FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Metaheuristic optimization of electro-hybrid powertrains using machine learning techniques

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Christopher Bacher
Matrikelnummer 0728088

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr. Günther Raidl
Mitwirkung: Dipl.-Ing. Thorsten Krenek, Bakk.techn.

Wien, 24.08.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Metaheuristic optimization of electro-hybrid powertrains using machine learning techniques

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Christopher Bacher
Registration Number 0728088

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr. Günther Raidl
Assistance: Dipl.-Ing. Thorsten Krenek, Bakk.techn.

Vienna, 24.08.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Christopher Bacher
Feldgasse 2, 2425 Nickelsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

Ich möchte mich bei allen Menschen bedanken, die mich im Laufe meines Studiums und bei dieser Arbeit unterstützt haben.

Spezieller Dank gilt meinen Betreuern Prof. Dr. Günther Raidl und Dipl.-Ing. Thorsten Krenek, ohne die es diese Arbeit nicht geben würde und für die Hilfe die ich erhalten habe um mich in das — für mich völlig neue — Gebiet der Hybridfahrzeugtechnik einzuarbeiten. Auch bei den Mitarbeitern des Instituts für Fahrzeugantriebe & Automobiltechnik, mit denen ich den vergangenen Monaten viel Zeit verbringen durfte, möchte ich mich für die freundliche Aufnahme und das gute Arbeitsklima bedanken. Ebenso bedanke ich mich bei den Mitarbeitern des Arbeitsbereiches für Algorithmen und Datenstrukturen mit denen ich in meiner Zeit als Tutor zusammenarbeiten durfte und von denen ich viel für mein Studium mitnehmen konnte.

Meinen Eltern und meiner Familie kann ich vermutlich nicht genug danken, dafür dass sie mich immer unterstützt und mir bei meinen Entscheidungen geholfen haben — und auch dafür dass sie meinen steigenden Stresslevel in den letzten Wochen ertragen haben. Dank gilt auch meinen Freunden für die gegenseitige Hilfe im Studium und für die vielen unterhaltsamen Stunden.

Abstract

Hybrid Electric Vehicles (HEVs) have risen in popularity in the last years. Therefore their fuel efficiency becomes increasingly important to manufacturers to satisfy consumer expectations and legal restrictions. The efficiency of a HEV depends strongly on the used operation strategy. As operation strategies are typically complex their optimal parameters cannot easily be determined.

In this thesis the operation strategy parameters of two HEV simulation models — Model A and Model B — are optimized using metaheuristics. A main problem in HEV optimization are the long simulation times of the simulation software. This has already been observed in [30]. A main goal of this thesis is to decrease the number of simulations needed in a metaheuristic while still getting good approximate solutions, and to this end different techniques are used.

First search space restrictions are imposed where applicable. Second a discretization of the continuous search space is employed to enable the usage of a solution cache. Third different regression models — based on neural networks and ensemble methods — are applied to replace the simulation software as a part of the used fitness functions. Different regression models are trained and compared in experiments. These methods are implemented into a new modular optimization platform developed by the author.

Metaheuristic optimization algorithms like Particle Swarm Optimization (PSO), Active Covariance Matrix Adaption Evolution Strategy (Active CMA-ES), and a genetic algorithm (GA) are adapted to incorporate trained regression models. For this purpose a two-phased optimization scheme is proposed. The first phase is concerned with generating solutions which are subsequently used in the training of regression models. The second phase uses the regression models in tandem with the simulation software. Different approaches to incorporating regression models are explored, like switching fitness functions or filtering of bad solutions.

The results of the two-phased optimization experiments are compared to reference experiments of an unmodified PSO and an Active CMA-ES. The two-phased optimization is able find solutions of approximately the same quality as the reference experiments. The fitness values of best solutions found in the reference experiments and the two-phased experiments differ only in 0.6% for Model A and 0.3% for Model B — a difference negligible in practice. As no reference values exist for the models it is hard to estimate the overall efficiency of the found solutions. Although the parameter settings for the models are in the expected range where one would expect good solutions. The required (overall) simulation time could be reduced by up to 27% if compared to the reference experiments. Last an analysis of the model parameters for Model A is given with the help of Self-Organizing Maps (SOMs).

Kurzfassung

Elektrohybride Kraftfahrzeuge (HEVs) gewannen in den letzten Jahren zunehmend an Bedeutung. Für die Hersteller wird es dadurch wichtiger deren Kraftstoffverbrauch zu senken um Kundenerwartungen als auch rechtliche Rahmenbedingungen zu erfüllen. Dabei ist der Kraftstoffverbrauch von Hybridfahrzeugen stark von der verwendeten Betriebsstrategie abhängig. Diese ist meist komplexer Natur, sodass deren optimalen Parametereinstellungen nicht einfach bestimmt werden können.

In dieser Diplomarbeit werden die Parameter von zwei Hybridfahrzeugmodellen — Modell A und Modell B — mit Hilfe von Metaheuristiken optimiert. Ein Hauptproblem dabei sind die langen Simulationszeiten der eingesetzten Simulationssoftware. Dies wurde bereits in [30] erkannt. Daher liegt das Hauptaugenmerk auf der Reduzierung der benötigten Simulationsdurchläufe. Zu diesem Zweck werden verschiedene Methoden eingesetzt.

Verschiedene Suchraumeinschränkungen werden angewandt wo dies möglich ist. Weiters wird der Suchraum diskretisiert um den Optimierungsmethoden die Verwendung eines Lösungs-Caches zu ermöglichen. Zuletzt werden verschiedene, auf neuronalen Netzwerken und Ensemble-Methoden basierende, Regressionsmodelle trainiert um die Simulation aus der verwendeten Fitnessfunktion zu entfernen. Die oben genannten Erweiterungen werden in einem zuvor vom Autor entwickelten Optimierungsframework implementiert.

Unterschiedliche Metaheuristiken wie Particle Swarm Optimization (PSO), Active Covariance Matrix Adaption Evolution Strategy (Active CMA-ES) und genetische Algorithmen werden für die Verwendung von Regressionsmodelle adaptiert. Zu diesem Zweck wird eine zweiphasige Optimierungsstrategie entwickelt. In der ersten Phase werden Daten für das Training der Regressionsmodelle gesammelt. In der zweiten Phase werden die Regressionsmodelle zur Approximation von Fitnesswerten verwendet. Verschiedene Möglichkeiten zu deren Verwendung, wie abwechselnde Verwendung von Regression und Simulation, als auch filtern mittels Regression, werden analysiert.

Abschließend werden die Ergebnisse der zwei-phasigen Optimierung mit Referenzlösungen von unmodifizierten Optimerungsalgorithmen verglichen. Dabei findet die zwei-phasige Optimierung ähnlich gute Lösungen wie die Referenzmethoden. Die Zielfunktionswerte der besten gefundenen Lösung aller Referenzoptimierungen und aller zwei-phasigen Optimierungen liegen zwischen 0.6% (für Modell A) und 0.3% (für Modell B) — eine in der Praxis vernachlässigbare Differenz. Die gefundenen Modellparametereinstellungen korrespondieren mit den Gebieten in denen gute Lösungen erwartet werden. Die erforderliche Simulationszeit konnte mit der zweiphasigen Optimierung um bis zu 27% verringert werden. Die evaluierten Modellparametereinstellung werden für Modell A mit Hilfe von Self-Organizing Maps (SOMs) analysiert.

Contents

CHAPTER 1

Introduction

1.1 Background & Motivation

For the longest time in automotive history a large part of developed vehicles has been powered by internal combustion engines. Although the design of internal combustion engines changed and improved over the last hundred years the main concept of combusting fossil energy storages stayed the same. During that time significant experience has been gained in the field of combustion engine construction.

It is undeniable that the advances in the field of internal combustion engines led to more fuel efficient vehicles and also that the CO_2 emissions per vehicle have reduced, as for example the 2013 report [10] of the US Environmental Protection Agency shows. Nevertheless, the climate change driving atmospheric CO_2 concentration reaches higher levels each year. Consequently any possible reduction of CO_2 emission has to be pursued vigorously.

Therefore, in recent years, the automotive industry faces increasing pressure from nations and potential customers to develop more efficient drive concepts. For example the European Union passed legislation [12] in 2009 prescribing CO_2 emission goals for the current decade. The regulation requires that all newly registered (light-duty) vehicles in the European Union emit less than $130\,\mathrm{g\,km^{-1}}$ by 2015 and further less than $95\,\mathrm{g\,km^{-1}}$ by 2020. Failure to do so will result in penalties for the manufacturers.

Besides passing such large scale measures more often local communities pass their own regulations concerning automotive vehicles. Examples for this can be found throughout the European Union where so called low emission zones[1] have been created — local areas where only vehicles complying to an emission standard may enter (or enter at a lower fee). These measures usually target more direct goals like reducing the emission of particulate matter — environmental pollutants which have been linked to respiratory diseases like asthma.

Currently, several more innovative (and less direct) concepts for promoting fuel efficient and lower emission technologies are set up throughout Europe. An excellent example for such a

[1] see `www.lowemissionzones.eu`, retrieved 2013-03-15

concept is the city of Madrid where electric vehicles and plug-in hybrid electric vehicles are exempt of parking regulations[2]. Further the city provides several free charging stations. A similar charging concept on a much larger scale is planned for London 2013, where about 1300 charging stations shall be built and made available for a low annual fee[3].

Due to the regulations and promoting concepts mentioned above, the incentive for the automotive industry to invest in alternative drive concepts has risen over the last years. As a typical goal of the intended developments "pure" electric long range vehicles are conceived (also called Zero Emission Vehicles (ZEVs) [21]), which by definition do not emit pollutants during operation[4]. According to [21] both fuel cell powered electric vehicles and battery powered vehicles are seen as potential candidates for ZEVs for everyday use.

Although these technologies have been around for some time, there are still several drawbacks associated with them. As drawbacks for fuel cells [21] lists cost-intensive production of Hydrogen and of the required fuel cells, as well as high volumetric requirements and high mass of the cell concept.

The specific short-comings of electric batteries differ between the battery types in use (e.g Li-Ion, Pb/PbO_2, Ni-Metal-hybrid, etc.), but some common problems can be found [21]. Typically the theoretical and — even more so — practical energy density is far lower than the energy densities of gas, diesel or H_2. Further batteries require additional control circuits and software for managing their state of charge, as failing to do so would limit their lifespan further. To make things worse their usual mode of operation is restricted to a rather small temperature range s.t. the batteries may require additional heating and/or cooling. Their practical use is also limited by their high recharge times although significant improvements have been made in recent years.

It can be seen that both energy technologies do not come close to their gas/diesel counterpart in matters of everyday use. It is believed that further technological advances are able to establish the competitiveness of fuel cell and battery technologies. In the mean time some bridge technologies can be employed to counter the environmental and economical problems (at least partially) caused by the use of traditional internal combustion engines.

This is where hybrid drive concepts come into play. Hybrid drive concepts combine the benefits of internal combustion engines with an alternative drive concept such as fuel cells or electric batteries to limit the drawbacks of both technologies. For achieving this goal Hybrid Electric Vehicles (HEVs) use both internal combustion engines and electric machines in its powertrain. There are several ways how internal combustion engines and electric machines can be combined, which is detailed in Chapter 2. In this diploma thesis only HEVs using electric batteries as second energy storage are considered.

As mentioned in the beginning, today's engineers may rely on years of experience if the design and improvement of internal combustion engines is concerned. In the case of HEVs,

[2] see
`www.c40cities.org/media/case_studies/switching-to-an-alternative-clean-transportation-the-promotion-of-electric-vehicles-in-madrid`, retrieved 2013-03-15

[3] see
`www.london.gov.uk/priorities/transport/green-transport/electric-vehicles`, retrieved 2013-03-15

[4] Emissions emitted during energy transformation e.g. production of H_2 in the case of fuel cells or power generation in a coal power plant are not taken into account.

however, this does not hold as — although the one of the first vehicles combining combustion engines with an electric drive has been built in 1899 [21] — HEVs have become a viable concept only recently due to advances in energy storage technologies. This lack of experience poses a severe challenge in the construction of HEVs which have significantly more parameters in its powertrain to be adjusted than a conventional vehicle[5].

For example powertrain parameters for conventional vehicles are the ignition timing of the engine or the transmission of the gear set. HEVs, on the other hand, possess further parameters for the powertrain which have a much stronger influence on the overall behaviour of the vehicle. Particularly, HEVs use different modes of operation like pure electric, hybrid, Internal Combustion Engine (ICE)-only, recuperating or boosting having a large impact on the performance of the vehicle. The current mode of operation is determined by different parameters like the current speed, axle torque and the current charge state of the battery. Further HEVs often use a different type of gear set, a so called planetary gear set (see Chapter 2) which allows the mechanical coupling of an ICE with an electric machine. The parameters of this coupling greatly influence the performance of the Hybrid Electric Vehicle (HEV).

With this explosion of the number of parameters controlling the behaviour of the powertrain, it gets infeasible to test and optimize a HEV's powertrain manually. Further as the testing of a vehicle in a test bed setup is usually cost- and work-intensive, alternatives have been sought. Therefore complex simulation software has been developed to simulate HEVs in a virtual setting and to evaluate output values such as fuel consumption, CO_2 emissions, temperatures, torques or battery charging state post simulation. The simulation software considered in this paper is GT-SUITE[6].

This thesis targets the optimization of different parameters of a hybrid powertrain on different driving cycles (see Chapter 2) using the above mentioned simulation software. The engineers are able to model many aspects of the actual vehicle with different accuracy in the simulation software i.e. using an engine map instead of simulating the complete engine, depending on the task for which the model is needed. With a tuned model the software then allows to evaluate different parameter settings in less time than by using the actual vehicle.

As the simulation software is proprietary no direct integration (with the software) can be achieved to facilitate the optimization. Therefore the software is treated as black box system and the methods for optimization employed in this thesis, are metaheuristic optimization techniques (see Chapter 3) such as genetic algorithms, particle swarm optimization or evolution strategies. Although the use of metaheuristics can — theoretically — not guarantee a successful optimization, many practical applications on problems like antenna design or rotor blade design have shown promising results.

The thesis builds upon the work of a previous thesis [31] and paper [30] which has shown the feasibility of the task at hand and achieved first good results. Nevertheless [31] showed that even with the available simulation software GT-SUITE, getting usable optimization results requires a substantial amount of time. There are two main reasons for this problem. First, there are the long simulation times of the models, ranging from a few minutes up to several hours, even with large variance of computation times in the same model — depending on the parameter settings.

[5] "Conventional vehicle" terms vehicles which are propelled by an Internal Combustion Engine (ICE) only

[6] trademark of Gamma Technologies, `www.gtisoft.com`

Second, there is the lack of practical possibility for parallelizing the simulations, as GT-SUITE is a proprietary software and requires an unique (costly) license for each parallel simulation instance.

As metaheuristics typically rely upon a large number of fitness/objective function evaluations — in this case simulations with different parameter settings — this is a severe restriction on the efficiency and the effectiveness of the optimization. Therefore one major motivation of this thesis is to explore ways how the number of unique simulations can be lowered or how the time spent by the optimization in the simulation software can be reduced.

The methods to achieve this goal are manifold. First methods are employed to restrict the search space of the optimization, by excluding unpromising parameter combinations, based on the knowledge of an expert — typically the author of the model under evaluation. Second the metaheuristics in use are adapted for faster convergence in promising regions of the search space and third machine learning and regression techniques are introduced into the optimization process to act as a fast (but less accurate) replacement for the simulation software.

As the above mentioned ideas are highly interchangeable, a proper software framework had to be developed for this thesis. This has been done by the author in previous computer science project, resulting in the "Yet Another Optimization Platform" (YAOP, see Chapter 4). This optimization platform defines the basic structure of the optimization task and is designed to be highly modular to allow for an easy integration of different ideas. Further it exposes a web fronted for controlling the optimization platform.

Another motivation for the thesis and the previous computer science project arises from the intended audience of the software system. On the one hand the users of the developed platform are expected to be an automotive engineers and on the other hand metaheuristics and machine learning techniques are powerful but complex tools on their own, best understood by computer scientists knowledgeable in these areas. To allow an effective usage of the optimization software without extensive knowledge in metaheuristics and machine learning techniques, feasible default values for the parameters of the used algorithms are set.

1.2 Thesis overview

Chapter 2 introduces the basic concepts and terminology of HEVs, as well as the basic concepts the used metaheuristics and machine learning techniques. In Chapter 3 previous work in the area of HEV optimization is summarized. Further used improvements for the metaheuristics and machine learning techniques of the previous chapter are discussed. The concrete implementation of the algorithms and introduced alterations are explained Chapter 4. The results of the conducted optimization and regression experiments are given and analysed in Chapter 5. Chapter 6 concludes this thesis by summarizing its main goals and achievements, as well as implying possible future work in the area of HEV optimization.

CHAPTER 2

Basics

2.1 Hybrid Electric Vehicles

Before discussing optimization of hybrid electric vehicles, a short introduction to their concepts and terminology is given. This section follows closely chapter 2 “Definitionen und Klassifizierungen der Hybridkonzepte” of [21].

The most basic definitions needed for understanding the discussed concepts are that of a hybrid vehicle and that of a hybrid electric vehicle. According to [21], a hybrid vehicle is a vehicle with at least two different energy converters used for propulsion and two different and integrated energy storage systems. In consequence a hybrid electric vehicle requires one of the energy converters to be an electric drive (typically an electric machine) and one energy storage to release its stored energy as electric power. The main electric energy storage system relevant for the optimization process are electric batteries.

Another important term for understanding this thesis is the vehicle’s powertrain. The powertrain of a vehicle consists of all parts relevant for propelling the vehicle, beginning at the vehicle’s engines, its shafts, the different gear sets and differentials, finally ending at the cars drive axles.

A HEV’s powertrain can be implemented in different ways and with different degrees of hybridization. This allows to classify the type of HEV in two different dimensions.

The first axis is the degree of hybridization, which refers — simplified — to the possible reliance on the electric drive as main drive of the vehicle, at least for a period of time. [21] distinguishes the following hybridization degrees:

- **Micro hybrids** — possess an additional electric starter motor. The electric motor is mainly used for start/stop (see below) and as power generator for board electronics. This allows that the power of the ICE can be solely used for propulsion.
- **Mild hybrids** — possess a low-power electric machine, which is used for boosting and basic load-point shifting (see below). As well as for recuperation and as a small generator. Further it may be possible to use the electric drive in a pure electric mode at low speeds.

- **Full hybrids** — possess a single or even multiple high-power electric machines. The same functions as for mild hybrids apply, but on a larger scale. The ICE and the electric drive can either be used exclusively or in a mixed drive mode.

- **Plug-in hybrids** — are today basically the same as full hybrids, but with the possibility to charge the battery via the power grid. In the future [21] assumes that plug-in hybrids may be more similar to pure electric vehicles, with an ICE as range extender which cannot be used for propulsion directly, but only for charging the battery.

In this thesis two different HEV models are considered for optimization. The first model — named "Model A" in the following — is classified as a plug-in hybrid and the second model — called "Model B" further on — is classified as a mild hybrid. More information about the vehicles dealt with in this paper can be found in Chapter 5

Hybridization concepts

The second axis for classification is the way how the HEV's different drives and energy storages are integrated. There are multiple ways for achieving this technically. Those relevant for the two vehicles considered in this paper are discussed in the following.

Series hybrid

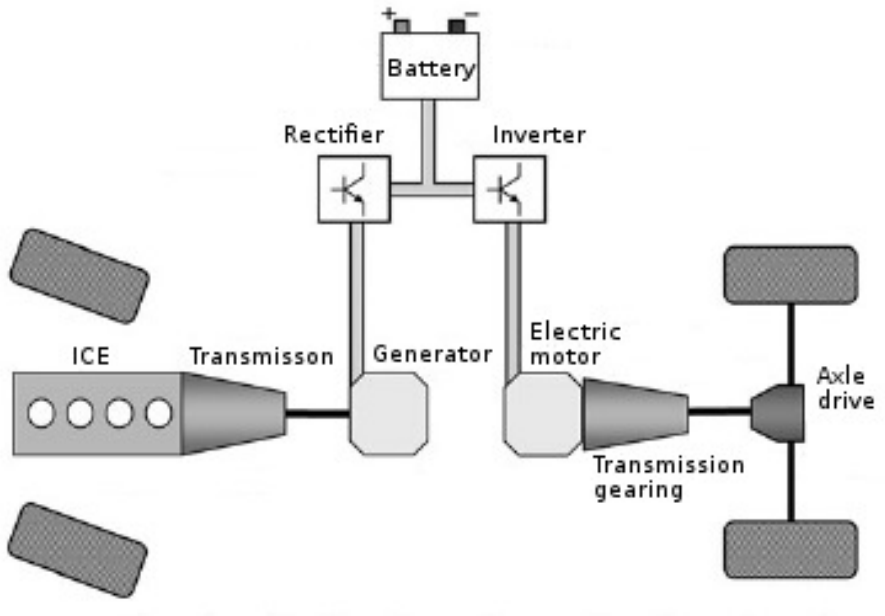


Figure 2.1: The concept of a series hybrid. (Source [21, p. 21] — modified, translated to English)

Although non of the vehicles discussed in this thesis are dedicated series hybrids, “Model A” is able to switch between a series hybrid mode and a power-split hybrid mode (see below). In series hybrids the arrangement of electric drive and ICE is designed so that only the electric drive can be used for propulsion directly. Therefore the ICE is mechanically decoupled from the driving shaft. The setup is depicted in Figure 2.1. The main task of the ICE in this setup is to charge the HEV’s battery. Further in some implementations of the concept it is also possible to bypass the battery and to supply the ICE’s power directly to the electric engine. In both cases a series hybrid needs at least two electric machines: one coupled to the driving shaft via a differential and one used as generator. But the number of electric machines used for propulsion is not limited to one, but can go up to be equal to the number of wheels (one engine per wheel) — a special case called “wheel hub drive” or “electric transmission”.

Series hybrids have several advantages e.g. it is possible to start the ICE delayed and use the electric drive for starting. Further it is possible to operate the ICE in its most fuel-efficient or emission-optimal point as torque and engine speed do not need to be varied to fit the current demands of the propulsion requirements. But there are also disadvantages as for example the high energy losses due to the numerous energy conversion steps in the powertrain.

Parallel hybrid

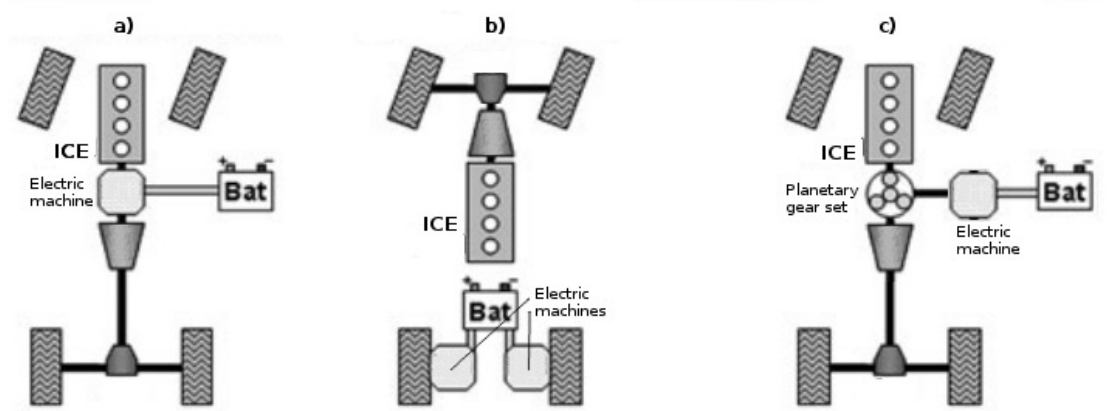


Figure 2.2: Different concepts for parallel hybrids. (Source [21, p. 17] — modified, translated to English)

In contrast to the series hybrids a parallel hybrid setup allows the ICE to propel the vehicle without taking the detour via the electric drive. Thereby the energy loss is reduced in comparison to a series hybrid as fewer energy conversion steps are necessary. But this advantage costs the parallel hybrid to possibility to operate the ICE independently from the current propulsion requirements. This may lead to a more inefficient operation of the ICE compared to a series hybrid. The electric machine is deployed as an alternative drive which may be used as an additional drive — or even stand-alone depending on the concrete realization. Parallel hybrids can be built in different forms:

- With the electric machine placed on the same shaft as the ICE using torque addition (see Figure 2.2a).

- With the electric machine and the ICE coupled via a gear set (shown in Figure 2.2c) e.g. a planetary gear set (see below).

- With each the ICE and the electric drive attached to separate drive axles (shown in Figure 2.2b).

In this thesis the torque addition implementation is of high importance as the vehicle "Model B" is implemented as such.

Power-split hybrid

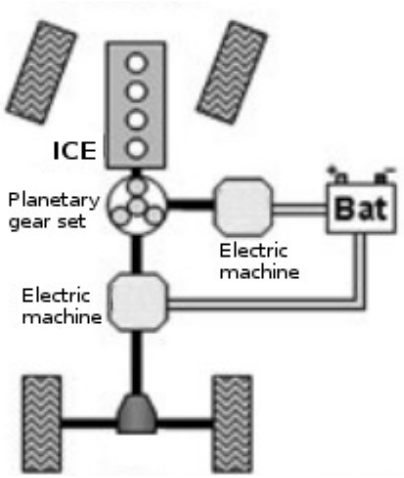


Figure 2.3: The concept of a power-split hybrid. (Source: [21, p. 17] — modified, translated to English)

Power-split hybrids try to achieve some of the benefits from both serial and parallel hybrids. Power-split hybrids are able to transfer the power output of the ICE to two paths. The mechanical path is directly coupled with the driving axles. This allows the vehicle to use the ICE as drive similar to parallel hybrids. The second path is the electrical path which allows to use the ICE to charge the battery via a generator, even while the ICE is used for propulsion. Further a second electric machine can be used as drive at the same time. The benefit of this behaviour is explained below.

The mechanical means for establishing the two paths typically make use of planetary gear sets (see below) for coupling the electric machines with both the ICE and the driving shaft. These special means of coupling allow that both paths are able to operate in parallel, thereby splitting the power output of the ICE — hence the name power-split hybrid.

Modes of operation

In contrast to conventional vehicles which do not allow for much variety in the way the powertrain is used in a specific driving situation (except for the choice of the shifting strategy), HEVs possess a more diverse and dynamic powertrain setup. This variability allows choosing between different modes of operation, dependent on the current vehicle parameters. The choice of the current operational mode is typically not left to the driver but is performed by software whose behaviour is defined in the vehicle's operation strategy.

The definition of a "good" operation strategy has great influence on the performance, the fuel consumption and therefore on the emissions of the HEV. Therefore the parameters of the operation strategy are ideal targets for the optimization algorithms described in this thesis.

The following paragraphs describe a HEV's possible modes of operation and their operational constraints. One has to be aware that the concrete implementation of these modes depend on the vehicle at hand and may be realized in multiple ways — or not at all if the vehicle's concept does not allow for it, or if the operation strategy is not designed for using a specific mode.

The actual realizations of the operational modes for the HEVs discussed in this paper are described in Section 5.2.

Battery state of charge (SOC)

The battery's State of Charge (SOC) is a parameter greatly influencing the operation strategy of the vehicle. Its unit is dimensionless and indicates the remaining charge of the battery in percent. Some operational modes of a HEV are only available at specific SOC ranges e.g. electric vehicle mode, recuperation (see below).

Further the SOC takes a special role in the optimization of the powertrain (see Section 4.2 and Section 5.2), as it is usually required of an operation strategy to perform in such a way that the SOCs at the beginning and at the end of a driving cycle (see Section 2.2) are approximately equal i.e. balanced. Otherwise if the SOC at the end of the cycle is lower than at the beginning, the strategy would not be comparable to another strategy with a balanced SOC. The first strategy would achieve better results merely by draining the battery. On the other hand if the SOC is higher at the end, the strategy could perform worse than a balancing strategy as it has not drained the battery enough — even if the unbalanced one achieved better, an even better strategy would be a similar one using the electric drive more effectively.

Start/stop

The idea of start/stop is to disable the ICE automatically while halting e.g. at traffic lights or during traffic jams. Although start/stop systems are also available in conventional vehicles, according to [21] HEVs may need less time for restarting the ICE by using the electric machine as a starter.

Recuperation

Recuperation or regenerative breaking terms the partial recovery of energy while decelerating. Instead of using the vehicle's breaks or its exhaust brake, a HEV is able use the generator for decelerating while charging the vehicle's battery. This improves the efficiency of the electric drive. If needed, the remaining breaking energy is supplied conventionally. One has to be aware that recuperation is only possible if the battery's state of charge is low enough to cope with the additional energy.

