FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Verbrauchsminimierung eines Hybridfahrzeuges im Neuen Europäischen Fahrzyklus

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsingenieurwesen Informatik

eingereicht von

Thorsten Krenek
Matrikelnummer 0427478

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn Günther Raidl
Mitwirkung: Univ.Ass. Dipl.-Ing. Mario Ruthmair

Wien, 28.07.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

Thorsten Krenek
Gertrude-Wondrack-Platz 5 / 1 / 2.03
1120 Wien

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen, Internet-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.

Wien, am 28. Juli 2011

_____

Thorsten Krenek

Kurzfassung

Für ein Computermodell eines Hybridelektrokraftfahrzeugs (Hybrid Electric Vehicle, HEV) der neuesten Generation soll der Treibstoffverbrauch im Neuen Europäischen Fahrzyklus (NEFZ) minimiert werden. Eine Besonderheit bei der Verbrauchsbestimmung von HEVs im NEFZ ist die Forderung eines nahezu gleichen Batterieladezustands (SOC) am Beginn und am Ende eines Fahrzyklus. Für das Fahrzeug soll u.a. die Schaltstrategie, Schwellwerte für die rein elektrische Fahrt und die Lastpunktanhebung optimiert werden. Dadurch ergeben sich in etwa zehn zu optimierende Parameter. Die Aufgabe besteht nun darin, eine Optimierungsstrategie zu finden, die es ermöglicht, in möglichst geringer Laufzeit ein Parameterset zu finden, das möglichst nahe am globalen Optimum liegt. Durch die hohe Komplexität des Modells ist eine direkte Bestimmung garantiert optimaler Parameterwerte praktisch ausgeschlossen, es können lediglich konkrete Parametersets durch relativ zeitaufwändige Simulationen des Modells bewertet werden.

Ausgehend von bereits vorhandenen einfacheren Standardoptimierungsmethoden wurden unterschiedliche, teils deutlich effizientere Algorithmen entwickelt, die spezielle Eigenschaften der Problemstellung ausnutzen. Weiters galt es das Optimierungspotenzial für ein Modell, dessen Parameterset bereits empirisch ermittelt wurde, abzuschätzen.

Als ersten Schritt wurden die vorhandenen Optimierungsmethoden der vorgegebenen Simulationssoftware analysiert und in weiterer Folge zu Vergleichszwecken herangezogen. Es wurde ein Framework für die zu implementierenden Algorithmen erstellt, das den Zugriff auf die Simulationssoftware des Modells ermöglicht. Für den eigentlichen Optimierungsprozess wurde zuerst ein Algorithmus entwickelt, der die Auswahl der zu optimierenden Parameter reduziert bzw. festlegt. In weiterer Folge wurden ein genetischer Algorithmus (GA), ein Downhill-Simplex Verfahren und ein Algorithmus, der auf Schwarmintelligenz beruht (PSO), implementiert. Beim GA wurde darauf geachtet, dass bei der Rekombination jeweils der Wert der Lösung akzeptiert wird, der eher zu einem ausgeglichenem SOC führt. Beim Downhill-Simplex Verfahren wurde auf eine gesamte Verkleinerung des Simplex verzichtet, weil dadurch alle Punkte des Simplex neu berechnet werden müssten und diese oft einen schlechteren Zielfunktionswert aufweisen. Beim PSO wurde die beste Lösung nach einer gewissen Anzahl an Iterationen mit einem Surface-Fitting Algorithmus verbessert. Aufgrund der beschränkten Anzahl an Iterationen wurden die Ausgangslösungen nicht zufällig, sondern mithilfe eines Monte-Carlo Suchverfahrens erzeugt. Schlussendlich wurden die einzelnen Metaheuristiken in einem gesamten Optimierungsprozess, unter Berücksichtigung der Stärken und Schwächen der einzelnen Verfahren für die Problemstellung, miteinander kombiniert.

Für das Modell des zu optimierenden HEVs konnte eine Treibstoffersparnis von etwa 33 Prozent im Vergleich zu einem konventionell betriebenen vergleichbaren Kraftfahrzeug erzielt

werden. Der Anteil unseres kombinierten Ansatzes liegt dabei etwa bei fünf Prozentpunkten. Die bereits vor der Diplomarbeit vorhandenen Optimierungsverfahren konnten vor allem aufgrund der hohen Anzahl an zu optimierenden Parametern keine nennenswerte Verbesserung bewirken. Weiters wurde der implementierte Algorithmus auch auf ein einfacheres Modell eines HEVs angewendet und es konnte gezeigt werden, dass auch hierbei bessere Ergebnisse als mit den vorhandenen Optimierungsverfahren erzielt werden können.

Abstract

For a computer model of the latest generation hybrid electric vehicle (HEV) the fuel consumption should be minimized considering the New European Driving Cycle (NEDC). A special feature of the fuel consumption measurement of a HEV in the NEDC is the requirement of a nearly identical state of charge (SOC) at the beginning and end of the driving cycle. For the vehicle among others the gear shifting strategy, thresholds for the electric cruise, and the load point increase should be optimized. For the optimization process about ten parameters will be selected. The goal is to find an optimization strategy making it possible to reliably find a parameter set that is close to optimal. Due to the high complexity of the model a direct determination of guaranteed optimal parameter values is almost impossible. Only specific parameters can be evaluated with relatively time-consuming simulations of the model.

Starting from simple existing standard optimization methods different, sometimes clearly more efficient algorithms were developed taking advantage of special properties of the problem. Furthermore, the achievable optimization potential for a model whose parameter set has been already determined empirically should be estimated.

As a first step the existing optimization methods were analyzed and used for comparative purposes. A framework has been developed giving access to the model of the simulation software. For the actual optimization process an algorithm was first developed that reduces the number of parameters and determines the parameter selection. Then a genetic algorithm (GA), a downhill simplex method, and an algorithm based on swarm intelligence (PSO) have been implemented. The GA's recombination operator accepts a better solution if it has a more balanced SOC. The simplex reduction in the downhill simplex method has not been implemented because it calculates all points of the new simplex and this mostly ends in worse objective function values. The best solution from the PSO algorithm has additionally been optimized with a surface-fitting algorithm. Given the limited number of iterations the solutions were not initialized randomly but by a Monte Carlo search procedure. Finally, the individual metaheuristics were combined to a hybrid optimization algorithm, taking into account the strengths and weaknesses of the single procedures.

For the model to be optimized a fuel saving of about 33 percent compared to a related conventionally powered vehicle could be achieved. The part our combined algorithm contributes is about five percent. The previously existing optimization methods could not significantly improve the model especially due to the high number of parameters to be optimized. Furthermore, it could be shown that the implemented algorithm achieves better results on a simpler HEV model, too.

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während meines Studiums und der Erstellung dieser Arbeit unterstützt haben.

Besonderer Dank gilt meinen Betreuern Herrn Prof. Dr. Günther Raidl und Herrn Univ.Ass. DI Mario Ruthmair, sowie den Mitarbeitern des Instituts für Fahrzeugantriebe & Automobiltechnik, insbesondere Herrn Prof. Dr. Bernhard Geringer, Herrn Prof. Dr. Thomas Lauer und Herrn Projektass. Michael Planer, MSc, die durch Ihre Unterstützung eine interdisziplinäre und interfakultäre Arbeit zu einem aktuell spannenden Thema erst möglich gemacht haben und mir immer mit Rat und Tat zur Seite gestanden sind.

Weiters möchte ich mich bei allen MitarbeiterInnen des Arbeitsbereichs für Algorithmen und Datenstrukturen bedanken, bei dem ich während meines Studiums mehrere Jahre als Tutor und Studienassistent tätig sein durfte und das dort durch viele interessante Gespräche erworbene Wissen bei der Arbeit gut einsetzen konnte. Besonders hervorheben möchte ich die TutorInnen, die meine Arbeit als Studienassistent durch ihre Mithilfe und ihr Engagement wesentlich erleichtert haben.

Abschließend möchte ich mich auch noch bei meiner Familie, meiner Freundin und meinen Freunden bedanken, die mich während des Studiums immer unterstützt haben und geduldig die ein oder andere Laune ertrugen.

Inhaltsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

KAPITEL 1

Einführung

1.1 Motivation und Zielsetzung

Die Idee, eine Arbeit im Bereich der Optimierung von Prozessen der Kraftfahrzeugtechnik zu verfassen, entstand durch den Besuch einschlägiger Lehrveranstaltungen im Bereich der Kraftfahrzeugtechnik und Optimierung. Die konkrete Problemstellung ergab sich durch ein aktuelles Hybridfahrzeug-Projekt des Instituts für Fahrzeugantriebe & Automobiltechnik (IFA)[1], mit welchem während des gesamten Projektes zusammengearbeitet wurde.

In der Automobilindustrie sind die Einstellungsmöglichkeiten des Kraftfahrzeuges in den letzten Jahren immer mehr gestiegen. Früher war es zumeist ausreichend innerhalb eines abgeschlossenen Systems einen Parameter wie zum Beispiel den Zündzeitpunkt am Prüfstand richtig einzustellen. Heutzutage gibt es eine Vielzahl an Parametern, die verändert werden können und teilweise stark voneinander abhängen. Beim Verbrennungsmotor wären das u.a. die Einspritzmenge, der Einspritzzeitpunkt, die Ventilsteuerzeiten und der Zündzeitpunkt. Dazu kann das Fahrzeug entweder vollständig oder partiell durch ein dazu gehöriges Computermodell simuliert werden.

Vor allem beim Treibstoffverbrauch gibt es in der Automobilindustrie immer noch Optimierungsbedarf. Ein Grund dafür liegt darin, dass der Trend in Richtung massiverer Fahrzeuge geht und zugleich die gesetzlichen Vorschriften einen immer geringeren Durchschnittsverbrauch vorsehen. Eine mögliche Lösung des Problems bieten Hybridelektrokraftfahrzeuge (Hybrid Electric Vehicles, HEV), siehe Kapitel 2.2. Sie profitieren nicht nur von der Weiterentwicklung der Verbrennungskraftmaschine (VKM), sondern es stehen zusätzlich mehr Freiheitsgrade bei der Energieumwandlung zur Verfügung und damit verbunden haben die Modelle auch erwartungsgemäß eine höhere Anzahl an zu optimierenden Parametern [Hofmann 09].

Um den Treibstoffverbrauch eines Fahrzeuges bewerten und vergleichen zu können, durchfährt ein Fahrzeug einen bestimmten Fahrzyklus. Dieser legt fest mit welchen Geschwindigkeiten und unter welchen Bedingungen ein Fahrzeug betrieben wird. Für die Ermittlung des

[1] IFA – Institut für Fahrzeugantriebe & Automobiltechnik, Technische Universität Wien, http://www.ifa.tuwien.ac.at/

Kraftstoffverbrauchs von Kraftfahrzeugen wird in der Europäischen Union der in Abbildung 1.1 gezeigte Neue Europäische Fahrzyklus (NEFZ) (siehe Kapitel 2.1) verwendet. Dieser besteht aus vier identen Stadtzyklen und einem Überlandzyklus. Die Gesamtzeit beträgt knapp 20 Minuten, wobei der Überlandzyklus etwa ein Drittel des gesamten Fahrzyklus ausmacht. Um den Treibstoffverbrauch sinnvoll bewerten zu können, ist es bei HEVs wichtig, dass zu Beginn und am Ende des Fahrzyklus der Ladezustand der Batterie (state of charge, SOC - siehe Kapitel 2.2) annähernd gleich ist. Dies muss in jedem Fall auch von den zu entwickelten Algorithmen berücksichtigt werden [Hofmann 09].

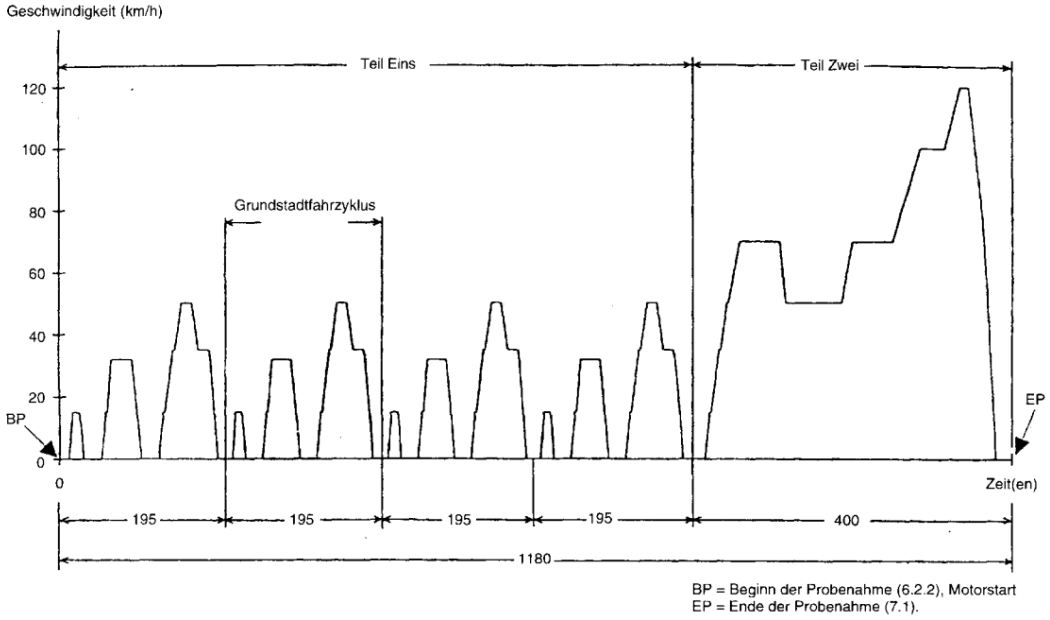

Abbildung 1.1: Neuer Europäischer Fahrzyklus (Bildquelle:[http://eur-lex.europa.eu/], Letzter Aufruf: 25.6.2011)

Ziel ist es nun für ein Modell, das den Antriebsstrang eines HEVs simuliert, eine Konfiguration zu finden, sodass einerseits der Treibstoffverbrauch möglichst gering ausfällt und andererseits der SOC ausgeglichen ist.

Das in dieser Arbeit im Speziellen betrachtete Fahrzeug (IFAHEV) ist ein sogenannter Mildhybrid (siehe Kapitel 2.2), welches sowohl mit der VKM, rein elektrisch, als auch in Kombination (Boosten) angetrieben werden kann. Weiters ist es möglich durch Lastpunktanhebung den Betriebspunkt der VKM zu verändern, sowie durch ein Start/Stopp System und die teilweise Rückgewinnung der Bremsenergie den Treibstoffverbrauch zu senken [Hofmann 09]. Da es sich beim IFAHEV um ein Fahrzeug der neuesten Generation handelt und noch in der Entwicklung ist, kann dieses in dieser Diplomarbeit weder genannt, noch können genaue Ergebnisse bekannt gegeben werden.

Das Fahrzeug wird mithilfe der Software GT-SUITE[2] (siehe Kapitel 2.3) vollständig simuliert. Das Modell für das Fahrzeug wird vom IFA zur Verfügung gestellt und kontinuierlich weiterentwickelt. Der Software GT-SUITE werden die Werte für die zu optimierenden Parameter übergeben und nach der Beendigung der Simulation erhält man den Treibstoffverbrauch und den SOC.

Für das IFAHEV soll die Lastpunktanhebung bzw. das Boosten (siehe Kapitel 2.2) und die Schwellwerte für den rein elektrischen Betrieb optimiert werden. Da das IFAHEV ein Automatikgetriebe hat, ist es erlaubt, für die Verbrauchsbestimmung im NEFZ auch die Schaltstrategie festzulegen und zu optimieren.

Für eine möglichst gute Einstellung der Parameter sind geeignete Optimierungsstrategien notwendig. Aufgrund dessen, dass der Treibstoffverbrauch und SOC durch keine direkt gegebene mathematische Funktion, sondern durch eine komplexe Simulation berechnet wird, können keine analytischen Methoden bzw. Gradientenstrategien (siehe Kapitel 2.4) auf das Problem angewendet werden. Um bei einer größeren Anzahl an zu optimierenden Parametern die Auswahl der einflussreichsten für die Optimierung sinnvoll bestimmen zu können, wurde eine Parameteranalyse entwickelt. In vergangenen Arbeiten konnten genetische Algorithmen (GA) und auf Schwarmintelligenz aufbauende Algorithmen (particle-swarm optimization, PSO) erfolgreich auf HEV-Modelle angewendet werden (siehe Kapitel 3), weshalb auch in dieser Arbeit zunächst auf diese Konzepte aufgebaut wurde. Beim GA wurde darauf geachtet, dass bei der Rekombination jeweils der Wert einer Lösung akzeptiert wird, der wahrscheinlicher zu einem ausgeglichenen SOC führt. Beim PSO wurde die beste Lösung nach einer gewissen Anzahl an Iterationen mit einem Surface-Fitting Algorithmus verbessert. Aufgrund der beschränkten Anzahl an Iterationen wurden die Ausgangslösungen nicht zufällig, sondern mithilfe eines Monte-Carlo-Suchverfahrens erzeugt. So kann die Wahrscheinlichkeit erhöht werden, dass zumindest ein paar Startlösungen mit ausgeglichenem SOC existieren.

Bei beiden Ansätzen ist der Grad an Diversifikation relativ hoch. Daher wurde weiters ein Downhill-Simplex-Verfahren implementiert, das pro Iterationsschritt nur einen begrenzten Bereich genauer betrachtet. Es wurde auf eine gesamte Verkleinerung des Simplex verzichtet, weil dadurch alle Punkte des Simplex neu berechnet werden müssten und diese, aufgrund eines unausgeglichenen SOCs, oft einen schlechteren Zielfunktionswert aufweisen. Trotz der Anpassungen an das Problem zeigten sich Schwächen der einzelnen Algorithmen. Der GA erwies sich als relativ robust, durch die freie Parameterwertzuweisung bei der Mutation können auch bei einem Pool aus schlechten Lösungen noch gute Ergebnisse erzielt werden. Andererseits ist es dadurch auch unwahrscheinlicher lokale Verbesserungen an Parametersets zu erzielen. Der PSO hängt wiederum sehr stark von den Ausgangslösungen ab; existiert darin keine gute Lösung, so liefert dieser Algorithmus auch schlechte Ergebnisse. Das Downhill-Simplex-Verfahren funktioniert nur dann gut, wenn sich die Lösungen in einem Bereich des Suchraums befinden, wo ein gutes lokales Optimum liegt. Ziel sollte es natürlich sein, dass nur ein Optimierungsverfahren letztlich zu Verfügung gestellt wird. Aufgrund dessen, dass jedes Verfahren verschiedene Stärken und Schwächen aufweist, wurden die einzelnen Metaheuristiken in einem hybriden Algorithmus adäquat kombiniert. Es hat sich gezeigt, dass dieser Ansatz für das IFAHEV als auch für ein weiteres Modell immer bessere Ergebnisse liefert als die einzelnen Heuristiken und die

[2]GT-SUITE ist eine Software der Gamma Technologies, Inc., http://www.gtisoft.com/

vor dieser Diplomarbeit bereits vorhandenen Ansätze. Vor allem kann eine im Vergleich zu den einzelnen Heuristiken höhere Robustheit beobachtet werden. Für das zu optimierende Modell des IFAHEVs konnte durch den kombinierten Ansatz die Treibstoffersparnis um weitere fünf Prozentpunkte verbessert werden. Dabei wurde das Modell bereits vorher teilweise durch die vorhandenen Methoden verbessert, wobei nur Parametergruppen zu zwei bis drei Parametern optimiert wurden, um u.a. einen ausgeglichenen SOC zu erhalten. Prinzipiell ist es möglich mit unserem Ansatz jedes beliebige Modell zu optimieren, da die Anpassungen an das HEV eine allgemeine Anwendung nicht ausschließen.

Um den Optimierungsprozess zu verwalten wurde weiters eine grafische Benutzeroberfläche erstellt. Mit dieser ist es einerseits möglich die Parametergrenzen und die Zielfunktion festzulegen und andererseits Informationen über den laufenden Optimierungsprozess anzuzeigen.

1.2 Aufbau und Kapitelübersicht

Am Anfang der Arbeit werden in Kapitel 2 die Grundlagen des HEVs erläutert, um die Besonderheiten der Optimierung eines HEVs erkennen zu können. Außerdem werden in dem Kapitel die zugrundeliegenden Strategien, die beim Optimierungsprozess verwendet wurden, erklärt. In weiterer Folge wird in Kapitel 3 der aktuelle Stand der Technik bezogen auf die bisherigen Arbeiten und der Fahrzeugsimulation in GT-SUITE und dessen Optimierungssoftware angeführt. Anschließend werden in Kapitel 4 die Optimierungsstrategien und Details der Implementierung erläutert. Die experimentellen Resultate für das zu optimierende Fahrzeugmodell sowie weitere Modelle, die zu Vergleichszwecken mit der bereits bestehenden Optimierungssoftware entwickelt wurden, werden in Kapitel 5 analysiert. Als Abschluss wird in Kapitel 6 ein Resümee der Arbeit sowie in Kapitel 7 ein Ausblick für zukünftige Projekte präsentiert.

