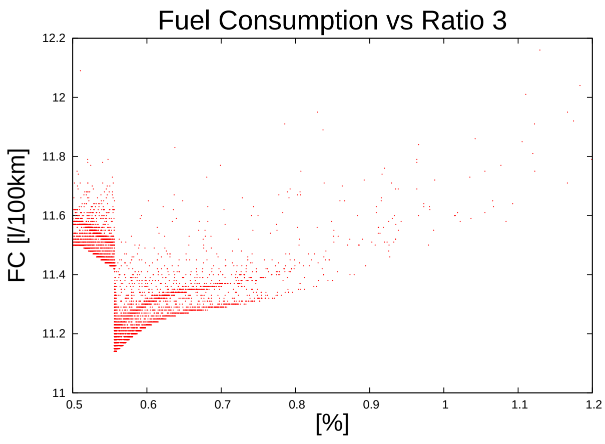


(b) Factor for the gear ratio of gear 3

Figure 6.5: CO2 Emission over the factors for the gear ratios



(a) Factor for the gear ratio of gear 2



(b) Factor for the gear ratio of gear 3

Figure 6.6: Fuel Consumption over the factors for the gear ratios

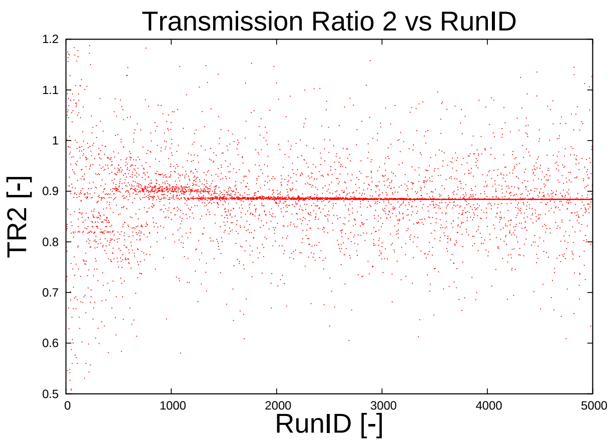
Each point in Figure 6.5 and Figure 6.6 shows the objective function value of one design point calculated using AVL CRUISE. Due to the smaller difference between the maximum and the minimum value of the Fuel Consumption there are much less different points in Figure 6.6 which shows the Fuel Consumption over the values of the design variables.

The values of the objective functions Fuel Consumption and CO2 Emission before and after the optimization of the gear ratios using the continuous optimization algorithm as described above are shown in the following table:

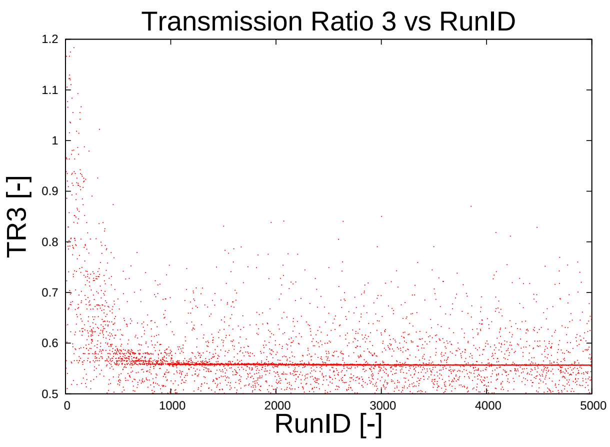
	Fuel Consumption [l/100km]	**CO2 Emission** [g/km]
Before	11.61	278.32
After	11.14	266.91

As one can see the optimization of the gear ratios results in a reduction of about 4.05 percent of the Fuel Consumption and a reduction of about 4.01 percent of the CO2 Emission.

During a run with the optimization routine with the AVL Design Explorer, the continuous design variables TR2 and TR3 converge to 0.883 and 0.556. This convergence can be seen in the pictures in Figure 6.7, where the design variables over the increasing number of design points are shown. These design points are the first 5000 of the 12160 computed points due to the 150 generations with populations of size 80. The design points are numbered with a RunID. In the following pictures the region between design points with RunID 5000 to 12160 would look similar to the region between 3000 and 5000 so it is not shown.



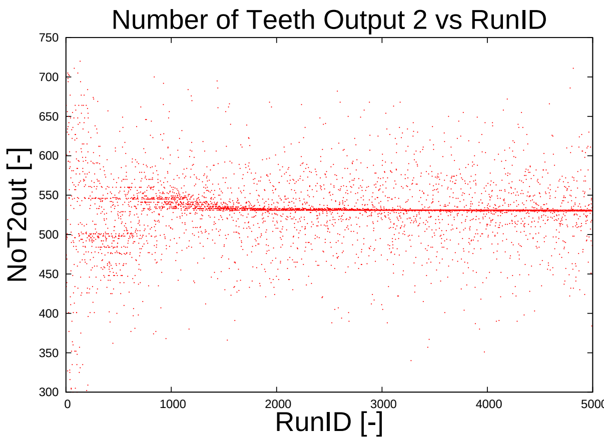
(a) Factor for the gear ratio of gear 2



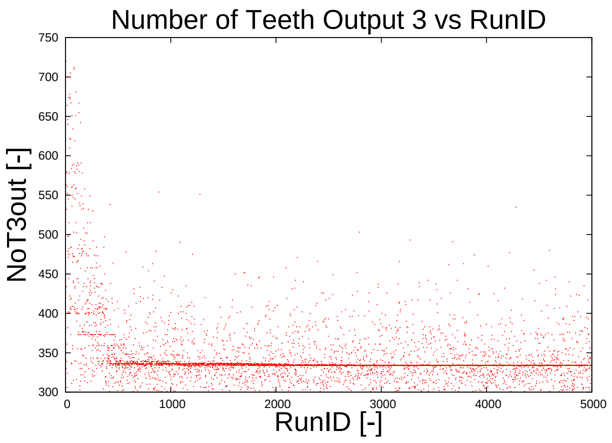
(b) Factor for the gear ratio of gear 3

Figure 6.7: Convergence of the factors for the gear ratios using continuous optimization