Load-point shifting

Besides recuperation, load-point shifting is another efficient technique for charging the HEV's battery. As the efficiency of ICEs is typically higher if the torque is near its peak for the current engine speed [21], it is reasonable to raise the torque if feasible. This can be achieved by using the torque not needed for propulsion for charging the battery by attaching the generator to the powertrain and thereby increasing the load on the ICE.

Boosting

Boosting[1] is to be understood as the use of the electric drive to support the ICE in propelling the vehicle. This may be required during hard acceleration phases, where the ICE would not be able to deliver the required power alone or if doing so would be inefficient for the ICE.

ICE-only mode

This propulsion mode is typically used if the state of charge is low and no "hybrid" mode is applicable, but also if the battery is charged but the electric drive cannot operate at the current speeds — either by boosting or through EV mode operation.

Electric Vehicle (EV) mode

Reciprocal to the ICE-only mode, the Electric Vehicle mode is the mode where the vehicle uses only its electric machine(s) for propulsion. An usual usage of this mode is in urban environments at low speeds, where electric machines may operate for a longer period of time without recharging. Further electric machines typically have higher torques at lower engine speeds than ICEs, thereby reducing the required energy for driving the first few meters after stopping if compared to an ICE. But depending on the available power output of the electric drive it may also be possible to use the EV mode cross-country at higher speeds.

This is beneficial as electric machines typically have higher torques at lower engine speeds than ICEs,

[1] The term "boosting" used in two-fold manner in this thesis. First it describes an operational mode of a HEV and second it is a machine learning concept for improving the performance of learning algorithms. Therefore the meanings should not be confused.

Planetary gear sets

In parallel and power-split hybrid concepts, a way of coupling the ICE and the electric machine(s) is by using a planetary gear set. Planetary gear sets are important components for realizing the different operational modes of a HEV and its gear ratio(s) directly influence the performance characteristics of the vehicle. Consequently its gear ratio(s) are a viable target for optimization.

Depicted in Figure 2.4 is the structure of such a gear set with its components. A planetary gear set consists of three components connected to different stashes:

- the **sun gear** at the center of the gear set
- the **planet gears** connected with a the planetary carrier and surrounding the sun gear
- the **gear ring**, an inverted gear interacting with the planet gears.

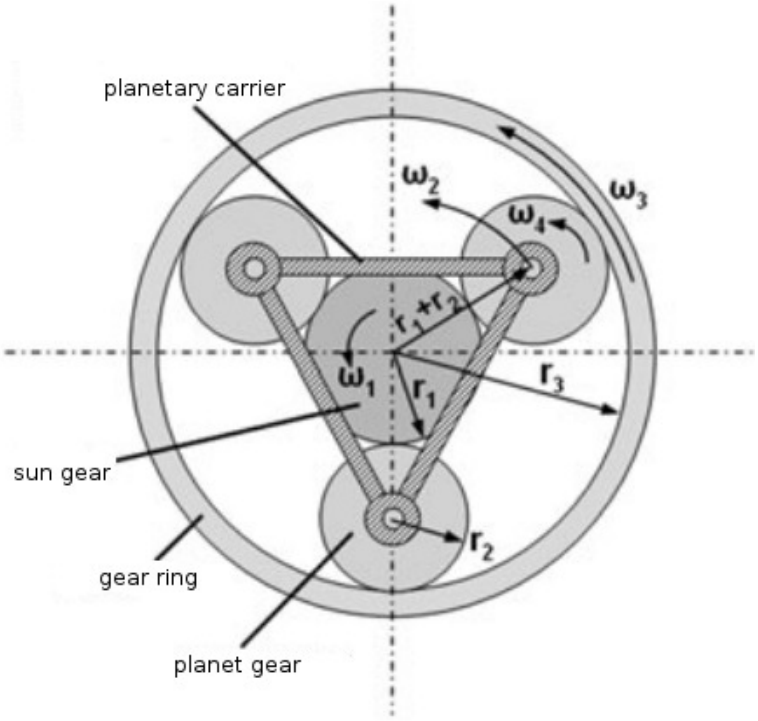


Figure 2.4: The structure of a planetary gear set. (Source [21, p. 28] — modified, translated to English)

A typical setup described in [21] connects the ICE's stash with the planetary carrier, the stash of an electric machine with the sun gear and the driving shaft with the gear ring. There are two modes in which planetary gear sets are able to operate:

- with a single fixed shaft and two rotatable shafts, enabling — depending on the fixed shaft — either an ICE-only mode, an EV mode or a charging mode. Fixing a shaft leads also leads to fixed gear ratios and it is important to choose proper ratios for a maximum of performance.
- with all three shafts freely rotatable, allowing for either using the sun gear for charging or inversely for boosting via the electric machine. In contrast to the mode above the gear ratios in this mode are load dependent but the torques of the separate shafts are fixed.

2.2 Standardized driving cycles

An important characteristic for the driver of any vehicle is the vehicle's every-day performance. Besides the vehicle's engine's energy output and its maximum torque, it gets increasingly important to compare fuel consumption and emissions, before buying a specific vehicle. Therefore these parameters are also important during design and construction of the vehicle.

Fuel consumption and emission differ greatly in every-day usage. So a standardized way is needed to compare these measures. A solution to this problem provide standardized driving cycles. These cycles predefine the targeted driving speed at any point of time of the cycle and its environmental constraints e.g. ambient temperature, engine preheating, etc. This shall guarantee that all measurements of fuel consumption, etc. for different vehicles are recorded with the same constraints in place.

Besides comparing different vehicles with the help of a standardized driving cycle, it can be used for comparing the same vehicle, with different parameter settings too. This approach is used in this thesis when applying meta-heuristics to HEV simulation models.

Both the USA and the European Union define official driving cycles — the EPA US06 and the NEDC — that are used for testing a vehicle's compliance with the current fuel consumption and emission standards. Due to the importance of these driving cycles they are used for evaluating the simulation models in this thesis and are explained below.

EPA US06

In the USA the emission standards for different kinds of vehicles are regulated by the Clean Air Act (CAA) in its latest version of 2008 [45]. According to the CAA, the United Stated Environmental Protection Agency (EPA) is responsible for defining regulations and test procedures for testing a vehicle's compliance to the pollution standards defined by the CAA.

The standardized driving cycles of the EPA for light-duty vehicles i.e. passenger cars, are the Federal Test Procedure (FTP) with a city and a highway version and the three Supplemental Federal Test Procedure (SFTP)s. The SFTPs address specific short comings of the original FTP. The EPA US06 is such a SFTP targeting higher speeds and a more aggressive driving behaviour [1] the original FTP. The US06 driving cycle — depicted in Figure 2.5 — is one of the cycles used in this thesis for assessing and optimizing different HEV parameter settings.

It can be seen that the US06 driving cycle depicted in Figure 2.5 is highly dynamic due to short acceleration and braking phases. Further the cycle is rather short (600 seconds) which results in short simulation times.

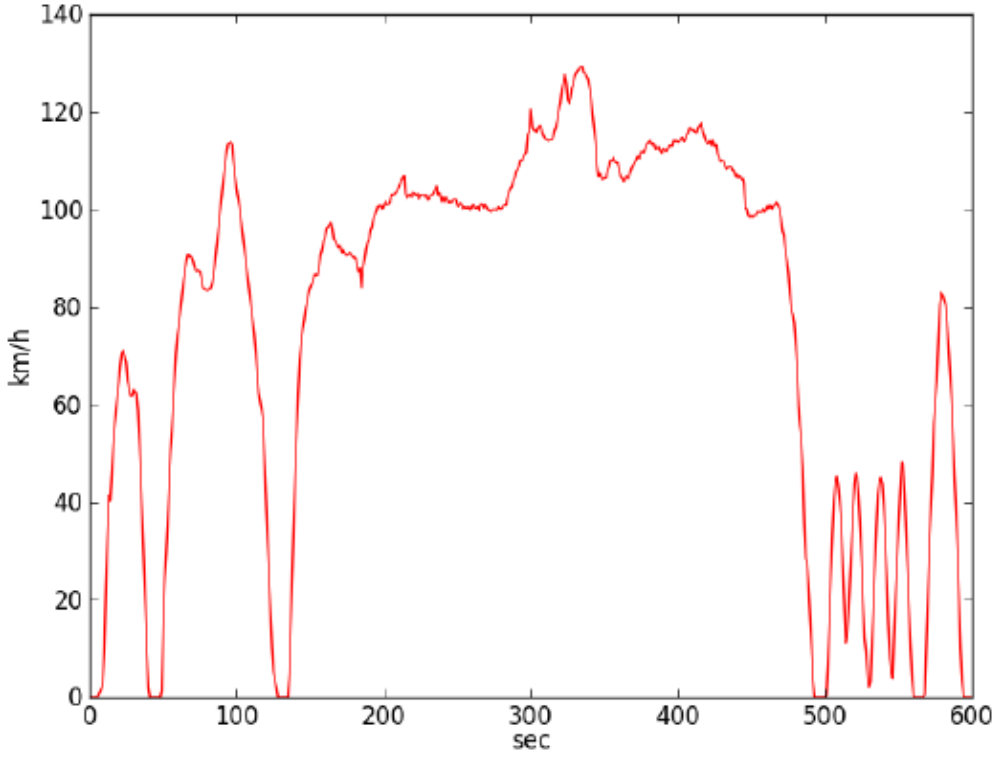


Figure 2.5: A plot of the EPA US06 driving cycle

New European Driving Cycle (NEDC)

The New European Driving Cycle (NEDC) is currently the driving cycle relevant for measuring fuel consumption and emissions for newly registered vehicles in the European Union [36]. The driving cycle is composed from two different driving cycles: the ECE-15 and the EUDC. The ECE-15 is a city driving cycle and is repeated four times at the beginning of the NEDC followed by a single iteration of the EUDC. Besides the US06 driving cycle, the NEDC is used for assessing the fuel consumption during the optimization process used in this thesis.

In difference to the US06 driving cycle is significantly longer and less dynamic. The NEDC is an artificial driving cycle consisting of linear acceleration and braking phases, interspersed with with plateaus of constant speeds.

2.3 Metaheuristics and Optimization

Optimization deals with minimizing or maximizing some objective function (also called fitness function). This thesis deals with continuous parameter optimization problems, with objective functions f of the form $f : \mathbb{R}^d \rightarrow \mathbb{R}$. In this thesis all optimization problems are treated as maximization problems as by Definition 1 — see Section 4.2 for details.

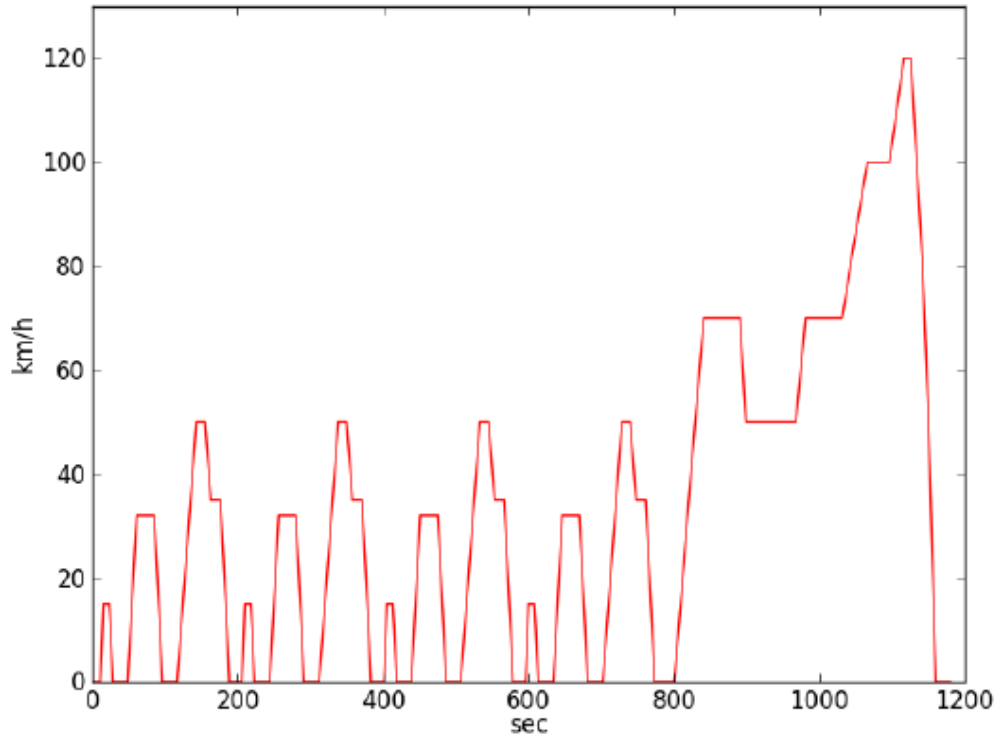


Figure 2.6: A plot of the New European Driving Cycle.

Definition 1. *A continuous parameter maximization problem for some objective function* $f : X \to \mathbb{R}$*, with* $X \subseteq \mathbb{R}^n$*, is defined as follows. Find some global optimum* $\vec{x} \in X$ *such that* $\forall \vec{s} \in X : f(\vec{x}) \geq f(\vec{s})$ *holds.*

As mentioned in Chapter 1 the optimization process uses a black box simulation system for implementing its objective function. Consequently mathematical techniques for solving systems of equations and gradient descent methods are not applicable. Therefore the author relies on metaheuristics which are able to operate on black box objective functions.

The main characteristic of any heuristic is that it is not guaranteed that the method finds a global optimum of its search space. Hence, their performance on a specific problem can only be evaluated empirically. Heuristics are usually problem-specific i.e. a heuristic designed for one problem may be useless for another problem. Metaheuristics provide more abstract concepts than conventional heuristics, as they define frameworks which in sequence use problem-specific heuristics to create new solutions or to modify existing ones. A benefit of using metaheuristics is the possibility to analyse the properties of the used metaheuristics in a generic way to understand specific properties of the metaheuristic itself e.g. like the schema theorem for genetic algorithms or influences of metaheuristic-specific parameters on the search behaviour. The author therefore relies on metaheuristics which have been (empirically) proven to be effective on other optimization problems. The basics of the used metaheuristics are explained below.

Genetic Algorithms (GA)

Genetic algorithms are a popular family of metaheuristics. Their general idea has been introduced by J.H.Holland in 1975 [22]. Further information can be found in [4]. Genetic algorithms are inspired by the concepts of evolution in biology.

In biology, evolution is the process how specific species/individuals came to fill their current biological niche. According to evolution, existing species/individuals exist because their ancestors possessed specific attributes that allowed them to perform better in a reproductive sense, than others in their current environment — this principle is also called natural selection. Further it is assumed that the descendants share at least some of these attributes with their ancestors, making them a similarly good "fit" to their current niche.

These principles are tried to be adapted to the field of optimization. Genetic algorithms belong to the class of population-based heuristics as they deal with a set of possible solutions at a time. Each solution is treated as an individual in a population and is encoded in a "chromosome". For each chromosome a fitness value can be calculated by means of the objective/fitness function of the optimization problem. After an initial population is created a genetic algorithm typically performs the following steps until some halting condition is fulfilled e.g. a number of iterations or convergence of the population.

1. **Selection** — In the selection step individuals are selected which are allowed to pass parts of their chromosome to the next generation. It is important that the selection is based on the fitness value of the individuals for genetic algorithms to work. Often used selection methods are roulette wheel selection, where a each individual is selected with a probability proportional to its fitness and tournament selection, where for each individual to be selected two or more are randomly chosen and the best of these is then selected.

2. **Recombination** — The individuals selected in the previous step are then recombined into new individuals. This step depends strongly on the encoding used for the chromosomes. Usually the attributes (genes) of two individuals are intermingled to produce a new individual.

3. **Mutation** — Commonly the recombination step reduces the variance in the attributes of the new population. This is due to using the information of a small set of selected ancestors to create a larger set of descendants, thereby losing information from the individuals not selected for recombination. As this behaviour would lead to fast convergence i.e. all individuals become identical, a small percentage of each descendant's genes are mutated to introduce more variance into the new population. Again this steps depends on the encoding scheme used for the chromosomes.

GAs usually target combinatorial optimization problems in contrast to continuous optimization problems, as some of the typically applied encoding/recombination/mutation concepts do not apply well to continuous variables. Although binary encodings like gray coding can be used for implementing continuous variables, a different encoding is used in this thesis as described in Section 4.2.

Nevertheless, due to modifications to the search space applied in this thesis (see Section 4.2), it is possible to apply GAs on discrete variables at least for some phases of the optimization process.

Evolution Strategies (ES)

Evolution Strategy (ES) have been introduced by Schwefel in [42] and are similar to genetic algorithms, with a few but nevertheless important differences. Whereas genetic algorithms are primarily used in combinatorial optimization, evolution strategies have been developed with continuous optimization in mind. An overview can be found in [5] and [6]

In ES, like in genetic algorithms, solutions are encoded as chromosomes. Although unlike in genetic algorithms, solutions are encoded as a real valued vector of the solution's parameters. Further the main operation in evolutionary strategies is not recombination but mutation. Mutation is usually implemented as addition of a normally distributed vector to an individual's chromosome. The recombination step is often omitted.

For selection an own notation has been introduced. It distinguishes between two types of evolutionary strategies.

1. $(\mu + \lambda)$-ES have a population size of μ and produce λ descendants per generation. From this pool of $\mu + \lambda$ individuals the best μ individuals are selected. This scheme introduces a concept called elitism to the ES, where an individual may survive several generations instead of one.

2. (μ, λ)-ES have a population size of μ and produce λ descendants per generation. The best μ descendants then form the new population.

A commonly applied modification to mutation in ES is to use a separate normal distribution with zero mean for each individual. The variances and covariances — the so called strategy parameters — of these normal distributions are then included into the chromosomes of the individuals. The strategy parameters are then also prone to selection. This technique works well as long as the number of dimensions is low as for higher dimensional counts the algorithm begins to suffer from the "curse of dimensionality". Alternative attempts for controlling the ES strategy parameters are described in Section 3.3.

Particle Swarm Optimization (PSO)

Like GA and ES, Particle Swarm Optimization (PSO) is another metaheuristic mimicking nature. Though PSO is not inspired by evolutionary concepts. PSO has been developed by Eberhart and Kennedy in 1995 [27].

PSO tries to emulate the food search behaviour of birds, which have a tendency to cluster around rich food sources. Similar to ES, PSO is typically used for continuous optimization — although discrete variants like Binary-PSO exist. In PSO each solution is represented as a particle, a real valued vector of the solution parameters, representing the position of the particle in the search space. Further each particle possesses a velocity in the search space. Each particle moves through the search space, evaluating the solution at its current position and orienting itself

towards its own best solution and the globally known best solution. The exact steps executed in each iteration are described below.

1. For each particle i, evaluate the objective function at its position $\vec{x_i}$.
2. For each particle update its own best solution $\vec{b_i}$ and the globally known best solution $\vec{g}$.
3. For each particle update its current velocity $\vec{v_i}$.

$$\vec{v_i} = \vec{v_i} + \phi_1 \cdot \text{rand}_{[0,1]} \cdot (\vec{b_i} - \vec{x_i}) + \phi_2 \cdot \text{rand}_{[0,1]} \cdot (\vec{g} - \vec{x_i}), \quad \phi_1 = \phi_2 = 2 \tag{2.1}$$

   Where $\text{rand}_{[0,1]}$ is function providing uniform random values in the range $[0, 1]$.

4. For each particle update its current position $\vec{x_i}$.

$$\vec{x_i} = \vec{x_i} + \vec{v_i} \tag{2.2}$$

There exist several variations of the update scheme like canonical particle swarm optimization (see Section 3.2). As for GA and ES, the steps above are executed until some halting condition is fulfilled.

2.4 Regression & machine learning techniques

According to [30] the evaluation of the used objective function requires a substantial amount of time. This is due to the need to simulate a driving cycle with the simulation software "GT Suite" for evaluating the objective function. The simulation poses a severe bottleneck to the whole optimization. As a major intent of this thesis is to lower the time needed by the optimization process, without increasing the number of parallel simulations, it is reasonable to think about ways for replacing the evaluation model. This has already been proposed in [30].

A possible way to do this is by creating a regression model for the objective function or at least for parts of the objective function (see Section 4.2). A regression model/function tries to capture the hidden dependencies between some input variables in $\mathbb{R}^d$ and some output variable in $\mathbb{R}$. Regression models require preobtained data for model fitting. This data will further be called training data or training set. In the case of the used objective functions the data is collected from previous simulations during the optimization process.

It is distinguished between different types of regression:

- **Parametric regression** uses regression models $\psi_{\beta_1,\ldots,\beta_k}(\vec{x})$ of a predefined form e.g. polynomial, exponential, logarithmic. The regression techniques are used to determine the parameters $\beta_i, 1 \leq i \leq k$ of the regression function $\psi : \mathbb{R}^d \to \mathbb{R}$.
- **Non-parametric regression** uses regression models of no predefined form. An example for non-parametric regression models are regression splines like multivariate adaptive regression splines [14].

- **Semi-parametric regression** is a combination of parametric and non-parametric regression i.e. models which allow adaption of the model by modifying parameters of functions included into the model as well as providing some variability in the form of the function.

The regression methods used in this thesis primarily are semi-parametric neural networks. But also parametric and non-parametric techniques are employed. The basics of these techniques are explained below. Advanced concepts used in the experiments are described in Section 3.5 and Section 4.3.

Linear regression

Linear regression is a form of parametric regression, where a linear function is used as regression model. The task of the regression analysis is then to determine $\alpha, \vec{\beta}$ of the function ψ below.

$$\psi_{\alpha,\vec{\beta}}(\vec{x}) = \alpha + \vec{\beta}^T \vec{x} \tag{2.3}$$

The parameters of the linear model are chosen such that they minimize some error function for a given set of training data $(\vec{x_i}, y_i), 1 \leq i \leq m$, with m the size of the training set. A commonly used error function is the Sum of Squares Error (SSE).

$$\text{SSE} = \frac{1}{2} \sum_{i=1}^{m} (y_i - \psi(\vec{x_i}, \alpha, \vec{\beta}))^2 \tag{2.4}$$

If the SSE is used as error function the method for determining the regression parameters is called "least squares method". The parameters can be calculated directly by setting their gradient to zero as the SSE for linear regression is a convex function.

For another method to fit a linear regression models see the artificial neural network section below.

Generalized linear regression

The concept of linear regression can be generalized by the use of basis functions $\phi_j : \mathbb{R}^d \to \mathbb{R}$. These functions allow to use arbitrary function forms for regression. The regression model than takes the following form.

$$\psi_{\vec{\beta}}(\vec{x}) = \sum_{j=1}^{k} (\beta_j \phi_j(\vec{x})) = \vec{\beta}^T \boldsymbol{\phi}(\vec{x}) \tag{2.5}$$

where k is the number of basis functions and $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^k$ is the vector-valued function combining the basis function. For example one-dimensional linear regression can be expressed in the framework of generalized linear regression with the basis functions shown below.

$$\boldsymbol{\phi}(x) = \begin{pmatrix} \phi_1(x) \\ \phi_2(x) \end{pmatrix} = \begin{pmatrix} 1 \\ x \end{pmatrix} \tag{2.6}$$

The parameter values $\vec{\beta}$ for generalized linear regression are acquired by minimizing some error function like SSE. In general it is not possible to minimize the error function by setting its gradient to zero as the error function is not required to be convex. Therefore other methods for function minimization are generally used, like gradient descent, Newton's method or the Levenberg-Marquardt algorithm (see Section 3.4).

If generalized linear regression is used the question arises which basis functions should be chosen for fitting the training set. Unfortunately there is no direct answer to this question and if no further information about the training set is available then proper basis functions have to be determined experimentally.

Artificial Neural Networks (ANN)

The term Artificial Neural Network (ANN) refers to models and methods inspired by the workings of biological neural networks. These biological networks are well known to possess exceptional computation, memory and learning capabilities. ANNs have been developed for different kinds of purposes like simulating their biological counterparts to gain better understanding of their workings, reconstructing damaged input signals, classification of inputs and — most relevant for this paper — regression of input and output data.

The methods for implementing ANNs are as diverse as their intended purpose. Network types that can be used for regression are for example radial basis networks or multi-layer perceptrons, which are explained in the following. ANNs are usually classified as semi-parametric regression techniques.

The Perceptron

Before introducing multi-layer perceptrons the model of a (single layer) perceptron is introduced. Multi-layer perceptrons are an extension of the perceptron neural network model. The perceptron has been devised by F. Rosenblatt [40]. A perceptron is a (very abstract) model of biological neuron and similarly possesses weighted inputs — the artificial equivalent to dendrites — and a single output — equivalent to the axon. Figure 2.7 depicts the basic structure of the perceptron model.

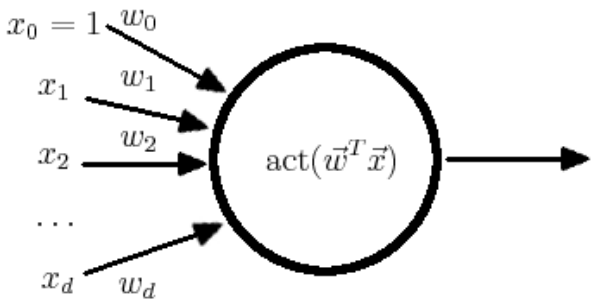


Figure 2.7: The structure of a perceptron.

The function computed by a perceptron is shown in Equation 2.7. The parameters — called "weights" in the ANN context — of the model are denoted as $\vec{w}$. The weight vector is of the same size as the input values $\vec{x}$. It has to be noted that the inputs of the regression problem are augmented by an additional first element which is always of the value 1. The additional value is also called "bias" or "bias neuron" and is an elegant way to implement a constant offset to the net (= netto input) function net : $\mathbb{R}^d \to \mathbb{R}$. The net function is used to calculate the input for the activation function from the inputs to the neuron. The activation function act : $\mathbb{R} \to \mathbb{R}$ calculates the actual output of the perceptron.

$$\psi_{\vec{w}}(\vec{x}) = \text{act}(\vec{w}^T \vec{x}) \tag{2.7}$$

There is a variety of activation functions like signum — used originally — or sigmoid functions. Observe that if the identity function is used as the activation function then the perceptron computes the same function as a multivariate linear regression. Therefore the training algorithms for the perceptron can also be used as training algorithms for linear regression.

Several perceptron training algorithms exist. One of the most flexible — with regard to the activation function — is the method of gradient descent applied to some error function (e.g. SSE) of the perceptron output.

Multi-layer Perceptrons (MLP)

Minsky and Papert showed in [34] that the single layer perceptron — with specific activation functions — has severe restrictions e.g. the XOR problem is not linearly separable. To overcome these limitations one possibility is to use a Multi-Layer Perceptron (MLP). MLPs do not use a single perceptron but several layers of perceptrons in sequence. Thereby the outputs of the neurons in one layer are connected to the inputs in the next layer as depicted in Figure 2.8.

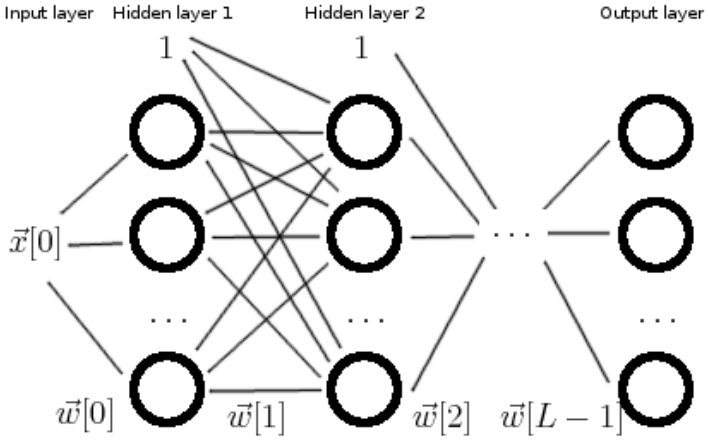


Figure 2.8: The schematic of a multi-layer perceptron.

This thesis uses layered feed-forward MLPs, where each neuron in layer i is connected to all neurons in layer $i + 1$. Further each layer except the last (output) layer contains a single

bias neuron whose output is always 1. The layers between the input and the output layer are called hidden layers. The number of neurons in the hidden layers and their activation functions greatly determine the class of functions which can be represented by a specific neural network. In general MLPs tend to be more flexible with regard to the form of their output function than generalized linear regression methods.