KAPITEL 2

Grundlagen

2.1 Neuer Europäischer Fahrzyklus (NEFZ)

Ein Fahrzyklus legt fest mit welchen Geschwindigkeiten und unter welchen Bedingungen ein Fahrzeug zur Berechnung des Treibstoffverbrauchs und der Emissionen betrieben wird. Ziel ist es möglichst die Realität abzubilden. Durchgeführt wird dieser meist auf einem Rollenprüfstand, was es ermöglicht vergleichbare Ergebnisse zu erzielen. Weiters ist er auch Bestandteil von Abgasvorschriften. Seit dem 1. Jänner 1996 erfolgt die Ermittlung des Kraftstoffverbrauchs von Kraftfahrzeugen in der Europäischen Union im in Abbildung 1.1 gezeigten NEFZ. Der gesamte Zyklus dauert in etwa 20 Minuten und besteht aus vier hintereinander folgenden identen Stadtzyklen und einem Überlandzyklus. Bei Fahrzeugen mit Automatikgetriebe kann die Schaltstrategie beliebig gewählt werden, bei Fahrzeugen mit manueller Schaltung wird diese vorgeschrieben [Wirtschaftsgemeinschaft 07].

2.2 Hybridelektrokraftfahrzeuge (Hybrid Electric Vehicles, HEV)

Wie in [Hofmann 09] beschrieben gibt es trotz der langjährigen Erfahrung der Automobilindustrie vor allem beim Treibstoffverbrauch und dem damit verbundenen CO_2-Ausstoß immer noch Optimierungsbedarf. Der Trend für den Kauf eines Neuwagens geht in Richtung größerer und stärkerer Fahrzeuge. Die Vorgaben der Europäischen Union sehen allerdings einen immer geringeren Durchschnittsverbrauch für Neuwagen vor. Diese Vorgaben können für diese Fahrzeuge durch die ausschließliche Weiterentwicklung des Antriebsstrangs nicht erreicht werden. Es werden daher Technologien benötigt, mit denen auch schwere, große und leistungsstarke Fahrzeuge möglichst effizient betrieben werden können. HEVs stellen diesbezüglich eine interessante Alternative zu konventionellen Autos mit VKM dar. Sie profitieren nicht nur von der Weiterentwicklung der VKM, sondern es stehen auch mehr Freiheitsgrade bei der Energieumwandlung zur Verfügung. Weiters kann auch die Bremsenergie teilweise zurückgewonnen und gespeichert werden.

Bei einem HEV müssen mindestens zwei Energieumwandler und zwei Energiespeichersysteme vorhanden sein. Als Speicher werden in der Praxis chemische Energieträger wie z.B. Benzin und wiederaufladbare elektrische Energiespeicher eingesetzt. Die Energie wird dann jeweils mithilfe eines Motors in mechanische Energie umgewandelt. Weiters kann die mechanische Energie auch noch in elektrische Energie umgewandelt werden. Bei konventionellen Ottomotoren erfolgt die Steuerung der Last durch Drosselung der angesaugten Luft. Dies führt im Teillastbetrieb zu einem relativ schlechten Wirkungsgrad und einem damit verbundenem höheren spezifischen Treibstoffverbrauch. Je nach Konzept kann bei HEVs durch Lastpunktanhebung und rein elektrischen Betrieb die Effizienz erhöht werden. Ein weiteres Einsparungspotenzial bieten Start/Stopp Systeme. Die Erklärung zu den einzelnen Begriffen folgt später im Kapitel.

HEVs lassen sich nach der elektrischen Leistung in Micro-, Mild-, und Fullhybride einteilen [van Basshuysen 11]. Bei einem Microhybrid ist nur ein Start-Stopp System vorhanden mit einer Leistung des Elektromotors (E-Maschine) von etwa fünf Kilowatt. Der Mildhybrid hat eine elektrische Leistung von etwa 20 Kilowatt und kann somit auch zum Antrieb des Fahrzeugs genutzt werden. Weiters ist auch eine Unterstützung der VKM möglich und die Bremsenergie kann zumindest teilweise genutzt werden. Beim Fullhybrid sind Leistungen der E-Maschine von bis zu 200 Kilowatt möglich. Dadurch kann auch nur mit der E-Maschine alleine gefahren werden und es können alle Hybridantriebsfunktionen noch effizienter genutzt werden. Beim IFAHEV handelt es sich um einen Mildhybrid, bei dem es auch möglich ist, rein elektrisch zu fahren. Weiters können HEVs noch in der Art der Anordnung des Elektro- und Verbrennungsmotors unterschieden werden. Beim seriellen Antrieb treibt die VKM direkt einen Generator an, der die elektrische Energie der E-Maschine liefert oder damit die Batterie ladet. Für den Antrieb des Fahrzeuges wir nur die E-Maschine verwendet. Beim parallelen Antrieb können sowohl die E-Maschine als auch die VKM zum Antrieb beitragen. Weiters existieren Mischhybride, bei denen beide Varianten möglich sind. Beim IFAHEV handelt es sich um einen parallelen Hybridantrieb. Im Folgenden werden die wichtigsten Begriffe eines HEVs lt. [Hofmann 09] erklärt.

Batterieladezustand (state of charge, SOC)

Der SOC gibt den Ladezustand der Batterie in Prozent an. Dieser muss sich innerhalb festgelegter Grenzen befinden. Je nach Ladezustand können nur bestimmte Betriebsstrategien angewendet werden. Ist die obere Grenze des SOCs erreicht, so können keine Strategien mehr angewendet werden, die zur Aufladung der Batterie führen. Sinkt der SOC unterhalb des Betriebsbereichs, so müssen Strategien angewendet werden, die zur Erhöhung des SOCs führen.

Lastpunktanhebung (LPA)

Bei der Lastpunktanhebung wird durch Erhöhung der Last mehr als die benötigte Leistung für den Fahrzeugantrieb erzeugt und damit der Energiespeicher über die E-Maschine aufgeladen. Dies soll die Effizienz des Antriebsstrangs steigern und einen niedrigeren spezifischen Treibstoffverbrauch zur Folge haben. Um dies beurteilen zu können, müssen die gesamten Leistungsflüsse betrachtet werden. Der spezifische Treibstoffverbrauch be lässt sich durch die Formel 2.1 berechnen.

$$be = \frac{BE}{P_{soll} + P_{EM} \cdot \eta_{gen} \cdot \eta_{batein} \cdot \eta_{bataus} \cdot \eta_{mot}} \tag{2.1}$$

be	...	Spezifischer Treibstoffverbrauch bei konstanter Drehzahl
BE	...	Absoluter Verbrauch der VKM
P_{soll}	...	Leistung der VKM
P_{EM}	...	Leistung der E-Maschine
η_{gen}	...	Wirkungsgrad der E-Maschine im generatorischen Betrieb
η_{batein}	...	Einspeisungswirkungsgrad der Batterie
η_{bataus}	...	Ausspeisungswirkungsgrad der Batterie
η_{mot}	...	Wirkungsgrad der E-Maschine für den Einsatz der elektrischen Energie

Abbildung 2.1 zeigt das Prinzip der Lastpunktanhebung bei konstanter Drehzahl n. M_{soll} gibt das erforderliche Drehmoment an und M_{VKM} das tatsächliche Drehmoment der VKM. Die Differenz beider Werte muss dem Drehmoment M_{EM} an der E-Maschine entsprechen. Dies entspricht der zusätzlichen Last an der VKM. Die Leistung $P[kW]$ lässt sich als Produkt von Drehmoment $M[Nm]$ und Drehzahl $n[\frac{1}{min}]$ mit der Formel 2.2 berechnen [Schicker 02].

$$P = \frac{M \cdot n \cdot 2 \cdot \pi}{60 \cdot 1000} \tag{2.2}$$

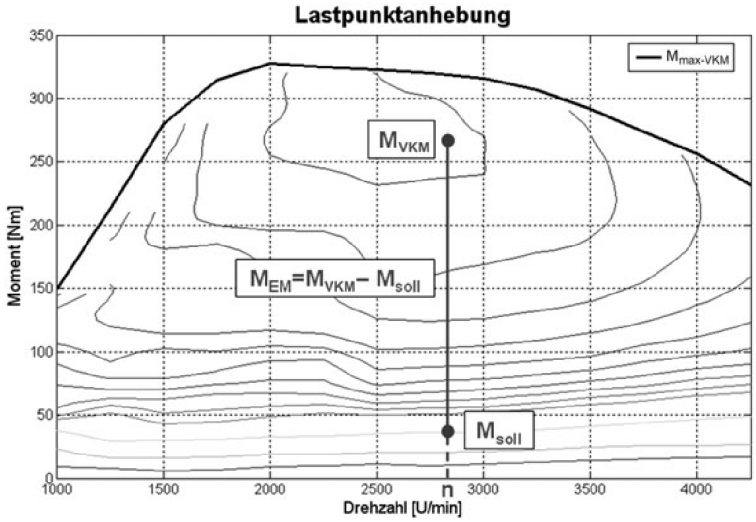

Abbildung 2.1: Lastpunktanhebung bei konstanter Drehzahl (Bildquelle: [Hofmann 09])

Boosten

Beim in Abbildung 2.2 dargestellten Boosten wird das Moment der E-Maschine M_{EM} bzw. $M_{EM\text{-}max}$ zusätzlich zum Moment der VKM M_{beopt} bzw. $M_{VKM\text{-}max}$ an das Getriebe übertra-

gen. Dies kann einerseits dazu dienen höhere Antriebsmomente $M_{VKM\text{-}max} + M_{EM\text{-}max}$ realisieren, oder andererseits eine Lastpunktabsenkung der VKM $M_{soll} \rightarrow M_{beopt}$ zu ermöglichen, die wiederum zu einem günstigeren spezifischen Treibstoffverbrauch führen kann.

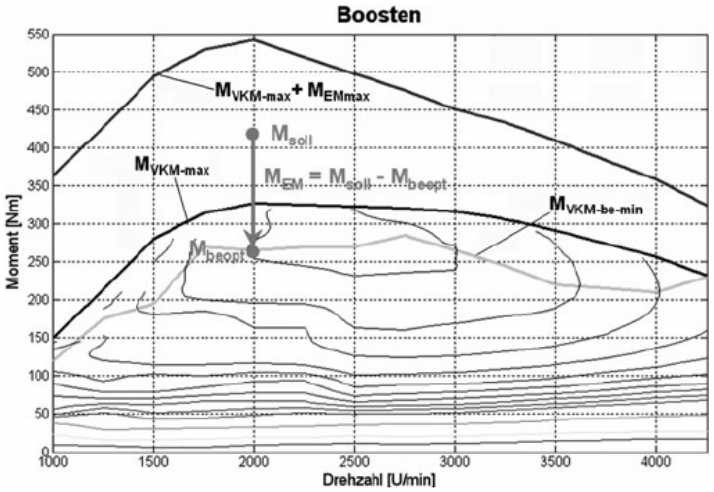

Abbildung 2.2: Prinziperklärung Boosten (Bildquelle: [Hofmann 09])

Schaltstrategie

Da das zu optimierende Fahrzeug ein Automatikgetriebe besitzt, darf auch die Schaltstrategie für den NEFZ selbst gewählt werden. Diese wird in Abhängigkeit der Drehzahl festgelegt. Für jeden Gang, von dem aus in einen nächst höheren Gang geschaltet werden kann, wird jeweils eine Drehzahl festgelegt, bei welcher dieser Schaltvorgang erfolgen soll. Für das Schalten in den nächst niedrigeren Gang wird nur eine Drehzahl für alle möglichen Schaltvorgänge definiert. Grund dafür ist, dass das Schalten in den nächst niedrigeren Gang einen wesentlich geringeren Einfluss auf den Treibstoffverbrauch hat, da nur bei Verzögerung des Fahrzeuges einen Gang hinunter geschaltet wird und somit der Schaltvorgang im Wesentlichen nur die Rekuperation betrifft.

Beim IFAHEV ist ein paralleles Hybridkonzept verfolgt worden. Sowohl die VKM als auch die E-Maschine können zum Antrieb beitragen und werden mit demselben Getriebe betrieben. Somit wirkt sich die Schaltstrategie auf beide Antriebskonzepte aus. Abbildung 2.3 zeigt einen möglichen Drehmomentverlauf für beide Konzepte. Die E-Maschine hat ein konstantes maximales Drehmoment bis etwa 2000 $\frac{U}{min}$, danach nimmt dieses kontinuierlich ab. Das Drehmoment der VKM steigt relativ steil bis etwa 1500 $\frac{U}{min}$ und bleibt bis 3000 $\frac{U}{min}$ in etwa konstant und nimmt dann wieder langsam ab. In diesem Fall würde sich die E-Maschine für rein elektrisches Fahren für geringe Lasten und niedrige Drehzahlen eignen. Das Moment für den optimalen spezifischen Treibstoffverbrauch für die VKM ergibt sich entlang der Linie $M_{VKM\text{-}be\text{-}min}$.

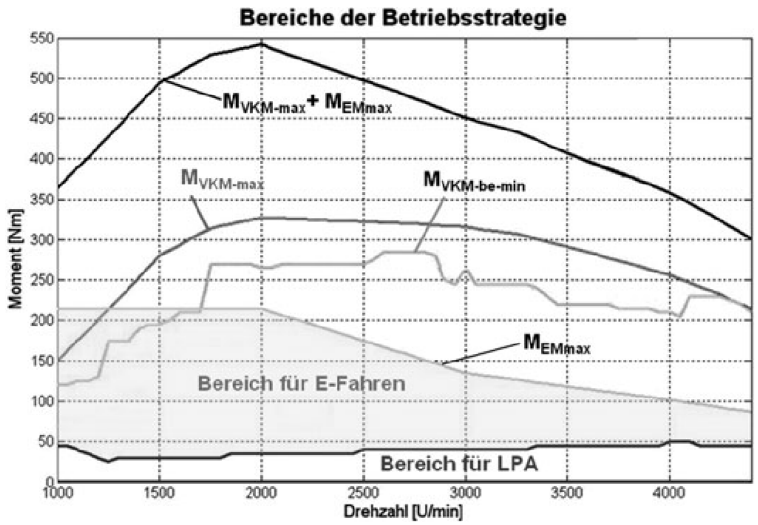

Abbildung 2.3: Bereiche der Betriebsstrategie (Bildquelle: [Hofmann 09])

Rekuperation

Bei Rekuperation wird ein Teil der Bremsenergie, die sonst in Wärme umgewandelt werden würde, für die Aufladung der Batterie genutzt. Die Nutzbarkeit hängt dabei sowohl vom Fahrwiderstand als auch von den Verlusten am Antriebsstrang ab. Weiters spielt auch noch der SOC eine wichtige Rolle, da dieser nicht über einer gewissen Grenze liegen darf. Grundsätzlich können zwei Arten der Rekuperation unterschieden werden. Einerseits die Bremsung, die durch Betätigung des Bremspedals erfolgt, andererseits die Simulation des Schleppbetriebes der VKM. Bei konventionellen Fahrzeugen wird im Schleppbetrieb die VKM für die Verzögerung des Fahrzeuges genutzt.

Start/Stopp

Im Stillstand des Fahrzeuges kann die VKM ausgeschaltet werden. Vor allem bei "Stop and Go"-Verkehr kann dies den Treibstoffverbrauch und die Emissionen erheblich senken.

Elektrisches Fahren

Beim elektrischen Fahren wird die erforderliche Leistung nur von der E-Maschine erzielt. Dazu ist es u.a. notwendig, dass das erforderliche Moment von der E-Maschine bereit gestellt werden kann und der SOC ausreichend hoch ist. Abbildung 2.3 zeigt einen möglichen Bereich für elektrisches Fahren in Abhängigkeit der Drehzahl und des benötigten Moments.

VKM Fahren

Das Fahrzeug wird dann ausschließlich durch die VKM angetrieben, wenn alle weiteren Betriebsstrategien nicht möglich oder ineffizient sind. Abbildung 2.4 zeigt die Betriebsmodi in Abhängigkeit der Geschwindigkeit und des benötigten Drehmoments.

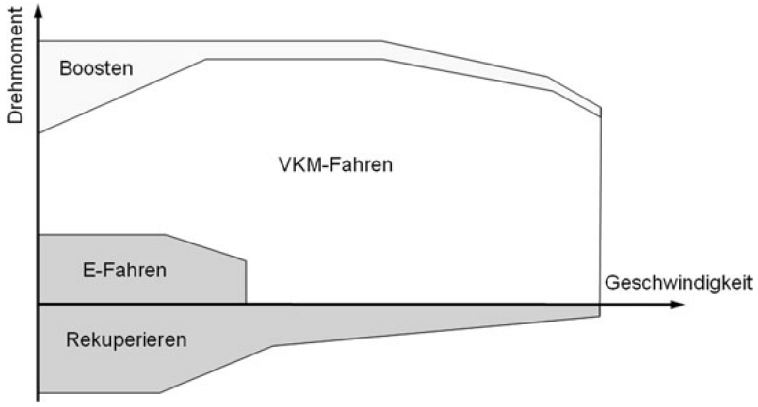

Abbildung 2.4: Betriebszustände in Abhängigkeit der Geschwindigkeit und des Drehmoments (Bildquelle: [Hofmann 09])

2.3 Simulation des HEVs mit GT-SUITE

Um den Treibstoffverbrauch im NEFZ zu berechnen, besteht die Möglichkeit einer Computersimulation des Fahrzeuges. Für das IFAHEV wurde GT-SUITE[1] V7.1 für die Modellierung verwendet, da der komplette Antriebsstrang des Fahrzeugs für einen vorgegeben Fahrzyklus berücksichtigt werden muss und diese Software eine führende Position in diesem Bereich aufweisen kann. Mit dieser Software ist es möglich das Fahrzeug im gesamten Fahrzyklus mit entsprechenden Bedingungen, wie z.B. Außentemperatur und Luftdruck, zu simulieren. Der NEFZ kann für das zu optimierende Modell mit einem Intel Xeon X5570 Prozessor mit 2.93 GHz in etwa sieben Minuten berechnet werden. Abbildung 2.5 zeigt einen Auszug eines Modells.

Damit möglichst gute Ergebnisse durch die Simulation erzielt werden können, ist es notwendig, dass das Modell an das reale Fahrzeug angepasst wird. Es gibt aber auch bereits fertige Gruppen, wie z.B. Motor, Getriebe und Karosserie, die man für erste Berechnungen anwenden kann. In weiterer Folge ist es aber notwendig diese durch eigene Modelle oder Modelle der Hersteller zu ersetzen. Um die Berechnungszeit zu beschleunigen, können auch komplexe Berechnungen durch Kennfelder, die vom Motorprüfstand gewonnen oder vom Hersteller zur Verfügung gestellt wurden, ersetzt werden. Es gibt auch die Möglichkeit einzelne Komponenten durch eine dreidimensionale Strömungssimulation (CFD) zu berechnen. Dies führt meist zu sehr genauen Ergebnissen, die allerdings mit sehr viel Rechenaufwand verbunden sind. Um

[1] GT-SUITE ist eine Software der Gamma Technologies, Inc.,http://www.gtisoft.com/

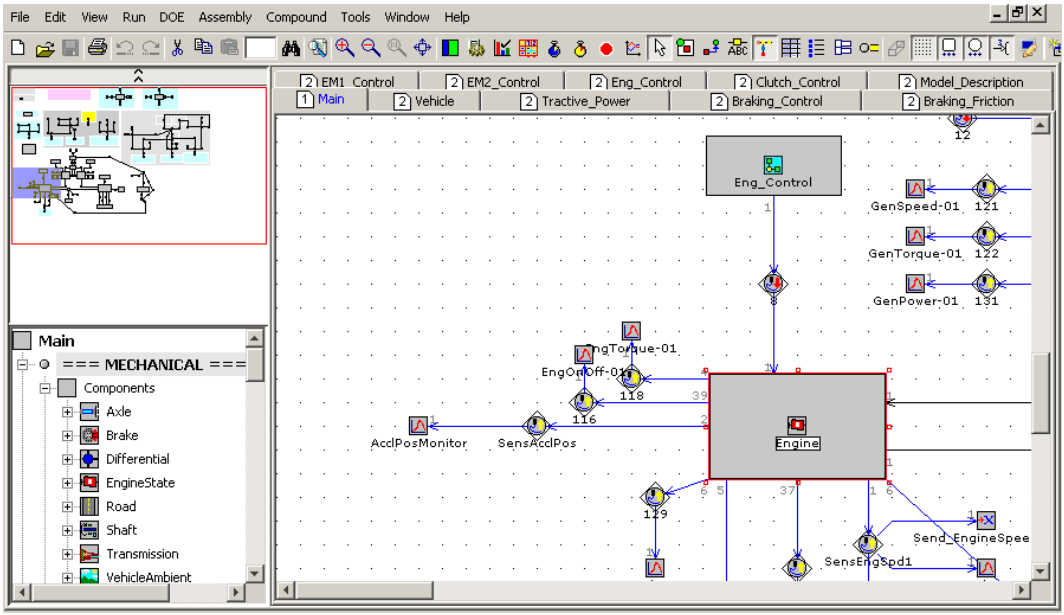

Abbildung 2.5: GT-SUITE

die Berechnungszeit möglichst gering zu halten, wurde beim IFAHEV-Modell wenn möglich auf Kennfelder zurückgegriffen. Ein Beispiel für den Einsatz eines Kennfeldes ist die Verbrauchsbestimmung einer VKM. Dabei werden, bei bestimmten Drehzahlen und Lasten, Verbrauchsmessungen durchgeführt, die dann in einer Tabelle gespeichert werden. Um auf bestimmte Drehzahl-Last-Kombinationen zugreifen zu können, müssen die vorhandenen Werte interpoliert werden.