Almost the same result figures can be observed for the integer optimization problem. For Figure 6.8 and Figure 6.9 the crossover operator Scattered Crossover is chosen.



(a) Factor for the gear ratio of gear 2

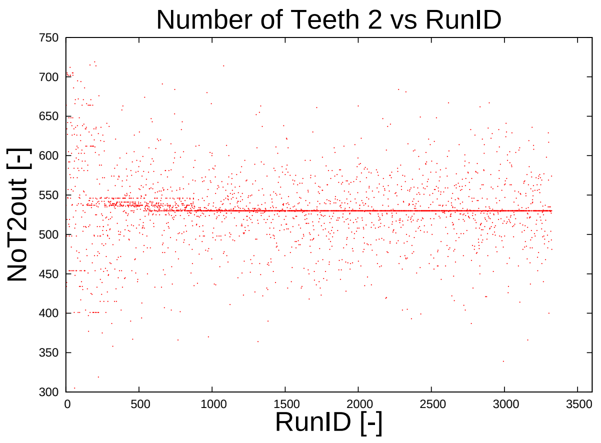


(b) Factor for the gear ratio of gear 3

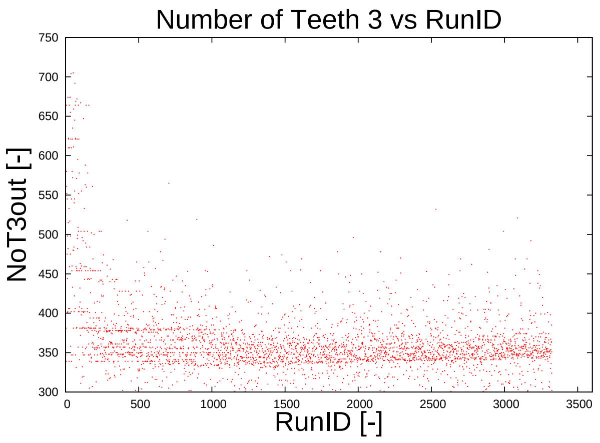
Figure 6.8: Convergence of the factors for the gear ratios using integer optimization

For additional tests a third objective function and constraints are added to the problem. The third objective function is the maximum achievable vehicle velocity. For obtaining the maximum velocity, also the fifth gear is important, which results in the necessity of constraints. These constraints assure as described above, that the transmission ratios for increasing gears decrease. The convergence of the number of teeth output of the factor for gear three for this expanded problem differs to the convergence of the ratio before the expansion of the problem as shown in Figure 6.9b. An additional test run with 8000 data points results in a similar picture. This is a consequence of the third objective function which restrains the number of teeth from converging to a specific value.
However, the resulting numbers of teeth of the best solutions for both gears coincide with the values without the third objective function and the constraints.

Data points with an unsatisfied constraint are deleted in the pictures in Figure 6.9. The best obtainable solution for the three objective functions is computed by 70 generations with a population size of 50 such that 3600 design points are generated. This number decreases due to the deletion of the design points with an unsatisfied constraint for the pictures in Figure 6.9.



(a) Factor for the gear ratio of gear 2



(b) Factor for the gear ratio of gear 3

Figure 6.9: Convergence of the factors for the gear ratios with Maximum Velocity as a third objective function

The optimization of the model including the maximization of the Maximum Obtainable Velocity results in similar objective values as for the case with only two objective functions. The minimum Fuel Consumptions coincide and the CO2 Emission (266.93 g/km instead of 266.91 g/km without the third objective function) is close to the best value. Additionally, the resulting maximum velocity is only 0.71 km/h smaller than without the optimization.

	Fuel Consumption [l/100km]	**CO2 Emission** [g/km]	**Maximum Velocity** [km/h]
Before	11.61	278.32	233.50
After	11.14	266.93	232.79

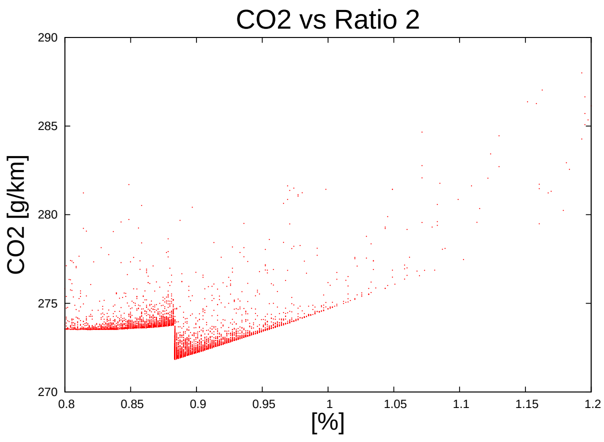
This best obtained solution with three objective functions has the following gear ratios for the gears one to five:

Gear [-]	**Gear Ratio [-]**
1	3.8975
2	1.9610
3	0.8406
4	0.5904

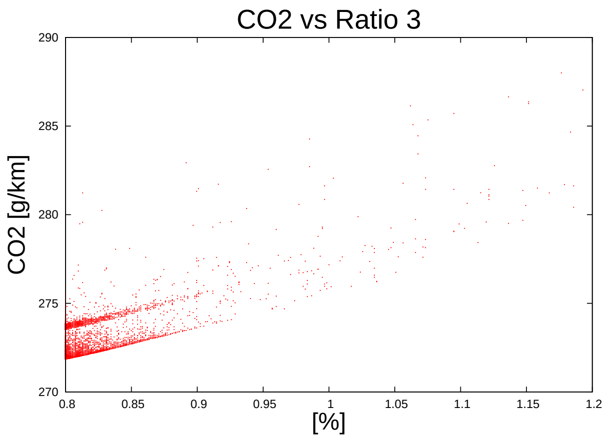
5	0.4350

The numbers of teeth for the gears two and three converge to the same values when using two objective functions (Figure 6.8) and when using three objective functions (Figure 6.9). For gear two the number of teeth converges to 530 (factor for the gear ratio of about 0.883) and for gear three to 334 (factor for the gear ratio of about 0.557). These values are similar to the ones from the continuous results above but due to the integer variables, it is not possible to reach the same optimal solution as in the continuous case.

The results above are all computed with the lower bound of 50% of the basis values for the transmission ratios of the different gears. This is done because if a lower bound of 80% were chosen, the optimal value for the factor for the gear ratio of gear three would be its lower bound.
For this reason, the optimal value is easy to find, because each time, when the value of the design variable is less than the lower bound, it is set to the lower bound and the optimal value is found. For a better comparison with the figures above, the transmission ratios and not the numbers of teeth are shown in the following figures, although the results are generated by integer optimization.



(a) Factor for the gear ratio of gear 2

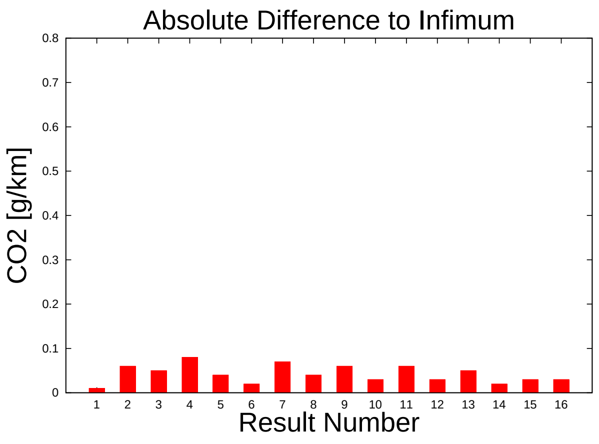


(b) Factor for the gear ratio of gear 3

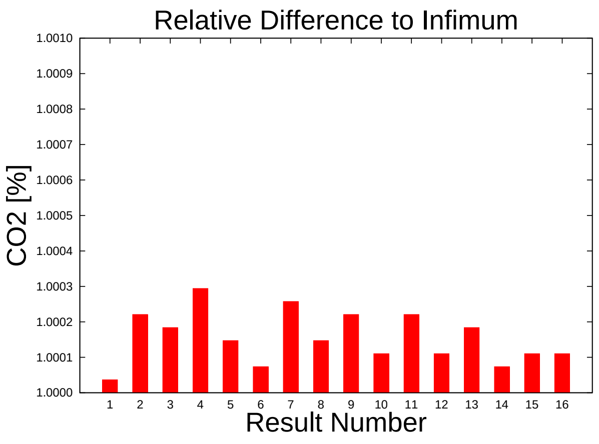
Figure 6.10: Lower bounds for the design variables:
80% of the basis values

Again, each point in Figure 6.10 is an objective function value of a design point. To illustrate the problem of a lower bound of 80% only the CO2 Emission is chosen because the pictures showing the Fuel Consumption look similar.

The following pictures in Figure 6.11 show the results of a number of optimizations which are numbered with Result Numbers. These are computed with lower bounds of 80% of the base values. The differences between the best result for the CO2 Emission using continuous optimization and 16 integer optimization runs are presented in Figure 6.11. The results 2 to 16 are generated using 20 generations of populations of size 10 (220 design points), result 1 is computed with 150 generations and a population size of 80 (12160 design points). The range of the y-axis is chosen for a better comparability of Figure 6.11, Figure 6.12, and Figure 6.14.



(a) Absolute difference to the best continuous solution



(b) Relative difference to the best continuous solution

Figure 6.11: Comparison Integer and Continuous with lower bound of 80% of the base values

The absolute and the relative difference of the CO2 Emission between the best solution computed with continuous optimization and 16 results using integer optimization are shown in Figure 6.11. Due to the fact, that good results can be found easily, the differences are less than 0.1 g/km CO2 Emission, which is only a difference of at most 0.4%.