In the following the terminology needed for explaining the computation of the neural network is introduced. Let $\vec{x}[l]$ be the input vector of layer $l+1$ i.e. $\vec{x}[0]$ denotes the augmented inputs of the whole neural network and $\vec{x}[l],\ 1 \leq l \leq L$ denotes the outputs of layer l with L the number of layers. Accordingly the matrices $\boldsymbol{w}[l],\ 0 \leq l < L$ denote the augmented weights between the neurons of layer l and $l+1$. Further let $\#(l)$ denote the number of neurons in layer l, not including the bias neuron. Thereby the matrix component $\boldsymbol{w}[l]_{ij},\ 0 \leq i \leq \#(l),\ 1 \leq j \leq \#(l+1)$ belongs to the weight between neuron i of layer l and neuron j of layer $l+1$. The regression function ψ described by the network is then recursively calculated as follows.

$$\begin{aligned} \vec{x}[0] &= \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_{\#(0)} \end{pmatrix} \\ \vec{x}[l+1] &= \text{act}[l+1](\boldsymbol{w}[l]^T \begin{pmatrix} 1 \\ \vec{x}[l] \end{pmatrix}),\ 0 \leq l < L \\ \psi_{\boldsymbol{w}[1],\ldots,\boldsymbol{w}[L-1]}(\vec{x}[0]) &= \vec{x}[L] \end{aligned} \tag{2.8}$$

There remains the problem of learning the weight values of the MLP. The learning algorithm most often explained in introductory lectures to neural network training is the (error) backpropagation algorithm [8], which is basically a gradient descent algorithm with a SSE error function. The main problem solved by backpropagation is how the gradients of the neurons in the hidden layers are computed.

Despite the popularity of backpropagation, it suffers from low convergence speed and is prone to drop to local minima. Early experiments during the work on this thesis have shown that backpropagation is not suitable for training neural networks for the regression problems in this thesis. Therefore a different learning algorithm — the Levenberg-Marquardt algorithm for neural networks (see Section 3.4) — is used for training the neural networks, which has been observed to perform better and to converge faster than backpropagation.

CHAPTER 3

State of the Art

3.1 Previous work

Introductory note: the mathematical variables, definitions and algorithms introduced in Chapter 2 are the same in their respective context and not described again. This holds true in the further chapters too.

Optimization of HEVs

Early efforts in optimizing operation strategies of HEVs has been done by Johnson, Wipke and Rausen in 2000. In [26] they developed a real-time control strategy for a parallel HEV model. For optimization the commercial "VisualDOC™" software[1] is used to optimize the strategy's parameters on a surface fitted model obtained from Design of Experiments (DoE). The optimization is carried out using Sequential Quadratic Programming. It is mentioned that the simulation of the HEV model using the ADVISOR[2] simulation software requires a substantial amount of time — up to ten times longer than the models considered in this thesis, although the simulation times cannot be directly compared due to the advancements in computational power.

Another approach for optimizing parallel HEVs is explored by Hu, Wang and Liao in [23]. Their intent is to minimize a multi-objective function compromised by the HEV's fuel consumption and several emissions by using a Genetic Algorithm (GA) variant for Pareto-optimization. Thereby the multi-objective case is treated quite differently to the approach chosen in this thesis (see Section 4.2 for a discussion). Further the paper does not address the computation times for the simulation.

Last the paper [30] and master thesis of Krenek [31] is an important reference for this thesis. The work deals first with the feasibility of using GT-Suite as objective function component and applies different metaheuristics to Model B (see Section 5.2) of this thesis. Further a hybrid optimization approach PSAGADO combining several metaheuristics has been developed. The

[1] `http://www.vrand.com/visualDOC.html`, accessed 2013-06-15

[2] Developed by the National Renewable Energy Laboratory (NREL) of the USA and commercialized by AVL.

problems of the long computation times, parallelization problems due to license restrictions and the ideas for the numerical restriction of some parameters and the use of approximative objective functions — which are addressed in this thesis — are presented in its outlook, too.

Combining metaheuristics with regression techniques

In [2] an introduction to simulation optimization is given. The authors of the paper state that it is common to use so called metamodels for simulation optimization to reduce computationally expensive simulation times. A metamodel is some kind of algebraic representation of the simulation i.e. some kind of regression function. It is stated that linear regression is one of the most common techniques, but also neural networks are named as a type of metamodel in more recent publications. According to [2] metamodels typically act as a filter which lets only solutions of a specific estimated quality be passed to the simulation model.

A concrete example where neural networks have been used as a metamodel is given in [28], where the physical parameters of a rectangular microstrip antenna are optimized. In contrast to [2] the neural network — trained beforehand with data from experiments — is used solely as objective function for the implemented GA, without ever relying on a more elaborate simulation model.

Jin, Olhofer and Sendhoff developed a framework for combining evolutionary algorithms with neural networks as metamodels in [25]. Besides the afore-mentioned filtering approach — called "managed individuals" in the paper — a different usage scenario for metamodels called "managed generations" is presented. Therein the neural network is used as main objective function. Although every few generations the original simulation model is used to evaluate a whole generation of individuals. Further the paper addresses the matter of retraining of the regression models and suggests a novel method for using the covariance matrix of the CMA-ES (see Section 3.3) algorithm for selecting new training samples.

3.2 Advances in PSO

In the years after its first introduction PSO has become a widely known technique for metaheuristic optimization and has been adapted to an extensive set of optimization problems. Further the number of PSO variants has grown nearly as fast. A good overview of these variants is given by Poli and Kennedy in [37].

The modifications to the original PSO range from rather simple adaptions as the introduction of velocity controlling parameters [43] to different neighbourhood topologies for the velocity update. As the developed system is intended to be used by engineers with rather limited experience with metaheuristic techniques, a tradeoff between flexibility of adaption to the particular to-be-optimized HEV model and general ease of use has to be made. According to these considerations a rather simple PSO variant has been chosen, requiring few parameters to be set and for whom reasonable standard settings are known.

Canonical Particle Swarm Optimization

The original PSO as described in Section 2.3 suffers from unbounded velocities as also mentioned in [37]. The velocity of the particles tends to grow ever larger thereby leading to oscillations in the search space. A velocity-limiting constant $V_{\max}$ may be used as in [27] s.t. if $||v_i|| > V_{\max}$ then $v_i = \text{sign}(v_i)V_{\max}$ to reduce this effects but it is noted in [37] that this is not sufficient.

In [37] a PSO variant with a so called "constriction coefficient" is discussed, an adaption originally proposed by Clerc [9]. This coefficient $\chi, 0 \leq \chi \leq 1$ is multiplied with the updated velocity v_i of each particle i. Thereby reducing the effective velocity of the particle. Further it is noted in [37] that although for a properly chosen χ — about $\chi = 0.7298$ — the algorithm is proven to converge. These modifications yield the so called "Canonical Particle Swarm Optimization".

Also it is mentioned in [37] that the Canonical PSO can also be expressed by using a different velocity update rule which uses an inertia weight. This adaption has been introduced in [43] and is one of the earliest PSO variants. The Equation 2.1 controlling the velocity update is changed to Equation 3.1.

$$\vec{v_i} = \omega\vec{v_i} + \phi_1 \cdot \text{rand}() \cdot (\vec{b_i} - \vec{x_i}) + \phi_2 \cdot \text{rand}() \cdot (\vec{g} - \vec{x_i}) \tag{3.1}$$

The new inertia factor ω decays the memory of the previous direction of the particle each iteration. By setting $\omega = 0.7928$ and $\phi_1 = \phi_2 = 1.49618$ the algorithm becomes identical to the Canonical PSO.

In this paper the Canonical PSO is implemented using the update rule with inertia, as this variant is widely used and the author considers its parameters more intuitive than the variant using the constriction coefficient.

3.3 Advances in Evolution Strategies

As mentioned in Section 2.3 managing the strategy parameters as part of the chromosome is a common method for automatically controlling the search behaviour of the ES. Its drawback is its limited scalability. In the following a different approach — the so called Covariance Matrix Adaption — for adapting the strategy parameters is introduced. It is expected that the use of this parameter adaption scheme leads to a faster convergence of the algorithm requiring fewer evaluations of the costly objective function.

Covariance Matrix Adaption (CMA)

In [20] Hansen and Ostermeier introduced the Covariance Matrix Adaption (CMA) strategy for adapting the strategy parameters of an ES. Instead of determining the strategy parameters of the ES indirectly by exposing them to selection, CMA computes them directly from the previously selected mutation steps. The underlying idea of CMA is that successful mutation steps from the previous generation will likely be successful in the next generation. The strategy parameters

are therefore adapted such that previously successful mutation steps become more likely to be produced again.

To explain the concept of the update scheme a special version of ES has to be introduced, namely the (μ_w, λ)-ES. This ES variant is similar to the (μ, λ)-ES with the exception that a weighted recombination is used. Weighted recombination computes a weighted-average chromosome $\vec{x}_w[j+1]$ from the individuals $\vec{x}_i[j]$ of the current population — of size N — and a set of associated weights $\vec{w}_i$, $1 \leq i \leq N$ as given in Equation 3.2. The computed weighted-average chromosome then acts as the center for the mutation step in iteration $j+1$. The newly generated individuals are seeded around this center according to the updated normal distribution of iteration $j+1$.

$$\vec{x}_w[j+1] = \frac{1}{\sum_{i=1}^{\mu} w_i} \sum_{i=1}^{\mu} w_i \vec{x}_i[j] \tag{3.2}$$

In the following the general idea of the update is explained but not the exact algorithm which can be found in [20]. As it is rather lengthy to describe the algorithm without proper introduction to its basic concepts. Although the algorithm for the CMA-ES variant which is actually used in the experiments is given in Section 4.2.

1. Let $\boldsymbol{C}[j]$ be the covariance matrix of the used normal distribution with zero mean in iteration j. Then $\boldsymbol{C}[j]$ can be described by its eigenvalues and eigenvectors. Let $\boldsymbol{B}[j]$ be a matrix with the normalized eigenvectors of $\boldsymbol{C}[j]$ as columns and $\boldsymbol{D}[j]$ be a diagonal matrix containing the square roots of the according eigenvalues.

   Further each individual i can be expressed as a tuple $(\vec{x}_w[j], \vec{z}_i[j])$ where $\vec{x}_w[j]$ is the distribution center of iteration j and $\vec{z}_i[j] \sim \boldsymbol{N(0, I)}$ is a standard normal-distributed vector. The actual position $\vec{x}_i[j]$ of individual i can be expressed as below.

$$\vec{x}_i[j] = \vec{x}_w[j] + \sigma \underbrace{\boldsymbol{B}[j]\boldsymbol{D}[j]\vec{z}_i[j]}_{\sim N(0,\boldsymbol{C})} \tag{3.3}$$

2. Calculate the weighted average $\vec{z}_w$ of the normal distribution realizations of the best μ individuals in iteration j. In this paper the implementation uses the arithmetic mean instead of a weighted average.

3. Calculate $\boldsymbol{C}[j+1]$ as below, where c_{cov} is a constant weighting the influence of the previous iteration's normal distribution.

$$\boldsymbol{C}[j+1] = (1 - c_{\text{cov}})\boldsymbol{C}[j] + c_{\text{cov}}(\boldsymbol{B}[j]\boldsymbol{D}[j]\vec{z}_w)(\boldsymbol{B}[j]\boldsymbol{D}[j]\vec{z}_w)^T \tag{3.4}$$

   The first component $(1 - c_{\text{cov}})\boldsymbol{C}[j]$ can be understood as a contraction of the axes of highest variance of $C[j]$ i.e. its eigenvectors/values. The second component rotates and stretches $C[j+1]$ towards $\vec{x}_w[j+1]$.

The idea of using the best individuals' mutation steps for adapting the covariance matrix of the mutation distribution is extended further by using so called evolution paths. An evolution

path accumulates the information of successful mutation realizations over several generations. To do this the term $\boldsymbol{B}[j]\boldsymbol{D}[j]\vec{z}_w$ in Equation 3.4 is replaced by the evolution path vector $\vec{p}_c[j+1]$ which is calculated as follows.

$$\vec{p}_c[j+1] = (1-c_c)\vec{p}_c[j] + \sqrt{c_c(2-c_c)}\boldsymbol{B}[j]\boldsymbol{D}[j]\vec{z}_w[j+1] \tag{3.5}$$

Thereby $\vec{p}_c[j+1]$ contains directional information from several generations with c_c controlling the memorization time of the directional information. It can be seen that c_c acts as an inverse inertia factor. Further a second evolution path $\vec{p}_\sigma$ is used for controlling the adaption speed of the mutation's step size σ independently of the adaption speed of the mutation's direction. The update for σ can be found in [20, p. 18].

The experiments in [20] showed that CMA-ES is a promising metaheuristic outperforming ES with mutative control of strategy parameters and other ES variants on a set of test functions.

Active Covariance Matrix Adaption

Although CMA-ES showed a good performance compared to other ES techniques it is noted in [20] that the adaption speed of the covariance matrix is somewhat slow. CMA-ES is expected to require several hundred generations for fully adapting its mutation distribution. For the problem at hand this is deemed infeasible due to the particularly high costs of function evaluations.

Therefore a variant of the CMA-ES is implemented for this thesis. The variant[3] called Active Covariance Matrix Adaption ES has been proposed by Jastrebski and Arnold in [24] and targets the particular problem of slow adaption of the covariance matrix. Where CMA-ES uses the information of successful mutation steps only, there Active CMA-ES uses additional information about unsuccessful mutation steps. Therefore Equation 3.4 is modified to Equation 3.6 by using the evolution path improvement and by adding a new term for incorporating information about the current generation's successful and unsuccessful mutations.

$$\boldsymbol{C}[j+1] = (1-c_{\text{cov}})\boldsymbol{C}[j] + c_{\text{cov}}\vec{p}_c\vec{p}_c^{\,T} + \beta\boldsymbol{Z}[j+1] \tag{3.6}$$

$$\boldsymbol{Z}[j+1] = \boldsymbol{B}[j]\boldsymbol{D}[j]\frac{1}{\mu}\Big(\sum_{k=1}^{\mu}\vec{z}_{\pi_k}\vec{z}_{\pi_k}^{\,T} - \sum_{k=\lambda-\mu+1}^{\lambda}\vec{z}_{\pi_k}\vec{z}_{\pi_k}^{\,T}\Big)(\boldsymbol{B}[j]\boldsymbol{D}[j])^T \tag{3.7}$$

In Equation 3.7 π denotes the permutation ordering the individuals by their decreasing fitness values. Therefore $\boldsymbol{Z}$ is a scaled and rotated covariance matrix which actively reduces the variance in the direction of the μ worst individuals of the current generation. The factor β controls the influence of $\boldsymbol{Z}$ on the covariance matrix update.

The experiments in [24] show that ACMA-ES reaches predefined stop values on a set of test functions, significantly faster than CMA-ES and the Hybrid-CMA-ES variant. Consequently the author considers ACMA-ES as an ES variant particularly suited for optimizing functions with high evaluation costs.

[3] More specifically the algorithm is a variant of the Hybrid-CMA-ES described in [19], which has not been accessible for the author.

3.4 Advances in neural network training

As mentioned in Section 2.4 the "standard" training algorithm for neural networks, error backpropagation, is deemed unusable for the requirements of this thesis. The use of the standard backpropagation algorithm or improvements like resilient backpropagation [38] have resulted in very slow convergence rates (even divergence at some times) and therefore long computation times.

A research for different training algorithms has brought up some alternatives, ranging from metaheuristic training techniques like using genetic algorithms [35] to standard numerical optimization method like Newton's optimization method, BFGS [39] and the Levenberg-Marquardt algorithm [18]. The choice has finally fallen upon the Levenberg-Marquardt algorithm as the results in [18] seem promising. Another reason for choosing the Levenberg-Marquardt algorithm has been the availability of an easy-to-use library (see Section 4.1). This is an important advantage as the proper implementation of a performance-critical numerical algorithm is not easily done even if the algorithm is known as (constant) performance improvements achieved by properly using CPU caches and memory alignments can be significant.

Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm for minimizing least squares error functions with nonlinear parameters has been developed independently by Levenberg in [32] and Marquardt in [33]. Marquardt describes the intent of the algorithm to combine both the gradient descent approach and the Gauss-Newton algorithm for finding local optima for least squares problems.

The gradient descent approach is used by the standard backpropagation algorithm and basically steps iteratively along the direction of the negative gradient $-\boldsymbol{J}(\boldsymbol{w}[l])$ of the backpropagated error.

The Gauss-Newton algorithm is another numerical optimization algorithm that can be used to fit the nonlinear parameters $\vec{\beta}$ of a function $f(\vec{x}, \vec{\beta})$ to a training set $(\vec{x}_i, y_i), 1 \leq i \leq m$. The algorithm is derived by using the first-order Taylor polynomial $F(\vec{x}, \vec{\beta} + \Delta\vec{\beta}) = f(\vec{x}, \vec{\beta}) + \nabla f(\vec{x}, \vec{\beta})\Delta\vec{\beta}$ to approximate the local neighbourhood of $f(.)$. The approximation is then used in the SSE function and the derivative of the SSE function is then set to zero to find an optimum of the local approximation. The change of the parameters $\Delta\vec{\beta}$ can then be computed by equation 3.8.

$$\Delta\vec{\beta} = (\boldsymbol{J}^T(\vec{\beta})\boldsymbol{J}(\vec{\beta}))^{-1}\boldsymbol{J}^T(\vec{\beta})(\vec{y} - \boldsymbol{f}(\boldsymbol{x}, \vec{\beta})) \tag{3.8}$$

The benefit of this optimization scheme is that additionally to the gradient $\boldsymbol{J}$ the information of the Hessian matrix — $\boldsymbol{J}^T(\vec{\beta})\boldsymbol{J}(\vec{\beta}$ is a linear approximation thereof — can be used. In general this allows for a faster convergence if compared to gradient descent, though Marquardt names divergence as a main problem of the algorithm in [33].

Therefore the Levenberg-Marquardt algorithm tries to interpolate between gradient descent and the Gauss-Newton algorithm by changing the update rule of Gauss-Newton to Equation 3.9

$$\Delta\vec{\beta} = (\boldsymbol{J}^T(\vec{\beta})\boldsymbol{J}(\vec{\beta}) + \lambda\boldsymbol{I})^{-1}\boldsymbol{J}^T(\vec{\beta})(\vec{y} - \boldsymbol{f}(\boldsymbol{x}, \vec{\beta})) \tag{3.9}$$

The new parameter λ is used to interpolate between the above mentioned methods. In the first case if λ is large then the product of the gradient is irrelevant for the update and the equation becomes similar to gradient descent. In the second case if λ is small then the equation is basically the same as the equation for the Gauss-Newton algorithm. λ is updated each iteration depending on the new value of the SSE function. If the update of the parameters would result in a larger value for the SSE then λ is multiplied by some constant factor γ turning the algorithm towards the more stable gradient descent. Otherwise if the SSE would be lower after the update then λ is divided by γ.

In [18] the Levenberg-Marquardt algorithm is incorporated into the training of neural networks. This is done by adapting the traditional backpropagation method by exchanging the gradient descent in the "weight space" with the Levenberg-Marquardt algorithm. The main problem left is the computation of the Jacobian $\boldsymbol{J}$ i.e. the gradient which is solved in [18] by a modification to the backpropagated error in the backpropagation algorithm.

It shall be noted that training algorithm for neural networks implemented by the used library[4], is a modified version of the Levenberg-Marquardt algorithm using the exact Hessian matrix $\boldsymbol{H}$ instead of the linear approximation $\boldsymbol{J}^T\boldsymbol{J}$.

3.5 Ensemble learning

Although neural networks which are complex enough can theoretically be used to fit any function, it is hard to determine the proper size to do so. On the one hand if the network is too small i.e. consists of too few neurons, then the ANN cannot represent the training data adequately and is said to underfit or to have a high bias. On the other hand if the network is too large then it is prone to overfit the training data. In this case its generalization performance degrades significantly i.e. its ability to predict the correct values of new inputs is worsened. For the task at hand both cases may worsen the effectiveness of the metaheuristics used to optimize the HEV models.

Therefore further machine learning techniques are evaluated to determine if they provide advantages over simple neural networks. The chosen techniques are of the family of the so called ensemble (learning) methods. The general idea of ensemble methods is to combine several base predictors — ANNs in this case — to create an ensemble of learners. In the following the basic principles of the ensemble methods that are evaluated are explained. Further details on the implemented methods are provided in Section 4.3.

Bootstrap Aggregating (Bagging)

Bootstrap Aggregating or "Bagging" for short is an ensemble method developed by Breiman in [7]. It works by training K base predictors $\varphi_k,\ 1 \leq k \leq K$ on different training sets

[4] See `http://www.alglib.net/dataanalysis/neuralnetworks.php` for details; accessed 2013-05-30 11:50

$T_k = \{(\vec{x}_i, y_i) | 1 \leq i \leq N\}$ for the same function. For predicting the target value y of some input $\vec{x}$ the average of the trained predictors is used as described by equation 3.10.

$$\varphi(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} \varphi_k(\vec{x}) \tag{3.10}$$

As noted in [7] typically only a single training set T is available for the learning task. The trick for applying bagging is therefore to sample K training sets T_k from the available set T to mimic the existence of several different training sets. The sampling is performed with replacement to approximate the underlying distribution of T.

In principle bagging works by reducing the variance of the used predictors through the averaging step. Consequently the method is suited for improving unstable predictors like ANNs. According to [7] the more unstable the used predictors, the higher the expected benefit of bagging gets.

Boosting

Another family of ensemble methods are categorized as boosting algorithms. Contrary to bagging, boosting algorithms are not concerned with trying to lower the variance of the base predictors in the first place, but to decrease the bias of them. Therefore boosting is expected to be well suited for the use of so called weak learners/predictors — prediction models that achieve marginally better results than random guessing [41]— as base predictors. The research of the author showed that boosting is much more common if machine learning is applied to classification tasks, such as the AdaBoost algorithm [13]. Nevertheless some boosting algorithms have been specifically developed with regression problems in mind.

Gradient Boosting

Gradient Boosting and its variant Stochastic Gradient Boosting are such boosting algorithms and have both been developed by Friedman in [16] and [15]. Gradient Boosting builds its ensemble predictor iteratively as a sum of base predictors, each trained to correct the error of the previous state of the ensemble predictor. Although the algorithm is specified for several different error functions, in the following only the SSE function is used.

1. Let $i = 0$ and let $\phi_i(\vec{x})$ be a constant estimate minimizing the error of the training set T.
2. Calculate the residual error $r_j = y_j - \phi_i(\vec{x}_j),\ 1 \leq j \leq |T|$
3. Find $\arg\min_{\rho_{i+1}, \vec{\beta}_{i+1}} \sum_{j=1}^{|T|} (r_j - \rho_{i+1} f(\vec{x}_j, \vec{\beta}_{i+1}))^2$ using line search, where $f(.)$ is a base predictor trained on T. It has to be considered that finding the optimal parameters β_{i+1} of the regression model and the weight ρ_{i+1} of the expansion step is non-trivial in general.
4. Set $i = i + 1$ and let $\phi_i(\vec{x}) = \phi_{i-1}(\vec{x}) + \rho_i f(\vec{x}, \vec{\beta}_i)$.
5. Repeat from step 2 until the maximum number of expansions has been reached.

The decision to rely on the SSE function has been made as SSE is used as error function for all other machine learning methods in this paper. Choosing a different error function would be counterproductive as the ANNs are optimized to reduce the SSE function. Therefore using another error measure is expected to lead to sub-optimal results.

Besides the original Gradient Boosting algorithm, the Stochastic Gradient Boosting [15] is evaluated too. The Stochastic Gradient Boosting uses a similar approach to bagging in modifying its training set T. For each iteration a different training set T_i is sampled from the original training set T without replacement.

CHAPTER 4

Implementation

4.1 Languages, libraries and licenses

Before discussing the implementation of the algorithms and concepts a short overview of used programming languages, libraries, programs and licenses is given.

The main programming language used for the implementation is Python 2.7[1]. Python is a very high-level programming language possessing built-in support for procedural, object-oriented and functional programming styles. The language has been chosen for its flexibility and good library support for the problem at hand. Though the standard implementation of the language is an interpreter, the performance is not limited thereby as only a small percentage of the overall computation time during optimizaton is spent inside the interpreter. The remaining time is taken up by the simulation software GT-Suite, which is used in version 7.2. Further the time-intensive tasks such as neural network training or linear algebra operations (e.g. eigenvalue decomposition) are handled by native libraries.

Neural network training is handled by `ALGLIB`[2] in version 3.6.0. The library is dual-licensed and available under the GNU General Public License (GPL) for non-commercial and academic purposes. The library is written in C which allows the training of ANNs — one of the most time-consuming tasks — to be performed faster than a pure Python library could have.

Linear algebra and advanced random functionality is provided by the `scipy` Python package. `scipy` relies on the well known `BLAS` and `LAPACK` libraries for implementing its linear algebra functionality and is therefore sufficiently fast. The package is licensed under the BSD-new license, which is considered GPL-compatible.

The results of GT Suite simulations are saved in a MySQL database. The connectivity is provided by the `MySQLdb` Python package.

Many XML-based configuration files need to be read (e.g. GT Suite model files, optimization platform configuration, algorithm configurations) and modified. As a typical SAX or DOM ap-

[1] `www.python.org`
[2] `www.alglib.net`

proach to XML parsing would require substantial effort for extracting and storing the required information, an XML-to-Object-Mapper (XOM) is used. The XOM in use is the `xmlflection` library which has been developed by the author for another project some years ago and is provided under the GNU Lesser General Public License (LGPL) license.

As a consequence of the libraries used above the current version of the optimization platform has to be licensed under the GPL — mainly due to the restrictions of the `ALGLIB` library.

4.2 The "Yet Another Optimization Platform" (Yaop)

The different algorithms, post-processing capabilities and strategies for fitness evaluation which are used in this thesis are integrated into the Yet Another Optimization Platform (Yaop). The basics of the platform have been developed by the author in a precursive project addressing some of the shortcomings of the optimization platform used in [31] e.g. multi-user support, easier integration of new HEV models, and license scheduling/planning.

The entity — of the optimization platform — most relevant for the user[3] is an optimization project. A project encapsulates the HEV model to be optimized, the settings for the adjustable model parameters, the optimization algorithm configuration and the configuration of the used fitness function.

The platform consists of four main components each running in a separate process:

- The **license scheduler** manages the GT Suite licenses which are currently available for use by optimization processes. Before a simulation is run, the optimization process must request a license from the scheduler and has to release it after the simulation has finished.

- The **license planner** controls the availability of GT Suite licenses for optimization processes over the course of a week. This is necessary as GT Suite licenses may be used by users who use GT Suite directly. For reducing license conflicts between these different usage scenarios only a portion of the overall available licenses is used by optimization processes at a time i.e. half the number of bought licenses during office hours and no restrictions on weekends.

- A **web interface** has been developed for creating, configuring and running optimization processes. Further it provides functionality for configuring the license planning component as well as the scheduler component.

- Each optimization project is run in a separate **optimization process**. Such a process carries out the actual optimization, following the specified algorithm configuration. Further they communicate with the license scheduler, the database and are controlled via the web interface.

[3] A user of Yaop is a person who wants to run an optimization on the platform, as well as a person who chooses and configures the used optimization algorithm.

Optimization algorithm template pattern and basic terminology

The two main principles of the platform's design are modularity and transparency. First the platform is designed in a modular way to allow a wide variety of optimization algorithms. For achieving this goal the main loop of the optimization process follows the well known "Template Method" design pattern originally identified in [11].

In the "Template Method" design pattern, the control flow and the steps of an algorithm are outlined at an abstract level. The outlined steps themselves are implemented as callbacks to interfaces which have to be implemented by the concrete parts of the algorithm (as described below). Before the algorithm template itself is explained some terminology has to be introduced for a better understanding of the design decisions.

In terms of the framework a potential solution is called a `Run` and a set of potential solutions is called a population, a `RunCollection`, or a set of `Runs`. `Runs` contain an internal state which is either

- **unevaluated** if the `Run` has not been passed through an evaluation scheme,
- **valid** if an objective function value has been assigned to the `Run`,
- or **invalid** if the `Run` represents a set of model parameters which are infeasible to be solved by the evaluation scheme.

`RunCollections` are generated by the optimization algorithm, which is defined by an algorithm configuration loaded from an XML file. The configuration defines the order of the executed optimization algorithms, their algorithm-specific parameters and their termination criteria. Further it determines the post-processing behaviour for potential solutions, as well as the way how these solutions are expected to be evaluated. Last if a hybrid optimization scheme is used i.e. if several different optimization algorithms are combined in sequence then the algorithm configuration allows to specify sources for solutions. These sources are queried either initially or when the algorithm becomes active the next time. Possible sources might be for example the best solutions from a database, random solutions, or combinations thereof.

The metaheuristics have to implement the `AbstractOptimizationAlgorithm` interface to be used by the platform. The required callbacks of the interface are described below.

- `prepare` is called initially to set up the algorithms inner state according to the passed `ParameterConfiguration`, defining the parameters to be optimized, their bounds and other optional options.
- `passRunCollection` is called on specific events like initialization or (re-)activation of the algorithm and sends a new population to the algorithm. This method is essential for the correct handling of hybrid algorithms.
- `nextIteration` is a method to implement an "Iterator" pattern [11] for generating new `RunCollections` which are evaluated consecutively.