GT-SUITE Solver

In der Modellierungssoftware ist ein Kommandozeilen-basierter Solver integriert, mit dem es möglich ist ein Modell, das mit GT-Suite entworfen wurde, zu berechnen. Benötigt wird dazu die Eingabedatei $\langle projektname \rangle$.dat und die Modelldatei $\langle projektname \rangle$.gtm eines GT-Suite Projektes. Nach dem Ausführen des Solvers wird dann eine Ausgabedatei $\langle projektname \rangle$.trn erzeugt, die es ermöglicht im Modell definierte Ausgabedaten für die zu entwickelnden Algorithmen zu nutzen. Damit man mit einer externen Software die Parameter verändern kann, muss die textbasierte Eingabedatei $\langle projektname \rangle$.dat verändert werden.

Optimierungsverfahren von GT-SUITE

In GT-SUITE stehen zwei Standard-Optimierungsstrategien zur Verfügung. Zum einen existiert eine Optimierungssoftware, die auf einem Direct-Search Verfahren [Hooke 61] beruht und zum anderen Design of Experiments (DOE) [Myers 09], das auf Basis einer Vielzahl an berechneten Lösungen ein mathematisches Modell konstruiert.

Das direkte Optimierungsverfahren

Es kann auf einen Wert, ein Maximum oder ein Minimum optimiert werden. Das Verfahren wurde so ausgelegt, dass es dann korrekt funktioniert, wenn nur ein Extremwert innerhalb des vorgegebenen Bereichs liegt. In der Software muss für jeden zu optimierenden Parameter des Modells ein Wert festgelegt werden. Diese Werte dienen dem Verfahren als Ausgangsparameterset. Die Parameter werden schrittweise von einem Ausgangswert um einen konstanten Betrag erhöht und vermindert. Jedem Parameter werden somit drei Werte zugewiesen. Anschließend wird jede mögliche Kombination der Parametereinstellung berechnet. Somit ergeben sich zum Beispiel für vier zu optimierende Parameter $3^4 = 81$ zu berechnende Lösungen pro Iterationsschritt. Für die weitere Vorgehensweise stehen zwei Varianten zur Verfügung, diskrete Gitter und die Brent-Methode.

- Diskrete Gitter: Nach jedem Schritt wird die Schrittweite halbiert und nur mehr in dem Bereich weiter gesucht, wo die beste Lösung im jeweiligen Iterationsschritt gefunden wurde. Hat beispielsweise Parameter p_1 den Ausgangswert 10 und den Bereich 12, so kann er die Werte 4,10 und 16 annehmen. Ist nach dem Iterationsschritt bei der besten Lösung der Parameterwert $p_1 = 16$ so wird im nächsten Iterationsschritt der Bereich halbiert und der Parameter p_1 kann die Werte 10, 13 und 16 annehmen. Das Verfahren wird solange wiederholt, bis der Bereich eine Mindestgröße unterschritten hat oder die Anzahl an durchzuführenden Iterationen erreicht ist.

- Brent-Methode: Bei dieser Methode wird die Schrittweite mithilfe einer parabolischen Funktion oder dem goldenen Schnitt berechnet. Dies führt vor allem bei steigender Parameteranzahl meist schneller zu einer lokal besten Lösung innerhalb des vorgegebenen Bereichs als mit einer konstanten Schrittweite. Nachteil ist die größere Wahrscheinlichkeit aus einem lokalen Optimum, das nicht das globale Optimum ist, nicht mehr entkommen zu können.

Design of Experiments

Jeder kontinuierliche Parameter bekommt gewisse diskrete Werte zugewiesen. Die Anzahl und die Diskretisierung hängt von dem ausgewählten Ansatz ab. Es stehen im wesentlichen Full Factorial, D-Optimum und Latin Hypercube zur Verfügung:

- Full Factorial: Hierbei muss ein Bereich und eine gewisse Anzahl an diskreten Werten pro Parameter innerhalb dieses Bereichs vorgegeben werden. Die Lösungen setzen sich aus allen möglichen Kombinationen der festgelegten Werte zusammen. Gibt es z.B. drei Parameter mit jeweils fünf diskreten Werten so müssen $5^3 = 125$ Lösungen erzeugt werden.

- D-Optimum: Im Vergleich zum vollständigen Ansatz, bei dem alle möglichen Kombinationen berechnet werden, kann hierbei die Anzahl der Lösungen begrenzt werden. Die Auswahl der Kombinationen der festgelegten Werte hängt dabei von der später im Verfahren verwendeten Ersatzfunktion ab. Somit muss vor den Lösungsberechnungen angegeben werden, ob es sich bei der Ersatzfunktion um ein Polynom ersten, zweiten oder dritten

Grades handeln soll. Erst in Abhängigkeit der angegebenen Anzahl an zu berechnenden Lösungen und des Polynomgrades wird dann eine Auswahl an Parameterwerten getroffen und diese berechnet.

- Latin Hypercube: Dieser Ansatz ist ähnlich dem D-Optimum Ansatz, nur dass die Struktur der Ersatzfunktion vor der Lösungsberechnung noch nicht angegeben werden muss. Es wird im Lösungsraum eine möglichst gut verteilte Auswahl an Lösungen getroffen und diese berechnet. Die Wahl der Ersatzfunktion erfolgt erst nach der Berechnung aller Lösungen. Somit ist es im Unterschied zur D-Optimum Methode möglich, die Art der Ersatzfunktion zu ändern ohne neue Lösungen berechnen zu müssen, was in der Regel mit einem enormen Rechenaufwand verbunden ist. Auf den Rechenaufwand bezogen, kann im Vergleich dazu die Bestimmung der Koeffizienten der Ersatzfunktion vernachlässigt werden.

Mit Hilfe der erzeugten Ersatzfunktion ist es dann in weiterer Folge möglich, Parametersets wesentlich schneller näherungsweise zu bewerten. Als Verbesserungsstrategie wird bei der verwendeten Software auf das mathematische Modell ein GA [Goldberg 89] angewendet.

2.4 Optimierungsverfahren

Falls nicht anders angegeben beziehen sich alle Optimierungsverfahren in dieser Arbeit auf ein Maximierungsproblem. Allgemein kann ein Optimierungsproblem, bei dem ein Maximum gesucht wird, wie folgt beschrieben werden:
Ein Parameter-Tupel $p^* = (p_1 \dots p_n)$ heißt Optimum wenn der Wert $f(p^*)$ der Zielfunktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$ maximal ist, d.h. $f(p^*) \geq f(p)$, $\forall\ p$, und die Nebenbedingungen erfüllt sind. Zu diesen zählen die Begrenzung der einzelnen Parameterwerte eines Tupels, sowie Gleichheits- und Ungleichheitsnebenbedingungen [Meywerk 07].

Da es sich bei dem hier untersuchten Modell um ein nichtlineares System handelt und jeder Parameter einen beliebigen Wert $x \in \mathbb{R}$ innerhalb gegebener Grenzen annehmen kann, werden Verfahren zur Optimierung von kontinuierlichen Parametern für nichtlineare Probleme angewandt. Verfahren der nichtlinearen Optimierung lassen sich grob in Such- und Gradientenstrategien einteilen.

Suchstrategien

Suchstrategien sind im allgemeinen nicht streng auf Konvergenz überprüfbar, aber gut für unstetige Zielfunktionen geeignet. Die eigentliche Zielfunktion muss nicht bekannt sein, lediglich die Fitness eines Parametersets muss berechnet werden können [Meywerk 07].

Monte-Carlo Suchverfahren

Beim Monte-Carlo Suchverfahren werden innerhalb eines bestimmten Bereiches Zufallslösungen berechnet. Abhängig von der besten Zufallslösung wird der Bereich neu gesetzt und verkleinert. Der Algorithmus bricht ab, wenn der Bereich eine bestimmte Mindestgröße unterschritten hat [Meywerk 07].

Downhill-Simplex-Verfahren

Das Verfahren, auch als Nelder-Mead Methode bekannt, basiert auf einem Simplex. Ein v-Simplex ist ein Polytop mit der Dimension v. Das Simplex wird somit in einem v-dimensionalen Raum aus $v+1$ Punkten aufgespannt, welches die konvexe Hülle dieser bildet. Jeder Punkt entspricht einem bestimmten Parameterset, zu welchem ein Funktionswert berechnet wird. Der Algorithmus kommt ohne Ableitungen der Zielfunktion aus. Dies ist für die Optimierung des Modells wichtig, da Teile der Zielfunktion nur durch den GT-SUITE Solver berechnet werden. Durch Vergleiche der Funktionswerte im Suchraum wird die Tendenz der Funktionswerte und des Gradienten Richtung Optimum angenähert. Der Punkt mit dem höchsten und niedrigsten Funktionswert wird bestimmt, wobei der Punkt mit dem höchsten Wert der bis zu diesem Zeitpunkt jeweils besten gefundenen Lösung entspricht. Der Punkt mit dem niedrigstem Wert wird durch einen neuen ersetzt. Algorithmus 1 zeigt einen Iterationsschritt des Algorithmus. Die Lösung m wird dadurch gebildet, dass von allen Lösungen, bis auf die schlechteste, das arithmetische Mittel der n Parametereinstellungen gebildet wird. Der Faktor a kann einen beliebigen positiven Wert annehmen. Er bestimmt, wie stark sich die Parameterwerte der Lösung x' von den Parameterwerten der Mittelwertlösung m entfernen. Die Faktoren b und c können einen beliebigen Wert im Bereich $]0, 1[$ annehmen und steuern das Ausmaß der Verkleinerung des Simplex. Der Algorithmus ist bei Unstetigkeiten relativ robust, jedoch kann er auch schnell zu lokalen Optima konvergieren, die nicht das globale Optimum darstellen [Nelder 65].

Jacob-Verfahren

Dieses Verfahren nutzt sowohl Gradienten- als auch Krümmungsinformationen. Es startet mit einer Hauptsuchrichtung, in der die Zielfunktion durch eine quadratische Funktion ersetzt wird, indem sie die Zielfunktion an drei Positionen interpoliert. Von dieser wird das Minimum bestimmt, von welchem aus in weiterer Folge $n-1$ Nebensuchrichtungen mithilfe eines Orthogonalisierungsverfahren ermittelt werden, wobei n die Parameteranzahl darstellt. Für diese Nebensuchrichtungen wird wiederum eine quadratische Funktion berechnet und das Minimum dieser, sowie der erste Punkt der ersten Hauptsuchrichtung, ergeben die weiteren Hauptsuchrichtungen. Dadurch, dass die quadratische Funktion durch drei Punkte approximiert wird, muss die eigentliche Zielfunktion nicht bekannt sein [Meywerk 07].

Genetische Algorithmen (GA)

GAs sind der biologischen Evolution nachempfunden. Eine Menge an Individuen bilden die Population. Aus dieser werden mithilfe eines Gütekriteriums bestimmte Individuen ausgewählt. Diese werden miteinander kombiniert und leicht verändert um daraus neue Generationen von Individuen zu erstellen [Michalewicz 96].

Ein GA beinhaltet grundsätzlich folgende Schritte [Goldberg 89]:

1. Erstellung der ersten Generation durch die Erzeugung unterschiedlicher Individuen.

2. Für jedes Individuum wird mithilfe einer Fitness-Funktion der Gütewert bestimmt.

Algorithmus 1 Downhill-Simplex Prinzip

$x =$ worst solution;
$y =$ second worst solution;
$m =$ mean value of all solutions except x;
$best =$ best solution;
// x' = reflexion of solution x on mean solution m with factor a ;
$x'_{1...n} = m_{1...n} - a \cdot (x_{1...n} - m_{1...n})$;
if $f(x') > f(best)$ **then**
  choose $a' \in \mathbb{N}$, $a' > a$.
  // $x'' =$ reflexion of solution x on mean solution m with factor $a' > a$;
  $x''_{1...n} = m_{1...n} - a' \cdot (x_{1...n} - m_{1...n})$;
  $x = best$;
  if $f(x'') > f(x')$ **then**
    $best = x''$;
  else
    $best = x'$;
  end if
else
  if $f(x') > f(y)$ **then**
    $x = y$;
    $y = x'$;
  else
    if $f(x') > f(x)$ **then**
      $x = x'$;
    end if
    // $x''' =$ take x nearer to mean solution m with factor b;
    $x'''_{1...n} = m_{1...n} + (1 - b) \cdot (x_{1...n} - m_{1...n})$;
    if $f(x''') < f(x)$ **then**
      // all points $v + 1$ get closer to the best solution $best$ with factor c.
      for all solutions z of simplex **do**
        $z_{1...n} = best_{1...n} + (1 - c) \cdot (z_{1...n} - best_{1...n})$;
      end for
    end if
  end if
end if

3. Selektion: Zufällige Auswahl von Individuen unter Berücksichtigung des Gütewerts, d.h. dass bessere Lösungen im Allgemeinen häufiger selektiert werden.

4. Rekombination: Parameter bzw. Werte der Auserwählten werden vermischt und neue Individuen gebildet.

5. Mutation: Werte der neuen Individuen werden zufällig verändert.

6. Aus den alten und neuen Individuen wird eine neue Generation erstellt.

Simulated-Annealing (SA)

Die Idee der simulierten Abkühlung (Simulated-Annealing) [Kirkpatrick 83] kommt aus der Werkstoffkunde. Nach dem starken Erhitzen beim Glühen eines Metalls brauchen die Moleküle Zeit sich zu ordnen um ein stabiles Kristallgitter zu bilden. Die Temperatur entspricht beim Algorithmus der Wahrscheinlichkeit, mit der sich eine Nachbarschaftslösung auch verschlechtern darf. Dadurch ist es im Vergleich zur einfachen lokalen Suche möglich einem lokalem Optimum zu entkommen. Initialisiert wird der Algorithmus mit einer Startlösung x, einer Fitness-Funktion $f(x)$ und einer Temperatur $T = T_{max}$. In jedem Iterationsschritt wird eine Lösung x' aus der Nachbarschaft $N(x)$ abgeleitet. Ist die Lösung x' besser als x, dann wird sie in jedem Fall als neue Ausgangslösung akzeptiert $(x = x')$. Falls nicht, dann wird x' mit der Wahrscheinlichkeit $e^{\frac{-f(x')-f(x)}{T}}$ als neue Ausgangslösung akzeptiert (Metropolis Kriterium). Die Temperatur T wird nach einer gewissen Anzahl an Iterationen gesenkt. Abgebrochen wird der Algorithmus entweder

- nach einer fixen Anzahl an Durchläufen,
- nachdem eine Temperaturgrenze erreicht wurde,
- nachdem sich die Lösung x über mehrere Iterationen nicht mehr verändert hat, oder
- nachdem eine ausreichende Fitness erreicht wurde.

Tabu-Suche (TS)

Um zu verhindern, dass beim Traversieren des Lösungsraums bereits besuchte Lösungen erneut in Betracht kommen, wird währenddessen eine Tabu-Liste erstellt [Glover 98]. Lösungen in dieser Liste verweilen eine gewisse Dauer t_L in der Liste und dürfen in dieser Zeit nicht als neue Ausgangslösung verwendet werden. Weiters ist es auch möglich nur einzelne Attribute, die von einem Zug verändert wurden, in die Tabu-Liste einzufügen. Dies verhindert die unmittelbare Umkehrung eines Zuges. Initialisiert wird der Algorithmus mit einer Startlösung x, einer Fitness-Funktion $f(x)$ und einer Tabu-Liste $T_L = \{\}$ mit einer Dauer t_L. In jedem Iterationsschritt wird eine Lösung x' aus der Nachbarschaft $N(x)$ abgeleitet, wobei $|N(x)| \geq 1$ gelten muss und x' die beste Lösung aus der Nachbarschaft ist, die sich nicht in der Tabu-Liste befindet. Anschließend wird die ursprüngliche Lösung x in die Tabu-Liste eingefügt und alle Lösungen, die älter als t_L sind, aus der Tabu-Liste entfernt.

Particleswarm-Optimization (PSO)

Dieses Optimierungsverfahren wurde ursprünglich aus dem Verhalten von Vogel- und Fischschwärmen abgeleitet [Kennedy 95]. Jede Lösung s_j, $j = 1 \dots d$, entspricht einem Individuum des Schwarms der Größe d, das sich innerhalb des Suchraums bewegen kann. Die Bewegung hängt einerseits von der besten Lösung, die das Individuum selbst kennt, ab und andererseits von der besten Lösung des gesamten Schwarms. Bei der Initialisierung wird jede Lösung s_j zufällig innerhalb gewisser Grenzen im Suchraum positioniert. Die beste bekannte Lösung s_j^{best} für das Individuum entspricht zu Beginn s_j. Falls s_j besser als die bisher beste Lösung $best$ ist, wird diese durch $best = s_j$ aktualisiert. Weiters wird für jede Lösung ein Vektor $\vec{f_j}$ definiert. Dieser gibt an, in welche Suchrichtung und mit welcher Geschwindigkeit sich ein Individuum im Suchraum bewegt. Bis ein bestimmtes Abbruchkriterium erfüllt ist, wird kontinuierlich der Vektor $\vec{f_j}$ und die Position, in Abhängigkeit der besten bekannten Position jeder Lösung s_j^{best} und der besten Lösung des Schwarms $best$, verändert. Ist das Abbruchkriterium erfüllt, entspricht $best$ der besten gefundenen Lösung.

Gradientenstrategien

Aufgrund dessen, dass das Modell nicht durch eine mathematische Funktion berechenbar ist, ist es auch nicht möglich Ableitungen zu bilden. Somit können Gradientenstrategien nicht direkt auf das Modell angewendet werden. Allerdings kann, wie bei DOE, eine differenzierbare Ersatzfunktion für das Modell erstellt werden, worauf auch Gradientenstrategien angewendet werden können.