The gear ratios for the relevant gears one, two, and three are a result of one of the optimal solutions computed by 12160 design points:

The following table shows the resulting gear ratios for result number 1 in Figure 6.11 above with lower bounds of 80%. The used crossover operator is Scattered Crossover.

Gear [-]	Gear Ratio [-]
1	3.9135
2	1.9617
3	1.2080

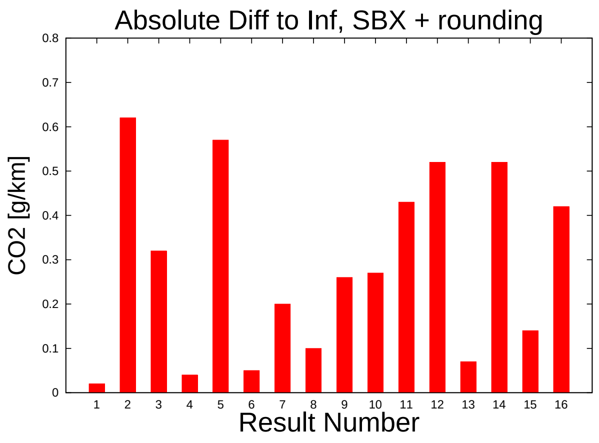
The following table shows resulting transmission ratios for result number 1 in Figure 6.12a below with lower bounds of 50%. The used crossover operator is Simulated Binary Crossover.

Gear [-]	Gear Ratio [-]
1	3.9096
2	1.9610
3	0.8406

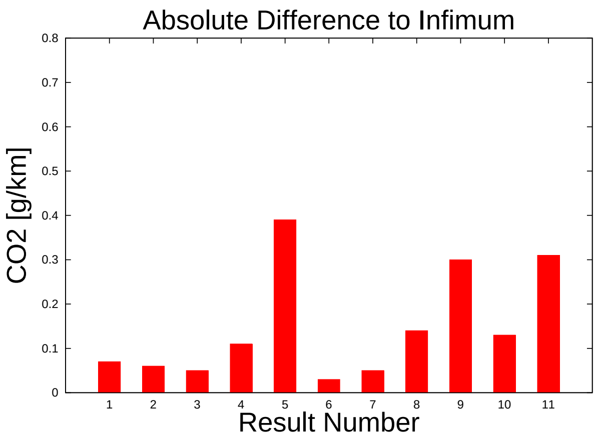
The following table shows the original gear ratios which are used in the standard model Man_FWD (refer to Section 6.1.1):

Gear [-]	Gear Ratio [-]
1	3.6200
2	2.2200
3	1.5100

In addition to the comparison between integer and continuous optimization, two different crossover operators for the integer optimization are also compared. The two crossover operators are the Simulated Binary Crossover (SBX) for continuous optimization using a rounding function and the Scattered Crossover. These operators are chosen from the ones shown in Section 5.2 due to initial test runs.



(a) Crossover: integer: SBX + rounding, continuous: SBX



(b) Crossover: integer: Scattered Crossover, continuous: SBX

Figure 6.12: Comparison Integer and Continuous with lower bound of 50% of the base values

For the pictures in Figure 6.12 the differences between the best obtained CO2 Emission of the continuous optimization problem and some solutions of the integer optimization problem are computed. Figure 6.12a shows 16 results using Simulated Binary Crossover and rounding for the integer optimization problem. Figure 6.12b shows 11 results computed using Scattered Crossover for the integer optimization problem.

6.3 Test Problem 2: Optimization of the Engine and Final Drive

The optimization of the engine and final drive is also done using a modified version of the standard model Man_FWD, which is introduced in Section 6.1.1.
Besides single variables like the gear ratios in the last section, whole components can also be used as values for an integer design variable. In the second test problem four engines with different engine displacements are defined.
The New European Driving Cycle NEDC is used and the applied simulation mode is Quasi-stationary, for more information about NEDC and Quasi-stationary refer to Section 4.2. Besides the three additional engines in the background of the schematic representation of the model in Figure 6.13, there are no changes to the standard model Man_FWD.