- `onRunCollectionFinished` is a callback method to update the iterator's state after the objective function of the previously generated `Runs` have been evaluated. It is not guaranteed by the framework that the set of previously generated `Runs` is the same as the one passed to the method.

In general an optimization algorithm is at least wrapped inside one `LoopController` which handles the chaining of several optimization algorithms within a loop. The controller itself implements the `AbstractOptimizationAlgorithm` interface and acts as a proxy for the controlled algorithm instances. It shall be noted that a `LoopController` may contain further `LoopControllers` to achieve more complex optimization schemes.

Beside optimization algorithms, a main component of the template method are so called `PostProcessors`. A `PostProcessor` is able to modify the population returned by an optimization algorithm before and possibly after passing them to the evaluation scheme. Their intended usage is to enforce search space restrictions without needing the actual optimization algorithm to care about it.

Ideally the optimization algorithms should be implemented to deal transparently with an unknown search space topology i.e. by holding no inner model or assumptions about the numerical search space and by adopting the solutions passed to the algorithm at the above defined entry points/methods. During development the transparency concept has been observed to be also highly relevant in the design of hybrid metaheuristics. The author's observation is that when implementing optimization algorithms for the platform, one should follow a principle of minimal state. Any internal state or "belief" stored inside an optimization algorithm might be rendered irrelevant by another employed optimization algorithm. Rather the state should be transferred to the used `Runs` as far as possible or recreated (and updated) when the control flow[4] reverts back to the optimization algorithm.

This kind of transparency enables a high reusability of the implemented optimization algorithms, while specific restrictions on the optimization are outsourced into typically shorter (in lines of code), less complex and more maintainable `PostProcessors`.

The optimization algorithm template is explained in the following.

1. The algorithm configuration is loaded from an XML file. A composite optimization algorithm is constructed by using the `AbstractOptimizationAlgorithmFactories` configured in the global Yaop configuration file.

2. All instantiated algorithms are `prepared` with a `ParameterConfiguration` created from the optimization project's settings (also an XML file).

3. The initial run source is invoked for the first algorithm, or an empty `RunCollection` is passed to it if no run source is defined. These two actions are handled transparently as the top-level `LoopController` proxies the appropriate method calls.

4. Until the top-level's `LoopController's` halting condition — for the conducted experiments it is a fixed number of loop iterations — is triggered the following steps are executed:

[4] Note that control flow and data flow are strongly coupled in the framework.

a) `nextIteration` is invoked on the top-level `LoopController` and returns a new `RunCollection`.

b) The returned `RunCollection` is passed through the post-processors defined for the currently active algorithm.

c) The post-processed set of `Runs` is passed through the evaluation scheme defined for the currently active algorithm. `ComputationStrategies` implement the evaluation scheme and are explained below.

d) The evaluated runs are optionally returned to the post-processors (in reversed order), allowing them to react to the computation results.

e) Finally `onRunCollectionFinished` is called with the evaluated set of `Runs` on the current algorithm for updating its state.

Objective/fitness functions

The above mentioned evaluation schemes are required to impose a total order on the returned `Runs` for the optimization to function properly. This order is imposed by the objective/fitness function of the `ComputationStrategy` assigned to the currently active optimization algorithm.

The algorithms of the framework have to deal with two different kinds of objective functions. First the exact "ground truth" objective which is calculated by using the target/output values of a GT Suite simulation. Second the inexact fitness function provided by the learned metamodels/regression models. Whereas the approximative metamodels try to automatically mimic features of the exact fitness function, the objective function to be used with the simulation model needs to be defined by the user of the platform.

Basic objective function

The objective function used in [31] is a function relative to some arbitrarily chosen reference solution. Further the penalization of solutions with unbalanced SOC[5] has been hard-coded in the optimization framework. As the optimization platform developed for this thesis intends to support a broader range of HEV models, some changes to objective function have been made.

Instead of using an arbitrary reference point, an absolute fitness function has been chosen. Its design goal is to allow a fine-grained control for multi-objective optimization[6]. Let t_i, $1 \leq i \leq M$ be the output/target values of the HEV model e.g. fuel consumption, SOC deviation, etc. for some input parameters $\vec{x}$, with M as the number of output values selected for optimization. Additionally the user has to specify — estimated — lower and upper bounds l_i, u_i for each target value t_i. Let w_i, $1 \leq i \leq M$ be the weights associated with each target value. Consequently w_i

[5] see Section 2.1

[6] The possibility to use Pareto-optimality has been declined in favour of a more direct influence on the optimization, as well as for a broad range of implementable standard algorithms.

can be used to either maximize or minimize t_i — $w_i > 0$ or $w_i < 0$ respectively. Then the basic objective function $o(\vec{x})$ is defined as in Equation 4.1.

$$o(\vec{x}) = \sum_{i=1}^{M} w_i \frac{t_i - l_i}{u_i - l_i} \tag{4.1}$$

The function is designed s.t. if $\forall i, j : |w_i| = |w_j|$ a percental change of any t_i within its bounds l_i, u_i has an equal absolute effect on $o(\vec{x})$ as the same percental change of another t_j within its respective bounds. Further the relative effect on the objective function for any t_i can be adjusted by using its weight w_i. So far it is not possible to achieve non-linear weighting via the framework. If such a behaviour is required then the appropriate function should be applied directly to the HEV model's output values in GT Suite.

Multi-case objective function

Another issue addressed with the new objective function is the possibility to evaluate multi-case simulation models. GT Suite provides means to define different cases for a simulation model. Each case assigns fixed values to model parameters not under optimization. These case parameters are typically of environmental nature under which the HEV model shall be examined e.g. different driving cycles, different initial SOCs, air resistance (wind), etc.

The platform provides different solutions for multi-case optimization. The user is able to configure the platform s.t. either case averaging is used or the fitness of the best or worst case is used. The fitness function is then one of the following.

$$o_{\mathrm{a}\,vg}(\vec{x}) = \frac{1}{|C|} \sum_{c \in C} o(\vec{x}) \tag{4.2}$$

$$o_{\mathrm{m}\,in}(\vec{x}) = \min o(\vec{x}) \tag{4.3}$$

$$o_{\mathrm{m}\,ax}(\vec{x}) = \max o(\vec{x}) \tag{4.4}$$

Care has to be taken if either Equation 4.3 or 4.4 is used as multi-case objective function. Different cases may have a different "base value" for some target value e.g. for fuel consumption if the driving cycles are of different length. Therefore the fitness values for different cases might not be comparable, leading to unintended search behaviour. A possible fix for this problem would be to assign a weight to each case. This is currently not implemented as the intended scenarios for multi-case optimization are unlikely to include such uncomparable case pairs and the effect on Equation 4.2 is lessened by averaging.

Dealing with invalid parameter sets

As the objective functions defined above are only applicable with parameter sets compatible with the HEV model at hand, a way has to be found to deal with invalid parameter sets. The reasons for a parameter set being invalid are manifold. Usually these invalid parameter sets can be divided into those which are knowable beforehand — e.g. gear sets underlie certain mechanical

restrictions, or up- and down-shifting speeds need to be ordered correctly — and those which result in problematic runtime behaviour — e.g. speed requirements of a driving cycle cannot be met.

The first category of invalid parameter sets — those foreseeable by the user — should be dealt with by repairing the affected sets if possible. In the terms of the optimization platform repairing of solutions should be handled by some post-processor implementing a side condition. The second category is harder to deal with as the simulation has possibly required some time to deem the used parameters as invalid. Even if the the effecting parameters could be determined and repaired, a substantial amount of time would be required to resimulate the configured model. Therefore a database cache is used to store invalid parameter sets and is queried before a configured model is submitted to the simulation software. Invalid model configurations are then removed from the current population and the optimization algorithm becoming active next is required to fill the empty spot.

Computation strategies

The association of a `Run` with its fitness value is performed by a `ComputationStrategy` which can be divided in exact and approximative computation strategies. If a hybrid optimization algorithm is used then it is possible to select a `ComputationStrategy` per component or falling back to a default strategy. This allows for a diverse set of evaluation schemes over the course of the optimization. A topic of this thesis is to evaluate the viability of some of these complex evaluation schemes. The different available `ComputationStrategies` are described next.

GT Suite local & distributed computation strategy

The platform currently ships with two exact computation strategies, both relying on the evaluation of the HEV simulation model under optimization with GT Suite. The main difference between the strategies are where the models are simulated.

- `LocalGtSuiteComputationStrategy` simulates the HEV model with the parameter sets of the passed `RunCollection` locally on the computer where the optimization process is executed. The number of parallel simulations is restricted by the number of local CPU cores and by the number of available licenses.

- `RemoteGtSuiteComputationStrategy` passes the HEV model instantiated with the passed parameter sets to a running GT Suite Distributed Queue. The "Distributed Queue" is a facility provided by GT Suite to queue simulation requests from several users and to dispatch the queued models to different computation nodes. This allows the optimization process to use a cluster of computers for parallel execution, whereas the local strategy is only able to use the multiple CPU cores on the local machine. Of course the same license restrictions apply as on the local strategy.

Both implementations are able to execute multiple simulations at once as simulation is important for an efficient use of the available computational resources. `Runs` are submitted in

batches per case if the HEV model is used for multi-case optimization i.e. for each parameter set the first case is simulated before the second case of any other `Run`. By doing this invalid parameter set can be detected and filtered early on. The remaining cases of the affecte `Run` are then not simulated as a single invalid case renders the `Run` itself invalid. For achieving the highest effectiveness with this evaluation scheme, the cases should be order by decreasing aggressiveness i.e. the case most likely to fail should be simulated first.

Predicting computation strategy

A simple approximative evaluation scheme is the `PredictingComputationStrategy` which requires an instance of a `LearnerFactory` to be passed. The passed factory is used to create a `Learner` — an implementation of a regression technique — which is then either trained with the data from the database, or whose previously saved state — called a continuation in terms of the platform — is loaded from a file.

The trained `Learner` is used to predict the fitness of the passed `Runs`. Currently the `Learner` is trained to predict the fitness value directly instead of predicting the single objective function components, which is done to reduce the training times of the regression models. Additionally to predicting the fitness value, the `PredictingComputationStrategy` flags the `Run`'s fitness as approximative s.t. the optimization algorithms may react to this circumstance.

The `PredictingComputationStrategy` allows to retrain the used regression model after a specified number of evaluated populations. This is helpful if a hybrid optimization strategy is employed to generate new exact solutions regularly. Currently it is not possible to retrain a regression model and continuing the optimization e.g. by using an exact evaluation scheme meanwhile. This would be beneficial as it may require some time to update the state of the `Learner` (see Section 5.5). As in the current setup the computational resources used for training and simulation are not the same, doing so would allow to evaluate more exact solutions in the same amount of optimization time. Due to the modularity of the platform this scheme may be implemented easily in the future as another `ComputationStrategy`.

Filtering computation strategy

In the spectrum of exact and approximative evaluation schemes exist some interesting combinations of those. The `FilteringComputationStrategy` is the first of two mixed evaluation schemes.

The `Runs` passed to this computation strategy are assigned an exact fitness value by passing the parameter sets to one of the `GTSuiteComputationStrategies`. Although, not the whole `RunCollection` is evaluated exactly but a fraction of it. Only the best `Runs` — based on their predicted fitness value — are be selected and in sequence passed to the simulation software. The main benefit of this evaluation scheme is that the optimization algorithms can not be misguided by inaccurate fitness values. Nevertheless the approximative fitness value may have a misguiding effect on the selection of the best `Runs`. To mitigate the possible negative influence on the optimization, not the best predicted `Runs` are selected, but a roulette-wheel selection is carried out to select a predefined portion of the `RunCollection`.

The usage of the `FilteringComputationStrategy` is limited to optimization algorithms which are able to produce a larger number of descendants than the number of individuals in the current population. Otherwise the population would be thinned out throughout the optimization process requiring the empty spots to be refilled with random or elitist solutions. Therefore this evaluation scheme is considered problematic if used in combination with PSO and search space discretization (see Section 4.2 below). This is due to the PSO's property that the variance of the possible descendants of some solution is limited to a hyper-conical area spanned by the weighted distances of the current solution to the global and local best solutions respectively. The variability is reduced further by the discretization of the search space.

Extrapolating computation strategy

The second mixed evaluation scheme is the `ExtrapolatingComputationStrategy`. In contrast to the filtering approach its computed fitness values are not exact but approximative.

The HEV models used by GT Suite can be configured to write the current values of the model's outputs, which are relevant for the objective function, to a file in regular periods e.g. the fuel consumed up until now. GT Suite also allows to halt the simulation at the end of such a time period. Then a trained `Learner` can be used to predict the value of the output at the end of the cycle based upon the information collected during simulation. (see Section 4.3 for details).

This evaluation method has some drawbacks if multiple output values are used to evaluate the objective function. Specifically, a single `Learner` is required per target value. Even if the time required for training of these regression models can be kept low, the problem of model selection has to be solved for each of the model's outputs.

Postprocessing

Postprocessing of `Runs` has been introduced to relieve the optimization algorithms from menial tasks as checking parameter bounds as well as to decouple other functionality used by several optimization algorithms e.g. discretization, repairing of invalid solutions, or non-database caching.

Discretization post-processor

A problem that may arise when continuous functions are optimized, is the occurrence of plateaus in some regions of the fitness function i.e. larger areas where the slope/first-order derivation is zero. If function evaluations are costly then computing multiple solutions in such a region is a waste of computation time. In simulation this problem is worsened as any non-trivial model possesses an inherent model error and a deviation between the calculated outputs and the actual outputs is unavoidable. In consequence if the difference between the fitness of two solutions is less then the expected deviation error then both solutions may simplified be considered of equal quality. Therefore the term plateau shall be applied to areas of shallow slope/small first-order derivation (less than the model error) in the context of simulation optimization.

A common technique to reduce computation times is to use a cache of already known results which is also known as memoization. As it is unlikely to hit a specific solution twice in a suffi-

ciently large continuous search space, speed ups achieved due to caching effects are expected to occur rarely. A method to increase the possibility of a cache hit is to discretize the search space. The possibility to discretize the search space is also discussed in [31]. "Similar" `Runs` are then mapped to the same point in the search space, which allows cache hits to occur more frequently if the optimization algorithm performs a locally restricted search.

A very simple discretization scheme is implemented by the `LinearDiscretization` post-processor which defines a static discretization of the search space. The grain of the discretization is determined by two parameters (per input value) `discretizationStart` and `discretizationEnd` which define the number of steps in the search space at the start and at the end of the optimization respectively. The number of discretization points are linearly interpolated during the optimization. This allows to start at with a very coarse grained space to estimate the overall fitness distribution and to use a finer grained space afterwards.

A parameter set $\vec{x}$ is then mapped to its discretized parameter set $d(\vec{x})$ as follows. Let $\vec{l}$ be the vector containing the lower bounds of all dimensions and let $\vec{u}$ be the vector of upper bounds respectively. Let $\vec{s}$ be the vector of the number of discretization steps per dimensions and let $\vec{s^{-1}}$ denote its component-wise inverse.

$$\vec{k} = \text{round}\left(\frac{\vec{x} - \vec{l}}{(\vec{u} - \vec{l})\vec{s^{-1}}}\right) \tag{4.5}$$

$$d(\vec{x}) = \vec{l} + \vec{k} \cdot (\vec{u} - \vec{l})\vec{s^{-1}} \tag{4.6}$$

The discretized parameter set $d(\vec{x})$ can then be used twofold. Either the `Run` representing $\vec{x}$ is replaced by a `Run` representing $d(\vec{x})$ permanently, or only temporarily for evaluation and is reverted to $\vec{x}$ afterwards. Both variants have advantages and drawbacks. If the "permanent discretization" is used then the optimization algorithm operates with the parameter values which have actually been evaluated. The algorithms do not need to deal with border cases like different fitness values for the same parameter set, if the number of discretization steps changes. Further algorithms which rely on estimated gradients are not confused by false plateaus inevitably introduced by the temporary discretization scheme. On the other hand if permanent discretization is used then optimization algorithms relying on relative steps in the search space e.g. like PSO, may run into problems as they need to jump a minimal distance $\Delta\vec{x}[i]$ in some dimension i of the search space. Only if some dimension i fulfills this requirement then the parameter set will be mapped to a different point in the discretized search space. The minimal distance is given by Equation 4.7.

$$\Delta\vec{x}[i] > \frac{\vec{u}[i] - \vec{l}[i]}{2\vec{s}[i]} \tag{4.7}$$

Currently only the permanent variant has been implemented and the temporary variant is left for future work due to time constraints.

Discrete domain & integer domain post-processor

The `DiscreteDomain` and `IntegerDomain` post-processors work similar to the post-processor described above, but have an entirely different intent. Both post-processors are used to force a specific parameter either to be integer or to be of a value from some predefined set.

The `DiscreteDomain` post-processor is useful if a unchangeable discretization shall be applied to a parameter or if the discretization points are not equidistant as enforced by the `Discretization` post-processor. The `IntegerDomain` may further be used to express boolean values if the bounds of the parameter are additionally set to $[0, 1]$.

Inequality-coupled parameters post-processor

In some cases the need arises that some parameters need to adhere to some half-order, like up-shifting speeds for consecutive gears e.g. the speed for the second gear has to be lower than the speed for the third gear. The `InequalityCoupledDomain` post-processor allows to force the value of a parameter to be less/greater-equal than the values of a set of other parameters.

If the restricted parameter violates these conditions it is repaired by determining the allowed range of the parameter's value and by uniformly sampling the parameter in this range. As a consequence it is important which parameter is marked as restricted as the restricted one will change its value while the other will keep its value. For example if two parameters a, b have to conform to $a \leq b$ and a is restricted then a will be resampled. Further if for example three parameters a, b, c have to adhere to $a \leq b \wedge a \leq c$ then a should be marked as its the infimum of the imposed half-order. Otherwise the conditions would be expressed as $b \geq a$ and $c \geq a$ with b, c as restricted parameters. In the worst case both b and c would change their value randomly and the modified `Run` would probably be located farther away from the original `Run`. This may hinder the effectiveness of the optimization scheme.

Available run sources

With different `RunSources` the platform provides ways to to couple multiple optimization algorithms to different hybrid optimization algorithms by controlling how `Runs` are passed from one algorithm to next. Besides the selection of the used optimization algorithms, the used `RunSources` are the primary way to create new hybrid optimization schemes for the platform. Basically there are three sources which can be accessed to retrieve `Runs`:

- The latest `RunCollection` generated by the **previous optimization algorithm** can be used as a source [7].
- The **database** used to store `Runs` with exact fitness value can act as a source too.
- The **`Empty`** source can be used, whereas the optimization algorithm has to purge its current population and to create a new one on its own.

Further each source might be accessed differently:

[7] Using the complete `RunCollection` of the previous algorithm is the default.

- **All** `Runs` can be used if the previous algorithm acts as source.
- A **random set** of `Runs` can be selected from either the database.
- Only the **best `Runs`** are selected from either the database.

Finally multiple `RunSources` can be combined to a compound source either by

- using them in parallel to retrieve a fixed number of `Runs`,
- or by using a secondary `RunSource` to fill up the requested number of `Runs` if the main `RunSource` is unable to satisfy the request.

`RunSources` are queried either at the first use of an optimization algorithm in the current loop or if the hybrid optimization scheme switches from one algorithm to another.

Used Metaheuristics

The metaheuristic algorithms used in the experiments largely conform to the descriptions as given in sections 2.3, 3.2, and 3.3, which in sequence correspond to the original proposals of these algorithms — except recombinant random sampling which is a simple GA variant designed for comparison purposes.

Special attention is payed to the circumstance that no selection scheme inside the optimization algorithms uses a kind of fitness-proportional selection, as when exact and approximate objective function values are not directly comparable as the actual and regressed values even of the known solutions are likely to diverge. The author considers that rank-based selection copes better with this effect if the exact and approximated fitness values do not differ beyond given bounds (see Section 4.3 for details). The optimization algorithms have not been modified further as the primary goal of this thesis is to evaluate the possibilities of integrating machine learning methods with the these algorithms, as well as to integrate search space modifications.

PSO

The implementation of the Particle Swarm Optimization follows closely the CPSO variant described in Section 3.2. It differs mainly in the way which solutions are accepted as global or local best solution. The modification of this mechanism has been necessary as the PSO is intended to be used alternately with the `PredictingComputationStrategy` and an exact strategy. The update scheme has to distinguish approximate and exact fitness values, due to the reason mentioned above. A `Run` $\vec{x}$ will replace a global or local best solution $\vec{b}$ if

- $o(\vec{x}) \geq o(\vec{b})$ and $\vec{x} \neq \vec{b}$ holds,
- or if $o(\vec{b})$ is approximative and $o(\vec{x})$ is exact — this gives precedence to exact solutions to correct for potential misguidance by approximated solutions.

Further a randomization mechanism is introduced to escape local optima, by randomly resampling a `Run`'s position and velocity, if it has not updated its personal best solution after a specified number of iterations. The velocity is generally limited to a maximum velocity expressed as a percentage of the size of the search space's dimensions e.g. a maximum velocity of 0.25 means that no particle will jump further than one quarter of a dimension's size in any dimension. This adaption is introduced as the PSO is required to use larger values for $\phi_i, i \in \{1, 2\}$ than usual — as given for the canonical PSO in [37] — to overcome the problems described in the discretization section.

Active CMA-ES

The Active CMA-ES has basically been implemented according to [24]. No adaption to the internal selection mechanism had to be introduced as the Active CMA-ES is intended to be used with the `FilteringComputationStrategy` and is working with exact solutions all the time. To enable the Active CMA-ES to be combined with other optimization algorithms, means have to be developed to estimate the covariance matrix $\boldsymbol{C}[j]$, the step width σ and the mutation center $\vec{x}_w[j]$ from the passed `RunCollection` for the current iteration j. A way to do this is given in Equation 4.8, and Equation 4.10 and Equation 4.11.

Let $\vec{x}_i$, $1 \leq i \leq N$ with N the size of the passed population, be the individuals of the passed population. Let π denote the permutation ordering the individuals by their decreasing fitness values. The normal distribution realizations $\vec{z}_i$ of the individuals are estimated by using the new mutation center $\vec{x}_w[j]$ and the individuals' positions $\vec{x}_i$, as given in Equation 4.9. Let $\vec{z}_w$ be the weighted averages of the normal distribution realizations z_i of the best μ individuals of the passed `RunCollection`.

$$\vec{x}_w[j] = \frac{1}{N} \sum_{i=1}^{N} \vec{x}_i \tag{4.8}$$

$$\vec{z}_i = \frac{\vec{x}_i - \vec{x}_w[j]}{||\vec{x}_i - \vec{x}_w[j]||} \tag{4.9}$$

$$\boldsymbol{C}[j] = \vec{z}_w \vec{z}_w^T + \frac{\beta}{\mu} (\sum_{k=1}^{\mu} \vec{z}_{\pi_k} \vec{z}_{\pi_k}^T - \sum_{k=\lambda-\mu+1}^{\lambda} \vec{z}_{\pi_k} \vec{z}_{\pi_k}^T) \tag{4.10}$$

$$\sigma = \sqrt{\frac{1}{\mu - 1} \sum_{i=1}^{\mu} \left(||\vec{x}_{\pi_i} - \vec{x}_w[j]|| - \frac{1}{\mu} \sum_{k=1}^{\mu} ||\vec{x}_{\pi_k} - \vec{x}_w[j]|| \right)^2} \tag{4.11}$$

The remaining values are initialized as usual.

Recombinant random sampling (RRS)

For initial training of the regression models, it is important to accumulate a sufficient amount of diverse training data. As explained in Chapter 5 the above mentioned algorithms showed some problems in fulfilling these criteria. Therefore a simple GA variant has been implemented to

generate an initial batch of solutions. The GA uses no special encoding of the individuals but the vector representation $\vec{x}_i$ of each individual $1 \leq i \leq N$.

1. Generate N random individuals $\vec{x}_i$, $1 \leq i \leq N$ and evaluate them.
2. Sample R new individuals randomly.
3. $N - R$-times iterate the following steps to generate the remaining individuals of the new population:
   a) Use two tournament selections (with replacement) with tournament size k to determine two ancestors $\vec{a}, \vec{b}$ for the new individual $\vec{x}_i$.
   b) Use $\vec{a}$ and $\vec{b}$ as the defining corners of a hypercube and randomly sample $\vec{x}_i$ within it.
   c) Resample the value of any of $\vec{x}_i$'s dimensions with probability P_{mut}.
4. Evaluate the new generation and jump to step 2 unless some halting condition is met.

The selection pressure of the algorithm is mainly controlled by the tournament size k. The variance of the population can either be controlled by means of the number R of randomly generated individuals or by the mutation probability P_{mut}. Thereby R is intended to explore new areas of the search space which is mainly interesting at the beginning of the optimization and for creating diverse training data. Whereas P_{mut} has been introduced for later optimization phases to escape local optima.

4.3 Applied regression & machine learning techniques

The methods explained in this section are considered as ensemble methods and follow the descriptions in Section 3.5. The base learner for all ensemble methods are neural networks trained with the Levenberg-Marquardt algorithm implementation of `ALGLIB`.

Error functions

The error measure for the used regression techniques is important for the training algorithm being able to adequately describe the underlying model of the training data. For ease of implementation the SSE function is used as a well known error function by by all of the implemented algorithms.

Although theoretical considerations suggest that the SSE values are not the main target of regression model optimization in the context of this thesis. This is according to the fact that the implemented optimization algorithms do not rely on an as-close-as-possible accuracy of the target values predicted by the models. As all of the used optimization algorithms use rank-based selection it is more important to predict the correct rank of some `Run` if compared to other `Runs`. That means instead of trimming the regression models to fit the target values, it could be more useful to train them to predict the total order of the evaluated `Runs` while neglecting the actual target values. Therefore a new error measure is proposed in Definition 2.

Definition 2. *The **Mean Total Order Deviation** (MTOD) of a regression model $\psi : X \to \mathbb{R}$ for an objective function $o : X \to \mathbb{R}$ and a set X with $x_i \in X, 1 \leq i \leq |X|$ is defined as follows.*

Let π_o be the permutation of all $x \in X$ s.t. $\forall 1 \leq i < j \leq |X| : o(x_{\pi_{o_i}}) \leq o(x_{\pi_{o_j}})$ holds. Let π_ψ be the permutation of all $x \in X$ s.t. $\forall 1 \leq i < j \leq |X| : \psi(x_{\pi_{\psi_i}}) \leq \psi(x_{\pi_{\psi_j}})$ holds. Further for any $a, b \in X$ s.t. $o(a) = o(b)$ with $\pi_o(a) < \pi_o(b)$, if $\psi(a) = \psi(b)$ holds, then $\pi_\psi(a) < \pi_\psi(b)$ has to hold i.e. if π_o is sorted to match π_ψ then the sort has to be stable. MTOD is then calculated as in Equation 4.12.

$$\mathrm{MTOD}(X) = \frac{1}{|X|^2} \sum_{i=1}^{|X|} |\pi_o(x_i) - \pi_\psi(x_i)| \tag{4.12}$$

The MTOD value of a set measures the mean percental deviation of its elements from their expected position in the total order induced by $o(.)$, to their position in the total order induced by $\psi(.)$. The main problem with the MTOD error function is its non-derivability. Therefore it is not possible to use the measure for regression model training with common numerical optimization techniques. As a consequence a surrogate error function should be used which is derivable and behaves similar to the MTOD. In Chapter 5 it is explored if the SSE is a suitable approximation of the MTOD for the given problem.

Parallelization and function call dispatching

As the major reason for using regression techniques is to achieve a speed up in the optimization process, it is important to make efficient use of the available computational resources. The computational power of the computation nodes for the simulation cannot be used for training regression models as other optimizations/simulations may require these resources.

Nevertheless parallelization is important to achieve an important practical speed up in the training process. The `IPython.parallel` module is used to connect user machines into a cluster, where new computational resource can be added or removed at any time. The ensemble methods described in the next sub sections employ `Dispatchers` for invoking training function calls. The concrete class of the `Dispatcher` determines how the passed function is invoked. Currently Yaop supports a `DefaultDispatcher` invoking the functions directly and an `IPythonDispatcher` delegating the function calls to the `IPython.parallel` cluster. The call, transmission of the parameters as well as receiving the return value is handled transparently by the `Dispatcher`.

Automatic (sub) model selection

Some of the ensemble methods require regression model selection steps during training. This need arises as for some (sub) prediction task it is not known which regression model will fit the underlying function best. If such a scenario is given then the ensemble methods typically provide two different (sub) model selection methods.

- **Training set based model selection** selects the regression model whose prediction error on the whole training set is the lowest. This method is fast as it only needs to compare the

errors of the regression models on training set but also expected to overfit the training set. This should result in a higher validation error.