Newton-Verfahren

Die Idee des Verfahrens [Nocedal 99] ist die Tangente in einem Ausgangspunkt x_u zu bestimmen und die Nullstelle x_{u+1} der Tangente wie in Formel 2.3 ersichtlich als Näherung der Nullstelle der Funktion zu verwenden. Diese dient wiederum als Ausgang für einen weiteren Iterationsschritt.

$$x_{u+1} = x_u - \frac{f(x_u)}{f'(x_u)}, f : \mathbb{R} \to \mathbb{R} \tag{2.3}$$

Dies wird soweit fortgeführt, bis die Änderung eine gewisse Grenze unterschritten hat. Wichtig ist, dass der Startwert bereits in der Nähe der gewünschten Nullstelle ist, ansonsten könnte die Folge divergieren. Wird das Verfahren auf mehrdimensionale Funktionen angewendet, so muss die Jacobi-Matrix $J(x)$, die alle partiellen Ableitungen enthält, wie in Formel 2.4 eingebunden werden.

$$x_{u+1} = x_u - \frac{f(x_u)}{J(x_u)}, f : \mathbb{R}^n \to \mathbb{R}^n \tag{2.4}$$

Quasi-Newton-Verfahren

Diese Verfahren [Deuflhard 04] basieren auf dem Newton-Verfahren. Die Funktion $f(x)$, die zweifach differenzierbar sein muss, wird bis zum zweiten Grad mit einer Taylor-Reihe

$f_T(x) \approx f(x)$ wie in Formel 2.5 angenähert.

$$f_T(x_{u+1}) = f(x_u) + (x_{u+1} - x_u) \cdot \nabla f(x_u) + \frac{1}{2}(x_{u+1} - x_u)^T \cdot H(x_k) \cdot (x_{u+1} - x_u) \quad (2.5)$$

Um den Rechenaufwand zu verkürzen wird die Hesse-Matrix $H(x)$ nicht direkt berechnet, sondern nur angenähert. Um ein Optimum (Minimum) zu erhalten muss die Ableitung der Funktion $f_T(x) = 0$ ergeben (siehe Formel 2.6).

$$\nabla f_T(x_{u+1}) = \nabla f(x_u) + H(x_k) \cdot (x_{u+1} - x_u) = 0 \quad (2.6)$$

Unter der Bedingung, dass die Hesse-Matrix positiv definit ist, gilt Formel 2.7.

$$x_{u+1} - x_u = -H^{-1} \nabla f(x_u) \quad (2.7)$$

Weiters muss die Sekantenbedingung (Formel 2.5) gelten.

$$\nabla f_T(x_{u+1}) = \nabla f(x_u) + H(x_k)(x_{u+1} - x_u) \quad (2.8)$$

Die approximierte Hesse-Matrix kann u.a. mit dem Broyden-Fletcher-Goldfarb-Shanno [Broyden 70] Algorithmus berechnet werden.

Broyden-Fletcher-Goldfarb-Shanno

Angenommen, es existiert eine Hesse-Matrix H_u in Iterationsschritt u und ein Ausgangspunkt x_u. Die Suchrichtung p_u kann durch die Lösung der Gleichung $B_u p_u = -\nabla f(x_u)$ berechnet werden. Anschließend wird der Punkt x_{u+1} durch $x_{u+1} = x_u + \alpha_u p_u$ bestimmt. α_u gibt dabei die Schrittweite an. y_u wird durch $y_u = \nabla f(x_{u+1}) - \nabla f(x_u)$ bestimmt. Daraus ergibt sich die Berechnung der approximierten Hesse-Matrix H_{u+1} in Formel 2.9.

$$H_{u+1} = H_u + \frac{y_u y_u^T}{y_u^T(\alpha_u p_u)} - \frac{H_u(\alpha_u p_u)(H_u(\alpha_u p_u))^T}{(\alpha_u p_u)^T H_u \, (\alpha_u p_u)} \quad (2.9)$$

KAPITEL 3

Stand der Technik

3.1 Vorangegangene Arbeiten

In [Markel 01] und [Wipke 01] wurden bereits mehrere Optimierungsalgorithmen auf HEV-Modelle angewendet und erkannt, dass der Suchraum dabei stark nicht-linear ist und nicht stetige Bereiche hat. Es wurde dabei jedoch nicht die Betriebsstrategie optimiert, sondern die Dimensionierung der E-Maschine und der Batterie. Ziel war es ebenfalls den Treibstoffverbrauch in einem vorgegebenen Fahrzyklus zu minimieren. Als Nebenbedingungen war außer eines möglichst ausgeglichenen SOCs auch noch eine Mindestanforderung an die Dynamik des Fahrzeuges gestellt. Als Simulationssoftware wurde ADVISOR[1] verwendet und die Optimierungsalgorithmen stammten aus der Software iSIGHT[2], VisualDOC[3] und MATLAB[4]. Für die Optimierung wurden die Gradientenstrategien FMINCON aus Matlab, VisualDOCs DGO und RSA, sowie SQP und die Suchstrategien DIRECT und ein GA (siehe Kapitel 2.4) angewendet. Das beste Ergebnis lieferte das DIRECT Verfahren, die Gradientenstrategien fanden lediglich lokale Optima, die nicht das globale Optimum waren. Die genauen Ergebnisse des GA wurden nicht angeführt, waren aber schlechter als das DIRECT Verfahren.

In [Huang 06] und [Montazeri-Gh 06] konnte jeweils ein GA erfolgreich auf ein Modell eines HEVs angewendet werden. Bei beiden Arbeiten wurde allerdings nicht nur der Treibstoffverbrauch sondern auch die Schadstoffemissionen berücksichtigt. Vergleiche mit anderen Verfahren wurden dabei nicht angeführt.

In [Gao 07] und [Gao 05] wurde u.a. die Simulationssoftware PSAT[5] und dessen Optimierungsalgorithmen DIRECT, GA, SA und PSO (siehe Kapitel 2.4) auf ein HEV-Modell ange-

[1] ADVISOR (Advanced Vehicle Simulator) ist eine Software von AVL bzw. NREL (bis 2002), http://www.avl.com/, http://www.nrel.gov,

[2] iSIGHT ist eine Software von Simulia, http://www.simulia.com/

[3] VisualDOC ist eine Software von VR&D, http://www.vrand.com/

[4] MATLAB ist eine Software von MathWorks, http://www.mathworks.de/

[5] PSAT (Powertrain System Analysis Toolkit) wurde von Argonne National Laboratory entwickelt, http://www.transportation.anl.gov/modeling_simulation/PSAT/

wendet. Als Nebenbedingung war eine Mindestanforderung an die Dynamik des Fahrzeuges gestellt. Ziel war es ebenfalls den Treibstoffverbrauch in einem vorgegebenen Fahrzyklus zu minimieren. Hierbei konnte der GA und der PSO vergleichsweise die geringsten Verbesserungen aufweisen, die Algorithmen SA und DIRECT waren am erfolgreichsten. Weiters wurde in [Gao 05] auch noch ein hybrider Algorithmus entwickelt. Dieser kombiniert das DIRECT mit einem SQP Verfahren. Es wurde allerdings nur auf eine Testfunktion und nicht auf das eigentliche Problem angewendet. Bei dieser konnte aber mit relativ wenigen Iterationen nahezu das globale Optimum gefunden werden.

Weiters wurde in [Wu 08a] und [Wu 08b] erfolgreich ein PSO Algorithmus auf ein Modell eines HEVs mit vorhandener Betriebsstrategie angewendet. Als Simulationssoftware wurde ebenfalls ADVISOR verwendet. Die Forderung eines ausgeglichenen SOCs wurde in die Zielfunktion miteinbezogen. Optimiert wurde nur die Betriebsstrategie, die Kenngrößen des Fahrzeugs waren vorgegeben. Es konnte eine Verbesserung gegenüber der vorhandenen Betriebsstrategie erzielt werden. Inwiefern diese bereits vorher optimiert wurde, ist nicht bekannt.

Im Vergleich zu GT-SUITE können Teile der Zielfunktion bei ADVISOR und PSAT durch Lösen einer mathematischen Funktion wesentlich schneller berechnet werden. Dadurch können einerseits Gradientenstrategien angewendet werden und andererseits ergeben sich meist erhebliche Geschwindkeitsvorteile bei der Berechnung eines Modells. Der Vorteil von GT-SUITE liegt allerdings in der weitaus genaueren Beschreibung des HEV-Modells. Bei den vorgestellten Anwendungen wurde nicht nur die Betriebsstrategie, sondern auch Kenngrößen wie die Batteriekapazität und die Anzahl an Batteriezellen, optimiert. Die Forderung eines ausgeglichenen SOCs wurde entweder als Nebenbedingung durch die Begrenzung der Abweichung zu einem ausgeglichenen SOC definiert oder als Penalty in die Zielfunktion eingegliedert. Wird sie als Nebenbedingung definiert, so führt dies vor allem bei einer kleinen Toleranz zu einer großen Anzahl an ungültigen Lösungen, was vor allem bei Verfahren wie DOE zu Problemen führen kann. Bei den bisherigen Arbeiten wurden meist Standard-Optimierungsverfahren aus bereits vorhandenen Bibliotheken verwendet. Interessant ist, dass bei sehr ähnlichen Problemstellungen immer unterschiedliche Optimierungsverfahren die besten Ergebnisse lieferten. Ein direkter Vergleich der unterschiedlichen Projekte ist leider nicht möglich, da keine genaueren Angaben zu den Konfigurationen der Optimierungsalgorithmen vorhanden sind. Somit ist es schwierig Rückschlüsse auf geeignete Verfahren für die Optimierung von HEVs zu ziehen.

3.2 Optimierungsverfahren von GT-SUITE

Bei DOE sind für brauchbare mathematische Modelle eine Vielzahl an Durchläufen durchzuführen. Es können nach [Meywerk 07] maximal zehn Parameter optimiert werden. Aus vorangegangenen Projekten wurde allerdings die Erkenntnis gewonnen, dass für jeden Parameter eine Diskretisierung in fünf bis zehn Bereiche notwendig ist. Mit der Annahme, dass ein Durchlauf zehn Minuten dauert und jedem Parameter zehn Werte zugewiesen werden, entspricht die Laufzeit bei der Full-Factorial Methode bei nur fünf zu optimierenden Parametern $10 \cdot 10^5 = 10^6$ Minuten (ca. 22 Tage), bei zehn Werten pro Parameter und zehn zu optimierenden Parametern ergibt sich bereits eine Laufzeit von $10 \cdot 10^{10} = 10^{11}$ Minuten bzw. ca. 11 Millionen Jahren. Als Alternative kann mithilfe der Latin-Hypercube Methode die Anzahl an Lösungen zwar rapide

herabgesetzt werden, allerdings wird dadurch bei zehn Parametern das mathematische Modell sehr stark vereinfacht. Aufgrund dessen kann das Modell unbrauchbar werden bzw. können daraus falsche Schlüsse, bezogen auf die Abhängigkeiten der Parameter untereinander, gezogen werden. Das direkte Optimierungsverfahren ist erst dann sinnvoll, wenn bereits eine gute Lösung existiert.

KAPITEL 4

Umsetzung

4.1 Zielfunktion

Für die Optimierung eines Modells müssen zu den Parametern Initialwerte angegeben werden. Mithilfe dieser Werte wird eine Referenzlösung berechnet. Zu jedem der q Ausgabewerte $o_k, k = 1 \dots q$, die in der Zielfunktion enthalten sind, muss eine Priorität e_k vergeben werden. Der Zielfunktionswert der Referenzlösung t^{ref} setzt sich aus der Summe der Prioritäten zusammen $\sum_{k=1}^{q} e_k^{ref}$. Der Zielfunktionswert t für ein Parameterset s wird dann wie in Formel 4.1 ersichtlich berechnet.

$$t = \sum_{k=1}^{q} e_k \cdot val; val = \begin{cases} \frac{o_k^{ref}}{o_k}, & \text{falls Ausgabewert minimiert werden soll} \\ \frac{o_k}{o_k^{ref}}, & \text{falls Ausgabewert maximiert werden soll} \end{cases} \tag{4.1}$$

Unabhängig davon ob es sich um einen zu minimierenden oder maximierenden Wert handelt, stellt ein höherer Zielfunktionswert somit immer ein besseres Ergebnis dar. Für das IFAHEV wurde einerseits der Verbrauch und andererseits die quadratische Abweichung zu einem ausgeglichenen SOC berücksichtigt (siehe Kapitel 2.2). Der Treibstoffverbrauch $cons$ wurde weiters mit einem Penalty P behaftet, falls die Batterie am Ende eines Zyklus weniger geladen war wie zu Beginn, bzw. besser bewertet, wenn diese einen höheren SOC aufwies. Die Berechnung von $cons_{new}$ und P wird in Formel 4.2 bzw. 4.3 gezeigt.

$$cons_{new} = cons \cdot P \tag{4.2}$$

$$P = 1 - \frac{\Delta E_{batt}}{E_{ges}} \tag{4.3}$$

ΔE_{batt} ist die Differenzenergie zu einem ausgeglichenem SOC. Ist die Batterie mehr aufgeladen als zu Beginn, so ist dieser Wert positiv, ist sie weniger aufgeladen, negativ. Bei ausgeglichenem SOC gilt: $\Delta E_{batt} = 0$ sowie $P = 1$. E_{ges} ist eine Konstante, die die gesamte Energie

angibt, die notwendig ist, um das Fahrzeug durch den Fahrzyklus bewegen zu können. $cons_{new}$ gibt den theoretischen Treibstoffverbrauch für einen ausgeglichenen SOC an.

Ziel sollte es aber sein, dass Lösungen einen möglichst ausgeglichenen SOC aufweisen. Somit wurde zusätzlich noch die quadratische Abweichung abw des SOC am Ende SOC_{end} zum SOC am Anfang SOC_{begin} mit einer Korrekturkonstante c berücksichtigt. Formel 4.4 zeigt die Berechnung von abw.

$$abw = c + (SOC_{begin} - SOC_{end})^2 \tag{4.4}$$

Der SOC ist bei vollständig aufgeladener Batterie 100 und bei leerer Batterie 0. Die Wahl von $c > 0$ beeinflusst vor allem bei kleinen Abweichungen die Bewertung. Wird c sehr klein gewählt, dann ergeben bereits kleine Abweichungen deutlich schlechtere Lösungen, wird es zu groß gewählt, dann hat die Abweichung nahezu keinen Einfluss mehr auf den Zielfunktionswert. Für die Referenzlösung muss gelten: $abw \approx c$. Ergibt beispielsweise die quadratische Abweichung $(SOC_{begin} - SOC_{end})^2 = 100$, so wäre bei $c = 10$ die Abweichung $abw = 110$. Die Referenzlösung hätte einen Abweichungswert von 10. Somit ergibt sich bei der Zielfunktionsberechnung, ohne Berücksichtigung der Prioritäten, $\frac{o_k^{ref}}{o_k} = \frac{10}{110} \approx 0.09$. Die Lösung würde somit wesentlich schlechter als die Referenzlösung bewertet werden. Wählt man $c = 1000$, dann würde sich bei der Zielfunktionsberechnung der Wert $\frac{o_k^{ref}}{o_k} = \frac{1000}{1100} \approx 0.9$ ergeben. Diese Lösung wäre nur etwas schlechter als die Referenzlösung. In Kombination mit der Penalty Berechnung hat sich durch verschiedene Tests die Einstellung für die Korrekturkonstante $c = 10$ und die Priorität für die SOC Abweichung $e^{soc} = 5$ ergeben.

4.2 Optimierungsstrategien

Bevor die Optimierungsalgorithmen angewendet werden können, muss entschieden werden, welche Parameter optimiert werden sollen. Dies geschieht entweder durch die Erfahrung mit bereits vorangegangenen Projekten, durch die in GT-SUITE vorhandene DOE, oder durch die entwickelte Parameteranalyse. Als nächsten Schritt wurde ein GA, ein PSO Algorithmus und das Downhill-Simplex Verfahren auf das Problem angewendet. Die Ausgangslösungen lieferte ein Monte-Carlo Suchverfahren. Schlussendlich wurden alle Verfahren in einem hybriden Ansatz kombiniert.

Parameteranalyse

Bei der Analyse ist darauf zu achten, dass sie im Vergleich zu den Optimierungsverfahren relativ wenig Zeit in Anspruch nimmt und für eine Anzahl von 5-20 zu analysierenden Parametern ausgelegt werden soll. Ziel ist es, Parameter die möglichst wenig Einfluss auf den Zielfunktionswert haben oder fast immer nur mit der selben Einstellung die beste Lösung erzielen, von der Optimierung auszuschließen.

Angenommen, es existieren n zu analysierende Parameter $p_i,\ i = 1 \ldots n$. Jeder dieser Parameter bekommt eine fixe Anzahl k an unterschiedlichen Werten $v_{ij},\ i = 1 \ldots n,\ j = 1 \ldots k$, $v_{ij} < v_{ij+1}$, die gleich auf den zulässigen Wertebereich aufgeteilt werden, zugewiesen. Für $n-1$

Parameter wird eine Zufallsbelegung festgelegt. Dem Parameter, der nicht Teil dieser Zufallsbelegung ist, werden die k Werte zugewiesen und jeweils eine Lösung berechnet. Diese Prozedur wird mit jeweils neuer Zufallsbelegung l mal wiederholt. Es müssen somit insgesamt $n \cdot k \cdot l$ Lösungen berechnet werden. Danach wird bestimmt, wie oft welcher Wert das beste Ergebnis erzielt hat und die l Verläufe der k berechneten Lösungswerte jedes Parameters werden analysiert. Befindet sich der beste Lösungswert immer bei der selben Einstellung, so wird dieser als empfohlene Einstellung gewählt und die Optimierungspriorität als niedrig eingestuft. Sind alle k berechneten Lösungswerte für eine Zufallsbelegung ident, so wird die Priorität ebenfalls niedrig eingestuft. Dies würde bedeuten, dass die Veränderung des Parameters keinen Einfluss auf den Zielfunktionswert hat. Ergibt die Analyse, dass sich der beste Lösungswert jeweils bei verschiedenen Einstellungen ergibt und auf den Verlauf der k berechneten Lösungswerte bezogen schließen lässt, dass es für alle l Zufallsbelegungen nur jeweils ein lokales Optimum gibt, wird eine mittlere Priorität zugewiesen. Ergeben sich mehrere lokale Optima, so erhält der Parameter eine hohe Optimierungspriorität. Weiters hat auch noch jeweils die Differenz des größten und kleinsten Zielfunktionswertes aller k berechneten Lösungen einen Einfluss auf die Priorität. Je höher diese ist, desto höher auch die Optimierungspriorität für diesen Parameter. Exakt wird die Priorität wie folgt berechnet: $(1.0 - \frac{h}{l} + \frac{w}{l}) \cdot diff_{max}$. h ist dabei die Häufigkeit, wie oft der beste Wert bei allen l Sets der beste Wert war und $diff_{max}$ die größte Differenz zwischen kleinstem und größtem Zielfunktionswert für alle l Sets. w gibt die Anzahl an Kurvenverlaufsänderungen, bezogen auf die k berechneten Lösungswerte, an.

Monte-Carlo Suchverfahren

Dieses Verfahren wurde hauptsächlich dazu verwendet, Ausgangslösungen für weitere Algorithmen zu erzeugen. Deswegen wurde für den Ausgangsbereich jedes Parameters der gesamte Wertebereich in Betracht gezogen. Folglich wurden im ersten Schritt ausschließlich Zufallslösungen erzeugt. In jedem der r Iterationsschritte werden d Lösungen erzeugt und der Bereich um den Faktor $resize$ verkleinert. Ausgehend von der besten Lösung $best$ wird im Iterationsschritt u der neue Bereich $[p^{start},\ p^{end}]$ eines Parameters durch die Parametergrenzen $p^{min},\ p^{max}$ und den Faktor $resize$ mit Formel 4.5 und Formel 4.6 berechnet.

$$p^{start} = p^{best} - \frac{(p^{max} - p^{min}) \cdot (resize^u)}{2} \tag{4.5}$$

$$p^{end} = p^{best} + \frac{(p^{max} - p^{min}) \cdot (resize^u)}{2} \tag{4.6}$$

Für den Fall, dass $p^{start} < p^{min}$ oder $p^{end} > p^{max}$ ist, wird $p^{start} = p^{min}$ bzw. $p^{end} = p^{max}$ gesetzt. Abgebrochen wird der Algorithmus nach r Iterationsschritten. Wird der Faktor $resize$ zu groß gewählt ($resize \approx 1$), so entsteht eine reine Zufallssuche. Bei einem zu kleinen Wert wird der Bereich sehr schnell eingeschränkt und die Wahrscheinlichkeit ist groß, dass die beste berechnete Lösung ein lokales Optimum, das nicht das globale Optimum ist, darstellt. Weiters wären die berechneten Lösungen nach wenigen Iterationen bereits sehr ähnlich und es wäre sinnvoll die Anzahl der Lösungen nach jedem Iterationsschritt herabzusetzen. Aufgrund dessen, dass der Algorithmus hauptsächlich Ausgangslösungen für weitere Verfahren erzeugen sollte,

wurde ein Faktor zwischen 0.8 und 0.9 gewählt und die Anzahl der berechneten Lösungen pro Iterationsschritt konstant gehalten. Um in weiterer Folge auf die berechneten Lösungen zugreifen zu können, werden diese in einer Datenbank (siehe Kapitel 4.3) abgespeichert. Algorithmus 2 zeigt die Implementierung des Monte-Carlo Suchverfahrens.