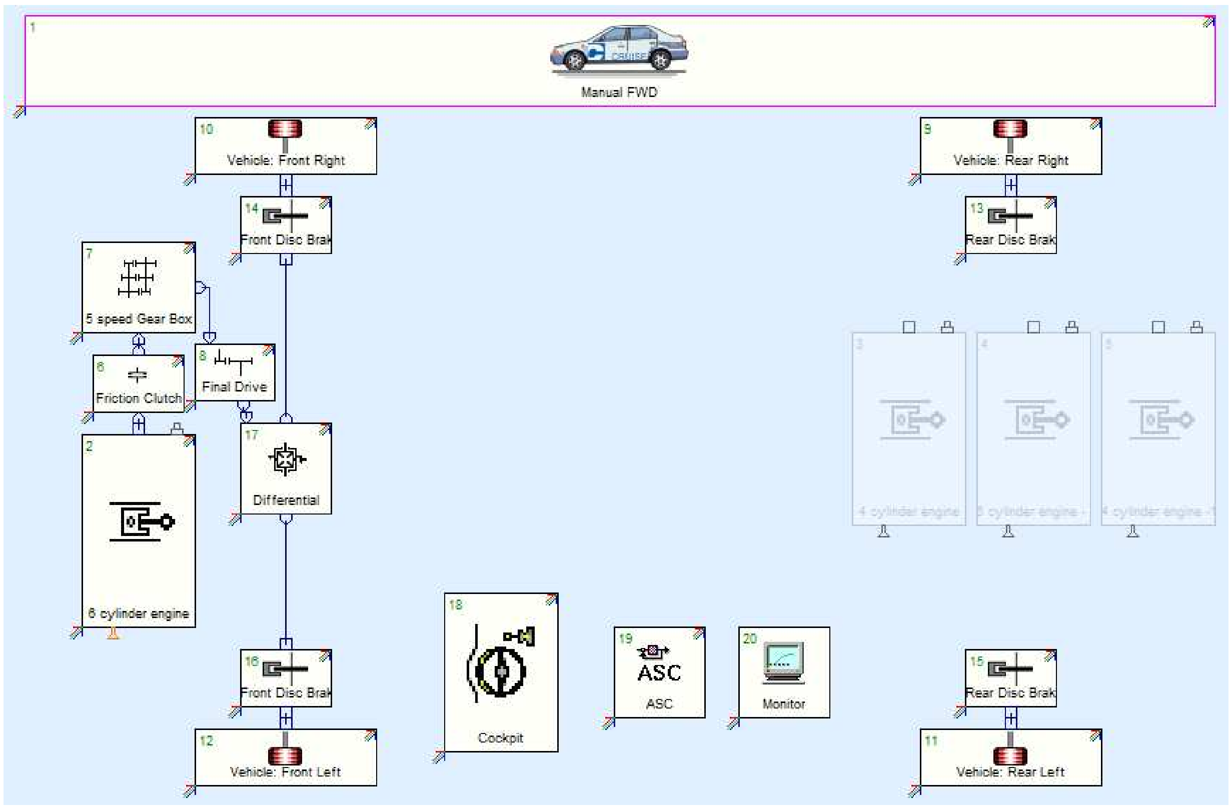


Figure 6.13: Schematic representation of the AVL CRUISE Model for component variation

For a detailed description of the modules in Figure 6.13 refer to Section 6.1.1.

6.3.1 Optimization Instance

The first design variable in this optimization instance can choose one out of four engines with different engine displacements:

1. 1478 cm^3
2. 3478 cm^3
3. 1878 cm^3 Turbo
4. 2478 cm^3

This means, that the variable has four possible values and according to this value the corresponding engine is activated and the objective functions are computed using the simulation software AVL CRUISE.

Additionally, the transmission ratio in the module Final Drive is chosen as a continuous design variable. For this design variable the lower bound 1.5 and the upper bound 4.5 are used. Due to this continuous design variable, a mixed integer optimization problem has to be solved.

6.3.2 Results

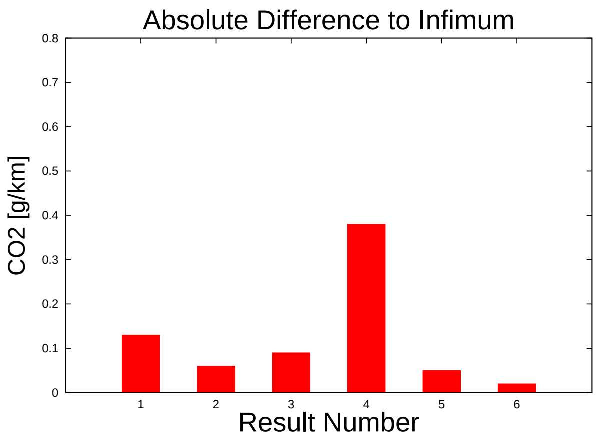
Figure 6.14 shows the differences of the resulting CO2 Emissions of the optimization instance, which is presented above, and the best solution obtained by an optimization using 150 generations with populations of size 80, which results in 12160 design points.
The best obtained value for the design variable transmission ratio in this optimization instance is about 1.669. Since the smallest engine displacement results in general in the lowest Fuel Consumption and CO2 Emission the first engine with 1478 cm^3 is selected by the AVL Design Explorer in all optimization runs. The resulting best objective values for Fuel Consumption and CO2 Emission are the following:

Fuel Consumption [l/100km]	**CO2 Emission** [g/km]
4.92	118.01

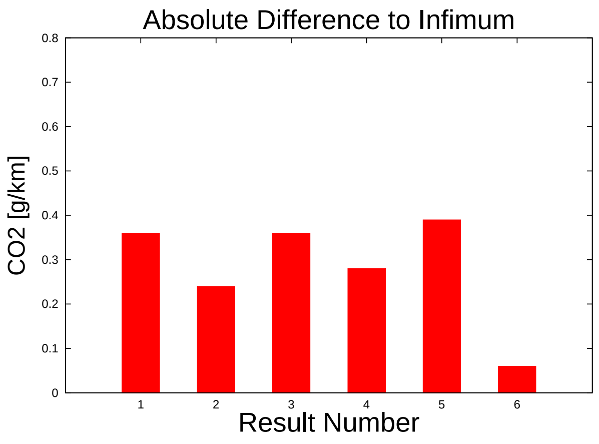
Since the complete engine is changed, a comparison with the Fuel Consumption and the CO2 Emission before starting the optimization is not meaningful.

The results in Figure 6.14a are computed by using the crossover operator Simulated Binary Crossover for integer and continuous design variables. For the results in Figure 6.14b the

SBX is used for the continuous and the Scattered Crossover is used for the integer design variables.