- **Validation set based model selection** trains L equally parametrized regression models ψ_l on subsets T_l, $1 \leq l \leq L$ randomly sampled without replacement from the original training set T. The size of each T_l is determined by the validation percentage parameter p_{val} s.t. $|T_l| = (1 - p_{\text{val}})|T|$. The validation errors on the sets $V_l = T \backslash T_l$ are then averaged and the regression model whose parametrization lead to the lowest average validation error is then selected and trained on the original training set T. It is expected that this method leads to lower validation errors in general, but requiring a higher cost than the training set based model selection. Usually one would employ a crossvalidation model selection as it is done in Chapter 5 but its computational impact has been considered too high by the author.

Bagging

The `BaggingEnsemble` predictor implements the bootstrap aggregation described in Section 3.5. The parameters of the regression model are

- the base learner factory used to create learners for the bagging ensemble,
- the size of the ensemble K,
- and the bagging percentage p controlling the size of each training set.

The training sets T_k, $1 \leq k \leq K$ are sampled with replacement from the original training set T. The size of each T_k is given by $|T_k| = p \cdot |T|$. Each of the calculated training sets T_k is trained with its own base learner created by the base learner factory. As no dependencies exist between the created learners, it is possible to dispatch all training function invocations in parallel.

K-Means divide-and-conquer regression

A ensemble method not treated in Chapter 3 is the K-Means divide-and-conquer regression implemented by the `PartitioningLearner`. The method is based on a similar method given in [17] where a fuzzy C-Means clustering is used to partition the search space into non-disjunct subsets for the subsequent training of neural networks for classification.

The idea of this regression ensemble is to partition the training set T into K disjunct subsets T_k, $1 \leq k \leq K$ and to train a single base learner for each of these subsets. The partitioning is performed by the K-Means++ algorithm described in [3]. This clustering algorithm is intended to partition the search space s.t. similar input values belong to the same cluster. Further it is believed that by using training sets with similar input values, it is possible to use simpler base learners. As due to the nature of the HEV models, it is expected that the output values are similar too. A similar assumption is made in [17]. Training a model with less parameters provides a further speed up. Similar to the `BaggingEnsemble` it is possible to dispatch all training function invocations in parallel.

Another option during training is to overlap the the clusters with each other. Thereby the closest data points of each other cluster are added to the currently trained cluster. The size of the overlap is controlled by the overlap percentage parameter, which defines the number of data points taken from other clusters as a fraction of the size of the current cluster. The number of data points taken from an different cluster is indirectly proportional to the distance between the current cluster center and the other cluster center.

A main problem is how the base learners for each cluster are chosen. There is no simple answer to the question which base learner will produce the best results when used for prediction. Therefore the `PartitioningLearner` provides the ability to train multiple different base learners for each cluster and subsequently chooses the "best" one. But even if multiple learners are trained per cluster deciding which regression model is the "best" is open question as it could only be answered after using all the learners for prediction. To solve this problem the `PartitioningLearner` provides automatic model selection as described above.

For predicting a new inputs the `PartitioningLearner` provides two possibilities. The first method is to use the base predictor responsible for the closest cluster. The "closest" cluster in this context is considered the cluster with the smallest distance between the new input and the cluster's center. Although different definitions of "closeness" could be applied like single-linkage distance i.e. the smallest distance between the new input and any element of the cluster, or complete-linkage distance i.e. the the largest distance between the new input and any element of the cluster.

The second method uses the base learners of all clusters by combining their outputs indirectly proportional to the distance of the new input $\vec{x}$ to the cluster centers as given by Equation 4.13. Let C denote the set of cluster centers $\vec{c}_i$, $1 \leq i \leq |C|$ and $\phi_i(.)$ denote the base learner associated with cluster center $\vec{c}_i$. Further assume that $\forall i < j : ||\vec{x} - \vec{c}_i|| \leq ||\vec{x} - \vec{c}_j||$ holds i.e. that C is ordered by increasing distance.

$$\psi(\vec{x}) = \frac{1}{\sum_{i=1}^{|C|} ||\vec{x} - \vec{c}_i||} \sum_{i=1}^{|C|} \frac{\psi_i(\vec{x})}{||\vec{x} - \vec{c}_{|C|-i+1}||} \tag{4.13}$$

Least-Squares Gradient Boosting with finite learner domains

Gradient boosting as described in Section 3.5 is implemented by `LSGradientBooster` using a SSE function as error measure. In [16] the gradient boosting approach is described to select the best predictor from a class of predictors for the current residual prediction step. The selection is carried out only by using a training set T but no validation set. This setup is kept by the implementation.

The domain of the class of predictors is finite as the `LSGradientBooster` takes a set of `LearnerFactories` L as a parameter, which are used in sequence to create the tested base predictors for each iteration of the gradient boosting algorithm. The line search in step 3 of the boosting algorithm described in Section 3.5 is replaced by a direct determination of ρ_{i+1}.

This is done by treating the SSE function as a function of ρ only and by applying a modified linear least squares approach. This is possible as $f(\vec{x}_j, \vec{\beta}_{i+1})$ — with $x_j \in T$ and $\vec{beta} \in L$ — can be computed beforehand because both T and L are finite. The required values of $f(.)$ can

then be treated as constants. An optimum of the function $\mathrm{SSE}(T, \vec{\beta}) = \arg\min_{\rho_{i+1}} \sum_{j=1}^{|T|}(r_j - \rho_{i+1} f(\vec{x}_j, \vec{\beta}_{i+1}))$ can then be deduced — by setting its first derivate to zero — as follows. Let $f_j = f(\vec{x}_j, \vec{\beta})$.

$$\begin{aligned} \frac{\delta}{\delta\rho}\left(\sum_{j=1}^{|T|}(r_j - \rho f_j)^2\right) &= 0 \\ 2\sum_{j=1}^{|T|}(r_j - \rho f_j)(-f_j) &= 0 \\ -\sum_{j=1}^{|T|} r_j f_j + \sum_{j=1}^{|T|}\rho f_j^2 &= 0 \\ \rho &= \frac{\sum_{j=1}^{|T|} r_j f_j}{\sum_{j=1}^{|T|} f_j^2} \end{aligned} \tag{4.14}$$

It remains to be shown that the computed value of ρ is a minimum, by showing that the second-order derivation is larger than zero.

$$\begin{aligned} \frac{\delta}{\delta^2\rho}\left(\sum_{j=1}^{|T|}(r_j - \rho f_j)^2\right) &> 0 \\ \frac{\delta}{\delta\rho}\left(-\sum_{j=1}^{|T|} r_j f_j + \sum_{j=1}^{|T|}\rho f_j^2\right) &> 0 \\ \sum_{j=1}^{|T|} f_j^2 &> 0 \end{aligned} \tag{4.15}$$

Equation 4.15 shows that ρ is a minimum of the SSE function if at $f_j \neq 0$ holds for some j.

Gradient boosting has some drawbacks in terms of parallelization as only the base learners passed to the `LSGradientBooster` can be executed in parallel for a single iteration of the boosting algorithm. The iterative expansions have to be computed in series, therefore worsening the time needed for the ensemble training, regardless of the number of computation cores available.

The maximum number of expansions E is a parameter of the algorithm. Further the bagging percentage i.e. the percentage p of T used for training, can be specified too. If $p < 1$ holds then the algorithm becomes the stochastic variant described in [15]. The sampling of the training sets T_i for each expansion step $1 \leq i \leq E$ is performed without replacement (if $p < 1$).

Partial simulation and extrapolation

As mentioned in Section 4.2 it is possible to collect the output values of the simulated HEV models at periodic time steps. In the partial simulation and extrapolation regression scheme, a

unique learner for each output value of the HEV, relevant for the calculation of the objective function, is used.

It is expected that it is possible to increase the accuracy of the prediction by simulating the first p percent of the driving cycle and then passing the HEV model parameters as well as the collected output value at time step $p \cdot D$ — where D is the duration of the whole driving cycle — as inputs to a neural network for prediction. The information of the collected output value is thereby extrapolated from time step $p \cdot D$ to time step D. The additional information from time step $p \cdot D$ about the output value, which is to be predicted, is consequently also used as additional input during training.

Time-progressive-learning ensembles

Time-progressive-learning ensembles are an extension of the extrapolation idea of the previous section. Thereby the driving cycle is split into K time slices at the time steps t_i, $1 \leq i \leq K$ and a separate regression model is used for any of these time slices. The scheme further requires such a set of regression models for each output of the HEV model, which is relevant for the objective function.

For any output value v, a set of base learners is trained for each of the K time slices. A base learner for such a time slice is expected to model the behaviour of the HEV model in that part of the driving cycle. Any base learner ψ_i for time slice $1 < i \leq K$ takes the following inputs.

- The inputs $\vec{x}$ of the HEV model — like any of the other regression schemes,
- and the collected output value v_{i-1} at time step t_{i-1}.

These inputs are then used to train the base learner to predict the output value v_{t_i}. For any time slice the "best" predictor is selected by using automatic model selection as described above.

For prediction the input $\vec{x}$ is passed to ψ_1. The prediction $\Psi_1 = \psi_1(\vec{x})$ and $\vec{x}$ is then passed to ψ_2 and so on. This is repeated until the output of $\psi_K(\vec{x}, \Psi_{K-1}) = v_K$ has been computed. The output value is thereby predicted "progressively over time".

An option of the time-progressive-learning ensemble is to use not only the tuples $(\vec{x}, v_{i-1})$ as training data for $\psi_i(.)$, but also the predicted tuples $(\vec{x}, \Psi_{i-i})$. It is believed that by using the predicted tuples additionally as "jitter" in the training set — a common technique in neural network training — the function represented by the net can be smoothed to produce more accurate predictions. The drawback of this method is that the training time is increased as a larger training set is used and as all regression models of time slice i have to be computed before the training for time slice $i + 1$ can be started leading to a lower degree of parallelization.

CHAPTER 5

Experiments

5.1 Experimental setup

There are two different experimental setups used in this thesis. The first one is the setup of the optimization platform as well as the computation nodes for the optimization experiments. The Yaop software is installed on an VirtualBox™4.2 virtual machine on an Intel Core2 Quad and is dedicated 2 cores (Intel VT-x enabled) and 1456 MB of RAM. The virtual machine runs Fedora 17 and has the Yaop packages and GT Suite v7.2 installed. The "exact" computation strategy in use is the `RemoteGtSuiteComputationStrategy` unless noted otherwise. The GT Suite simulations are run on separate computation nodes with a Intel Xeon X5570 2.93GHz core and 12 GB RAM each. A maximum of 8 computation nodes is available, due to license restrictions. As a consequence all populations have been chosen to be a multiple of 8 to utilize the computational resources efficiently. Ideally a batch of 8 runs is executed at once and finishes at approximately the same time. Simplying it can be said that batches form a hardware-independent time measure[1]. The training of regression models during optimization is performed on user PCs in virtual machines (see below).

The second setup is concerned with the training of regression models for the evaluation of their performance, pre-optimization. The two setups are physically separated s.t. the computational power of the training setup cannot be used for training during optimization. The training is performed using the `IPythonDispatcher` with its `ipengines` deployed in virtual machines running Linux Mint[2]. Two virtual machines are deployed, one on an Intel Core i7-2620M with 4 dedicated cores (Intel VTx enabled) — one per `ipengine` — and another one on an Intel i5-21xx with 4 dedicated cores.

[1] In practice a new simulation is submitted as soon as a license is released. In most cases it is expected that this happens nearly at once for all previously submitted simulations.

[2] A different Linux distribution is used here due to network-related software problems in Fedora.

Problems with high computation times

The limited computational resources for optimization have large impact on the number of conducted tests. At most 10 different runs have been run per setup. For the control experiments without search space discretization it is even less. Although the low number of runs makes statistical explanations problematic, it is shown that the behaviour of the algorithms is consistent by using a qualitative approach.

5.2 HEV models for optimization

In the following the two HEV models under optimization are detailed and their parameters are explained.

Model A

The first model — Model A further on — has been developed by the first author of [30] and has the ability to act both as a series and a power-split hybrid. The vehicle relies on a single ICE and two electric machines — termed "generator" and "motor" further on – for propulsion. The ICE and the "generator" are directly coupled via a shaft, but may be detached by opening a clutch. Though the distribution of the required power between the machines is variable and is determined by the operation strategy. The ICE/generator unit and the "motor" are coupled to the driving shaft by a planetary gear set. Although the ICE/generator unit can be mechanically decoupled from the planetary gear set.

By using the above setup the following modes of operation are enabled.

- **EV1** is a pure-electric mode for low vehicle velocities where only the electric "motor" is used for propulsion. The ICE/generator unit is deactivated and detached from the planetary gear set.
- **EV2** is a pure-electric mode for high vehicle velocities where both electric machines are used for propulsion. The "generator" is attached to the gear set, but the ICE is deactivated and detached from the "generator".
- **ER1** is the first range-extension mode where the HEV is operated in a series hybrid setup. Therefore the ICE/generator unit is decoupled from the planetary gear set and is used to generate electrical power for the "motor" and for charging the battery. This range-extension mode is typically used for low velocities.
- **ER2** is the second range-extension mode where the HEV is operated in a power-split setup. All clutches are closed and all machines are active and thereby the power output of the ICE is split into a mechanical and electrical path. This mode is typically used for high velocities.

The selection of the proper mode of operation is determined by the vehicle's operation strategy, which is pre-determined for the HEV. The behaviour of the strategy is parametrized by the parameters explained in the next section.

Parameters

- **ringteeth** determines the number of teeth for the ring gear of the planetary gear set. The domain of the parameter is integer. Further the parameter has been limited to the values $\{70, 80, 90, 100, 110, 120, 130\}$ as using a finer grained domain is considered redundant.

- **sunteeth** determines the number of teeth for the sun gear of the planetary gear set. The domain of the parameter is restricted to the values $\{20, 30, 40, 50, 60\}$. Let r be the number of ring teeth and s be the number of sun teeth. Then the following constraints have to hold s.t. **ringteeth** and **sunteeth** form a valid parameter tuple.

  - $(r - s) \bmod 2 = 0$
  - $r > s$

  The both constraints are enforced by the domains chosen for the parameters.

- **generatorpowermin** determines the minimal power output of the electric "generator". The domain of the parameter is integer and limited to $[3, 20]$.

- **torqueev1up** $\in [200, 800]$ determines the torque on the driving shaft above which no switching from mode EV1 to EV2 is performed. The domain of the parameter is integer as fractional values are not considered significant.

- **speedev1up** $\in [60, 140]$ determines the vehicle speed required to switch from mode EV1 to EV2. The domain of the parameter is integer as fractional values are not considered significant.

- **torqueer1up** $\in [200, 1000]$ determines the axle torque above which no switching from mode ER1 to ER2 is performed. The domain of the parameter is integer as fractional values are not considered significant.

- **speeder1up** $\in [40, 140]$ determines the vehicle speed required to switch from mode ER1 to ER2. The domain of the parameter is integer as fractional values are not considered significant.

- **speedermin** $\in [0, 80]$ determines the minimal vehicle speed required to switch from a pure-electric mode to ER1. The domain of the parameter is integer as fractional values are not considered significant. Further its value is required to be less than or equal to the value of **speeder1up**.

- **socband** $\in [0.01, 0.1]$ determines the allowed percental deviation of the battery's initial SOC — in both directions. Low values lead to higher rates of starting/stopping the ICE as it is tried to keep the SOC nearly constant. Higher values lead to may lead to larger deviations at the end of the simulated driving cycle and to a penalization of the solution in sequence (as described in the next section). Further the **socband** determines when the switch from pure-electric to range-extension modes is performed.

- **chargepowerhigh** $\in [20, 100]$ is the maximum power used to charge the battery. This amount of power has to be generated if the SOC is far lower than the initial SOC.

- **chargepowerlow** $\in [0, 30]$ is the minimum power used to charge the battery. This amount of power has to be generated if the SOC is higher or only a bit lower than the initial SOC. The domain is integer and the parameter's value has to be lower than **chargepowerhigh**. The actual power generated by the "generator" is determined by interpolating between **chargepowerlow** and **chargepowerhigh**, depending on the SOC's level in comparison to the initial SOC.

- **usepowerdemandcharge** $\in \{0, 1\}$ is boolean and indicates if the "generator" should try to cover the current power requirement of the electric "motor".

Objective function

Model A is evaluated on the EPA US06 driving cycle, as it is short driving cycle and as it is designed to resemble "real" driving behaviour. As objective function the single case objective function — as described in Section 4.2 — is used with a single target value. The **fuelcons** value measures the fuel consumption in L/100km on the driving cycle plus a penalization for SOC deviation. The penalization for a negative deviating SOC is calculated by estimating how much fuel would be needed to charge the battery to its original state using the ICE/generator combo at stop. It is possible that the penalization is negative if the SOC deviation is positive. The lower bound

Model B (IFAHEV)

Model B is a version of the IFAHEV model in [31], but a larger set of parameters has been chosen for the optimization. The operation strategy for Model B is specifically designed for the NEDC as described in Section 2.2. As this model is protected by an NDA it is not allowed to detail the achieved results as well as the model itself. Therefore the model serves mainly for verification for some aspects of the evaluated methods.

Parameters

- **GearRPM$_\mathbf{i}$**, $i \in \{2, 3, 4, 5, 6, 7\}$ determines the RPM whiche the ICE needs to reach to switch from gear $i - 1$ to gear i.

- **LPS$_\mathbf{x}$**, $x \in \{\text{ECE-15}, \text{EUDC}\}$ determines the percental load-point shifting during the constant speed phases of the ECE-15 and EUDC parts of the NEDC respectively.

- **LPS$_\mathbf{i}$**, $i \in \{2, 3, 4, 5\}$ determines the percental load-point shifting during different acceleration phases. LPS_2 and LPS_3 control the load-point shifting during the second and third acceleration phase in the ECE-15 parts of the NEDC. LPS_4 controls the load-point shifting during the first two acceleration phases in the EUDC part of the NEDC and LPS_5 in the last two acceleration phases of the EUDC part.

- **EMVehicleSpeed$_x$**, $x \in \{\text{ECE-15}, \text{EUDC}\}$ determines the vehicle speed up to which acceleration is handled by the electric machine solely. The ECE-15 and EUDC variants control this parameter for the respective phases of the driving cycle.

Objective function

As fitness function the single case objective function with two parameters has been used. The first parameter is the fuel consumption and the second parameter is the SOC deviation. The SOC deviation has been modified s.t. a deviation less than one percent does not contribute to the fitness function. The weights of the parameters have been split 70/30. The estimated lower/upper bounds chosen for the parameters, as well as the exact calculation of the fuel consumption with SI units, cannot be given. Doing so would allow to reverse the fitness calculation giving insight in protected data.

5.3 Experiments without discretization

The first set of experiments has been performed with the constraints of the parameters enabled, but without discretization of the search space. Further only exact evaluation of the HEV models is used using the `RemoteGtSuiteComputationStrategy` with 8 computation nodes at maximum. The number evaluations per algorithm iteration has been set to be a multiple of 8 to maximize the load on the computation nodes if all computation nodes are available to the simulation. For each model the empirical distribution of the measured simulation times is given first. This time measures will be used throughout the chapter to determine the idealized duration of the optimization experiments. Otherwise the real measured optimization times could be very different, as for a large percentage of the time the computation nodes have been used for other tasks too.

Results for Model A

From the experiments conducted for Model A, the simulation time measurements have been collected. Overall 16892 simulations have been run during the experiments without discretization. The collected data is depicted in the box plot in Figure 5.1. The different simulation times arise as the simulation software computes the driving cycles with variable time steps i.e. one second of the driving cycle might be simulated by a varying number of computation steps. Typical simulation times range from 6 to 10 minutes. The mean simulation time is 7.49 minutes. This value will be used to compute the computation times for Model A further on.

PSO without discretization

The first set of experiments have been conducted with a PSO without discretization and with the parameters set to the values in Table 5.1. The values for ω, ϕ_1, and ϕ_2 are the same as the values for the Canonical PSO in [37].

Figure 5.2 shows the best fuel consumption values for each iteration for the conducted experiments. Although the number of experiments is very low, the plots show a similar progression

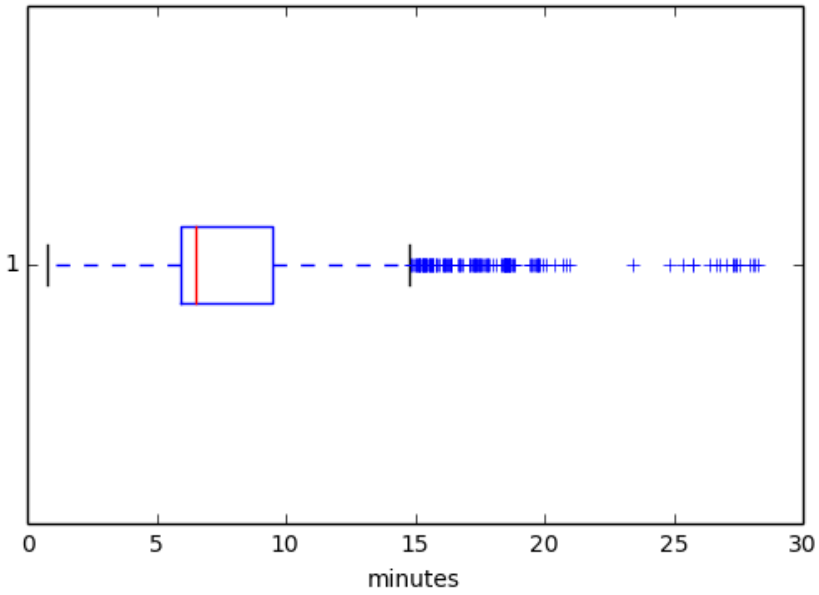


Figure 5.1: A box plot of the empirical distribution of the simulation times for Model A. Generated from 16892 simulations.

#Iterations	#Particles	ω	ϕ_1	ϕ_2	#Non-impr. iterations before reset
200	16	0.7298	1.49618	1.49618	20

Table 5.1: Model A/B: Parameters for PSO without discretization

for the fuel consumption of the found solutions. The best fuel consumption found in these experiments is about 5.83L/100km. All experiments found solutions below 6.0L/100km.

The first drop below a value of 5.9L/100km occurred for all experiments approximately after 100 iterations. An analysis of the cache hits showed that over all PSO experiments only 12 cache hits occurred. As the simulations are — ideally — executed in batches of 8 and finish at the same time, the time spent in the simulation software is 2.08 days per experiment, with the hardware mentioned above. This is equal to 400 simulated batches.

Active CMA-ES without discretization

Next a set of experiments with the Active CMA-ES has been run. Again three experiments have been conducted for with the settings given in Table 5.2. The values for β, c_{cov}, c_c, and c_σ are estimated as given in [24].

Figure 5.2 depicts the best fuel consumption values for each iteration for the conducted Active CMA-ES experiments. It can easily been seen that the performance of the Active CMA-ES is significantly worse than the performance of the PSO. The best parameter sets found have

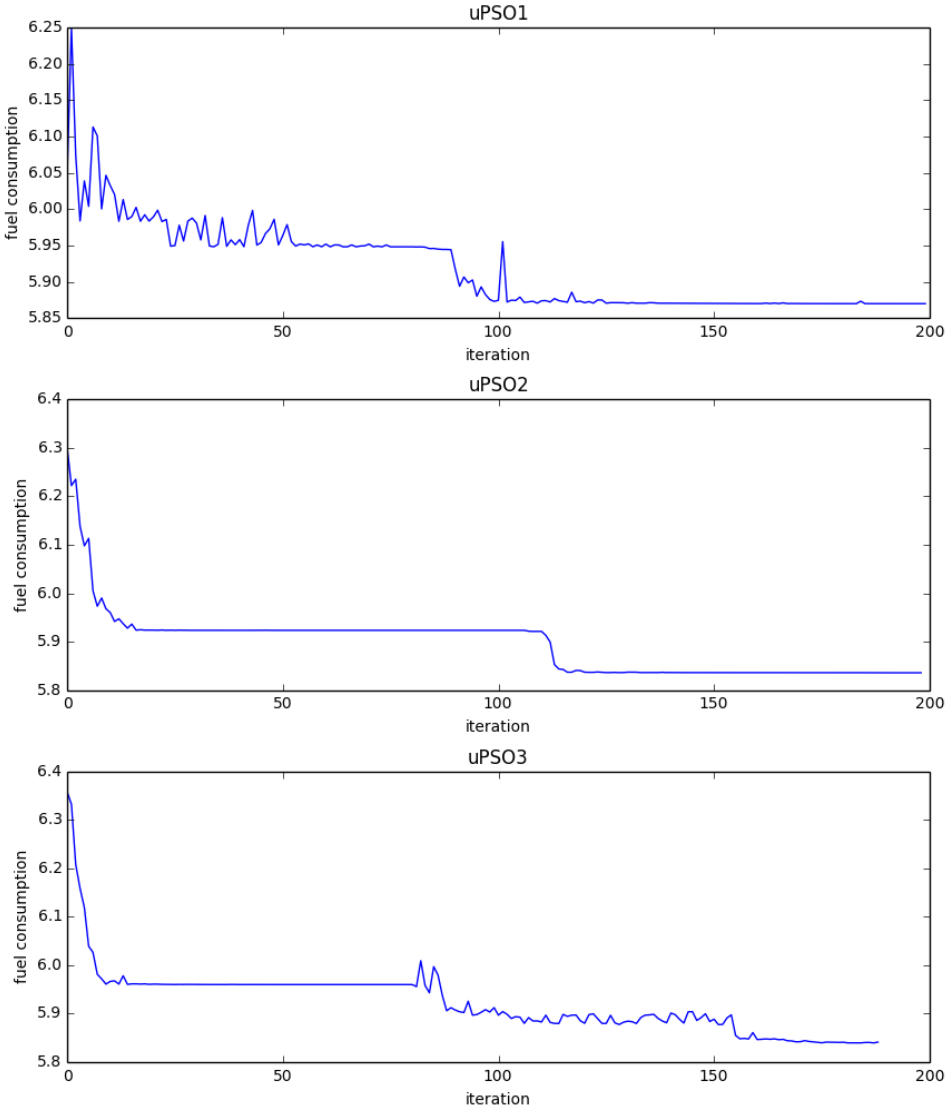


Figure 5.2: Model A: A plot of the best fuel consumption values per iteration for three PSO runs without discretization.

a fuel consumption of 6.00L/100km which have been achieved by all experiments.

The question arises why the Active CMA-ES performs worse than the PSO. It may be an-

#Iterations	μ	λ
200	4	16

Table 5.2: Model A/B: Parameters for Active CMA-ES without discretization

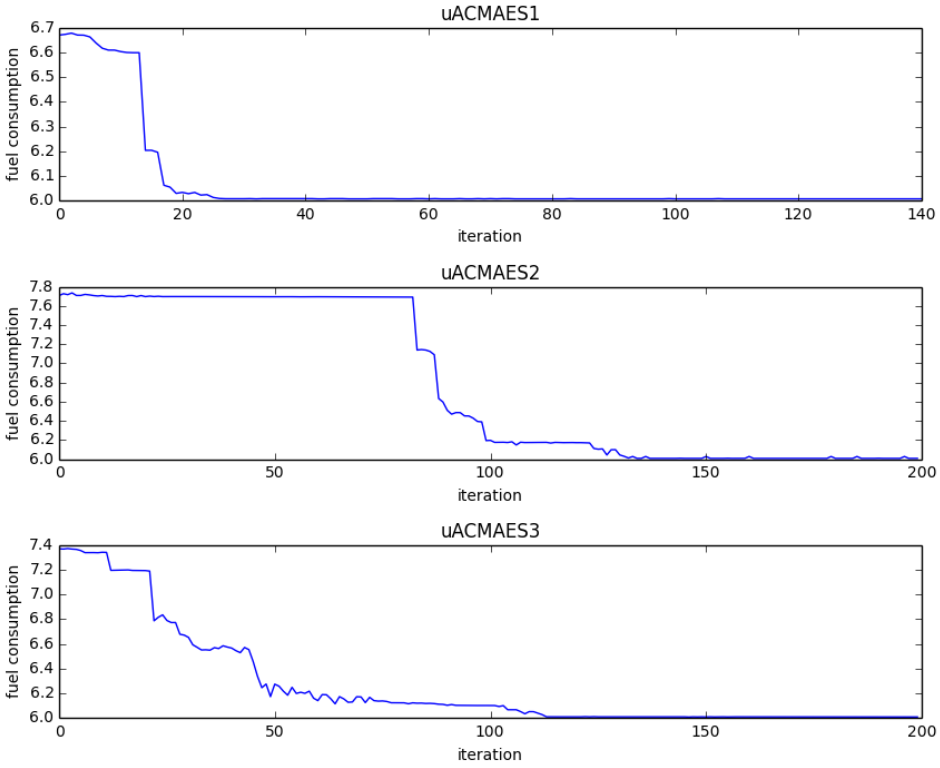


Figure 5.3: Model A: A plot of the best fuel consumption values per iteration for three Active CMA-ES runs without discretization.

swered by comparing the variance of the solutions' fuel consumptions per iteration. Therefore these values are shown in the box plot in Figure 5.4 for the third PSO experiment and the first Active CMA-ES experiment. These two experiments have been chosen as they show a similar plateau of the fuel consumption value at the same time. It is assumed that for the HEV model at hand, similar fuel consumption values correlate with plateaus in the search space. This assumption is supported by the fact that the solutions for Active CMA-ES are sampled around a single centroid. Therefore it is assumed that if the fuel consumption variance is low, then the variance of the solutions' parameters is low too.