Algorithmus 2 Monte-Carlo Suchverfahren MC

$t^{best} = -1$; // best objective value
$s^{best} = NULL$; // best solution
for $i = 1 \dots n$ **do**
  $p_i^{start} = p_i^{min}$;
  $p_i^{end} = p_i^{max}$;
end for

for $u = 1 \dots r$ **do**

  // calculate solutions
  for $j = 1 \dots d$ **do**
    for $i = 1 \dots n$ **do**
      $p_i = random(p_i^{min}, p_i^{max})$;
    end for
    $s_j \dots$ new solution with parameter setting p;
    if $t_j > t^{best}$ **then**
      $t^{best} = t_j$;
      $s^{best} = s_j$;
    end if
  end for

  // update range depending on s_{best}
  for $i = 1 \dots n$ **do**
    $p_i^{start} = p_i^{best} - \frac{(p_i^{max} - p_i^{min}) \cdot (resize^u)}{2}$;
    $p_i^{end} = p_i^{best} + \frac{(p_i^{max} - p_i^{min}) \cdot (resize^u)}{2}$;
    if $p_i^{start} < p_i^{min}$ **then**
      $p_i^{start} = p_i^{min}$;
    end if
    if $p_i^{end} > p_i^{max}$ **then**
      $p_i^{end} = p_i^{max}$;
    end if
  end for
end for

Genetischer Algorithmus (GA)

Beim GA wurden Lösungen von dem Monte-Carlo Suchverfahren für die Erzeugung der ersten Population verwendet. Jedes Individuum wird durch den Vektor der Parameterwerte repräsentiert. Die Auswahl der jeweiligen Lösungen aus der Population für die Rekombination erfolgt per Zufall. Im Laufe des Verfahrens ersetzen die erzeugten Lösungen, die einen höheren Zielfunktionswert haben, mit höherer Wahrscheinlichkeit die bereits existierenden. Aufgrund dessen und der insgesamt relativ kleinen Anzahl an zu berechnenden Lösungen wurde auf eine Fitnessproportionale Selektion verzichtet.

Bei der Rekombination von zwei Lösungen s_x, s_y wird pro Parameter ein Wert einer Lösung übernommen. Somit erhält man als Ergebnis der Rekombination nur eine neue Lösung. Die Auswahl, welcher Parameterwert von welcher Lösung übernommen wird, hängt von der Abweichung zu den besten d^{best} Lösungen der bereits berechneten Lösungen in der Datenbank ab. Dabei wird jeweils die Standardabweichung std_p^x, std_p^y für die Parameterwerte p^x, p^y der beiden Lösungen, wie in Algorithmus 3 ersichtlich, berechnet.

Algorithmus 3 Berechnung der Abweichung eines Parameters p STDDEV

$std_p^x = 0, std_p^y = 0$
$dbVals$ = parameter values of the d^{best} best solutions from the database;
for $j = 1 \dots d_{best}$ **do**
  $std_p^x = std_p^x + \frac{|dbVals_p^j - p^x|}{d^{best}}$;
  $std_p^y = std_p^y + \frac{|dbVals_p^j - p^y|}{d^{best}}$;
end for
return std_p^x, std_p^y

Die Wahrscheinlichkeiten P_x, P_y der Auswahl eines Wertes können dann durch das Verhältnis der Standardabweichungen mit der Formel 4.7 und Formel 4.8 berechnet werden.

$$P_x = \frac{std_p^x}{std_p^x + std_p^y} \tag{4.7}$$

$$P_y = 1 - P_x \tag{4.8}$$

Wichtig ist die Wahl von d^{best}. Wird sie zu klein gewählt, dann werden neu erzeugte Lösungen sehr schnell ähnliche oder gleiche Werte wie die bereits beste gefundene Lösung aufweisen. Bei zu großem d^{best} entwickelt sich die Auswahl in Richtung der durchschnittlichen Parameterwerte der Lösungen aus der Datenbank. Weiters kann noch eine Mutationswahrscheinlichkeit P_{mut} angegeben werden. Diese gibt die Wahrscheinlichkeit an, mit der ein Parameter zufällig einen neuen Wert zugewiesen bekommt. Aufgrund dessen, dass bei einer kleinen Mutationswahrscheinlichkeit pro Lösungsberechnung meist nur ein Parameter mutiert wird und es auch möglich sein sollte aus einer Kombination von zwei ähnlichen Lösungen mit unausgeglichenem SOC eine gute Lösung zu erzeugen, wurden nicht nur kleine Veränderungen mit höherer Wahrscheinlichkeit zugelassen, sondern dem Parameter ein zufälliger Wert innerhalb der zulässigen Grenzen $[p^{min}, p^{max}]$ zugewiesen. Weitere noch nicht implementierte Ansätze zur Rekombination und Mutation werden in Kapitel 7 erwähnt.

Nachdem eine neue Lösung s_x erzeugt und der Zielfunktionswert t_x berechnet wurde, wird per Zufall eine Lösung s_y der Population ausgewählt. Die neu berechnete Lösung ersetzt diese mit der Wahrscheinlichkeit P aus Formel 4.9.

$$P = \frac{t_x - corrVal}{t_x + t_y - 2 \cdot corrVal} \tag{4.9}$$

Der Korrekturwert $corrVal$ dient zur Veränderung des Einflusses der Zielfunktionswerte auf die Wahrscheinlichkeit P. Je höher dieser gewählt wird, desto eher wird eine neue bessere Lösung gewählt werden bzw. eine neue schlechtere Lösung nicht gewählt werden. Die Differenzen $t_x - corrVal$ und $t_y - corrVal$ müssen jeweils größer als 0 sein.

Algorithmus 4 zeigt die Implementierung des GAs.

Particle-Swarm-Optimization (PSO)

Bei der Initialisierung werden die Zielfunktionswerte t_j und Parametersets s_j, $j = 1 \dots d$, von d Individuen (Lösungen) zufällig aus der Datenbank gewählt, die bereits durch das Monte-Carlo Suchverfahren berechnet wurden. Zu jedem Individuum wird der beste von diesem erreichte Zielfunktionswert $t_j^{localbest}$ und das dazugehörige Parametersetting $s_j^{localbest}$ gespeichert. Bei der Initialisierung entsprechen diese den Werten aus t_j bzw. s_j. Außerdem wird der Zielfunktionswert $t^{globalbest}$ und das Parametersetting $s^{globalbest}$ der besten bekannten Lösung aller Individuen gespeichert. In jedem Iterationsschritt wird das Parameterset der Individuen in Abhängigkeit der lokal und global besten bekannten Lösung verändert. Dazu wird für jedes Individuum s_j ein Vektor $\vec{f_j} \in [-1; 1]^n$, wobei n die Anzahl der zu optimierenden Parameter darstellt, definiert und mithilfe der Formeln 4.10 und 4.11 berechnet.

$$\vec{f_j^i} = \vec{f_j^i} + factor_{local} \cdot \frac{p_i^{localbest,j} - p_i^j}{range_i} + factor_{global} \cdot \frac{p_i^{globalbest} - p_i^j}{range_i} \tag{4.10}$$

$$\vec{f_j^i} = \vec{f_j^i} + randVal \tag{4.11}$$

i gibt den Parameter an und $range_i$ entspricht jeweils der Differenz des maximal p_i^{max} und minimal p_i^{min} zulässigen Wertes des Parameter i. $factor_{local}$ und $factor_{global}$ steuern den Einfluss der lokal und global besten bekannten Lösung auf die Berechnung des Kräftevektors. Es gilt: $factor_{local} + factor_{global} = 1, factor_{local} \in [0; 1], factor_{global} \in [0; 1]$. $randVal$ ist ein Zufallswert $\in [-0,1; 0.1]$. Sollte $\vec{f_j^i}$ größer als 1 oder kleiner als -1 sein, dann wird der Wert auf 1 bzw. -1 gesetzt. Das Parameterset s_j des Individuum j wird dann für jeden Parameter i wie in Formel 4.12 ersichtlich aktualisiert.

$$p_i^j = p_i^j + \frac{range_i}{stepsize} \cdot \vec{f_j^i} \tag{4.12}$$

$stepsize \geq 1$ steuert die Schrittweite, mit der ein Parameter verändert wird. Wird $stepsize$ groß gewählt, verändern sich die Parametersettings nur sehr langsam, wird es klein gewählt, dann ist die Veränderung pro Iterationsschritt größer. Ist ein Parameter p_i^j größer als der maximal oder kleiner als der minimal zulässige Wert des Parameters, dann wird p_i^j auf den maximal bzw. minimal zulässigen Wert gesetzt.

Algorithmus 4 Genetischer Algorithmus GA

$population[1 \dots d] \dots d$ random parametersets from the database
while termination criterion not reached **do**
  $s_x, s_y \dots$ random solutions from $population$, $s_x \neq s_y$;
  $s_{new} \dots$ new solution;
  for $i = 1 \dots n$ **do**
    $randVal = random(1, 100)$;

    // mutation
    if $randVal \leq P_{mut}$ **then**
      $p_i^{new} = random(p_i^{min}, p_i^{max})$;
    else

      // recombination
      $randVal = random(1, 100)$;
      std^x, std^y = STDDEV(s_x, s_y, i);
      $P_x = \frac{std^x}{std^x + std^y} \cdot 100$
      if $randVal < P_x$ **then**
        $p_i^{new} = p_i^x$;
      else
        $p_i^{new} = p_i^y$;
      end if
    end if
  end for

  // population update
  $s_x \dots$ random solution from $population$;
  $randVal = random(1, 100)$;
  $corrVal = \min_j\{t_j\} - 2, j = 1 \dots d$;
  $P = 100 \cdot \frac{t_{new} - corrVal}{t_{new} + t_x - 2 \cdot corrVal}$
  if $randVal < P$ **then**
    replace solution s_x with s_{new} in $population$;
  end if
end while

Als Abbruchkriterium wurde die Anzahl an Durchläufen vorgegeben. Nachdem dieses erfüllt ist, enthält $s^{globalbest}$ die beste gefundene Lösung. Algorithmus 5 zeigt die Implementierung des PSO Algorithmus.

Algorithmus 5 Particle-Swarm Optimization PSO

$s \ldots d$ random parametersets from the database;
$s^{localbest} = s$;
$s^{globalbest} = NULL$;
$t^{globalbest} = -1$;
for $j = 1 \ldots d$ **do**
  if $t_j^{localbest} > t^{globalbest}$ **then**
    $t^{globalbest} = t_j^{localbest}$;
    $s^{globalbest} = s_j^{localbest}$;
  end if
  $\vec{f_j^i} = \vec{0}$;
end for
while termination criterion not reached **do**
  for $j = 1 \ldots d$ **do**
    for $i = 1 \ldots n$ **do**
      $randVal = random(-0.1, 0.1)$;
      $\vec{f_j^i} = \vec{f_j^i} + factor_{local} \cdot \frac{p_i^{localbest,j} - p_i^j}{range_i}$;
      $\vec{f_j^i} = \vec{f_j^i} + factor_{global} \cdot \frac{p_i^{globalbest} - p_i^j}{range_i} + randVal$;
      $\vec{f_j^i} = \max(\min(\vec{f_j^i}, 1), -1)$;
      $p_i^j = p_i^j + \frac{range_i}{stepsize} \cdot \vec{f_j^i}$;
    end for
    if $t_j > t_j^{localbest}$ **then**
      $s_j^{localbest} = s_j$;
      $t_j^{localbest} = t_j$;
    end if
  end for
  for $j = 1 \ldots d$ **do**
    if $t_j^{localbest} > t^{globalbest}$ **then**
      $t^{globalbest} = t_j^{localbest}$;
      $s^{globalbest} = s_j^{localbest}$;
    end if
  end for
end while

Downhill-Simplex (SIMPLEX)

Auch beim Downhill-Simplex Verfahren wurden $v + b$ Lösungen aus der Datenbank für die Initialisierung verwendet. v ist die Größe des Simplex und b gibt die Anzahl der schlechtesten Lösungen w_l, $l = 1 \dots b$ an, deren Parametersetting in jedem Iterationsschritt neu berechnet werden soll. Um die Parallelisierung besser nutzen zu können werden im Vergleich zum klassischen Verfahren somit nicht nur eine Lösung, sondern b Lösungen ersetzt. Weiters werden drei Faktoren, $reflex$, $expand$ und $contract$, im Bereich von 0.01 bis 1 definiert. Diese werden in jeder Iteration zufällig bestimmt. Als nächsten Schritt muss von allen Lösungen $s_j, j = 1 \dots v$, die Teil des Simplex sind, der Mittelwert $mean_i$ für jeden Parameter i, mit der Formel 4.13 berechnet werden.

$$mean_i = \frac{1}{v} \cdot \sum_{j=1}^{v} p_i^j \tag{4.13}$$

Durch Reflexion der schlechten Lösungen w_l an den berechneten Mittelwerten $mean$ wird versucht diese mithilfe der Formel 4.14 zu verbessern.

$$p_i^l = mean_i - reflex \cdot distance_i^l \tag{4.14}$$

$distance_i^l$ entspricht der Differenz $p_i^l - mean_i$. Die weitere Vorgehensweise wird in zwei Phasen aufgeteilt. In der ersten Phase werden alle schlechten Lösungen w zuerst überprüft, ob diese besser sind als die bisher beste Lösung. Wenn das zutrifft, dann wird die Reflexion durch den Faktor $expand$ wie in Formel 4.15 ersichtlich verstärkt.

$$p_i^l = p_i^l - expand \cdot distance_i^l \tag{4.15}$$

Falls nicht, wird noch überprüft, ob der neu berechnete Wert der Lösung besser ist als der alte. Trifft dies zu, so wird die neu berechnete Lösung übernommen und in der zweiten Phase ist nichts mehr zu tun. Ist der neu berechnete Zielfunktionswert nicht besser als der Ausgangswert, so wird der Abstand vom ursprünglichen Parameterwert zum Mittelwert mit der Formel 4.16 verkürzt.

$$p_i^l = mean_i + contract \cdot distance_i^l \tag{4.16}$$

Bei allen Berechnungen der Parametereinstellung p_i^l muss überprüft werden, ob diese kleiner oder größer als die zulässigen Parametergrenzen p_i^{min} und p_i^{max} ist und gegebenenfalls begrenzt werden. Wurde die Reflexion durch den Faktor $expand$ verstärkt, so wird in der zweiten Phase überprüft, ob diese Veränderung eine Verbesserung bewirkt hat. Ist dies der Fall, so wird das berechnete Parameterset als neue Lösung übernommen. Falls nicht, dann wird das Ergebnis nach der ersten Reflexion übernommen. Wurde der Abstand der Ausgangslösung zum Mittelwert verkürzt, wird bei Verbesserung des Parametersets im Vergleich zur Ausgangslösung das neu berechnete Set übernommen. Falls keine Verbesserung möglich ist, so bleibt die Ausgangslösung bestehen. Tritt dieser Fall bei allen schlechten Lösungen auf, so wird der Algorithmus mit einem neuen Datenset neu gestartet. Es wurde bewusst auf das Verkleinern des kompletten Simplex verzichtet. Dies könnte dazu führen, dass mehrere dieser Lösungen schlechtere Ergebnisse,

aufgrund eines nicht ausgeglichenen SOC, aufweisen könnten. Es müssten somit alle Parameter dieser Lösungen Schritt für Schritt minimal verändert werden um wiederum einen sinnvollen neuen Punkt im Simplex zu erhalten, was einen erheblichen Rechenaufwand erfordern würde. Algorithmus 6 und 7 zeigt die Implementierung des Downhill-Simplex Verfahrens.

Algorithmus 6 Downhill-Simplex SIMPLEX

$s \dots v + d$ random parametersets from the database;
$improvement = true$;
while $improvement == true$ **do**
  $sort(s)$ // sorts all solutions by t_j in ascending order
  $reflex, contract, expand \dots random(0.01, 1)$;
  $improvement = false$;
  for $i = 1 \dots n$ **do**
    $mean_i = \frac{1}{v} \cdot \sum_{j=1}^{v} p_i^j$;
  end for
  $w = s_{v+1 \dots v+d}$;
  for $i = 1 \dots n$ **do**
    for $l = v + 1 \dots v + d$ **do**
      $distance_i^l = p_i^l - mean_i$;
      $p_i^l = mean_i - reflex \cdot distance_i^l$;
      $p_i^l = \max(\min(p_i^l, p_i^{max}), p_i^{min})$
    end for
  end for
  $modifySimplex()$;
end while

Surface-Fitting

Vorlage für den Surface-Fitting Algorithmus war das in [Meywerk 07] vorgestellte Jacob-Verfahren. In dem angeführten Beispiel wurde ein Problem mit zwei zu optimierenden Parametern vorgestellt. Für diese musste in jedem Iterationsschritt eine quadratische Funktion erzeugt werden, die möglichst nah an drei berechneten Zielfunktionswerten liegt. Bei der Problemstellung in dieser Arbeit sind es aber in etwa zehn zu optimierende Parameter, sodass eine quadratische Funktion, die von zehn Parametern abhängt, nicht mehr trivial erzeugt werden kann. Bei zwei Parametern p_1, p_2 erhält man durch die Form der Ersatzfunktion $fit(p_1, p_2) = a + b \cdot p_1 + c \cdot p_2 + d \cdot p_1^2 + e \cdot p_2^2 + f \cdot p_1 \cdot p_2$ sechs zu bestimmende Koeffizienten. Bei zehn Parametern wären dies bereits ein Koeffizient für einen konstanten Betrag, jeweils zehn Koeffizienten für lineare und quadratische Anteile der Parameter und $\sum_{i=2}^{10} \binom{10}{i}$ Koeffizienten für alle Parameterabhängigkeiten. Selbst wenn man diese auf zwei abhängige Parameter begrenzen würde, dann wären es immer noch $1 + 2 \cdot 10 + \binom{10}{2} = 66$ zu bestimmende Koeffizienten. Grundsätzlich wird bei diesem Verfahren immer nur ein Parameter verändert und alle weiteren stehen mit einem gewissen Faktor in Bezug zu diesem. Dadurch entsteht ein immer größerer Fehler je mehr Parameter optimiert werden sollen.

Algorithmus 7 Downhill-Simplex modifySimplex

for $l = v + 1 \dots v + b$ **do**
  $w = s_l$;
  $t^w = t_l$;
  if $t^l > t^{best}$ **then**
    $improvement = true$;
    for $i = 1 \dots n$ **do**
      $p_i^l = p_i^l - expand \cdot distance_i^l$;
      $p_i^l = \max(\min(p_i^l, p_i^{max}), p_i^{min})$;
    end for
    if $t^l \leq t^w$ **then**
      $s_l = w$;
      $t_l = t^w$;
    end if
  else if $t^l > t^w$ **then**
    $improvement = true$;
  else
    for $i = 1 \dots n$ **do**
      $p_i^l = mean_i + contract \cdot distance_i^l$;
      $p_i^l = \max(\min(p_i^l, p_i^{max}), p_i^{min})$;
    end for
    if $t^l > t^w$ **then**
      $improvement = true$;
    else
      $s_l = w$;
      $t_l = t^w$;
    end if
  end if
end for

Aufgrund dessen wurde ein Surface-Fitting Algorithmus entwickelt, der in jedem Iterationsschritt nur zwei zufällig gewählte Parameter verändert und für diese versucht mithilfe von mindestens sechs berechneten Parametersets eine quadratische Funktion zu approximieren. Dies wurde mithilfe der GNU Scientific Library (GSL) realisiert. Die Anzahl zu berechnender Parametersets wird in Abhängigkeit der zur Verfügung stehenden GT-SUITE Lizenzen festgelegt. Sie muss jedoch mindestens so groß sein, wie es Koeffizienten zu bestimmen gibt. In weiterer Folge muss von der erzeugten Ersatzfunktion das Optimum berechnet werden. Dafür wurde ebenfalls die GSL verwendet. Diese stellt als Gradientenstrategie u.a. den Broyden-Fletcher-Goldfarb-Shanno Algorithmus und als Suchstrategie den Downhill-Simplex Algorithmus (siehe Kapitel 2.4) zur Verfügung. Die implementierten Algorithmen sind nur für Minimierungsprobleme ausgelegt, sodass alle Lösungswerte mit -1 multipliziert werden müssen. Parametergrenzen sind dabei jeweils so als Nebenbedingung definiert, dass alle Lösungs-

werte die außerhalb dieser Grenzen liegen einen Lösungswert von $+1$ haben.