(a) Crossover: integer: SBX + rounding, continuous: SBX



(b) Crossover: integer: Scattered Crossover, continuous: SBX

Figure 6.14: Comparison Integer and Continuous with lower bound of 50% of the base values

In the comparison of these results, the Simulated Binary Crossover is better than the operator Scattered Crossover. For results of the optimization of the gear ratios in Section 6.2 the Scattered Crossover is better as one can see in Figure 6.12.

Additionally, the optimization of the engine and the final drive is done as a pure integer optimization problem, which would be the better way to solve this optimization problem due to the reasons mentioned in the sections before. For the purpose of integer optimization the number of teeth output of the module Final Drive is chosen as the second design variable with a base value of 300, a lower bound of 150, and an upper bound of 450, which is equivalent to the base value and the bounds in the continuous variant. The following objective function values result for the pure integer problem using Scattered Crossover as crossover operator for three test runs using 20 generations with populations of size 10 (220 design points):

Fuel Consumption [l/100km]	**CO2 Emission** [g/km]
4.93	118.03

This table shows small differences to the best obtained solution, which is shown above. These small differences of 0.01 l/100 km Fuel Consumption and 0.02 g/km CO2 Emission results from less possible values, which are possible for the number of teeth, so the best value can be found easier than in the continuous optimization. Considering the results of the pure integer optimization problem, Scattered Crossover for the integer design variables is a good choice.

6.4 Test Problem 3: Optimization of PID Control and Final Drive

The third test problem concerns the optimization of the parameters of the modules PID Control and Final Drive. The underlying model is the Range Extender model considered in Section 6.1.2.
The design variables of the optimization problems considered in Sections 6.2 and 6.3 are mainly the transmission ratios which are mechanical parameters. Changing the values of these parameters results in new parts which have to be manufactured. This means, that also costs and availability of these parts have to be taken into account. The advantage of using control parameters like the parameters of the module PID Control as design variables is, that this sort of problems does not arise.

6.4.1 Optimization Problem

The design variables of the third test problem are the continuous variables Proportional and Derivative Parameter from the module PID Control and the integer variable Number of Teeth Output from the module Final Drive:

Variable Name	Lower Bound	Upper Bound	Unit
Proportional Parameter	10.0	30.0	[-]
Derivative Parameter	0.00125	0.00375	[s]
Number of Teeth Output	50	92	[-]

Since the Fuel Consumption and the Emission would decrease significantly if the initial battery charge were much higher than the battery charge after the computation, a constraint is added to the problem. This constraint is only satisfied, when the difference of the battery charge after the computation and the initial charge is at least -0.01, which implies that the state of charge does not decrease too much.

$$\text{Delta\_SOC} \geq -0.01$$

Due to the possibility to define the final drive with the transmission ratio, this optimization problem could also be solved as a continuous optimization problem. The same arguments as for the case of the optimization of the gear ratios in Section 6.2 show, that choosing the number of teeth output as a variable has to be preferred over the continuous approach.

In contrast to the previous two test problems, the calculation mode Simulation instead of Quasi-stationary and the New European Driving Cycle (NEDC) instead of the Urban Driving Cycle are used. The NEDC consists of four consecutive UDCs and an additional part with higher velocity. For more information about the NEDC and the calculation mode Simulation refer to Section 4.2.

6.4.2 Results

The following table shows the reduction of the Fuel Consumption and the CO2 Emission in the best obtained solution compared to the Fuel Consumption and the CO2 Emission using the design variable values of the model Range Extender.

	Fuel Consumption [l/100km]	**CO2 Emission** [g/km]
Before	3.81	91.26
After	2.96	70.82

This is a reduction of about 22.3 percent of Fuel Consumption and of about 22.4 percent of CO2 Emission.

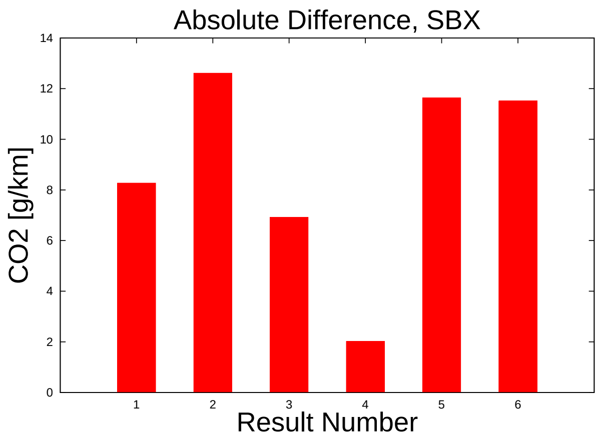
The following table shows the values of the design variable values, which are used in the standard model Range Extender:

Variable Name	**Value**	**Unit**
Proportional Parameter	20.0	[-]
Derivative Parameter	0.0025	[s]
Number of Teeth Output	71	[-]