The assumption above is further borne out if iteration 84 of the depicted PSO run is considered. Analysis of the PSO's behaviour shows that after iteration 75 to iteration 83, several of the

particles got resampled by the mechanism described in Section 4.2. This leads to an increase of the solutions' variance. The Active CMA-ES lacks a similar feature and is therefore unable to escape the local plateau.

Although the variance of the solutions of the Active CMA-ES is lower than for the PSO, only 25 database cache hits occurred overall experiments without discretization. The number of batches simulated per experiment is with 400 the same as for the PSO.

Results for Model B

The simulation times for Model B have been collected and evaluated similarly to Model A. The box plot in Figure 5.5 shows the distribution of the time measurements. The mean simulation time is 8.65 minutes and has been calculated from 15769 simulations. It is further used to determine the computation times of the experiments.

PSO without discretization

The PSO has been configured with the same parameter values as for Model A, which are given in Table 5.1. As the actual fuel consumption values might not be published, the fitness values for the evaluated solutions are given, where higher fitness values indicate better solutions. The fitness values of the best solution in each iteration are shown in Figure 5.6. The best solution found by the PSO experiments has a fitness value of -44.74. For all PSO experiments the best solutions have been found during the last iterations of the optimization.

No cache hits occurred during the PSO experiments. Consequently 400 batches have been simulated per experiment equalling a computation time of 2.4 days.

Active CMA-ES without discretization

The experiments with the Active CMA-ES have been conducted with the parameter settings given in Table 5.2. Again the results of the Active CMA-ES have been worse than the PSO results for each experiment. The best solution found has a fitness value of -54.71. The best fitness values for each iteration are depicted in Figure 5.7. The Active CMA-ES exhibited similar behaviour as in the experiments for Model A. Under the same assumption as for Model A that similar fitness values correlate with similar regions in the search space, the algorithm shows little variance in the generated solutions.

5.4 Two-phased optimization I

The second set of experiments deals with incorporating search space discretization and regression techniques into the optimization process. To do this the optimization is split into two phases. In the first phase it is important to aggregate a diverse set of solutions which act as training data for the used regression models. Nevertheless, the search space cannot be explored fully due to its size. It is important not to waste too much computation time on search space areas which are unlikely to be visited further in the optimization process, while still being able to classify these

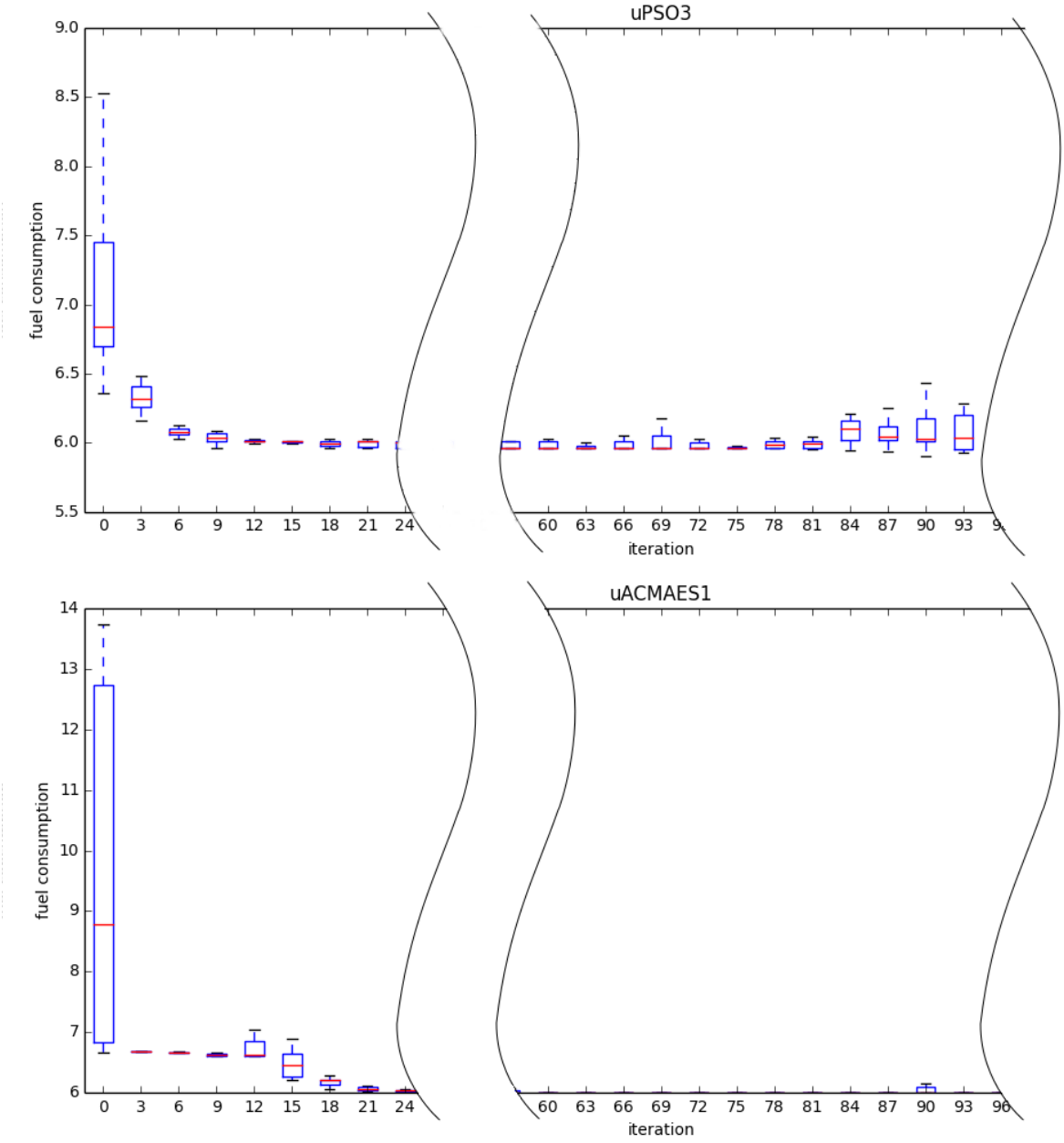


Figure 5.4: Model A: Comparison of the fuel consumption variance per iteration between a PSO and an Active CMA-ES run.

areas correctly. Therefore a tradeoff has to be found between solution diversity and solution quality.

During this phase the search space discretization is configured to be coarse-grained. The number of discretization steps is kept at a constant level throughout this phase. The number of

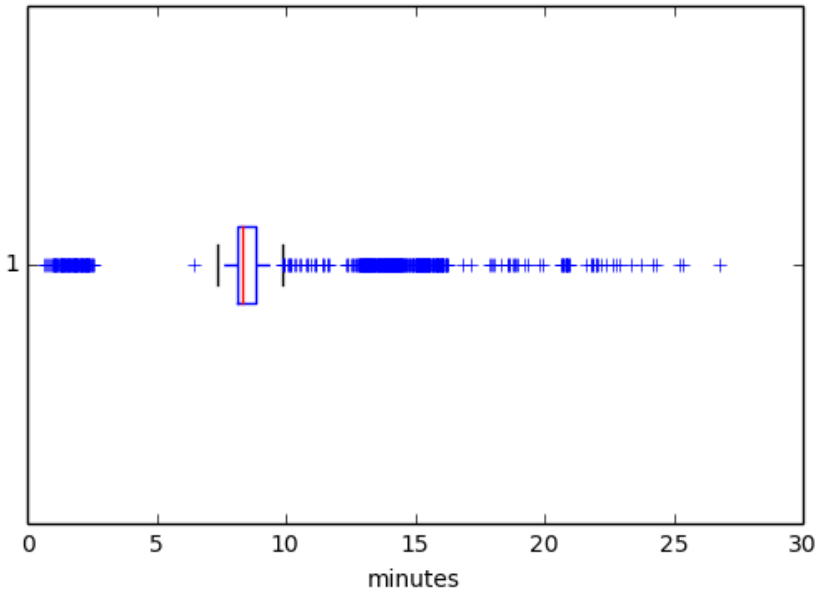


Figure 5.5: A box plot of the empirical distribution of the simulation times for Model B. Generated from 16892 simulations.

discretization points have been set to 6 for each dimension, except those discretized beforehand[3]. Further for all experiments in the first phase the number of iterations has been reduced to 65.

Experiments with the Active CMA-ES have not been conducted because of the low variance in the solutions generated in each iteration. It is expected that the Active CMA-ES would generate many duplicate solutions due to the introduced discretization. As explained below even the PSO has problems in generating new solutions in a discretized setup, which supports the decision above as the variance of the solutions generated by the Active CMA-ES is lower in general.

Phase I results for Model A

PSO with discretization

For the PSO experiments with discretization the parameter settings from Table 5.3 have been used. The values for ϕ_1 and ϕ_2 have been increased as early experiments with the PSO showed that for values less than 2 the algorithm prematurely became stuck somewhere in the search space. The reason for this behaviour is that the minimal velocity required to perform a jump in the search space[4] could not be maintained. The increase of ϕ_1 and ϕ_2 lessened the occurrence

[3] This concerns only the following parameters of Model A: **sunteeth**, **ringteeth**, and **usepowerdemandcharge**
[4] See Section 4.2 and Equation 4.7)

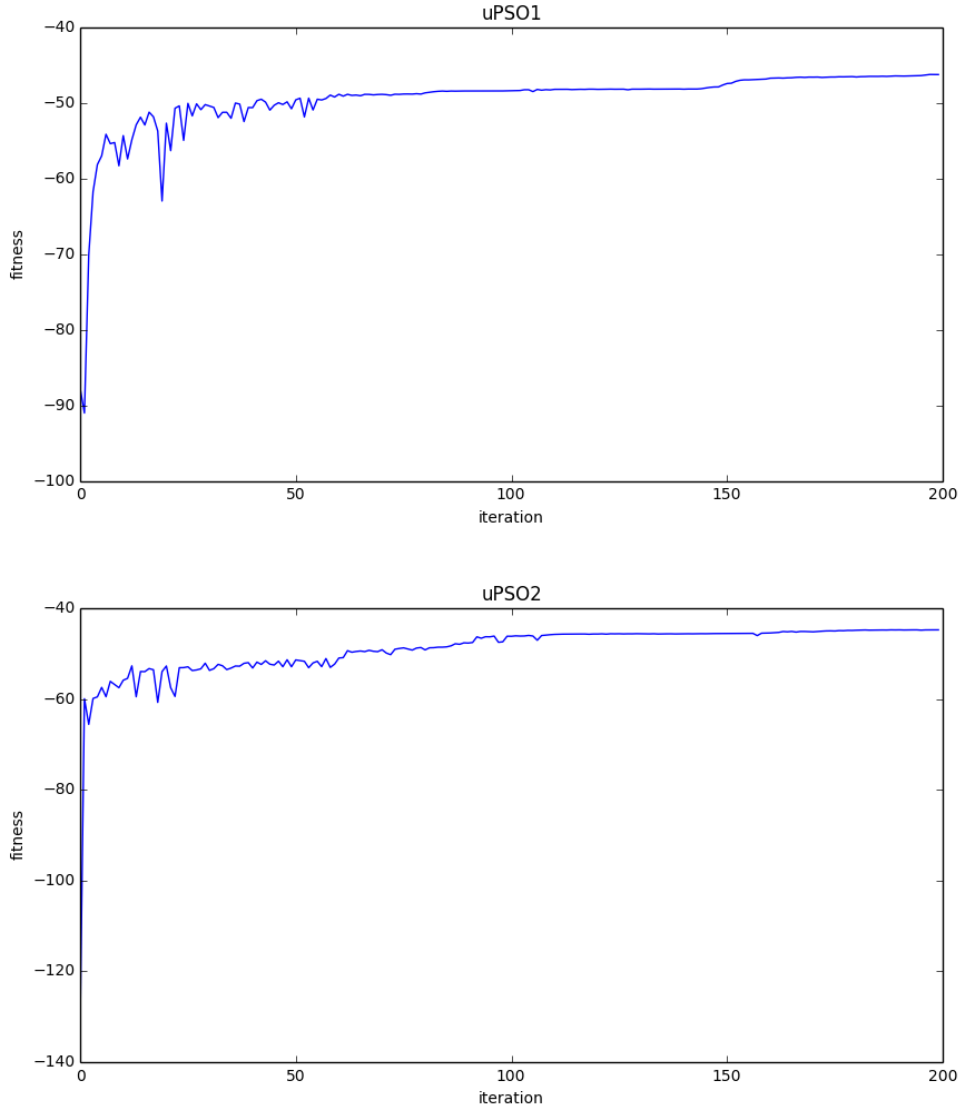


Figure 5.6: Model B: A plot of the best fuel consumption values per iteration for two PSO runs without discretization.

of this problem, but has not been able to mitigate it completely. Trying to increase the value of ω lead to erratic jumps through the search space. Therefore its value has not been modified.

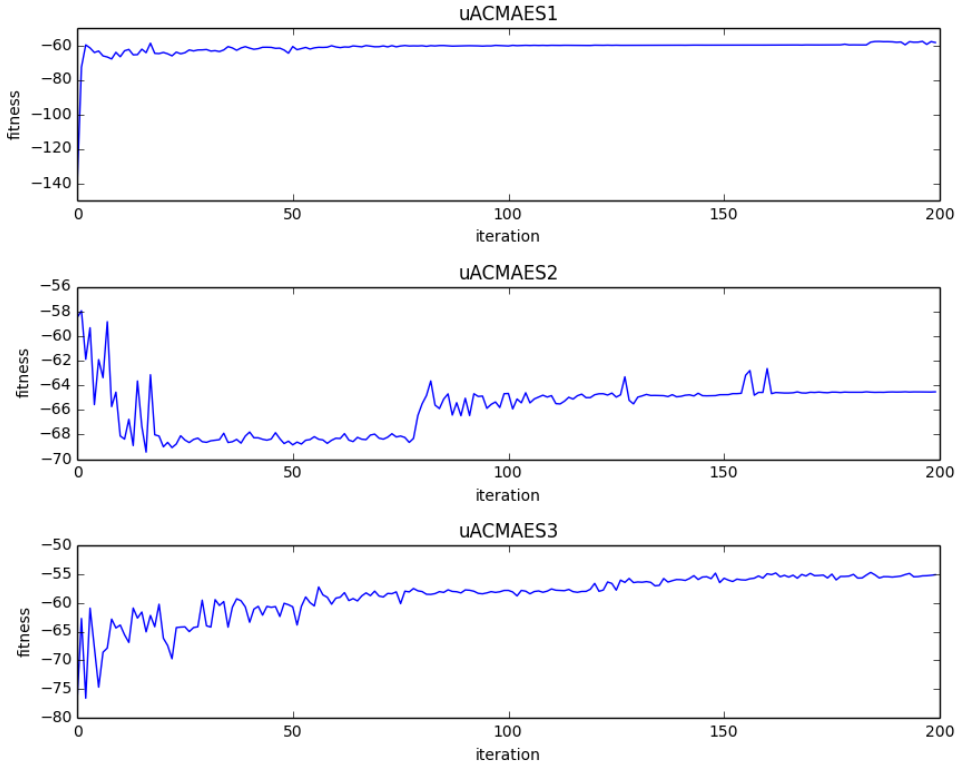


Figure 5.7: Model B: A plot of the best fuel consumption values per iteration for three Active CMA-ES runs without discretization.

#Iterations	#Particles	ω	ϕ_1	ϕ_2	#Non-impr. iterations before reset
65	16	0.7298	2.0	2.0	20

Table 5.3: Model A/B: Parameters for PSO with discretization

For the setup described above 10 experiments have been run. The comprehensive results are shown in Table 5.4. It can be seen that most of the time the PSO is able to find solutions below 6.0L/100km, thereby beating the Active CMA-ES without discretization. Nevertheless the solutions are still worse than results of the PSO experiments without discretization.

Up to iteration 65 the experiments without discretization would have computed 130 batches, requiring 16.22 hours on the given hardware. The analysis of the results shows that the PSO with discretization computes far less batches, due to a higher number of cache hits. Basic statistical measures for the number of computed batches are given in Table 5.5.

#Experiment	Iteration	Best solution (L/100km)
1	22	5.95
2	25	6.11
3	3	6.02
4	7	5.95
5	14	5.97
6	32	5.96
7	11	5.93
8	18	6.07
9	39	5.96
10	18	5.91

Mean: 5.98 L/100km Std. dev.: 0.06

Table 5.4: Model A: Results for the PSO in phase I

	Min. #batches	Max. #batches	Mean	Standard deviation
PSO	32	107	67.1	25.99
RRS	104	130	120.4	7.57

Table 5.5: Model A: Statistical measures about the computed batches in phase I

Recombinant Random Sampling with discretization

The biggest benefit of the PSO with discretization — good results with fewer computations — is also one of its biggest drawbacks if a regression model has to be trained. The solutions computed in the first phase of the optimization will be used as training data for the regression models. The number of training samples is an important factor for the performance of these models.

Therefore the Recombinant Random Sampling (RRS) algorithm described in Section 4.2 has been evaluated on the model, with discretization enabled. The configuration of the algorithm is given in Table 5.6. The tournament size k has been set to a low value as early experiments showed that using a larger value lead to strong elitism and converged too fast to a single points in the search space. The number of randomly generated solutions R has been set rather high to increase the probability to generate a diverse set of training data. The mutation for recombined individuals has been disabled to favor randomly generated solutions as source of entropy.

#Iterations	#Individuals	R	k	P_{mut}
65	16	5	2.0	0.0

Table 5.6: Model A/B: Parameters for Recombinant Random Sampling with discretization

Table 5.7 shows the results of 10 conducted experiments. It can be seen that the RRS generates not as good solutions as the discretized PSO, which might be due to the high randomness present in the algorithm. Nevertheless the RRS outperforms the Active CMA-ES algorithm.

#Experiment	Iteration	Best solution (L/100km)
1	9	6.01
2	36	5.98
3	8	6.04
4	57	6.04
5	59	6.00
6	62	6.06
7	56	5.99
8	50	6.02
9	40	6.00
10	27	6.04

Mean: 6.02 L/100km Std. dev.: 0.02

Table 5.7: Model A: Results for the RRS in phase I

As expected the number of evaluated batches could be increased, thereby generating more training data at the cost of longer computation times. The corresponding data is shown in Table 5.5.

Phase I results for Model B

PSO with discretization

For Model B 10 experiments with the discretized PSO variant have been conducted. The parameters of the PSO are given in Table 5.3. The results of the experiments are collected in Table 5.8. The PSO with discretization has not found solutions as good as the unconstrained PSO, but still outperformed the Active CMA-ES.

#Experiment	Iteration	Best solution
1	65	−51.61
2	65	−47.21
3	65	−53.50
4	65	−55.05
5	65	−47.99
6	65	−51.34
7	64	−54.98
8	64	−54.48
9	65	−50.56
10	65	−52.71

Mean: −51.94 Std. dev.: 2.62

Table 5.8: Model B: Results for the PSO in phase I

The best solutions found by the algorithm have been generated in the very last iterations of each experiment. It seems plausible that the PSO requires more iterations to converge to better solutions. This assumption is supported by considering the number of evaluated batches in Table 5.9. These are substantially higher than for Model A and indicate that new areas of the search space are still explored.

	Min. #batches	Max. #batches	Mean	Standard deviation
PSO	107	130	121.30	8.55
RRS	113	130	123.60	6.54

Table 5.9: Model B with discretization: Statistical measures about the computed batches in phase I

Recombinant Random Sampling with discretization

The experiments run with RRS for Model B follow the same settings as for Model A and are given in Table 5.6. The results of the experiments show that for Model B the RRS performs considerably better than the PSO in phase I. The number of computed batches is similar to those for Model A (see Table 5.9).

#Experiment	Iteration	Best solution
1	46	-51.16
2	19	-47.62
3	33	-47.53
4	21	-47.95
5	65	-47.69
6	53	-49.20
7	26	-46.91
8	59	-47.81
9	49	-48.50
10	7	-47.68

Mean: -48.20 Std. dev.: 1.14

Table 5.10: Model B: Results for the RRS in phase I

5.5 Evaluation of regression techniques

The next step in the phased optimization approach is the training of the regression model used in phase 2. The regression models use the same HEV model parameters as input, which the optimization algorithms are allowed to adjust during optimization. If not mentioned otherwise the target value of the regression models is the output of the objective function of the optimization.

In this intermediate step it has to be decided how the regression model shall be parametrized. For this cause different regression models are evaluated using cross-validation. The setup of the experiments is the following.

An exemplary RRS experiment of phase I is selected for Model A and Model B. The solutions of these experiments are then used as training data for the regression techniques. For both HEV models an experiment with a larger set generated solutions has been chosen to provide enough data for the training process. The training set for Model A contain 905 solutions and the set for Model B contains 863 solutions. Before cross-validation is applied to the training data, the data set is shuffled. Otherwise the order of the solutions — as generated by the optimization algorithm — would bias the folds to contain rather similar data. This could worsen the prediction performance on the validation fold. Further on all experiments use 10-fold cross-validation.

In the tables in the following sections, some acronyms and terms reappear several times. Their meaning is described in the following.

- Time ... the required computation time
- tMSE ... average Mean Square Error on the training set over all cross-validation folds
- vMSE ... average Mean Square Error on the validation set over all c.-v. folds
- tMTOD ... average Mean Total Order Deviation on the training set over all c.-v. folds
- vMTOD ... average Mean Total Order Deviation on the validation set over all c.-v. folds

Further all tables containing results are sorted by their vMTOD value.

Neural network experiments

The first regression model which has been explored are neural networks. These use the Levenberg-Marquardt training algorithm as implemented by `ALGLIB`. The activation function of neurons contained in the hidden layers is sigmoidal and the activation function of the output layer neurons is linear. The input normalization is directly handled by the used neural network library. This setups also holds true for the base learners of the following ensemble method experiments.

Different network architectures with up to two hidden layers have been evaluated. For the networks with a single hidden layer the number of neurons have been chosen to be a multiple of the input layer. A special case of the neural networks are those without hidden layer as they represent a linear regression. Further for each architecture different values for the weight decay parameter of the neural network have been chosen. According to the documentation of the neural network library, Tikhonov regularization is used as weight decay term in the error function. This method penalizes the use of large weights in the network, which consequently reduces the variance of the network output.

The results of the neural network experiments for Model A are given in Table 5.11 and the results for Model B in Table 5.12. Both tables are sorted by their average MTOD value — as discribed in Section 4.3 — of the validation sets over all folds. It can be seen that sorting the results by their validation MSE — the mean of the SSE error function — would result in

Layers	Decay	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
12,12,1	1.0	6.45	1.37	6.25	0.0746	0.0952
12,24,24,1	1.0	332.65	0.04	6.75	0.0173	0.0959
12,12,1	0.001	8.26	1.52	7.13	0.0752	0.0993
12,24,1	1.0	22.78	1.30	7.46	0.0612	0.1004
12,12,12,1	1.0	21.02	0.94	8.85	0.0640	0.1042
12,72,1	1.0	338.83	0.06	8.20	0.0203	0.1060
12,12,1	0.1	5.91	2.04	6.45	0.0796	0.1069
12,48,1	1.0	156.09	0.14	7.34	0.0300	0.1072
12,24,1	0.1	31.63	0.39	8.55	0.0511	0.1089
12,12,12,1	0.1	25.72	0.73	8.49	0.0644	0.1103
12,24,24,1	0.1	302.42	0.05	10.15	0.0139	0.1104
12,12,12,1	0.001	26.05	0.93	8.61	0.0691	0.1106
12,72,1	0.1	118.34	0.01	8.86	0.0069	0.1112
12,24,1	0.001	40.23	0.36	9.51	0.0479	0.1113
12,48,1	0.1	140.66	0.04	10.03	0.0166	0.1178
12,24,24,1	0.001	165.66	0.03	13.38	0.0138	0.1240
12,48,1	0.001	152.13	0.02	11.80	0.0131	0.1281
12,72,1	0.001	182.50	0.00	12.93	0.0008	0.1308
12,1	1.0	0.10	15.08	15.76	0.1502	0.1512
12,1	0.1	0.12	15.08	15.76	0.1502	0.1512
12,1	0.001	0.27	15.08	15.76	0.1502	0.1512

Legend
Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm

Table 5.11: Model A: Neural network results

a different order. As both orderings are only marginally different, the MSE may be used as a derivable approximation of the MTOD.

It is interesting to see that for both HEV models very different network architectures are rated best. Therefore it seems unavoidable to evaluate a large set of neural networks for each new HEV model, to deduce proper regression models.

Least-Squares Gradient Boosting

Gradient Boosting experiments have been conducted for Model A with two different sets of base learner candidates.

- The **weak** learner set contains a linear regression model and networks with a single hidden layer with either 3 or 6 neurons.
- The **medium** set contains a linear regression model and networks with a single hidden layer with 6, 12, or 24 neurons.

Layers	Decay	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
14,84,1	1.0	163.14	0.30	804.89	0.0034	0.0674
14,84,1	0.1	160.48	0.01	892.20	0.0010	0.0794
14,14,1	0.1	4.72	832.70	1762.95	0.0605	0.0888
14,14,1	0.001	4.80	974.58	1900.66	0.0626	0.0891
14,84,1	0.001	128.41	0.02	1281.89	0.0002	0.0929
14,56,1	1.0	55.14	12.16	1284.58	0.0130	0.0939
14,28,28,1	1.0	104.06	286.41	1819.73	0.0587	0.0952
14,28,1	0.1	14.02	288.42	1812.49	0.0416	0.0952
14,14,1	1.0	4.04	1044.52	2132.58	0.0644	0.0953
14,56,1	0.1	59.56	20.98	1498.88	0.0115	0.0956
14,28,1	1.0	14.41	324.84	1731.42	0.0398	0.0959
14,28,28,1	0.1	104.23	303.18	2121.63	0.0575	0.0996
14,28,28,1	0.001	98.23	400.33	2204.43	0.0605	0.1005
14,56,1	0.001	65.70	14.16	1811.86	0.0097	0.1029
14,28,1	0.001	14.85	350.42	1885.74	0.0417	0.1032
14,14,14,1	0.1	15.15	1112.13	2105.18	0.0893	0.1050
14,1	1.0	0.13	1646.20	1741.05	0.1035	0.1068
14,1	0.1	0.13	1646.15	1741.15	0.1035	0.1068
14,1	0.001	0.19	1646.15	1741.17	0.1035	0.1068
14,14,14,1	0.001	15.11	1572.44	2689.04	0.1073	0.1167
14,14,14,1	1.0	15.31	1356.03	2618.17	0.0960	0.1259

Legend Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm

Table 5.12: Model B: Neural network results

Both learner sets are evaluated with stochastic gradient boosting and simple gradient boosting. For the stochastic variant the sampled training sets are 0.5 and 0.75 times the size of the original training set. The number of expansion steps is fixed to 16. The results of the experiments are given in Table 5.13

Examination of the data shows that the training errors for the experiments without stochastic boosting are lower than the training errors of their base learners (as given in the previous section). This implies that boosting is able to reduce the training error as expected. This is detailed in Table 5.14 where the progression of the error values is shown for each expansion step for an exemplary experiment.

The regression method behaves differently if the stochastic variant is used. The performance worsens as the used training sets grow smaller and expansion steps tend to increase the MSE on both the training and the validation set. Another observation is that the gradient boosting approach seems to overfit the training data thereby resulting in high validation errors.

Due to the unpromising results for Model A the gradient boosting experiments for Model B have been omitted.