Der entwickelte Algorithmus wird mit einer Ausgangslösung s^{start} initialisiert. Per Zufall werden zwei Parameter p_1, p_2 gewählt. Dabei werden nach jeder Auswahl von p_1 und p_2 diese in einer Tabu-Liste $tabu$ der Größe $tabu_{size}$ gespeichert und falls die Tabu-Liste voll ist, werden die zwei am längsten in der Tabu-Liste verweilenden Parameter wieder gelöscht. In Abhängigkeit der Ausgangslösung und der gewählten Parameter werden d Lösungen s_j, $j = 1 \dots d$, erzeugt. Der Bereich des Parametersets einer Lösung wird mithilfe von drei Faktoren eingegrenzt. Der erste Faktor $factor_{area}$ wird mit 1 initialisiert und nach jeder vierten Lösung um 1 erhöht. Den weiteren zwei Faktoren ($factor_1, factor_2$) werden fortlaufend die Werte $(-1,-1)$; $(1,-1)$; $(-1,1)$; $(1,1)$ zugewiesen. Für alle n Parameter werden, in Abhängigkeit der Differenz der Parametergrenzen, Konstanten $consVal_i$, $i = 1 \dots n$, festgelegt. Sie geben den Abstand der Parameterwerte der neu berechneten Lösungen zur Ausgangslösung in Abhängigkeit der Faktoren $factor_{area}, factor_1$ und $factor_2$ an. Eine Lösung s_j wird mit den Formeln 4.17 und 4.18 berechnet:

$$p_1^j = p_1^{start} + factor_1^j \cdot factor_{area}^j \cdot consVal_1 \tag{4.17}$$

$$p_2^j = p_2^{start} + factor_2^j \cdot factor_{area}^j \cdot consVal_2 \tag{4.18}$$

Von allen berechneten Lösungen wird für beide Parameter der kleinste und größte Wert p_1^{low}, p_2^{low}, p_1^{high}, p_2^{high} ermittelt. Sind die Werte kleiner oder größer als die Parametergrenzen p_1^{min}, p_1^{max}, p_2^{min}, p_2^{max}, so werden die entsprechenden Parameterwerte p_i des Parameters i aller Lösungen s_j um den Betrag $p_i^{min} - p_i^{low}$ bzw. $p_i^{high} - p_i^{max}$ verschoben. Dadurch wird gewährleistet, dass die Parameterwerte jeder Lösung bei einem geringen $consVal$ Wert innerhalb der Parametergrenzen liegen. Mithilfe der GSL Methode `gsl_multifit_linear` wurden die sechs Koeffizienten $a \dots f$ der Ersatzfunktion $fit(.) = a + b \cdot p_1 + c \cdot p_2 + d \cdot p_1^2 + e \cdot p_2^2 + f \cdot p_1 \cdot p_2$ bestimmt. Mit dem Downhill-Simplex Verfahren der GSL wurde dann ein mögliches Optimum berechnet. Sowohl der Broyden-Fletcher-Goldfarb-Shanno Algorithmus, als auch das Downhill-Simplex Verfahren brachten bei der Berechnung der quadratischen Funktion immer die selben Ergebnisse. Beim Downhill-Simplex Verfahren erspart man sich allerdings für die GSL die Definition der zweiten Ableitung der Ersatzfunktion. Algorithmus 8 zeigt die Implementierung des Surface-Fitting Verfahrens.

Hybrider Ansatz (PSAGADO)

Bei den einzelnen Verfahren zeigten sich unterschiedliche Schwächen. Der GA konnte im Schnitt die besten Ergebnisse erzielen, da durch Mutation aus ungünstigen Bereichen des Suchraums entkommen werden konnte. Allerdings konnten relativ gute Lösungen oft nicht weiter verbessert werden. Würde man die Mutation dahingehend anpassen, dass mit kleinerer Wahrscheinlichkeit Parameterwerte größeren Änderungen unterzogen werden, sinkt wiederum die Wahrscheinlichkeit aus ungünstigen Bereichen des Suchraums zu entkommen. Die Ergebnisse des PSO und des Downhill-Simplex Verfahrens hängen sehr stark von den gewählten Ausgangslösungen ab. Würde man nur den PSO alleine anwenden, müssten die Lösungen möglichst gut im Suchraum verteilt sein und möglichst viele Lösungen bereits einen nahezu ausgeglichenen SOC aufweisen. Dies kann allerdings aufgrund der begrenzten Anzahl an Ausgangslösungen

Algorithmus 8 Surface-Fitting FITTING

start solution s^{start} ;
substitute function $fit(.) = a + b \cdot p_1 + c \cdot p_2 + d \cdot p_1^2 + e \cdot p_2^2 + f \cdot p_1 \cdot p_2$;
tabu list $tabu$;
while termination criterion not reached **do**
  p_1, p_2 not in $tabu$, $p_1 \neq p_2$.
  determining the d parameter sets $s_j, j = 1 \dots d$;
  if $p_1^{high} > p_1^{max}$ **then**
    shift parameter value p_1 of all solutions by $p_1^{high} - p_1^{max}$;
  else if $p_1^{low} < p_1^{min}$ **then**
    shift parameter value p_1 of all solutions by $p_1^{min} - p_1^{low}$;
  end if
  if $p_2^{high} > p_2^{max}$ **then**
    shift parameter value p_2 of all solutions by $p_2^{high} - p_2^{max}$;
  else if $p_2^{low} < p_2^{min}$ **then**
    shift parameter value p_2 of all solutions by $p_2^{min} - p_2^{low}$;
  end if
  $(a \dots f) =$ `gsl_multifit_linear(`s`);`
  s^{best} = `gsl_multimin_fminimizer(`$fit(.), s$`)` ;
  calculate t^{best} with GT-SUITE Solver;
  if $t^{best} > t^{start}$ **then**
    $s^{start} = s^{best}$;
    $t^{start} = t^{best}$;
  end if
  remove oldest two parameters from $tabu$ if $tabu_{size}$ is reached;
  add p_1, p_2 to $tabu$;
end while;

praktisch nicht realisiert werden. Beim hybriden Ansatz (Particle-Swarm And Genetic Algorithm with Downhill-Simplex Optimization, PSAGADO) werden die zuvor vorgestellten Algorithmen miteinander kombiniert und dadurch versucht, die einzelnen Schwächen auszugleichen. Als zentraler Algorithmus dient der PSO. Die Ausgangslösungen für diesen liefern die berechneten Lösungen des Monte-Carlo Suchverfahrens. Somit erspart man sich die Startlösungen neu zu berechnen und erhält trotzdem noch Lösungen, die nicht nur einen kleinen Bereich im Suchraum abdecken. Aufgrund dessen, dass über den Suchraum wenig bekannt ist, ist der PSO gut geeignet das Zentrum des gesamten Algorithmus zu bilden, da er ein robustes Verfahren ist welches anfangs Lösungen mit hoher Diversität betrachtet. Nach einer bestimmten Anzahl an Iterationen wird die beste Lösung des PSO Algorithmus mit dem Surface-Fitting Verfahren versucht weiter zu verbessern. Das Surface-Fitting wird nur auf die beste Lösung angewendet, da nur ein sehr begrenzter Suchraum betrachtet wird und dies bei einer schlechten Lösung unwahrscheinlich zu einer neuen besten Lösung führen würde. Weiters wäre der Aufwand dafür auch viel zu hoch, da für jede Lösung mehrere Punkte für den Surface-Fitting Algorithmus berechnet werden müss-

ten. Anschließend wird auf die letzten berechneten Lösungen des PSO der GA angewendet. Das Pool des GAs enpspricht somit zu Beginn den Lösungen des PSOs.

Durch den GA kann durch Rekombination und Mutation eventuell schneller eine bessere Lösung gefunden werden. Sind alle Individuen im Suchraum nahe beieinander, so kann dies trotzdem durch Mutation zu besseren Lösungen führen. Gibt es mehrere Lösungen verteilt im Suchraum, so kann dies auch durch Rekombination zu einer besseren Lösung führen. Wird beim GA eine neue Lösung berechnet, werden zufällig zwei Lösungen aus dem Pool gewählt und mit der Wahrscheinlichkeit, die für den GA angegeben ist, eine oder beide Lösung(en) ersetzt. Für den Fall, dass beide Lösungen ersetzt werden, wird eine Lösung durch das neu berechnete Parameterset und die andere durch eine Zufallslösung aus der Datenbank ersetzt. Kann durch den GA zumindest eine Lösung verbessert werden, so werden die Hälfte der Lösungen, die der besten Lösung, bezogen auf das Parameterset, am ähnlichsten sind, durch Zufallslösungen aus der Datenbank ersetzt. Die Differenz D_j einer Lösung s_j zur besten Lösung s^{best} wird mit Formel 4.19 berechnet.

$$D_j = \sum_{i=1}^{n} \left(\frac{|p_i^j - p_i^{best}|}{p_i^{max} - p_i^{min}} \right)^2 \tag{4.19}$$

n ist die Anzahl der zu optimierenden Parameter, p_i^{best} der Wert des Parameters i der besten Lösung und p_i^{max}, p_i^{min} die Parametergrenzen. Für die beste Lösung gilt: $D^{best} = 0$. Anschließend wird wieder der PSO Algorithmus ausgeführt. Sollte der GA keine Verbesserung erzielen, so wird das Simplex Verfahren angewendet. Dies tritt meist dann auf, wenn fast alle Lösungen des PSO, bezogen auf die Parametereinstellungen, sehr ähnlich sind. Das bedeutet einerseits, dass man bereits die meisten Lösungen im Bereich eines möglichen Optimums hat und andererseits, dass noch schlechte Lösungen existieren könnten, die noch eine Verschiebung des Simplex und somit eine mögliche neue beste Lösung bewirken könnten. Führt das Simplex-Verfahren zu einer Verbesserung, so wird mit dem PSO fortgefahren. Sind sich alle Lösungen des PSO bereits sehr ähnlich und konnte durch Mutation beim GA keine Verbesserung mehr erzielt werden, dann ergibt das Simplex Verfahren meist auch keine Verbesserung mehr. Falls das so ist, dann werden alle bestehenden Lösungen durch neue Lösungen aus der Datenbank ersetzt. Die Information über die bisher beste gefundene Lösung bleibt allerdings bestehen und der Algorithmus beginnt wieder beim PSO. Für den Fall, dass die Ausgangslösungen nicht gut im Suchraum verteilt sind und keinen ausgeglichenen SOC aufweisen, wäre es sinnvoller den GA vor dem PSO auszuführen. Aufgrund dessen, dass diese im Verlauf des Optimierungsprozesses aber praktisch abwechselnd ausgeführt werden, spielt dieser Aspekt keine tragende Rolle. Algorithmus 9 zeigt die Implementierung und Abbildung 4.1 den Aufbau des hybriden Ansatzes.

4.3 Details der implementierten Software

Die Optimierungssoftware und die grafische Oberfläche wurden in C++ für die Linux Distribution CentOS[1] und Windows entwickelt. Zur Parallelisierung wurde die Boost Library[2] und für das

[1] CentOS - Community Enterprise Operating System, http://www.centos.org/

[2] Boost C++ Libraries, http://www.boost.org/

Algorithmus 9 Hybrider Algorithmus PSAGADO

```
execute Monte-Carlo search method;
take randomised solutions for the PSO algorithm from the database;
while termination criterion not reached do
  execture PSO;
  apply FITTING on the best solution of the PSO;
  execute GA;
  if GA found at least one better solution then
    replace half of the solutions with lowes D_j;
  else
    execute SIMPLEX;
    if SIMPLEX could not found a better solution then
      take randomised solutions for the PSO algorithm from the database;
    end if
  end if
end while
```

Surface-Fitting die GNU Scientific Library (GSL)[3] verwendet. Als Datenbank wurde MySQL[4] genutzt. Die grafische Benutzer Oberfläche wurde mithilfe von GTK+[5] entwickelt.

Optimierungssoftware

Die Optimierungssoftware wird über eine Konfigurationsdatei gesteuert, die folgende Informationen enthält:

- Den GT-SUITE Solver Pfad
- Verzeichnisse in denen die Verarbeitung durchgeführt werden soll
- Pfade zu Ein- und Ausgabedatei
- Die IP-Adresse des Rechners, auf dem die Optimierungssoftware installiert ist
- Die IP-Adresse und Zugangsdaten des MySQL Servers
- Datenbank Informationen
- Grenzen, Initialwert und Eigenschaften der Eingabeparameter
- Prioritäten der zu optimierenden Ausgabewerte

[3] GNU Scientific Library, http://www.gnu.org/software/gsl/
[4] MySQL Datenbank, http://www.mysql.com/
[5] GTK+ - The GIMP Toolkit, http://www.gtk.org/

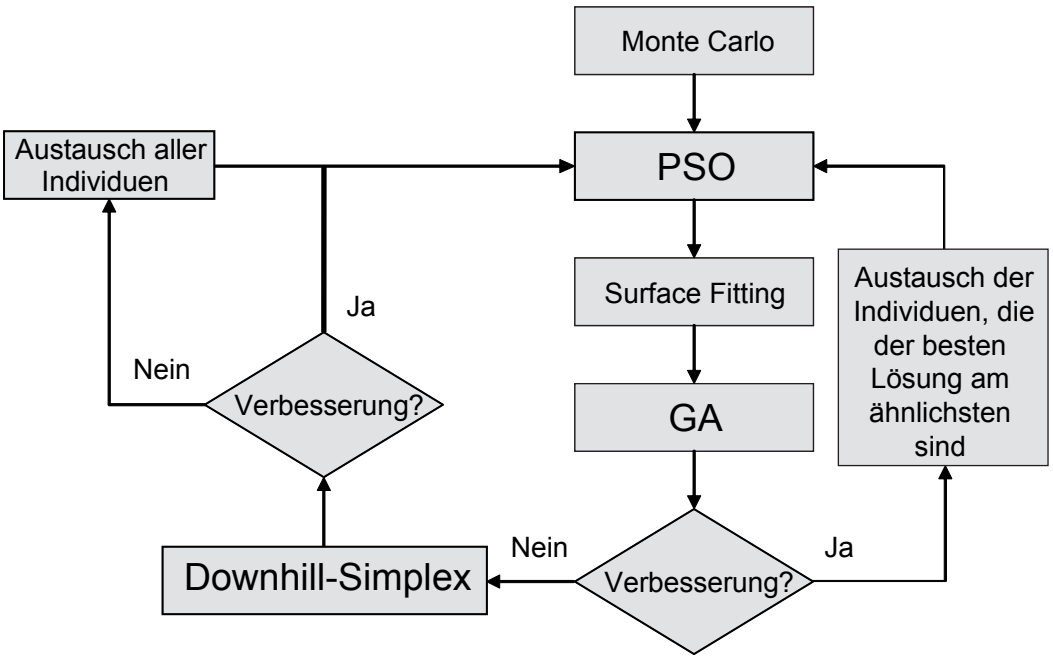

Abbildung 4.1: PSAGADO Aufbau

Diese Konfigurationsdatei kann entweder manuell oder mithilfe der grafischen Oberfläche erzeugt werden. Sie dient einerseits zur Konfiguration des GT-SUITE Solvers und anderseits der Konfiguration der Algorithmen. Um auf bereits berechnete Lösungen zurückgreifen zu können, werden alle Lösungen in einer Datenbank abgespeichert. Aufgrund dessen, dass eine Lösungsberechnung auf einem Intel Xeon X5570 Prozessor mit 2.93 GHz in etwa sieben Minuten dauert und mehrere Lizenzen und Rechenkerne zur Verfügung stehen, wurde bei den Optimierungsalgorithmen auf einen möglichst hohen Parallelisierungsgrad geachtet.

Datenbank

Für jedes Projekt und jede damit verbundene Versionsnummer wird in der Datenbank eine Tabelle $\langle projektname \rangle\_\langle version \rangle$ angelegt. In dieser Tabelle werden zu Vergleichszwecken nicht nur die zu optimierenden, sondern alle möglichen Parameter und Ausgabedaten gespeichert. Weiters wird aus den Ausgabedaten ein Zielfunktionswert bestimmt, der ebenfalls gespeichert wird.

Parallelisierung

Über die Konfigurationsdatei kann die Anzahl der maximal zu verwendenden Lizenzen eingeschränkt werden. Die Einstellungen der Optimierungsalgorithmen werden dann dementsprechend angepasst, sodass möglichst die maximale Anzahl an Lizenzen genutzt wird. Wird ein Pool an Lösungen für den Optimierungsalgorithmus benötigt, so ist die Größe des Pools ein

Vielfaches der zur Verfügung stehenden Lizenzen. Diese Lösungen werden in Blöcken berechnet, dessen Größe durch die Anzahl an Lizenzen bestimmt wird. Erst wenn alle Lösungen eines Blocks komplett berechnet wurden, werden die Lösungen des nächsten Blockes berechnet. Dies ist zwar nicht so effizient, als wenn sofort nachdem eine Lösung fertig berechnet wurde, die nächste Berechnung beginnen würde, jedoch würde es so zu einer permanenten Inanspruchnahme der Lizenzen kommen. Dies wäre ein Problem, falls sonst keine freien Lizenzen mehr zur Verfügung stehen würden und die Optimierungssoftware weitere Projekte blockieren würde. Die Unterschiede in den Berechnungszeiten liegen allerdings nur im Bereich von ein paar Sekunden, sodass die Verzögerung kaum eine Rolle spielt. Die Anzahl der zur Verfügung stehenden Prozessoren ist um ein Vielfaches höher als die Anzahl der vorhandenen Lizenzen, sodass darauf keine Rücksicht genommen werden muss.

Grafische Benutzeroberfläche (GUI)

Die grafische Oberfläche (Abbildung 4.2) dient hauptsächlich der Konfiguration der Algorithmen. Es können alle wesentlichen Dateipfade, der Befehl für den GT-SUITE Solver, die zu optimierenden Parameter, Werte für die Zielfunktion, Datenbankeinstellungen sowie Laufzeit bestimmende Einstellungen bestimmt werden. Alle getätigten Einstellungen werden in eine Konfigurationsdatei geschrieben. Diese muss dann von der Optimierungssoftware ausgelesen werden. Es gibt die Möglichkeit, die Konfigurationsdatei, sowie alle relevanten Projektdateien, mithilfe der GUI an einen beliebigem im Netzwerk befindlichen Computer zu kopieren und den Optimierungsprozess zu starten, falls die erforderliche Software vorhanden ist.

Es ist auch möglich den Prozess zu pausieren und abzubrechen. Weiters können auch Informationen über die bisher beste Lösung und die gesamte Laufzeit der Optimierungssoftware angezeigt werden. Es gibt auch die Möglichkeit, sich Flächen des Surface-Fitting (siehe Abbildung 4.3) anzeigen zu lassen.