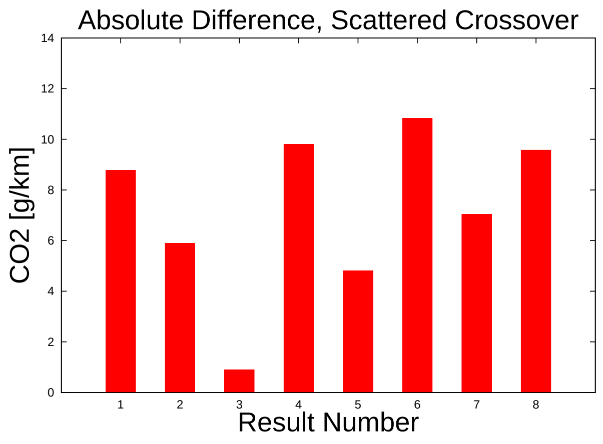
The values suggested by the results of the optimization approach are shown in the table below:

Variable Name	**Value**	**Unit**
Proportional Parameter	12.435179	[-]
Derivative Parameter	0.002742	[s]
Number of Teeth Output	68	[-]

In Figure 6.15 the differences of the objective values of test runs using 10 generations with populations of size 10 (120 design points) to the best obtained objective values are shown. Using these pictures, the two crossover operators Simulated Binary Crossover with rounding and Scattered Crossover for the integer design variables are compared. These two operators are chosen according to initial test runs. The crossover operator for the continuous design variables is always chosen to be the Simulated Binary Crossover.



(a) Crossover: integer: SBX + rounding, continuous: SBX



(b) Crossover: integer: Scattered Crossover, continuous: SBX

Figure 6.15: Absolute differences to the best obtained solution

The results of Simulated Binary Crossover and Scattered Crossover for the integer design variables are similar to each other. The Scattered Crossover operator performs slightly better since there is a smaller maximum difference to the best obtained solution and also a smaller minimum difference.

7 Conclusion

The need for optimization algorithms and efficient implementations will further increase in the automotive industry in the upcoming years. One of the reasons for this is the increasing demand to reduce CO2 emission. While most of the design variables in optimization problems arising in the automotive industry are continuous variables, in some cases also the need for integer variables occurs (for example the number of teeth in Section 6.2).
The goal of this thesis has been to extend the genetic algorithm for multicriteria optimization within the AVL Design Explorer to the integer and mixed-integer case. For this reason, mutation and crossover operators for handling integer design variables were added to the implemented genetic algorithm Non-dominated Sorting Genetic Algorithm II. These modifications were tested using test instances based on models of vehicles, which are created by AVL CRUISE.

It can be seen from the computational experiences that the version of the Non-dominated Sorting Genetic Algorithm II which uses the Scattered Crossover for the integer design variables behaves better than the version which uses the Simulated Binary Crossover with a rounding function. The Simulated Binary Crossover remains for the continuous variables, since test runs with alternative crossover operators did not result in better objective values. Additionally, for one test model (gear ratios, Section 6.2) a comparison between the results of the integer model and a pure continuous model has been undertaken. This comparison showed similar result quality for both variants of the optimization problem. Nevertheless, the integer variant of the problem is the preferred one since the optimal solution for the continuous optimization problem could result in gear ratios with an infeasible or costly number of teeth.

8 References

[1] AVL Design Explorer v2011. *User Guide.* AVL List GmbH, Graz, Austria, July 2011. Distributed as pdf file with AVL Design Explorer.

[2] M. M. Bai and N. Gan. Improvement and analysis of particle swarm optimization. *Journal of Science Mosaic*, 7(4):23–26, 2008.

[3] H.-G. Beyer and K. Deb. On the desired behaviors of self-adaptive evolutionary algorithms. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 59–68. Springer, 2000.

[4] G. Bilchev and I. C. Parmee. The ant colony metaphor for searching continuous design spaces. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 25–39. Springer, 1995.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 1st edition, 2004.

[6] AVL CRUISE v2011. *User Guide.* AVL List GmbH, Graz, Austria, June 2011. Distributed as pdf file with AVL CRUISE.

[7] K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, 1st edition, 2001.

[8] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. Technical report, Indian Institute of Technology, 1994.

[9] K. Deb and S. Agrawal. A niched-penalty approach for constraint handling in genetic algorithms. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 235–243. Springer, 1999.

[10] K. Deb and H.-G. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation*, 9:197–221, 2001.

[11] K. Deb and T. Goel. Controlled elitist non-dominated sorting genetic algorithms for better convergence. In *Evolutionary Multi-Criterion Optimization*, pages 67–81. Springer, 2001.

[12] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50. Morgan Kaufmann Publishers Inc., 1989.

[13] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, volume 1917, pages 849–858. Springer, 2000.

[14] K. Deb, R. Steuer, R. Tewari, and R. Tewari. Bi-objective portfolio optimization using a customized hybrid NSGA-II procedure. In *Evolutionary Multi-Criterion Optimization*, volume 6576 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2011.

[15] M. Dorigo, M. Birattari, and T. Stützle. Ant colony optimization artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*, 1 (4):28–39, 2006.

[16] L. dos Santos Coelho. An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications. *Reliability Engineering & System Safety*, 94(4):830–837, 2009.

[17] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization.* Springer, 1st edition, 2006.

[18] M. Ehrgott. *Multicriteria Optimization.* Springer, 2nd edition, 2005.

[19] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: A review. *Engineering Letters*, 13(2):124–137, 2006.

[20] H. A. Eschenauer. Multicriteria optimization for highly accurate systems. *Multicriteria Optimization in Engineering and Sciences*, 19:309–352, 1988.

[21] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms*, pages 187–202. Morgan Kaufmann Publishers Inc., 1992.