Learner set	Training set size	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
medium	1.0	183.88	0.16	5.49	0.0330	0.0984
medium	0.75	161.49	1.00	7.89	0.0639	0.1122
weak	1.0	30.75	1.49	7.18	0.0789	0.1213
weak	0.75	24.94	5.70	12.36	0.1142	0.1538
medium	0.5	137.97	8.86	17.36	0.1304	0.1636
weak	0.5	20.21	23.24	34.68	0.1710	0.1854

Legend
Learner set — the set of used base learner candidates
Training set size — the factor determining the size of the sampled training sets

Table 5.13: Model A: LS Gradient Boosting results

#Expansion	ρ	Layers	tMSE	vMSE	tMTOD	vMTOD
0	1.0217	12,24,1	0.7219	5.9861	0.0625	0.0894
1	1.3739	12,24,1	0.4181	6.2430	0.0522	0.0869
2	1.0839	12,12,1	0.3035	6.8463	0.0471	0.0872
3	1.1446	12,12,1	0.2396	6.9097	0.0415	0.0908
4	1.2013	12,12,1	0.2025	6.8802	0.0388	0.0937
5	1.0397	12,6,1	0.1616	6.7969	0.0344	0.0952
6	1.2797	12,12,1	0.1576	6.8794	0.0345	0.0964
7	1.0159	12,6,1	0.1555	6.9151	0.0338	0.0966
8	0.2521	12,6,1	0.1544	6.9226	0.0335	0.0966
9	1.0061	12,24,1	0.1544	6.9226	0.0335	0.0966
10	1.0061	12,24,1	0.1544	6.9226	0.0335	0.0966
11	1.0061	12,24,1	0.1544	6.9225	0.0335	0.0966
12	1.0061	12,24,1	0.1544	6.9225	0.0335	0.0966
13	1.0201	12,6,1	0.1518	7.1598	0.0330	0.0983
14	1.0061	12,24,1	0.1518	7.1598	0.0330	0.0983
15	1.0061	12,24,1	0.1518	7.1597	0.0330	0.0983

Legend
#Expansion — index counting the number of neural networs trained on residuals
ρ — the weight for computed for the added base learner
Layers — the number of neurons per layer

Table 5.14: Model A: Example for the expansion steps of a **medium** learner set

Bagging

The Bagging experiments have been conducted for both HEV models with the neural network architectures already evaluated in the neural network experiments. The number of learners per ensemble has been fixed to 24. Different sample sizes for the used training sets have been used too. The results for Model A are given in table 5.15 and for Model B in Table 5.16.

The experiments clearly show that, although the training error may be higher for a bagging ensemble if compared to their single network counterparts, the validation error drops significantly. Further bagging seems to favor large neural networks by reducing their variance through

Layers	Decay	Training set size	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
12,24,24,1	1.0	1.25	653.30	0.89	5.00	0.0339	0.0783
12,24,24,1	1.0	1.0	790.51	0.82	5.18	0.0337	0.0795
12,24,24,1	1.0	0.75	478.34	1.46	5.37	0.0421	0.0804
12,12,12,1	1.0	1.0	86.64	1.02	4.46	0.0495	0.0821
12,12,12,1	1.0	1.25	87.37	1.05	4.67	0.0511	0.0826
12,12,1	1.0	1.25	23.43	1.47	4.13	0.0631	0.0831
12,12,12,1	1.0	0.75	70.47	1.41	4.84	0.0513	0.0832
12,12,1	1.0	1.0	23.80	1.43	4.10	0.0637	0.0841
12,24,1	1.0	1.25	66.21	1.07	5.46	0.0488	0.0842
12,12,1	0.001	1.25	20.82	1.28	4.09	0.0616	0.0860
12,72,1	1.0	1.0	837.09	1.10	6.59	0.0371	0.0863
12,12,1	1.0	0.75	39.29	1.87	4.68	0.0650	0.0865
12,24,1	1.0	1.0	67.31	1.14	5.53	0.0488	0.0867
12,72,1	1.0	0.75	633.84	1.91	7.26	0.0464	0.0871
12,72,1	1.0	1.25	851.83	1.12	6.82	0.0371	0.0884
12,12,1	0.001	1.0	21.21	1.31	4.47	0.0623	0.0888
12,24,1	1.0	0.75	50.41	1.87	6.51	0.0535	0.0890
12,12,1	0.001	0.75	17.20	1.70	5.12	0.0640	0.0909

Legend
- Layers — the number of neurons per layer
- Decay — the value for the weight decay parameter of the training algorithm
- Training set size — the factor determining the size of the sampled training sets

Table 5.15: Model A: Bagging results

averaging. An examination of the behaviour of the validation error when adding a new base learner base learner shows that it leads to a decrease of the error almost any time — even if the ensemble is nearly complete. It seems possible that by adding further base learners, the validation error may be reduced further.

The obvious drawback back of the bagging approach is the rise in computation time. But due to the independence of single neural networks for other networks in the ensemble, bagging is an ideal candidate for massive parallelization.

Least-Squares K-Means divide-and-conquer regression

Least-Squares divide-and-conquer regression has been performed on both the training sets of Model A and Model B. The neural network architectures trained for each cluster are given in Table 5.17. The neural network to be used for a cluster is selected using automatic sub-model selection with validation sets, as described in Section 4.3. The number L of networks trained for validation is set to 5 and the validation percentage is set to 0.1.

Experiments have been conducted with 5, 10, and 15 clusters as well as overlap percentages of 0, 0.15, 0.3, and 0.5. As prediction methods both variants described in Section 4.3 have been used. The results for Model A with "closest" cluster prediction are given in Table 5.13 and the

Layers	Decay	Training set size	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
14,84,1	0.1	1.25	738.27	80.99	518.72	0.0219	0.0554
14,84,1	0.1	1.0	811.22	82.84	546.64	0.0223	0.0556
14,84,1	0.1	0.75	517.38	142.41	582.41	0.0284	0.0578
14,56,1	1.0	1.0	225.25	98.54	560.74	0.0251	0.0588
14,56,1	1.0	1.25	223.40	99.64	583.61	0.0248	0.0599
14,84,1	1.0	1.0	558.09	107.60	685.97	0.0227	0.0605
14,84,1	1.0	1.25	554.26	105.65	678.21	0.0226	0.0609
14,56,1	1.0	0.75	176.78	160.64	636.96	0.0303	0.0625
14,14,1	0.001	1.0	16.37	735.58	1196.59	0.0442	0.0649
14,14,1	1.0	1.0	16.59	719.32	1204.91	0.0442	0.0654
14,14,1	0.001	1.25	16.43	729.18	1210.51	0.0440	0.0675
14,84,1	1.0	0.75	419.10	213.81	878.19	0.0316	0.0686
14,14,1	1.0	1.25	16.50	716.79	1236.42	0.0450	0.0692
14,14,1	0.001	0.75	12.69	804.17	1279.30	0.0476	0.0692
14,14,1	1.0	0.75	18.79	798.94	1299.85	0.0470	0.0699

Legend
Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm
Training set size — the factor determining the size of the sampled training sets

Table 5.16: Model B: Bagging results

Model A		Model B	
Layers	Decay	Layers	Decay
12,1	0.1	14,1	0.1
12,3,1	0.1	14,3,1	0.1
12,3,1	1.0	14,3,1	1.0
12,6,1	0.1	14,7,1	0.1
12,6,1	1.0	14,7,1	1.0
12,12,1	0.1	14,14,1	0.1
12,12,1	1.0	14,14,1	1.0
12,24,1	0.1	14,28,1	0.1
12,24,1	1.0	14,18,1	1.0

Table 5.17: Neural network archictures for LS K-Means divide-and-conquer regression

results for Model B are shown in Table 5.19.

Compared to the other regression techniques the validation error is rather high for both HEV models. The results for "distance weighted" prediction are similar to those using "closest cluster" prediction and are not able to lower the validation error significantly. Therefore the results of this prediction method have been omitted.

In another set of experiments the automatic sub-model selection has been changed to use training set based selection and their findings are depicted in Table 5.20 for Model A and in

Ov.%	#Clusters	Time (secs)	tMSE	vMSE	tMTOD	vMTOD
0.5	5	120.64	3.25	15.89	0.0749	0.1298
0.15	5	102.77	3.78	16.77	0.0757	0.1399
0.3	10	96.79	4.99	21.39	0.0886	0.1489
0.0	5	97.30	4.41	21.45	0.0833	0.1495
0.5	10	111.09	5.05	20.04	0.0840	0.1502
0.5	15	136.07	4.87	27.23	0.0912	0.1524
0.15	10	79.93	5.10	21.79	0.0881	0.1554
0.3	5	110.27	21.38	35.42	0.1113	0.1572
0.0	10	78.79	4.80	27.52	0.0807	0.1577
0.15	15	93.03	4.29	28.22	0.0745	0.1578
0.3	15	114.50	4.44	28.55	0.0860	0.1641
0.0	15	117.12	4.75	32.37	0.0803	0.1660

Legend Ov.% — the overlap percentage
#Clusters — the number of clusters

Table 5.18: Model A: LS K-Means divide-and-conquer regression results, with validation set based model selection, predicting with the closest cluster only

Ov.%	#Clusters	Time (secs)	tMSE	vMSE	tMTOD	vMTOD
0.15	5	116.60	1245.05	2003.80	0.0863	0.1068
0.3	5	125.18	1137.92	2063.40	0.0821	0.1091
0.5	5	94.72	1244.60	2016.64	0.0888	0.1114
0.0	5	112.16	1260.83	2446.94	0.0872	0.1131
0.0	10	48.74	1008.44	2988.64	0.0814	0.1143
0.3	10	57.57	1138.64	2824.10	0.0830	0.1146
0.5	10	64.02	1191.69	2813.99	0.0817	0.1147
0.5	15	64.62	973.77	3510.19	0.0733	0.1188
0.15	10	50.53	1058.02	2831.55	0.0800	0.1190
0.15	15	53.62	956.81	4256.36	0.0768	0.1256
0.0	15	52.14	1027.64	4299.10	0.0761	0.1288
0.3	15	58.03	866.24	4311.31	0.0736	0.1302

Legend Ov.% — the overlap percentage
#Clusters — the number of clusters

Table 5.19: Model B: LS K-Means divide-and-conquer regression results, with validation set based model selection, predicting with the closest cluster only

Table 5.21 for Model B. It can be seen that the training error is far lower than in the experiments with validation set based selection. Although the validation error is increased further as the training set based selection mechanism favors overfitting.

The most interesting result of these experiments is that automatic model selection based on validation sets is able to improve the validation performance, if compared to training set based

model selection. This approach might also be useful for the gradient boosting approach.

Ov.%	#Clusters	Time (secs)	tMSE	vMSE	tMTOD	vMTOD
0.5	5	21.83	0.06	14.89	0.0160	0.1408
0.15	5	18.33	0.03	19.21	0.0105	0.1444
0.0	5	18.71	0.02	18.30	0.0084	0.1487
0.3	5	19.98	0.02	18.52	0.0097	0.1512
0.15	10	15.44	0.00	30.03	0.0044	0.1616
0.0	10	16.51	0.01	28.45	0.0047	0.1619
0.3	10	19.56	0.01	27.84	0.0057	0.1639
0.5	10	20.12	0.01	25.65	0.0057	0.1642
0.5	15	20.67	0.00	37.00	0.0045	0.1675
0.3	15	18.01	0.00	36.06	0.0040	0.1705
0.15	15	14.96	0.00	35.62	0.0039	0.1708
0.0	15	14.73	0.00	37.94	0.0036	0.1752

Legend Ov.% — the overlap percentage
#Clusters — the number of clusters

Table 5.20: Model A: LS K-Means divide-and-conquer regression results, with training set based model selection, predicting with the closest cluster only

Ov.%	#Clusters	Time (secs)	tMSE	vMSE	tMTOD	vMTOD
0.3	5	10.75	55.07	3749.14	0.0161	0.1247
0.5	5	11.55	61.10	3486.46	0.0180	0.1250
0.15	5	9.85	51.31	3622.05	0.0155	0.1287
0.3	10	10.90	9.45	4064.00	0.0077	0.1293
0.0	5	8.82	27.64	4233.74	0.0126	0.1320
0.5	10	11.76	12.99	4865.17	0.0088	0.1359
0.15	10	10.16	10.24	4829.17	0.0076	0.1377
0.5	15	12.75	6.50	5398.87	0.0063	0.1402
0.0	15	10.94	4.15	6434.68	0.0057	0.1444
0.0	10	9.49	9.46	5328.93	0.0073	0.1521
0.3	15	11.80	5.21	6666.51	0.0065	0.1548
0.15	15	11.37	4.89	6259.56	0.0059	0.1555

Legend Ov.% — the overlap percentage
#Clusters — the number of clusters

Table 5.21: Model B: LS K-Means divide-and-conquer regression results, with training set based model selection, predicting with the closest cluster only

Partial simulation and extrapolation

In contrast to the regression techniques evaluated above, partial simulation and extrapolation does not try to predict the fitness value for given parameter set but the output values of the HEV models directly. So for Model A a single neural network tries to predict the fuel consumption output of the model. For Model B two separate neural networks are used, the first one predicts the fuel consumption and the second one for predicts the SOC deviation.

In the experiments for both HEV models 25%, 50%, and 75% of the respective driving cycles are simulated. The neural network for Model A which predicts the fuel consumption then takes the last fuel consumption value generated by the simulation as additional input. The same holds true for the first neural network of Model B. Its second neural network takes the last known value of the SOC deviation as additional input.

The results for Model A are given in Table 5.22. The prediction results for the fuel consumption of Model B are given in Table 5.23 and the SOC deviation results are summed up in 5.24. The MSE values of the neural networks used for Model B are omitted, as they would allow to deduce the actual fuel consumption and SOC deviation of Model B.

The extrapolation results for Model A are very promising. The MTOD values for the validation sets are comparable between the fitness predicting neural networks and the neural networks extrapolating the fuel consumption. The reason is that Model A uses only a single output value to compute its fitness value. The scaling between the fuel consumption and the fitness value is linear and would not change the order of the solutions. Therefore it can be said that the best regression models for each simulated portion of the driving cycle exhibit lower validation errors than the best neural networks experiments. In the experiments where 75% of the driving cycle are simulated, the regression models even outperform the bagging ensembles trained for Model A. Although this comes at the cost of simulating the a larger portion of the driving cycle.

The MTOD values for Model B cannot be directly compared as the fitness value of a solution is a linear combination of multiple output values of the HEV model, which might change the order of the solutions. Nevertheless it can be seen that the validation errors for fuel consumption and SOC deviation are rather low. If the experiments, where 75% of the driving cycle are simulated, are considered then the validation errors are for both output values are close to zero. Further these prediction results are achieved by a linear regression model. In the case of Model B it seems possible to simulate only the first 75% of the NEDC driving cycle s.t. the expected quality of the solutions changes much.

Finally the observation can be made that for all extrapolation experiments it holds true that the prediction performance is increased if the simulated portion of the driving cycles are increased. Partial simulation and extrapolation seems to be a good option if the simulation time of a HEV model shall be decreased and very accurate predictions are required — which cannot be achieved with non-extrapolating prediction techniques.

Time-progressive-learning ensembles

Time-progressive learning as described in Section 4.3 have been applied on the fuel consumption output of Model A. The results are given in Table 5.25. Different numbers of time slices have been evaluated. Further experiments with jitter and validation set based automatic model

selection have been conducted. If validation sets are used for model selection then 3 validations with a validation percentage of 0.1 are conducted per base learner. For each time slice, neural networks with a single hidden layer of the sizes 0, 0.25, 0.5, 1, 3, and 6 times the number of input layer neurons are trained.

The results show that validation error is about the same as for neural networks. Analysis of the validation errors of each time slice show that in general the error during the first time slices is the highest and then drops in successive slices. This correlates with the observation that the fuel consumption — which is given in L/100km — fluctuates at the beginning of the driving cycle. It is assumed that time-progressive-learning ensembles perform better if monotonic output values shall be predicted. Unfortunately the use of jitter could not improve the validation error.

5.6 Two-phased optimization II

The regression models evaluated in the previous step are now used in the second optimization phase. As regression model the best bagging ensemble parametrization has been chosen for both HEV models, as they provided the best performance but the extrapolation methods. In the second optimization phase two different schemes for the combination of the optimization techniques with the regression model have been explored. Both methods are similar to the methods described in [25].

The first method incorporates a PSO with a `PredictingComputationStrategy` as follows.

1. For k iterations repeat the following
   a) Generate a new population p using the PSO algorithm.
   b) Evaluate p with using the trained regression model.
2. Take the last generated population p and evaluate it with the simulation software.
3. Pass p to the PSO algorithm and update its internal state as described in Section 4.2.
4. Until some halting condition has been met, repeat from step 1.

The second method combines a `FilteringComputationStrategy` with an Active CMA-ES algorithm. Thereby the parameter λ is set to large value s.t. large number of descendants is generated in each generation. The λ new descendants are then evaluated using the trained regression model. The best 16 individuals of the previous step are then passed back into the Active CMA-ES algorithm. The remaining individuals are discarded.

Besides these two new setups, the number of discretization points in each dimension of the search space are increased from 5 to 15. Further the used regression models are retrained every 15 iterations to incorporate the new solutions. In the case of the PSO scheme described above only the iterations of the outer loop are counted. All optimization algorithms are run for 60 (outer loop) iterations in phase II. This number has been chosen as the reference optimization algorithms without discretization have not improved the fitness significantly after 120 to 150 iterations. If phase I and phase II are combined then overall number of iterations sums up to 125.

Phase II results for Model A

PSO with generational re-evaluation

The parameter settings for the PSO in phase II are nearly the same as in phase I and are given in Table 5.26. It has initially been intended to start the PSO from the best solutions found in phase I. The idea has been discarded as first experiments showed that the PSO converges within a few (outer loop) iterations even if only few solutions are taken from the database. Therefore population is initially sampled randomly. For each outer loop iteration, seven inner loop iterations are performed. Due to time constraints it has not been possible to experiment with more settings for the inner loop. For all experiments a bagging ensemble with 24 neural networks is used. The architecture of the used neural networks is $(12, 24, 24, 1)$, with a weight decay of 1.0 and with a training set size factor set to 1.0.

The result for 10 PSO experiments in phase II are given in Table 5.27. It has even been possible to achieve results of approximately the same quality as the best found solution for Model A, in a few of the experiments. The number of computed batches is, similar to phase I, very low and a summary is given in Table 5.28. The average number of computed batches, if RRS is used in phase I, is 181.8. This is far lower than the number of computed batches for the optimization algorithms without discretization. If the algorithms without discretization are run for 125 iterations 250 batches are computed. This results in a effective speedup of the optimization of 27%.

Active CMA-ES with filtering

In phase II experiments with the Active CMA-ES are run again with filtering enabled. The algorithm generates 100 potential solutions to be evaluated by with the trained regression model. The best 16 solutions are then evaluated with the simulation software. The initial population is generated by using the 30 best solutions from phase I, as well as 20 random solution from phase I. The best solution are used to start in proximity to good areas in the search space. The random solution are added to increase the values in the estimated covariance matrix. The initial estimation of the algorithm state i.e. centroid, σ, and covariance matrix, is carried out as described in Section 4.2.

For all experiments a bagging ensemble with 24 neural networks is used. The architecture of the used neural networks is $(12, 24, 24, 1)$, with a weight decay of 1.0 and with a training set size factor set to 1.0. The configuration of the Active CMA-ES is summarized in Table 5.29.

The results of the experiments are given in Table 5.30. Interestingly the performance of the Active CMA-ES has been vastly improved by implementing the regression models for filtering. It even outperforms the PSO in phase II in terms of solution quality. Figure 5.8 shows the variance in the fuel consumption every two iterations for an exemplary Active CMA-ES run. It can be clearly seen that the variance is far higher than in the original Active CMA-ES experiments.

The number of computed batches is given Table 5.28. If summed with the RRS in phase I the 238.8 batches are computed on average over the whole two-phased optimization. Compared to the optimizations without discretization the speed up is about 4.5%.

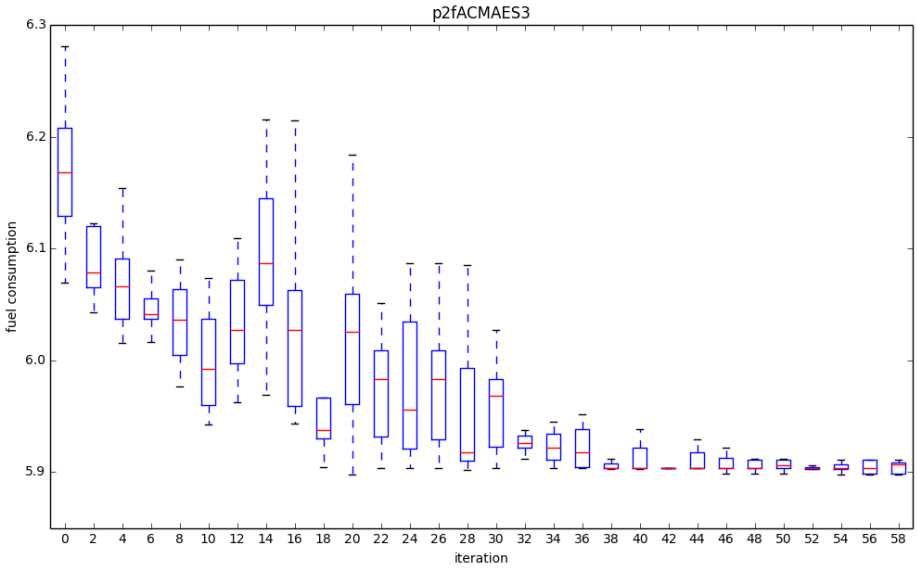


Figure 5.8: Model A: Fuel consumption variance per iteration of an exemplary Active CMA-ES run in phase II.

Phase II results for Model B

Active CMA-ES with filtering

The Active CMA-ES has also been applied to Model B in phase I. The settings for the algorithm are given in Table 5.31. The number of best solutions taken from phase 2 has been reduced if compared to Model A, as first experiments showed that the estimated parameters for the ES resulted in a narrow search area. Further an additional mechanism has been implemented to double σ if there had been no improvement for 8 iterations. This is necessary as otherwise σ grew smaller each iteration — a problem not exhibited in Model A. Consequently the search stagnated. Repeated doubling of σ solved the issue.

The results are given in Table 5.32. With the modifications mentioned above the performance of the algorithm has been boosted significantly. The best solution for Model B has even been found by the Active CMA-ES in phase 2.

The number of computed batches is given Table 5.33. If summed with the RRS in phase I the 242.4 batches are computed on average over the whole two-phased optimization. Compared to the optimizations without discretization the speed up is about 3%.

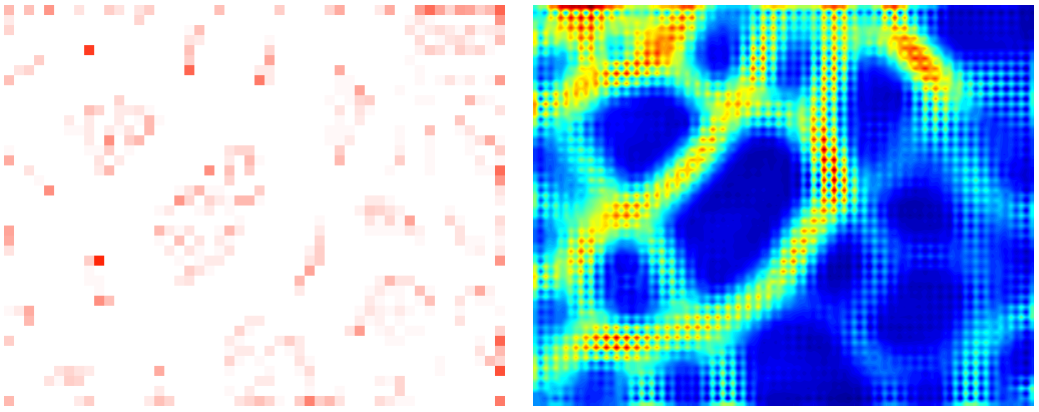
5.7 Analysis of the evaluated HEV model parameter sets

In this section an analysis of the parameter sets for Model A is given. It is not only important to find good parameter settings for the HEV model. Its further important to understand which

parameter settings yield the best solutions, which of the optimized parameters have high or low influence on the fuel consumption, and what discriminates good from bad solutions.

As the parameter space of the HEV model is high-dimensional, a proper analysis is difficult, as correlations might be hard to detect — especially if they are non-linear. Therefore a Self-Organizing Map (SOM) is trained on the evaluated solutions. A SOM is an unsupervised machine learning technique described in [29], which creates a non-linear two-dimensional mapping of a high-dimensional space. The trained SOM consists of units (points) which are connected in a two-dimensional grid. Each unit has an assigned position the high-dimensional space, which is determined by the training process. The inputs i.e. the HEV model parameter sets, are then projected to the closest unit in the search space. Due to the non-linearity of the mapping areas of the same size in the grid of units, may map to areas of different sizes in the input space.

The following SOMs are generated with the SOMToolbox software[5] created by the "Institute of Software Technology and Interactive Systems" of the "Technical University of Vienna". The inputs for the SOM training are the solutions generated by an exemplary two-phased optimization, with a RRS as first phase and an Active CMA-ES as second phase. Maps with different sizes have been created and their quantization errors and topological errors have been analysed to ensure that the SOMs are properly trained. The following visualizations are displayed on a selected 50x40 SOM.



(a) Hit histogram: Red indicates a high number of mapped solutions.

(b) U-Matrix: Blue indicates short distances and red indicates long distances between units.

Figure 5.9: Basic SOM unit measures

First the trained SOM gives insight into the distribution of the solutions generated by the optimization algorithm. A hit histogram visualizing the number of mapped solutions per unit is depicted in Figure 5.9. The more solutions are mapped to a unit the darker the unit is colored. Further the image displays a U-Matrix as described in [44]. A U-Matrix shows the distances — in the input space — between neighbouring units in the grid. From the information contained in both visualizations it can be deduced that the optimization algorithm generates a large number

[5] `http://www.ifs.tuwien.ac.at/dm/somtoolbox/index.html`, accessed: 2013-08-21

of solution in the right half of the map which maps to a rather small area in the input space. In the following it can be seen that this area also contains the best solutions of the optimization algorithm.

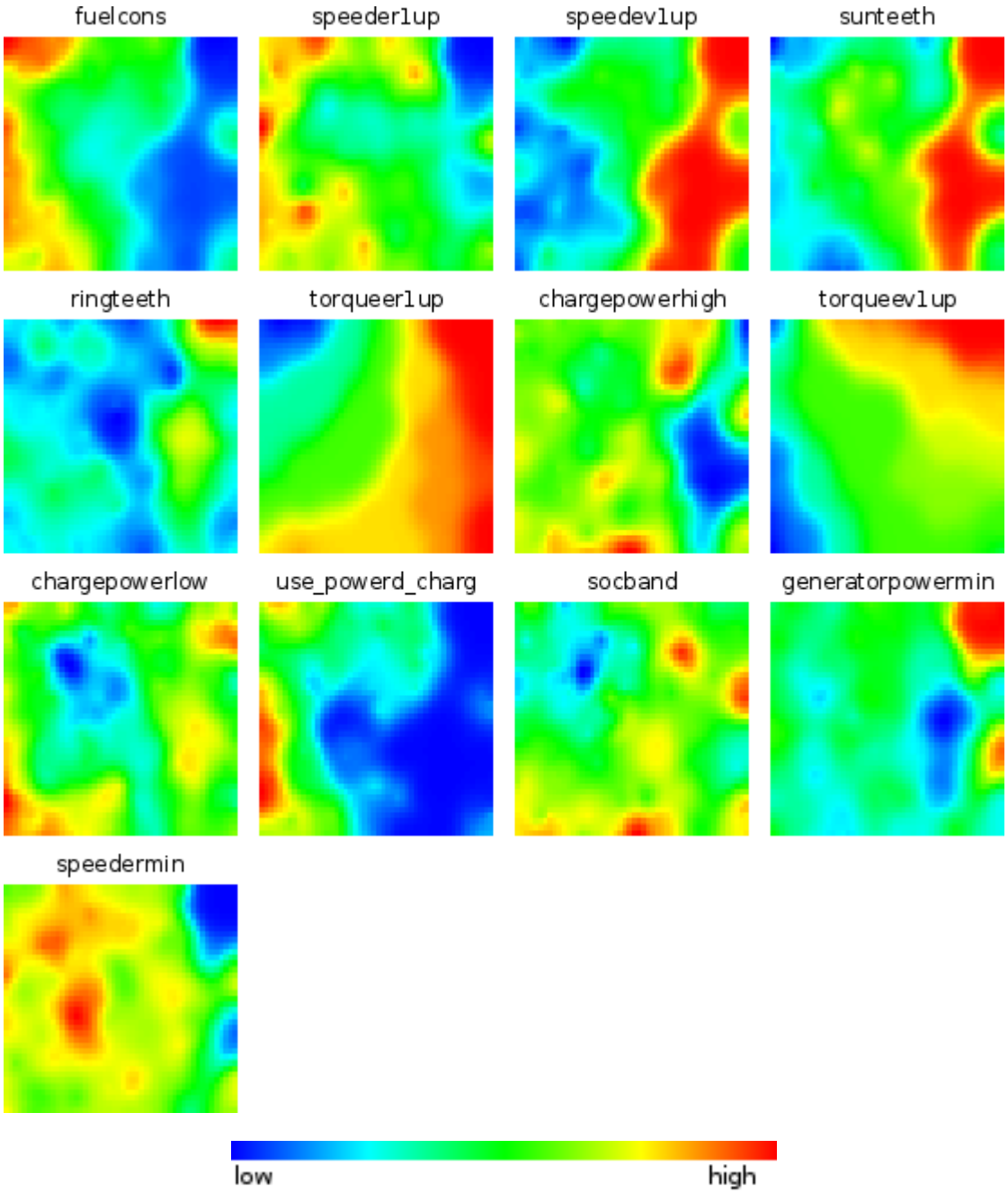


Figure 5.10: Model A: Component planes for the solutions of the evaluated optimization experiment.