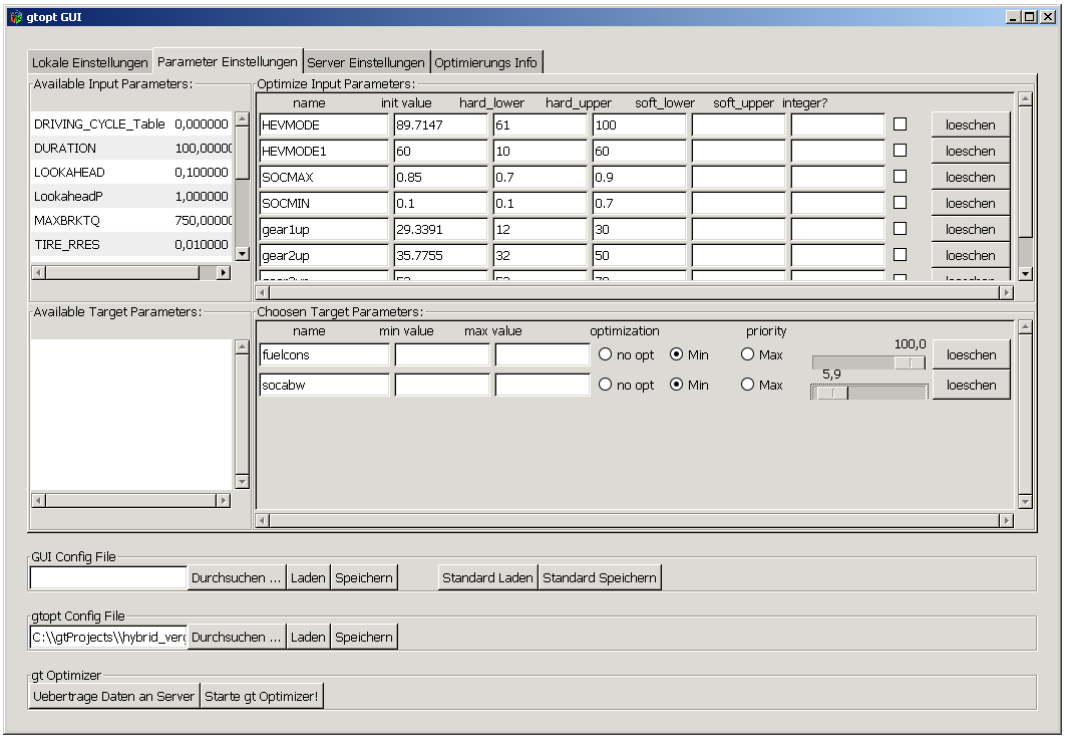

Abbildung 4.2: Grafische Benutzeroberfläche

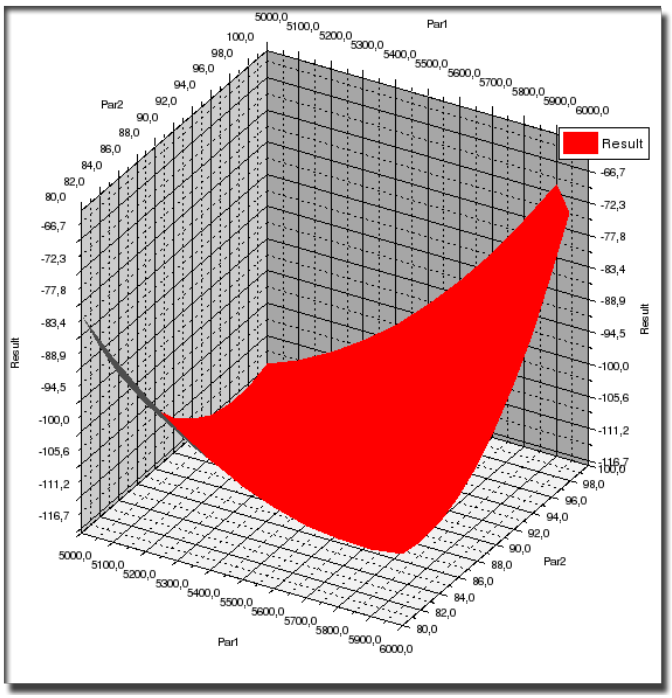

Abbildung 4.3: Surface-Fitting Beispielfläche

KAPITEL 5

Experimentelle Resultate

Für das IFAHEV wurden mehrere Betriebsstrategien mit einer unterschiedlichen Anzahl an zu optimierenden Parametern festgelegt. Während der Entwicklung der Optimierungsalgorithmen wurde auch das Modell und dessen Betriebsstrategie weiter entwickelt. Erste Erkenntnisse hatten gezeigt, dass es nicht ausreichte bei einem unausgeglichenem SOC einen Penalty beim Treibstoffverbrauch zu berücksichtigen (siehe Kapitel 4.1), sondern zusätzlich die quadratische Abweichung zu einem ausgeglichenem SOC betrachtet werden muss. Dadurch wurde es möglich, dass alle besten Lösungen einen nahezu ausgeglichenen SOC aufwiesen und ein objektiver Vergleich möglich war. Bei allen Strategien kam eine Start-Stopp Automatik zum Einsatz, die im NEFZ in etwa zehn Prozent Treibstoffeinsatz, im Vergleich zu einem konventionellen Modell, spart. Weiters konnte auch bei jeder Verzögerung des Fahrzeugs Rekuperation angewendet werden, sodass diese nicht optimiert werden musste.

Zu Beginn wurde nur die Schaltstrategie, ab welcher Drehzahl in den nächst höheren Gang geschaltet werden soll, mit fünf Parametern optimiert. Die Drehzahl, bei welcher der nächst niedrigere Gang verwendet werden sollte, wurde konstant festgelegt. Alle HEV-spezifischen Einstellungen wurden so gewählt, dass am Ende des Fahrzyklus ein ausgeglichener SOC gegeben ist. Die Schaltstrategie hat vor allem Einfluss auf den Treibstoffverbrauch des Fahrzeugs, aber auch einen geringen Einfluss auf die Änderung des SOCs. Dies ist darauf zurückzuführen, dass die Schaltstrategie auch die E-Maschine beeinflusst und diese ebenfalls einen drehzahlabhängigen Wirkungsgrad aufweist. In der Praxis wird meist davon ausgegangen, dass, wenn bei niedrigeren Drehzahlen bzw. Geschwindigkeiten bereits in den nächst höheren Gang geschaltet wird, der Treibstoffverbrauch auch niedriger ausfällt. Diese Tendenz hat sich auch bei diesem Modell bewahrheitet. Das Problem dabei ist allerdings, dass die Dynamik des Fahrzeugs bei zu niedrigen Drehzahlen zu gering ist und somit eine vom Hersteller vorgegebene untere Grenze der Fahrdynamik einzuhalten ist. Dies hatte zur Folge, dass die unteren Drehzahlgrenzen angehoben wurden.

Als nächster Schritt wurde zusätzlich zur Schaltstrategie die Lastpunktanhebung bei beschleunigter und konstanter Geschwindigkeit im Optimierungsprozess mit einbezogen. Somit wurden insgesamt sieben Parameter optimiert. Die Lastpunktanhebung hat massiven Einfluss auf

den SOC. Aufgrund dessen, dass sonst keine Parameter optimiert wurden, die den SOC maßgeblich beeinflussen, führte eine Erhöhung oder eine Verminderung von nur einem der Parameter immer zu einem unausgeglichenen SOC. Somit musste, wenn von einer Lösung mit ausgeglichenem SOC ausgegangen wird, jeweils einer der Parameter erhöht und der andere vermindert werden, um wieder einen ausgeglichenen SOC zu erhalten. Das beste Parameterset brachte eine Treibstoffeinsparung von etwa 25 Prozent im Vergleich zu einem konventionellen Modell.

In weiterer Folge wurde die Lastpunktanhebung bei beschleunigter und konstanter Geschwindigkeit jeweils durch mehrere Parameter in Abhängigkeit der Position im NEFZ repräsentiert. Insgesamt wurden zehn Parameter optimiert. Diese Maßnahme brachte eine Treibstoffeinsparung von etwa 28 Prozent im Vergleich zu einem konventionellen Modell.

Als bisher letzter Schritt wurden auch noch die Geschwindkeitsschwellwerte für den rein elektrischen Betrieb zum Optimierungsprozess hinzugefügt. Weiters wurden die Parametergrenzen der Lastpunktanhebung in soweit verschoben, das damit auch das Boosten ermöglicht wurde. Dabei führen negative Parameterwerte zu einer Lastpunktanhebung und positive Werte zum Boosten. Bei diesem Modell wurden 12 Parameter optimiert. Die Parameteroptimierung dieses Modells brachte eine Treibstoffeinsparung von etwa 33 Prozent im Vergleich zu einem konventionellen Modell.

Die Einstellungen des PSAGADOs sind in Tabelle 5.1 angeführt. It gibt die Iterationen des Algorithmus an und $Samples/It$ die zu berechnenden Lösungen pro Iteration. Die Spalte $Samples$ gibt die gesamte Anzahl an zu berechnenden Lösungen für einen Optimierungsprozess an. Die tatsächliche Anzahl an zu berechnenden Lösungen hängt davon ab, wie viele Lösungen bereits während des Optimierungsprozesses aus der Datenbank entnommen werden können und wie oft der SIMPLEX Algorithmus tatsächlich ausgeführt wird. Als Vorgabe sollen höchstens 3600 Berechnungen erfolgen, wobei der Optimierungsprozess z.B. bei zu langer Laufzeit unterbrochen werden kann.

Verfahren	It	$Samples/It$	$Samples$	Aufwand/%	Misc. Settings
MC	15	35	525	11,1	$resize = 0.89$
PSAGADO	10	420	4200	88,9	
PSAGADO-PSO	10	25	2500	52,9	$factor_{local} = 0.3$ $factor_{global} = 0.7$
PSAGADO-FITTING	5	12	600	12,7	
PSAGADO-GA	10	5	500	10,6	
PSAGADO-SIMPLEX	15	3	600	12,7	
Gesamt			4725		

Tabelle 5.1: IFAHEV-Modell, Einstellungen des PSAGADOs

Um den Zeitaufwand für eine Lösungsberechnung zu verringern, war angedacht die Berechnung vorzeitig abzubrechen, sobald die Tendenz des Treibstoffverbrauchs auf eine schlechte Lösung hinweist. Um diesen Prozess, wann die Berechnung abgebrochen werden soll, zu steuern, wurde ein Simulated-Annealing Verfahren (siehe Kapitel 2.4) im GT-SUITE Modell angewendet. Die Idee war, je weiter der Optimierungsprozess fortgeschritten ist, die Toleranzgrenze für

den Treibstoffverbrauch zu senken. Das Problem dabei war, dass man erst relativ spät im Fahrzyklus eine Aussage treffen kann, ob es sich um eine wahrscheinlich gute oder schlechte Lösung handelt. Somit wurde diese Idee wieder verworfen, da man somit nur relativ wenig Zeit spart und die Werte für die Zielfunktion nicht speichern kann, wenn der Prozess frühzeitig abgebrochen wird.

Aufgrund dessen, dass die Veröffentlichung der Ergebnisse des IFAHEVs seitens des Herstellers bzw. Auftraggebers untersagt ist, und DOE in vertretbarer Zeit nicht ausführbar ist, wurde der Optimierungsalgorithmus auch auf ein bereits in GT-SUITE mitgeliefertes Modell angewendet. Somit ist es möglich, Vergleichswerte zwischen der integrierten DOE, den einzelnen Metaheuristiken und dem PSAGADO zu erhalten. Das für Vergleichszwecke herangezogene Modell simuliert einen parallelen hybriden Antriebsstrang (siehe Kapitel 2.2) bei dem auch in einen seriellen Modus geschalten werden kann. Um die Rechenzeit wesentlich zu verkürzen, wurde statt des NEFZ, der eine Dauer von etwa 20 Minuten hat, ein 100 Sekunden langer Fahrzyklus (siehe Abbildung 5.1) mit einer Überland- und Stadtstrecke verwendet.

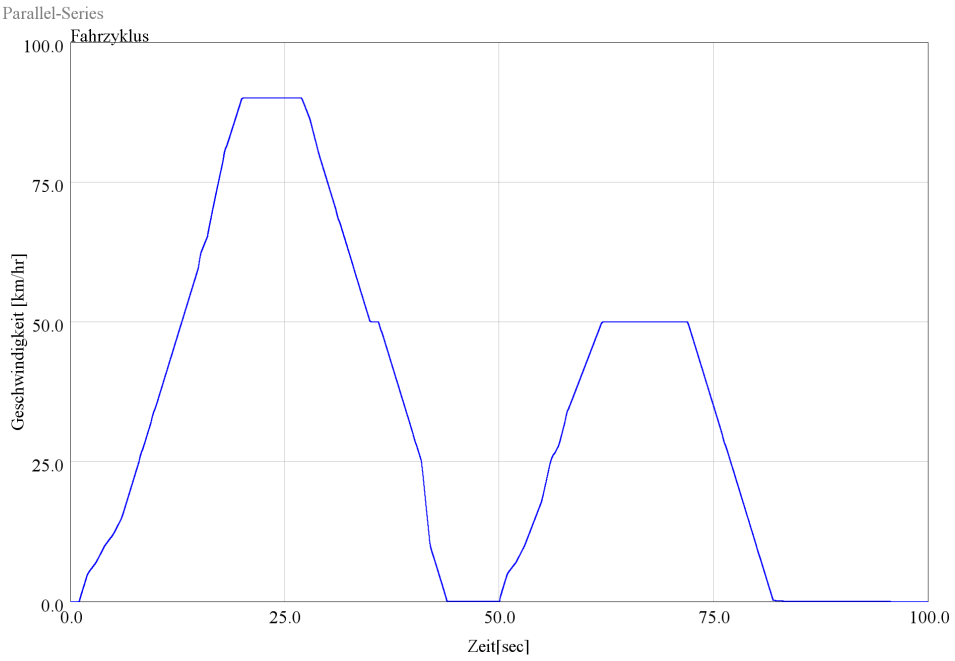

Abbildung 5.1: Fahrzyklus

Optimiert wurde die Schaltstrategie $gear1up \ldots gear4up$, die Ent- und Beladungsgrenzen der Batterie SOC_{max}, SOC_{min} und die Schwellwerte $hev1$, $hev2$, bei welchen Geschwindigkeiten zwischen parallelen und seriellen Betrieb umgeschaltet werden soll. Die Bereichsgrenzen sind in Tabelle 5.2 angeführt.

parameter	lower b.	upper b.
$hev1$ [km/h]	65	100
$hev2$ [km/h]	10	60
SOC_{max}	0,7	0,9
SOC_{min}	0,1	0,7
$gear1up$ [km/h]	12	30
$gear2up$ [km/h]	32	50
$gear3up$ [km/h]	52	70
$gear4up$ [km/h]	72	100

Tabelle 5.2: Paralleles Hybrid Modell - Parametergrenzen

Die Zielfunktion setzt sich ebenfalls aus dem Treibstoffeinsatz und der Abweichung des SOC zusammen, allerdings ohne einer Anpassung (Penalty) des Treibstoffverbrauchs, falls der SOC nicht ausgeglichen ist (siehe Kapitel 4.1). Die Abweichung wurde mithilfe der Formel 5.1 bestimmt.

$$SOC_{abw} = 10 + ((SOC_{ende} - SOC_{start}) * 300)^2 \tag{5.1}$$

Der Treibstoffeinsatz $fuelcons$ gemessen in Gramm wurde mit 3600 multipliziert. Für die beiden Extremfälle, ausgeglichener SOC und hoher Treibstoffeinsatz, sowie ein SOC mit großer Abweichung zum Ausgangs-SOC und kein Treibstoffeinsatz, ergibt sich damit ungefähr derselbe Zielfunktionswert. Die Zielfunktion $t = SOC_{abw} + fuelcons$ ist in diesem Fall zu minimieren.

Für DOE wurden 3600 Lösungen berechnet. Um die Verfahren vergleichen zu können, wurden bei dem PSAGADO ebenfalls nur Resultate berücksichtigt, die bis 3600 Lösungsberechnungen entstanden sind. Dies entspricht auch in etwa dem Zeitaufwand von 60 Stunden, den eine Berechnung in der Praxis nicht überschreiten sollte. Eine Lösungsberechnung für das Modell mit dem verkürzten Zyklus dauert in etwa 15 Sekunden. Somit ergibt sich eine Gesamtlaufzeit der DOE von etwa 15h. Für die Auswahl der Parametersets in DOE wurde die Latin-Hypercube Methode gewählt. Das mathematische Modell wurde sowohl quadratisch als auch kubisch approximiert. Der Zeitaufwand für die Berechnung der Koeffizienten der Ersatzfunktion, sowie die anschließende Optimierung mithilfe eines GAs, kann im Vergleich zur Lösungsberechnung vernachlässigt werden. Es existiert weiters noch die Möglichkeit eines komplexen Modells mit radialer Basisfunktion (RBF) [Buhmann 00], allerdings konnte dieses für die Anzahl an zu berechnenden Lösungen und zu optimierenden Parameter am Testsystem, aufgrund von unzureichenden Rechnerressourcen, nicht verwendet werden. Die Parametereinstellungen sind in Tabelle 5.3 und Tabelle 5.4, die dazugehörigen Zielfunktionswerte in Tabelle 5.5 und Tabelle 5.6 angeführt.

Die Differenz des Ergebnisses der DOE und der tatsächlichen Zielfunktionswerte ergibt sich dadurch, dass bei DOE die Optimierung auf das approximierte Modell angewendet wird. Somit kann es auch, wie in Tabelle 5.5 und Tabelle 5.6 ersichtlich, zu einem negativen Treibstoffverbrauch kommen, der praktisch nicht möglich ist. Dieser Effekt kann beim PSAGADO nicht auftreten, da auch alle Zielfunktionswerte während des Optimierungsprozesses den tatsächlichen

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	65.0	60.0	0.7	0.31877	24.38710	32.03519	58.73900	78.97947
2	65.0	60.0	0.7	0.31877	25.03812	32.0	58.73900	80.95015
3	65.0	60.0	0.7	0.31877	25.03812	32.03519	58.73900	81.03226
4	65.0	60.0	0.7	0.31818	24.87977	32.05279	58.96774	80.51222
5	65.0	60.0	0.7	0.31760	25.39003	32.0	59.19648	80.950147
6	65.0	60.0	0.7	0.31877	25.10850	32.03519	58.93255	80.84067
7	65.0	60.0	0.7	0.31818	24.95015	32.03519	58.73900	80.73118
8	65.0	60.0	0.7	0.31877	25.17889	32.03519	58.96774	80.95015
9	65.0	60.0	0.7	0.31877	25.02053	32.03519	58.68622	80.95015
10	65.0	60.0	0.7	0.31877	24.40470	32.0	58.73900	78.979472

Tabelle 5.3: Paralleles Hybrid Modell - Berechnete Parametersets der DOE (quadratisches Modell)

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	100.0	60.0	0.9	0.1	25.95308	44.72141	53.72434	76.21505
2	100.0	60.0	0.9	0.1	25.86510	44.68622	53.72434	76.13294
3	100.0	60.0	0.9	0.1	25.95308	44.68622	53.72434	76.21505
4	100.0	60.0	0.9	0.1	25.95308	44.72141	53.72434	76.21505
5	100.0	60.0	0.9	0.1	25.46041	44.73900	53.67155	76.21505
6	100.0	60.0	0.9	0.1	25.39003	45.30205	53.86510	76.5435
7	100.0	60.0	0.9	0.1	26.0235	44.82698	53.68915	76.10557
8	100.0	60.0	0.9	0.1	26.09384	40.99120	53.44282	76.32454
9	100.0	60.0	0.9	0.1	26.14663	40.99120	53.44282	76.32454
10	100.0	60.0	0.9	0.1	26.14663	45.58358	53.44282	76.21505

Tabelle 5.4: Paralleles Hybrid Modell - Berechnete Parametersets der DOE (kubisches Modell)

Werten entsprechen. Es ist zwar möglich anzugeben, dass der Treibstoffverbrauch z.B. größer gleich 10 sein muss, jedoch werden durch diese Einschränkung noch schlechtere Ergebnisse erzielt. Aus diesem Grund wurde auf eine Einschränkung verzichtet.

Beim PSAGADO wurden beim Monte-Carlo Suchverfahren 20 Durchläufe mit jeweils 25 Samples durchgeführt. Der Optimierungsprozess des PSAGADOs wurde mit 16 Samples in zehn Durchläufen durchgeführt. Insgesamt wurde der Algorithmus zehn Mal ausgeführt. Die sonstigen Einstellungen der Algorithmen wurden wie in Tabelle 5.1, Spalte Misc. Settings vorgenommen. Um die Durchläufe sinnvoll vergleichen zu können, wurde jeweils eine leere Datenbank verwendet, sodass keine berechneten Lösungen der vorangegangenen Durchläufe verwendet werden konnten. Aufgrund der Parallelisierung konnten am Testsystem zwei Lösungen gleichzeitig berechnet werden und der Zeitaufwand belief sich auf acht Stunden pro Durchlauf. Die Parametereinstellungen der jeweils besten Lösung sind in Tabelle 5.7 und die dazugehörigen

	Berechnete Lösungswerte			Tatsächliche Lösungswerte		
Set	SOC_{abw}	$fuelcons$	$targetval$	SOC_{abw}	$fuelcons$	$targetval$
1	156.706732	-0.847423	155,859309	147.15863	66.092450	213,25108
2	157.453056	-1.669649	155,783407	147.41412	66.345870	213,75999
3	157.433677	-1.618445	155,815232	147.39129	66.335433	213,726723
4	157.348521	-1.520508	155,828013	148.08298	66.347728	214,430708
5	157.958635	-2.171511	155,787124	149.02718	66.481011	215,508191
6	157.590418	-1.780531	155,809887	148.11776	66.356784	214,474544
7	157.31011	-1.494235	155,815875	147.39930	66.307476	213,706776
8	157.678015	-1.868305	155,80971	148.29475	66.407338	214,702088
9	157.386331	-1.568208	155,818123	147.27769	66.325199	213,602889
10	156.745605	-0.917685	155,82792	147.06921	66.129301	213,198511
		Maximum	155,859309		Maximum	215,508191
		Minimum	155,783407		Minimum	213,198511
		Durchschn.	155,81546		Durchschn.	214,03615
		Std.Abw.	0,021440679		Std.Abw.	0,726057021

Tabelle 5.5: Paralleles Hybrid Modell - Zielfunktionswerte der DOE (quadratisches Modell)

Zielfunktionswerte in Tabelle 5.8 angeführt. In den Tabellen geben die Spalten $quadratisch/\%$ bzw. $kubisch/\%$ an, um wieviel Prozent der Durchlauf des PSAGADOs besser war, als der beste DOE Durchlauf mit quadratischer bzw. kubischer Ersatzfunktion. Der beste Durchlauf wird deswegen als Vergleich herangezogen, da der Zeitaufwand, im Vergleich zur Lösungsberechnung, für die Optimierung mit dem GA in DOE vernachlässigbar ist. Der hauptsächliche Aufwand bei DOE ergibt sich durch die zu berechnenden Lösungen für die Erstellung des approximierten mathematischen Modells. Der GA kann praktisch beliebig oft auf dieses angewendet werden, ohne dass sich die Gesamtlaufzeit wesentlich erhöht, wobei sich die Lösungen der unterschiedlichen Runs kaum unterscheiden.