[22] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann Publishers Inc., 1993.

[23] D. P. Giesy. Calculation of pareto-optimal solutions to multiple-objective problems using threshold-of-acceptability constraints. *IEEE Transactions on Automatic Control*, 23:1114–1115, 1978.

[24] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[25] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Longman Publishing Company, Inc., 1st edition, 1989.

[26] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann Publishers Inc., 1991.

[27] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 41–49. Lawrence Erlbaum Associates Inc., 1987.

[28] T. F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics.* Chapman & Hall/CRC Computer & Information Science Series. Chapman & Hall/CRC, 1st edition, 2007.

[29] A. Göpfert and R. Nehse. *Vektoroptimierung. Theorie, Verfahren und Anwendungen.* Teubner, 1st edition, 1990.

[30] Y. Y. Haimes, L. S. Lasdon, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(3):296–297, 1971.

[31] R. Hartley. On cone-efficiency, cone-convexity and cone-compactness. In *SIAM Journal on Applied Mathematics*, volume 34, pages 211–222. Society for Industrial and Applied Mathematics, 2001.

[32] R. Hassan, B. Cohanim, O. De Weck, and G. Venter. A comparison of particle swarm optimization and the genetic algorithm. In *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, pages 1–13. AIAA, 2005.

[33] A. S. Hedayat, N. J. A. Sloane, and J. Stufken. *Orthogonal Arrays: Theory and Applications.* Springer, 1st edition, 1999.

[34] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

[35] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches.* Springer, 3rd edition, 2003.

[36] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[37] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[38] N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. In *International Genetic and Evolutionary Computation Conference (GECCO2000)*, pages 987–994. Morgan Kaufmann Publishers Inc., 2000.

[39] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[40] J. Lu, L. Zhang, and H. Yang. Combining strategy of genetic algorithm and particle swarm algorithm for reactive power optimization. In *Proceedings of the 2010 International Conference on Electrical and Control Engineering*, pages 3613–3616. IEEE, 2010.

[41] MATLAB. The MathWorks Inc., Natick, Massachusetts, USA. `www.MATLAB.com`.

[42] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 3rd edition, 1996.

[43] R. D. Moen, T. W. Nolan, and L. P. Provost. *Quality Improvement Through Planned Experimentation.* McGraw-Hill, 2nd edition, 1998.

[44] MOMHLib++. A. Jaszkiewicz and R. Ziembinski, Boston, Massachusetts, USA. `home.gna.org/momh/`.

[45] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[46] T. Nomura and T. Miyoshi. Numerical coding and unfair average crossover in GA for fuzzy rule extraction in dynamic environments. In *Selected Papers from the EEE/Nagoya-University World Wisepersons Workshop on Fuzzy Logic, Neural Networks, and Evolutionary Computation*, pages 55–72. Springer, 1996.

[47] P. M. Pardalos and H. E. Romeijn, editors. *Handbook of Global Optimization Volume 2.* Kluwer Academic Publishers, 1st edition, 2002.

[48] V. Pareto. *Manuale di Economia Politica.* Societa Editrice, 1st edition, 1906.

[49] R. U. Pfau. *Numerical Algorithms for the Calculation and Optimization of Vehicle Driving Performance and Fuel Consumption.* PhD thesis, Johannes Kepler University, Linz, Austria, 2002.

[50] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 3rd edition, 2007.

[51] S. S. Rao. *Engineering Optimization: Theory and Practice.* John Wiley & Sons, 3rd edition, 1996.

[52] R. Rosenberg. *Simulation of Genetic Populations with Biochemical Properties.* PhD thesis, University of Michigan, Ann Arbor, USA, 1967.

[53] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd edition, 2010.

[54] J. D. Schaffer. *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, Nashville, USA, 1984.

[55] R. Scheucher. *Analysis and Application of Mathematical Methods to Efficiently Optimize the Vibro-Acoustic Behavior of Engine Components*. PhD thesis, Technical University Graz, Graz, Austria, 2005.

[56] K. Schittkowski. NLPQL: A Fortran subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1986.

[57] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1945–1950. IEEE, 1999.

[58] P. K. Shukla and K. Deb. On finding multiple pareto-optimal solutions using classical and evolutionary generating methods. *European Journal of Operational Research*, 181 (3):1630 – 1652, 2007.

[59] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2:221–248, 1994.

[60] M. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[61] European Union. Council Directive 70/220/EEC of 20 March 1970 on the approximation of the laws of the Member States relating to measures to be taken against air pollution by gases from positive-ignition engines of motor vehicles . *Official Journal of the European Union L 076*, 1970.

[62] European Union. REGULATION (EC) No 715/2007 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 20 June 2007 on type approval of motor vehicles with respect to emissions from light passenger and commercial vehicles (Euro 5 and Euro 6) and on access to vehicle repair and maintenance information. *Official Journal of the European Union L 171*, 2007.

[63] Y. Yun. Hybrid genetic algorithm with adaptive local search scheme. *Computers and Industrial Engineering*, 51:128–141, 2006.

[64] G. Zhang, M. Dou, and S. Wang. Hybrid genetic algorithm with particle swarm optimization technique. In *International Conference on Computational Intelligence and Security*, volume 1, pages 103–106. IEEE, 2009.

[65] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8:173–195, 2000.