In Figure 5.10 the Component Planes visualization is shown. Component Planes are described in [46]. A Component Plane shows the distribution of the values for single input space dimension. The planes are useful to detect correlations between data points mapped to the same area in the grid. The plane in the upper left corner shows the distribution of the fuel consumption, where blue areas indicate low fuel consumption. The best solutions — below 5.9 L/100km — are clustered in the upper right corner of the map. It can be seen that fuel consumption correlates inversely with the number of teeth in the sun gear of the planetary gear set. A lower number of teeth for the ring gear can be found in bad solutions. Further it can be observed for good solutions that the switch between the operational modes EV1 to EV2 and ER1 to ER2 should not be inhibited by lower axle torque values.

Although the Component Planes of the SOM above indicate correlations between lower fuel consumptions and some parameter values, it is not possible to distinguish between the good solutions — fuel consumptions about 5.95 L/100km — and the best solutions — below 5.9 L/100km. Therefore a 30x20 SOM has been trained on the best 20% of solutions generated by the optimization process. The Component Planes for this map are depicted in Figure 5.11. It should be mentioned that the parameter values are no longer distributed across their complete value range in the new SOM. The visualization shows that the solutions with the lowest fuel consumptions possess a large number of teeth in the ring gear. The influence of the SOC band cannot be determined as its value range in the SOM above is narrow and about 0.52 and 0.6. Finally the *chargepowerhigh* parameter should not be set near its lower bound.

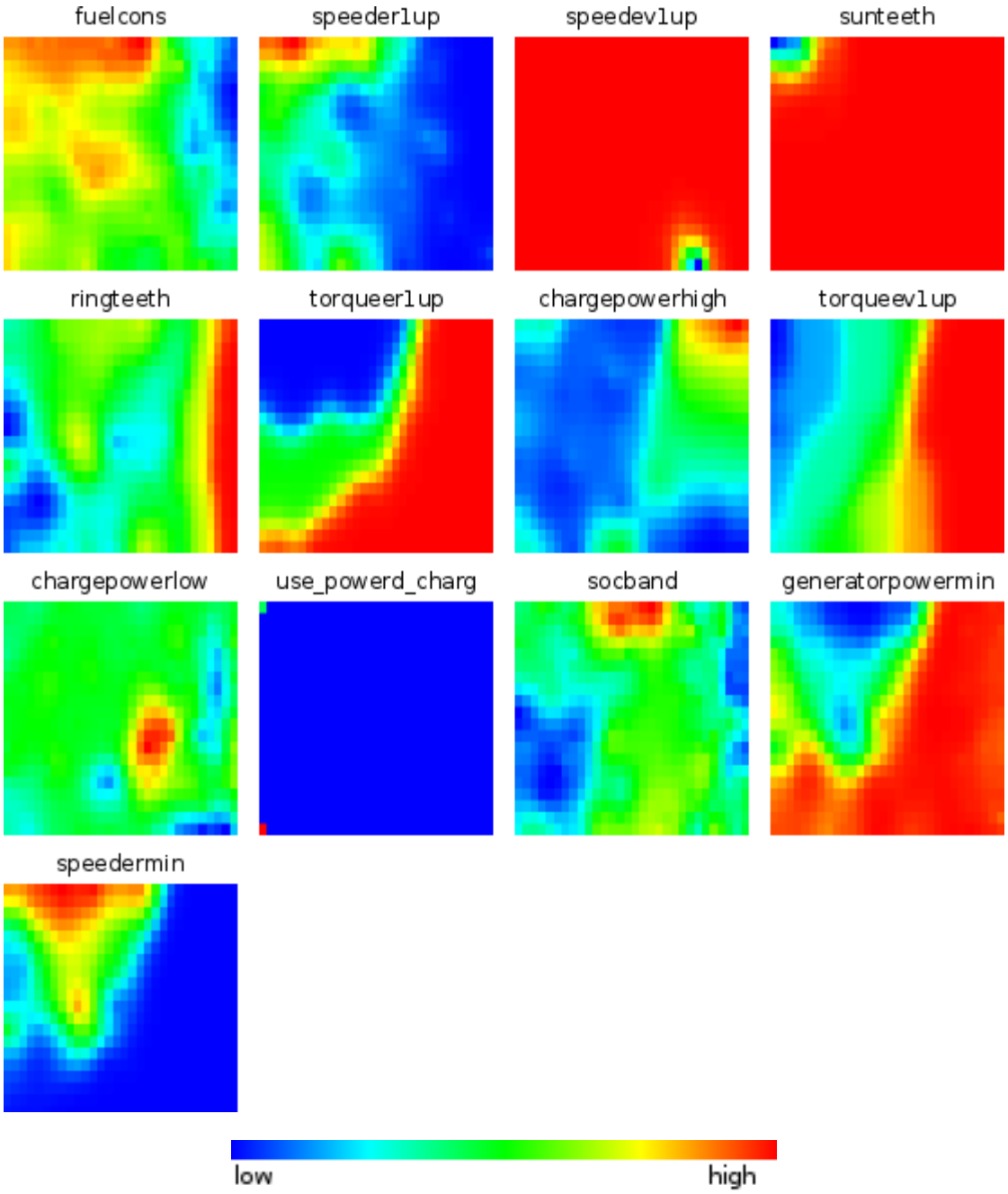


Figure 5.11: Model A: Component planes for the best 20% of the evaluated solutions. The color scale is adapted to parameter values of the best 20% of solutions.

Layers	Decay	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
13,13,1	0.1	14.96	0.01	0.02	0.0615	0.0810
13,13,13,1	0.1	46.40	0.00	0.03	0.0497	0.0824
13,26,1	1	34.86	0.01	0.02	0.0665	0.0829
13,26,26,1	1	256.18	0.00	0.02	0.0603	0.0838
13,78,1	0.1	433.81	0.00	0.03	0.0343	0.0853
13,78,1	0.001	200.72	0.00	0.06	0.0096	0.1175
13,52,1	0.001	188.52	0.00	0.06	0.0125	0.1213
13,1	0.1	0.12	0.07	0.08	0.1505	0.1536
13,1	0.001	0.14	0.07	0.08	0.1505	0.1536
13,1	1	0.11	0.07	0.08	0.1505	0.1536

Model A: Results for extrapolation after simulating 25% of the EPA US06 driving cycle

Layers	Decay	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
13,13,13,1	0.1	39.25	0.00	0.01	0.0489	0.0703
13,13,13,1	1	39.03	0.01	0.01	0.0637	0.0746
13,78,1	1	203.06	0.01	0.01	0.0681	0.0751
13,52,1	1	98.99	0.01	0.01	0.0690	0.0754
13,13,1	0.1	14.83	0.00	0.01	0.0555	0.0756
13,1	1	0.10	0.01	0.01	0.0899	0.0914
13,26,1	0.001	75.73	0.00	0.04	0.0350	0.0934
13,26,26,1	0.001	256.32	0.00	0.04	0.0088	0.0978
13,78,1	0.001	295.65	0.00	0.05	0.0084	0.1095
13,52,1	0.001	202.77	0.00	0.05	0.0126	0.1126

Model A: Results for extrapolation after simulating 50% of the EPA US06 driving cycle.

Layers	Decay	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
13,78,1	1	186.94	0.00	0.00	0.0351	0.0387
13,1	0.001	0.13	0.00	0.00	0.0357	0.0388
13,1	0.1	0.12	0.00	0.00	0.0357	0.0388
13,52,1	1	80.85	0.00	0.00	0.0352	0.0388
13,26,1	0.001	66.75	0.00	0.01	0.0244	0.0522
13,13,13,1	0.001	71.39	0.00	0.01	0.0240	0.0530
13,26,26,1	0.001	246.93	0.00	0.01	0.0101	0.0647
13,78,1	0.001	252.53	0.00	0.02	0.0116	0.0668
13,52,1	0.001	176.02	0.00	0.02	0.0146	0.0685

Model A: Results for extrapolation after simulating 75% of the EPA US06 driving cycle

Legend
Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm

Table 5.22: Model A: Neural network results for extrapolating the fuel consumption Results with intermediate validation error have been omitted from all tables.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,30,30,1	0.001	295.65	0.0196	0.0463
15,15,1	0.001	29.82	0.0268	0.0468
15,15,15,1	0.001	67.09	0.0279	0.0477
15,30,1	0.001	83.81	0.0272	0.0567
15,30,1	1	21.16	0.1327	0.1370
15,90,1	1	154.20	0.1328	0.1370
15,1	1	0.10	0.1330	0.1375
15,1	0.1	0.10	0.1330	0.1376
15,1	0.001	0.11	0.1330	0.1376

Model B: Results for fuel consumption extrapolation after simulating 25% of the NEDC driving cycle.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,30,30,1	0.001	323.23	0.0187	0.0350
15,15,15,1	0.001	49.45	0.0221	0.0351
15,30,1	0.001	83.41	0.0188	0.0369
15,15,1	0.001	27.15	0.0243	0.0406
15,60,1	0.001	169.02	0.0207	0.0459
15,30,1	1	22.25	0.1116	0.1153
15,90,1	1	137.00	0.1117	0.1153
15,1	1	0.11	0.1121	0.1157
15,1	0.1	0.12	0.1121	0.1158
15,1	0.001	0.13	0.1121	0.1158

Model B: Results for fuel consumption extrapolation after simulating 50% of the NEDC driving cycle.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,1	0.001	0.11	0.0000	0.0000
15,1	0.1	0.11	0.0001	0.0003
15,90,1	0.001	306.70	0.0011	0.0016
15,1	1	0.10	0.0013	0.0017
15,30,30,1	0.001	262.21	0.0016	0.0019
15,30,1	1	25.03	0.0157	0.0182
15,60,1	1	85.22	0.0158	0.0184
15,90,1	1	180.87	0.0158	0.0184
15,15,15,1	1	22.84	0.0426	0.0460
15,30,30,1	1	121.78	0.0434	0.0468

Model B: Results for fuel consumption extrapolation after simulating 75% of the NEDC driving cycle.

Legend Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm

Table 5.23: Model B: Neural network results for extrapolating the fuel consumption. Results with intermediate validation error have been omitted from all tables.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,30,30,1	1	369.18	0.0086	0.0542
15,90,1	1	608.95	0.0097	0.0553
15,60,1	1	240.18	0.0109	0.0557
15,15,15,1	1	48.48	0.0207	0.0575
15,15,1	0.001	13.42	0.0344	0.0615
15,90,1	0.001	244.17	0.0017	0.0789
15,60,1	0.001	121.77	0.0021	0.0857
15,1	0.1	0.13	0.1001	0.1057
15,1	0.001	0.17	0.1000	0.1057
15,1	1	0.13	0.1001	0.1057

Model B: Results for SOC deviation extrapolation after simulating 25% of the NEDC driving cycle.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,30,30,1	1	297.73	0.0085	0.0442
15,90,1	1	661.64	0.0095	0.0510
15,15,1	1	9.98	0.0342	0.0534
15,60,1	1	355.25	0.0108	0.0536
15,30,30,1	0.1	246.30	0.0056	0.0538
15,90,1	0.001	255.12	0.0017	0.0765
15,60,1	0.001	123.63	0.0021	0.0768
15,1	1	0.13	0.1001	0.1057
15,1	0.1	0.12	0.1001	0.1058
15,1	0.001	0.13	0.1001	0.1058

Model B: Results for SOC deviation extrapolation after simulating 50% of the NEDC driving cycle.

Layers	Decay	Time (sec)	tMTOD	vMTOD
15,1	0.001	0.10	0.0014	0.0015
15,1	0.1	0.11	0.0018	0.0022
15,15,1	0.001	10.21	0.0031	0.0040
15,30,30,1	1	205.98	0.0033	0.0052
15,1	1	0.11	0.0043	0.0055
15,60,1	1	188.42	0.0056	0.0140
15,60,1	0.001	117.49	0.0030	0.0188
15,90,1	0.001	265.08	0.0019	0.0202
15,30,30,1	0.001	222.93	0.0031	0.0215

Model B: Results for SOC deviation extrapolation after simulating 75% of the NEDC driving cycle.

Legend
Layers — the number of neurons per layer
Decay — the value for the weight decay parameter of the training algorithm

Table 5.24: Model B: Neural network results for extrapolating the SOC deviation. Results with intermediate validation error have been omitted from all tables.

K	Val.	Jitter	Time (sec)	tMSE	vMSE	tMTOD	vMTOD
3	N	N	385.97	0.00	0.04	0.0433	0.0917
2	N	N	373.65	0.00	0.04	0.0410	0.0942
5	N	N	597.07	0.00	0.04	0.0461	0.0957
5	Y	Y	46726.49	0.01	0.04	0.0494	0.0990
2	N	Y	816.68	0.00	0.05	0.0337	0.1017
5	N	Y	2221.32	0.01	0.04	0.0468	0.1059
3	N	Y	1220.32	0.01	0.05	0.0463	0.1068
2	Y	Y	10405.43	0.01	0.04	0.0476	0.1075
3	Y	Y	24850.91	0.02	0.05	0.0502	0.1078
2	Y	Y	2701.68	0.00	0.03	0.0390	0.1174

Legend
- K — the number of time slices
- Val. — 'Y' if validation set based automatic sub-model selection is used
- Jitter — 'Y' if the predicted values of a time slice are used as training data for the next slice

Table 5.25: Model A: Time-progressive-learning ensemble results

#Iterations (outer/inner)	#Particles	ω	ϕ_1	ϕ_2	#Non-impr. iterations before reset
60/7	16	0.9	2.0	2.0	20

Table 5.26: Model A: Parameters for PSO in phase II

#Experiment	Iteration	Best solution (L/100km)
1	35	5.93
2	34	5.97
3	41	5.97
4	39	5.89
5	12	6.10
6	59	5.90
7	11	6.01
8	32	5.88
9	40	5.98
10	24	5.93

Mean: 5.95L/100km Std. dev.: 0.06

Table 5.27: Model A: Results for the PSO in phase II

	Min. #batches	Max. #batches	Mean	Standard deviation
PSO	36	113	61.4	22.87
A. CMA-ES	114	120	118.4	2.33

Table 5.28: Model A: Statistical measures about the computed batches in phase II

#Iterations	μ	λ (unfiltered/filtered)	#Best/Random solutions from DB
60	4	100/16	30/20

Table 5.29: Model A: Parameters for Active CMA-ES in phase 2

#Experiment	Iteration	Best solution (L/100km)
1	56	5.87
2	52	5.92
3	20	5.89
4	31	5.89
5	49	5.87

Mean: 5.89L/100km Std. dev.: 0.02

Table 5.30: Model A: Results for the Active CMA-ES in phase II

#Iterations	μ	λ (unfiltered/filtered)	#Best/Random solutions from DB
60	4	100/16	20/20

Table 5.31: Model B: Parameters for Active CMA-ES in phase 2

#Experiment	Iteration	Best solution
1	58	-46.13
2	39	-46.65
3	58	-44.61
4	43	-46.09
5	52	-45.82

Mean: -45.86 Std. dev.: 0.68

Table 5.32: Model B: Results for the Active CMA-ES in phase II

	Min. #batches	Max. #batches	Mean	Standard deviation
A. CMA-ES	114	120	118.8	2.4

Table 5.33: Model B: Statistical measures about the computed batches in phase II

CHAPTER 6

Conclusion & Future Work

In this thesis different metaheuristic optimization algorithms have been applied to improve the fuel consumption of two different hybrid electric vehicles. The different hybridization concepts relevant for the optimized HEVs have been described. The first model — Model A — is built with two electric machines and a single ICE. The vehicle is able to operate as a series hybrid as well as a parallel hybrid. The second model — Model B — is a parallel hybrid with a single ICE and a single electric machine. The actual mode of operation is determined by the vehicles' operation strategy. The operation strategies respond to configurable parameters, which are optimized with the metaheuristics described in this thesis.

The metaheuristics used in this thesis are implemented in a new optimization platform — the "Yet Another Optimization Platform" — developed by the author. The platform is highly modular and allows to integrate new optimization algorithms easily. The implemented optimization algorithms can be configured with configuration files. These files also allow the user to combine several algorithms into new hybrid optimization schemes, without programming effort. In the scope of this thesis a Particle Swarm Optimizer, an Evolution Strategy with Covariance Matrix Adaption and a simple Genetic Algorithm have been implemented.

The optimization platform further allows to modify potential solutions independently of the used optimization algorithms, with so called post-processors. Different post-processors have been implemented to constrain the allowed values for the operation strategy parameters. It is possible to enforce specific parameter domains such as integer domains and domains with predefined values. Modelling of simple inequations between the several parameters has also been implemented as a post-processor.

For the concerned HEVs, simulation models are already available. The HEV models are simulated with the GT Suite simulation software. During simulation the models follow a specified driving cycle, which determines for each simulated second the expected vehicle speed. The fuel consumption as well as the SOC deviation of the battery are measured throughout the driving cycle. The collected outputs are then used to calculate an objective value for a specific operation strategy parameter set. The framework implements configurable objective functions which are able to weight different optimization objectives.

A main problem in HEV optimization are the long simulation times. Metaheuristics usually require a substantial number of objective function evaluations. In the case of HEV optimization, evaluating the objective function requires to run a simulation of the HEV model. The high computation times thereby limit the practical applicability of metaheuristics. This problem has already been mentioned in [31]. In this thesis it is tried to lower the negative effects of high computation times. For this cause different approaches are implemented.

First it is tried to restrict the search space of the optimization algorithms with the post-processors described above. Second it has been observed that small changes in the operation strategy parameters often have no relevant effect on the fuel consumption of the vehicles. Therefore the search space is discretized and a cache is used for already computed solutions. The discretization increases the probability of a cache hit without large negative effects on the performance of the optimization algorithms. Experiments show that the number of simulation runs required to find good initial solutions can be decreased.

Third it is tried to replace the HEV simulations with predictions from trained regression models. As regression models, artificial neural networks have been chosen in the thesis as they are able to fit a wide range of functions in practice. Further different ensemble methods combining multiple neural networks have been implemented to reduce the error of the regression models. Theoretical considerations about the properties of the required error measures have been made and came to the conclusion that it is not of primary import to predict the exact value of the replaced objective function. But rather its total order induced on the solutions. To address this issue a new error measure, the "Mean Total Order Deviation" has been introduced.

Gradient Boosting is one of the implemented ensemble methods. It tries to match not only the original target value, but also the residual errors of the previously trained regression models by chaining multiple neural networks. Experiments show that Gradient Boosting is able to reduce the training errors, but worsens the validation errors if compared to single neural networks.

Bagging is another ensemble method, which tries to reduce the variance of overfitting neural networks by averaging the outputs of several networks. The experiments have shown that Bagging yields better validation results than simple neural networks.

It is further tried to cluster the training data with a K-Means++ algorithm and to train a neural network per cluster. Automatic model selection — based either on training sets or validation sets — has been implemented to find proper neural networks for the ensemble.

Next a combination of simulation and prediction has been evaluated. The first part of the driving cycle is simulated and the intermediate outputs of the model — as well as the operation strategy parameters — are passed to a trained neural network. The network is trained to extrapolate the intermediate output values for the rest of the driving cycle. The experiments show that the validation performance can be enhanced significantly using this technique. For Model B it is even possible to predict the HEV outputs, after simulating 75% of the NEDC driving cycle, nearly perfectly.

Last it is tried to split the driving cycle into slices of equal size. Neural networks are trained to fit the intermediate HEV outputs of these slices. The networks are then used for prediction in a cascade. Although the validation error could not be enhanced further. Nevertheless it has been observed that the highest errors occur directly at the beginning of the driving cycle. It is expected that its due to strong fluctuations of the output values at the beginning of the simulation.

It is suggested to use output values with monotonic behaviour over the simulated time span, if possible. This might enhance the prediction performance in the first time slices.

The trained regression models are then incorporated into a new two-phased optimization approach. The first phase uses optimization algorithms to sample training data for the subsequent training of the regression models. In the second phase the optimization algorithms are intended to use the regression models as their objective function and to evaluate only selected solutions directly. For measuring the effectiveness of the implemented improvements a set of reference optimization's with a PSO and an Active CMA-ES algorithm have been run. It has been shown that the PSO performed far better than the ES, as the ES exhibited low variance in its solutions.

The experiments for the first phase showed that a PSO with a coarse grained search space, is able to find good solutions for both Models while using less computational resources due to the effects of discretization. Although a genetic algorithm has been used in the first phase to generate solutions for the subsequent training of regression models, as the PSO produces too few solutions. For the second phase a bagging predictor has been chosen as regression model for both HEV models. In the second phase a PSO uses the regression model as main objective function and evaluates only every seventh iteration with the simulation software. The experiments showed that for Model A the PSO is able to achieve similar good solutions as the reference optimization's in 73% of the time. Further an Active CMA-ES has been run using the regression models as filter for solutions which shall be evaluated with the simulation software. The problem concerning the solution variance is removed by this approach. The algorithm achieved as good results as the reference optimization's for Model A and even better results for Model B. Although the required time could only be reduced by 3 to 4%.

The quality of the found solutions is expected to be in viable ranges for the models. It is problematic to determine if the solutions are near the optimum or better than solutions found by other methods, as no other reference solutions exist[1]. Although it can be said that the devised parameter settings are found in areas of the parameter space which have been expected to contain good solutions.

Finally the parameter sets for Model A have been analysed with the help of SOMs. The analysis showed that a low fuel consumption requires a high number of teeth for both the sun gear and the ring gear of the vehicle's planetary gear set. Further low axle torque values should not inhibit the switch between different modes of operation.

The experiments have shown that there are still open issues with the introduced improvements. First the effect of search space discretization on the used algorithms has to be analysed more thoroughly. The behaviour of some algorithms changes drastically with its introduction, e.g. PSO particles require a specific minimal velocity to change their position. Measures have to be taken to cope with these effects or new optimization algorithms have to be developed, which are able to mix discrete and continuous optimization. Another issue with the current implementation of search space discretization is its static behaviour. Ideally the discretization should adapt to the responsiveness of the search space dimension. The discretization algorithm could be modified to dynamically add or remove discretization points in specific areas based on the

[1] The solutions for a variant of Model B in [31] cannot be directly compared, as a different objective function has been used.

distribution of the objective values of the surroundings. In this case, optimization algorithms would be needed which are able to react to a changing search space.

Second the optimization algorithms should be further improved to deal with changing objective functions. Currently the effects of using multiple objective functions are not well understood. Further experiments should be designed to measure potential misguidance of the optimization process by badly approximating regression models. Optimization algorithms could be designed to treat the return value of a objective function not only as a concrete value but as a probability distribution which incorporates the quality of the used approximation.

Third the used regression models may be revised as they currently do not include information about the structure of the approximated HEV model. It could be beneficial to try to automatically deduce surrogate models for the HEV model at hand. Also currently no input parameter selection is performed for the trained neural networks. Doing so may help to stabilize the networks further as unnecessary inputs are removed. Further it could be tried to use different base learners than neural networks. Especially for the filtering approach, methods like tabu list or artificial immune systems could be implemented to filter potential solutions not only by their predicted objective value, but also directly by their parameter settings.

Fourth completely different approaches for HEV optimization could be explored. Currently only the parameters of a fixed operation strategy are optimized. It might be possible to deduce complete operation strategies by using techniques like genetic programming. An even more radical approach could be to move the optimization process into vehicle itself s.t. the vehicle may adapt to the current driving situation based on an inner model of itself, which would require the development of real-time capable optimization algorithms.

Bibliography

[1] EPA detailed test information. http://www.fueleconomy.gov/feg/fe_test_schedules.shtml. Accessed: 2013-04-19.

[2] Jay April, Fred Glover, James P. Kelly, and Manuel Laguna. Practical introduction to simulation optimization. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 71—-78, 2003.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027—-1035, 2007.

[4] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. Institute of Physics Publishing, 1997.

[5] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary computation 1: Basic Algorithms and operators*. Institute of Physics Publishing, 2000.

[6] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary computation 2: Advanced Algorithms and Operations*. Institute of Physics Publishing, 2000.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[8] AE Bryson and Yu-Chi Ho. Applied optimal control. 1969. *Blaisdell, Waltham, Mass*.

[9] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58—-73, 2002.

[10] US EPA. Light-duty automotive technology, carbon dioxide emissions, and fuel economy trends: 1975 through 2012. Technical Report EPA-420-R-13-001, US EPA, March 2013.

[11] Gamma Erich, Helm Richard, Johnson Ralph, and Vlissides John. Design patterns: elements of reusable object-oriented software. *Reading: Addison Wesley Publishing Company*, 1995.

[12] European Parliament. Regulation (EC) no 443/2009 of the european parliament and of the council of 23 april 2009 setting emission performance standards for new passenger cars

as part of the community's integrated approach to reduce CO2 emissions from light-duty vehicles, 2009. Accessed: 2013-03-15.

[13] Yoav Freund and Robert Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

[14] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, page 1–67, 1991.

[15] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

[16] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[17] Dimitrios Frosyniotis, Andreas Stafylopatis, and Aristidis Likas. A divide-and-conquer method for multi-net classifiers. *Pattern Analysis & Applications*, 6(1):32–40, 2003.

[18] M.T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989 –993, 1994.

[19] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[20] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.

[21] Peter Hofmann. *Hybridfahrzeuge*. Springer-Verlag Vienna, Vienna, 2010.

[22] J. H. Holland. *Adaption in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[23] Xiaolin Hu, Zhongfan Wang, and Lianying Liao. Multi-objective optimization of HEV fuel economy and emissions using evolutionary computation. In *Society of Automotive Engineers World Congress and Exhibition*, volume SP-1856, pages 117–128, 2004.

[24] G.A. Jastrebski and D.V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, pages 2814–2821, 2006.

[25] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *Evolutionary Computation, IEEE Transactions on*, 6(5):481–494, 2002.

[26] Valerie H. Johnson, Keith B. Wipke, and David J. Rausen. HEV control strategy for real-time optimization of fuel economy and emissions. *Society of Automotive Engineers transactions*, 109(3):1677–1690, 2000.

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

[28] Bonomali Khuntia, Shyam S Pattnaik, Dhruba C Panda, Dipak K Neog, S Devi, and Malay Dutta. Genetic algorithm with artificial neural networks as its fitness function to design rectangular microstrip antenna on thick substrate. *Microwave and Optical Technology Letters*, 44(2):144–146, 2005.

[29] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 2001.

[30] T. Krenek, M. Ruthmair, G. R. Raidl, and M. Planer. Applying (hybrid) metaheuristics to fuel consumption optimization of hybrid electric vehicles. In C. Chio et al., editor, *Applications of Evolutionary Computation*, volume 7248 of *Lecture Notes in Computer Science*, pages 376–385. Springer Berlin Heidelberg, 2012.

[31] Thorsten Krenek. Verbrauchsminimierung eines Hybridfahrzeuges im Neuen Europäischen Fahrzyklus. *Master's thesis, Technical University of Vienna*, 2011.

[32] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[33] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[34] Marvin Minsky and Papert Seymour. *Perceptrons.* MIT Press, 1969.

[35] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 1, pages 762—-767, 1989.

[36] European Parliament. 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 text with EEA relevance.

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

[38] Martin Riedmiller and Heinrich Braun. Rprop-a fast adaptive learning algorithm. In *Proc. of ISCIS VII*. Citeseer, 1992.

[39] B. Robitaille, B. Marcos, M. Veillette, and G. Payre. Modified quasi-newton methods for training neural networks. *Computers & Chemical Engineering*, 20(9):1133–1140, 1996.

[40] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton*. Cornell Aeronautical Laboratory, 1957.

[41] R. E. Schapire. The boosting approach to machine learning: An overview. *Lecture Notes in Statistics New York Springer Verlag*, page 149–172, 2003.

[42] Hans-Paul Schwefel. Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. *Master's thesis, Technical University of Berlin*, 1965.

[43] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69—-73, 1998.

[44] A. Ultsch and H. P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proc. INNC'90, Int. Neural Network Conf.*, pages 305–308, Dordrecht, Netherlands, 1990. Kluwer.

[45] US Congress. Clean air act. http://www.gpo.gov/fdsys/pkg/USCODE-2008-title42/pdf/USCODE-2008-title42-chap85.pdf, 2008. Text of the Clean Air Act; Accessed: 2013-04-19.

[46] Juha Vesanto. Som-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.