Der Wertebereich der Zielfunktionswerte ist relativ klein. Dies liegt vor allem an dem relativ kurzen Fahrzyklus. Sieht man sich den Verlauf des Optimierungsprozesses an, so erkennt man mehrere lokale Optima von denen man durch die Veränderung von nur einem Parameter nicht entkommen kann. Tendiert das Monte-Carlo Suchverfahren zu einem lokalen Optimum, dessen Zielfunktionswert schlechter ist als die endgültige Lösung des Prozesses, so kann es relativ lange dauern, bis der Algorithmus diese endgültige Lösung erreicht. Dies liegt vor allem daran, dass bereits relativ viele Lösungen in der Datenbank mit ähnlichen Parametersettings durch das Monte-Carlo Suchverfahren vorhanden sind und diese vom Optimierungsprozess als Ausgangslösungen genutzt werden. Um dies zu verhindern, könnte der Bereichsverkleinerungsfaktor erhöht, oder die Anzahl der Durchläufe beim Monte-Carlo Suchverfahren herabgesetzt werden. Eine andere Möglichkeit wäre gänzlich auf das Monte-Carlo Suchverfahren zu verzichten und nur Zufallslösungen zu verwenden. Allerdings muss man bedenken, dass nur 3600 Berechnungen zur Verfügung stehen und somit kann es sinnvoll sein, bereits bei der Generierung der Zufallslösungen eine gewisse Gewichtung des Suchraums vorzunehmen, auch wenn

	Berechnete Lösungswerte			Tatsächliche Lösungswerte		
Set	SOC_{abw}	$fuelcons$	$targetval$	SOC_{abw}	$fuelcons$	$targetval$
1	242.040243	-124.924856	117,115387	145.51055	64.730499	210,241049
2	241.844808	-124.727637	117,117171	145.45462	64.732089	210,186709
3	242.029918	-124.914509	117,115409	145.70094	64.727435	210,428375
4	242.040243	-124.924856	117,115387	145.51055	64.730499	210,241049
5	241.634092	-124.469291	117,164801	145.51113	64.764658	210,275788
6	242.376304	-125.143376	117,232928	145.50602	64.813311	210,319331
7	241.94125	-124.822253	117,118997	145.61234	64.711133	210,323473
8	242.529408	-125.067451	117,461957	146.15772	64.715177	210,872897
9	242.567238	-125.102741	117,464497	146.01495	64.734751	210,749701
10	242.327348	-125.060149	117,267199	145.67409	64.663025	210,337115
		Maximum	117,464497		Maximum	210,872897
		Minimum	117,115387		Minimum	210,186709
		Durchschn.	117,2173733		Durchschn.	210,3975487
		Std.Abw.	0,140355593		Std.Abw.	0,229598277

Tabelle 5.6: Paralleles Hybrid Modell - Zielfunktionswerte der DOE (kubisches Modell)

man dadurch riskiert in den Bereich eines lokalen Optimums zu kommen, das nicht das globale Optimum ist. Wie in Tabelle 5.8 ersichtlich, liefert der PSAGADO immer bessere Ergebnisse. Dies ist einerseits auf ein ungenau approximiertes Modell von DOE zurückzuführen und andererseits auf die Auswahl der zu berechnenden Parametersets. Diese werden zwar möglichst gleichmäßig auf den Suchraum verteilt erzeugt, jedoch gibt es keinerlei Gewichtung. Dies kann dazu führen, dass gewisse Bereiche, in denen mit ziemlicher Sicherheit ausgeschlossen werden kann, dass sich dort das globale Optimum befindet, trotzdem in größerem Maße berücksichtigt werden und somit für die relevanten Bereiche zu wenig Lösungen zur Verfügung stehen.

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	65	60	0.79	0.5	29.9316	47.84	57.5267	72
2	65	59.9954	0.700349	0.556227	30	47.4223	57.79	72
3	65	60	0.807692	0.493948	30	46.7909	57.3753	72
4	65	60	0.7	0.36914	30	47.5334	52.1723	72.1723
5	65	60	0.786806	0.507374	29.4562	48.0416	57.5695	72.3901
6	65	60	0.7	0.619494	29.6498	47.3688	57.5825	72
7	65	60	0.727707	0.5	29.12	46.5826	57.5693	72
8	65	60	0.840951	0.586519	30	46.4543	57.5545	72.2661
9	65	60	0.786806	0.507374	29.4562	48.0416	57.5695	72.3901
10	65.0791	60	0.7	0.247083	30	47.7688	57.7835	72

Tabelle 5.7: Paralleles Hybrid Modell - Berechnete Parametersets des PSAGADOs

Run	SOC_{abw}	$fuelcons$	$targetval$	$quadratisch/\%$	$kubisch/\%$
1	141,968	64,7216	206,6896	3,05	1,66
2	142,04	64,7371	206,7771	3,01	1,62
3	142,023	64,7182	206,7412	3,03	1,64
4	143,385	64,139	207,524	2,66	1,27
5	142,127	64,7489	206,8759	2,97	1,58
6	142,177	64,7169	206,8939	2,96	1,57
7	142,224	64,7201	206,9441	2,93	1,54
8	142,127	64,75	206,877	2,97	1,57
9	142,127	64,7489	206,8759	2,97	1,58
10	142,361	64,6564	207,0174	2,90	1,51
		Maximum	207,524		
		Minimum	206,6896		
		Durchschn.	206,92161		
		Std.Abw.	0,232558592		

Tabelle 5.8: Paralleles Hybrid Modell - Zielfunktionswerte des PSAGADOs

Die einzelnen Metaheuristiken wurden zu Vergleichszwecken ebenfalls auf das Modell angewendet. Alle Algorithmen wurden ebenfalls zehn Mal ausgeführt und dieselben 500 durch das Monte-Carlo Suchverfahren berechneten Parametersets als Ausgangslösungen benutzt. Die beste Lösung des MCs hatte einen Zielfunktionswert von 230,59. Der GA hat dabei von den einzelnen Algorithmen am besten abgeschnitten, da durch Mutation vom Lösungsbereich in dem die VKM nicht verwendet wird, entkommen werden konnte. Die Ergebnisse des PSO hängen sehr stark von den gewählten Ausgangslösungen ab. Würde man nur den PSO alleine verwenden, so müsste man garantieren, dass die Lösungen möglichst gut im Suchraum verteilt sind. Weiters müsste man sicherstellen, dass zumindest eine Lösung einen ausgeglichen SOC hat. Sind sich die Lösungen zu ähnlich, so können keine Verbesserungen mehr erzielt werden. Ähnlich verhält es sich beim Downhill-Simplex Verfahren. Werden schlechte Ausganslösungen gewählt kann kaum eine bessere Lösung gefunden werden. Die Ergebnisse sind in Tabelle 5.9 ersichtlich. It gibt die Iterationen des Algorithmus an und $Samples/It$ die zu berechnenden Lösungen pro Iteration. Beim PSAGADO ist der maximale Wert angegeben, der tatsächliche Wert hängt davon ab, ob das Downhill-Simplex Verfahren in der Iteration angewendet wurde oder nicht. $target/min$, $target/max$ und $target/avg$ gibt dabei jeweils den kleinsten, größten und durchschnittlich erreichten Zielfunktionswert an.

Jeweils zwei charakteristische Verläufe jedes Verfahrens des Optimierungsprozesses sind in Abbildung 5.2 dargestellt. Der PSAGADO1 Verlauf ist charakteristisch für den kombinierten Optimierungsansatz. PSAGADO2 stellt die Ausnahme dar. Im Vergleich zum Downhill-Simplex Verfahren und PSO (Simplex1/PSO1) kann hier allerdings noch von dem möglichen Optimum, das nicht das globale Optimum ist, entkommen werden. Bei den Runs Simplex1 und PSO1 war der Verlauf nahezu ident, sodass diese in der Grafik zusammengefasst wurden. Bei Simplex2 sieht man bei etwa 1900 berechneten Lösungen, dass keine Verbesserung mehr erzielt werden

kann und der Algorithmus mit neuen Lösungen neu gestartet wird. Dadurch ergibt sich eine weitere erhebliche Verbesserung. An die Lösungswerte des PSAGADOs kommt das Verfahren allerdings nicht heran. Der Verlauf des GAs sieht bei allen Durchläufen ähnlich aus, man sieht hier nach 500 berechneten Lösungen einen deutlichen Sprung, der durch Mutation ausgelöst wurde. Sobald dadurch eine wesentlich bessere Lösung gefunden wurde, konnten in weiterer Folge auch durch Rekombination bessere Ergebnisse erzielt werden. Beim PSAGADO1 konnte ebenfalls durch Mutation ein deutlicher Sprung erreicht werden. Die weiteren Verbesserungen ergaben sich dann meist durch den PSO Algorithmus und die Rekombination beim GA. Dadurch, dass die Mutation eine Änderung eines Parameters im gesamten zulässigen Wertebereich bewirken kann, ist der GA meist für große Sprünge im Verlauf verantwortlich, die kleineren Verbesserungen werden meist vom PSO erzielt.

Verfahren	It	$Samples/It$	$target/max$	$target/min$	$target/avg$
PSAGADO	10	420	207,524	206,6896	206,92161
PSO	100	40	229,934	207,2231	212,432
GA	400	10	208,6398	206,98	207,2321
SIMPLEX	1000	4	230,57	207,94	215,9323

Tabelle 5.9: Paralleles Hybrid Modell - Zielfunktionswerte der Metaheuristiken und des PSAGADOs

Weiters wurden die HEV-spezifischen Parameter und die Schaltstrategie getrennt mit jeweils vier Parametern optimiert. Aufgrund der geringeren Parameteranzahl wurde die Anzahl der zu berechnenden Lösungen auf 1000 begrenzt. Dies verkürzt die Ausführungszeit von DOE auf etwa vier Stunden. In realer Anwendung würden 1000 zu berechnende Lösungen in etwa der Anzahl an Lösungen entsprechen, die zwischen zwei Arbeitstagen berechnet werden können. Aufgrund dessen, dass die DOE mit dem kubischen Vergleichsmodell bessere Ergebnisse erzielt hat, wurde nur mehr dieser Ansatz betrachtet. Weiters wurde auch nur ein Durchlauf berechnet, da sich die Lösungswerte von verschiedenen Durchläufen unwesentlich unterscheiden und nur für Vergleichszwecke herangezogen werden.

Die Parameterwerte der besten gefundenen Lösung von DOE für die HEV-spezifischen Parameter sind in Tabelle 5.10 und die zugehörigen Zielfunktionswerte in Tabelle 5.11 angeführt. Die Werte der Schaltstrategie sind ebenfalls angeführt, wurden aber nicht optimiert, sondern als Konstanten festgelegt. Die Ergebnisse für die Abweichung des SOCs und der Treibstoffverbrauch sind diesmal beide negativ. Die Lösung würde einer rein elektrischen Fahrt entsprechen. Berechnet man das Modell mit den vorgeschlagenen Parametern, so erhält man wiederum ein komplett unterschiedliches Ergebnis. Im Vergleich zu dem Optimierungsprozess mit acht Parametern, wo auch negative Zielfunktionswerte auftraten, ist diesmal das Ergebnis wesentlich schlechter im Vergleich zu den Ergebnissen des PSAGADOs (siehe Tabelle 5.13).

Beim PSAGADO wurden beim Monte-Carlo Suchverfahren 10 Durchläufe mit jeweils 15 Samples und einem Bereichsverkleinerungsfaktor von 0.89 durchgeführt. Der Optimierungsprozess des PSAGADOs wurde mit 12 Samples in fünf Durchläufen angewendet. Insgesamt wurde der Algorithmus zehn Mal ausgeführt.

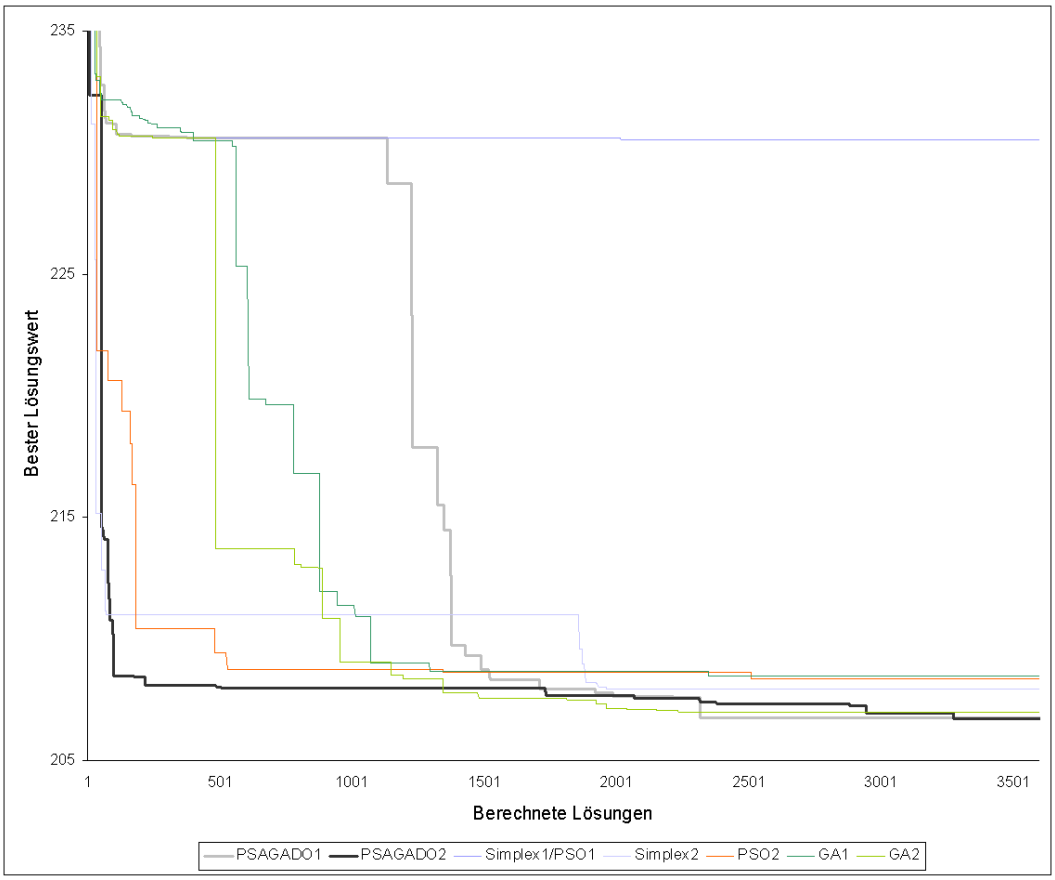

Abbildung 5.2: Paralleles Hybrid Modell - Verlauf des Optimierungsprozesses der Metaheuristiken und des PSAGADO

Run	*hev1*	*hev2*	SOC_{max}	SOC_{min}	*gear1up*	*gear2up*	*gear3up*	*gear4up*
1	100	10	0.5	0.11955	25	35	55	70

Tabelle 5.10: Paralleles Hybrid Modell, HEV-spez. Parameter - Berechnete Parametersets der DOE (kubisches Modell)

Die Parameterwerte der besten gefundenen Lösung des PSAGADOs sind in Tabelle 5.12 und die dazugehörigen Zielfunktionswerte in Tabelle 5.13 angeführt. Bis auf einen Durchlauf liegen wieder alle Ergebnisse innerhalb eines sehr kleinen Wertebereichs. Bei dem Durchlauf mit dem schlechtesten Ergebnis, welches bereits durch das Monte-Carlo Verfahren erzielt wurde, konnte dann keine Verbesserung mehr durch den weiteren Optimierungsprozess erzielt werden. Grund dafür war, dass beim PSO bereits Lösungen gewählt wurden, die der besten Lösung des Monte-Carlo Suchverfahrens ähnlich waren. In diesem Fall wäre es besser gewesen den Be-

	Berechnete Lösungswerte			Tatsächliche Lösungswerte		
Set	SOC_{abw}	$fuelcons$	$targetval$	SOC_{abw}	$fuelcons$	$targetval$
1	-1583,56	-13,004571	-1596,564571	222.34424	10	232.34424

Tabelle 5.11: Paralleles Hybrid Modell, HEV-spez. Parameter - Zielfunktionswerte der DOE (kubisches Modell)

reichsverkleinerungsfaktor zu senken bzw. weniger Durchläufe im Monte-Carlo Suchverfahren durchzuführen. Aufgrund der stark begrenzten Anzahl an gesamten Durchläufen ist es schwierig einen guten Ausgleich zwischen Intensivierung und Diversifizierung zu finden.

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	50.0577	15.2701	0.507594	0.431547	25	35	55	70
2	50.0626	30	0.869711	0.342976	25	35	55	70
3	50.0905	30	0.614425	0.1	25	35	55	70
4	50.0173	30	0.847602	0.1	25	35	55	70
5	50.1175	29.7121	0.664156	0.336195	25	35	55	70
6	50.0352	30	0.716934	0.459446	25	35	55	70
7	50.0191	30	0.839728	0.34	25	35	55	70
8	50.0811	30	0.9	0.114089	25	35	55	70
9	50.12	30	0.698	0.458	25	35	55	70
10	50.0832	30	0.73	0.49	25	35	55	70

Tabelle 5.12: Paralleles Hybrid Modell, HEV-spez. Parameter - Berechnete Parametersets des PSAGADOs

Die Parametereinstellungen der DOE für die Schaltstrategie sind in Tabelle 5.14 und die dazugehörigen Zielfunktionswerte in Tabelle 5.15 angeführt. Die Werte der HEV-spezifischen Parameter sind ebenfalls angeführt, wurden aber nicht optimiert, sondern als Konstanten festgelegt. Bei diesem Modell wären die berechneten Zielfunktionswerte in einem theoretisch möglichen Bereich, jedoch ist die Differenz zu den tatsächlichen Werten wiederum sehr groß. Der tatsächliche Zielfunktionswert ist diesmal aber wesentlich kleiner als beim vorangegangenen Modell und ist auch nur geringfügig schlechter als die Ergebnisse des PSAGADOs.

Beim PSAGADO wurden beim Monte-Carlo Suchverfahren 10 Durchläufe mit jeweils 15 Samples und einem Bereichsverkleinerungsfaktor von 0.89 durchgeführt. Der Optimierungsprozess des PSAGADOs wurde mit 12 Samples in fünf Durchläufen angewendet. Insgesamt wurde der Algorithmus zehn Mal ausgeführt.

Die Parametereinstellungen des PSAGADOs sind in Tabelle 5.16 und die dazugehörigen Zielfunktionswerte in Tabelle 5.17 angeführt.

Eine Übersicht über alle Verfahren liefert die Tabelle 5.18. min, max und avg geben dabei jeweils den kleinsten, größten und durchschnittlichen Zielfunktionswert an. $stadev$ gibt die Standardabweichung an.

Run	SOC_{abw}	$fuelcons$	$targetval$	$kubisch/\%$
1	119,41	91,8493	211,2593	9,07
2	116,458	90,3302	206,7882	11,00
3	116,676	90,2511	206,9271	10,94
4	116,464	90,3908	206,8548	10,97
5	116,611	90,2736	206,8846	10,96
6	116,408	90,336	206,744	11,02
7	116,464	90,3908	206,8548	10,97
8	116,408	90,336	206,744	11,02
9	116,558	90,2459	206,8039	10,99
10	116,546	90,2863	206,8323	10,98
		Maximum	211,2593	
		Minimum	206,744	
		Durchschn.	207,2693	
		Std.Abw.	1,403159886	

Tabelle 5.13: Paralleles Hybrid Modell, HEV-spez. Parameter - Zielfunktionswerte des PSAGADOs

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	65	60	0.7	0.3	15	25	40	69.41349

Tabelle 5.14: Paralleles Hybrid Modell, Schaltstrategie - Berechnete Parametersets der DOE (kubisches Modell)

	Berechnete Lösungswerte			Tatsächliche Lösungswerte		
Set	SOC_{abw}	$fuelcons$	$targetval$	SOC_{abw}	$fuelcons$	$targetval$
1	51.515021	33.21851	84,733531	147.15707	62.096246	209.253316

Tabelle 5.15: Paralleles Hybrid Modell, Schaltstrategie - Zielfunktionswerte der DOE (kubisches Modell)

Run	$hev1$	$hev2$	SOC_{max}	SOC_{min}	$gear1up$	$gear2up$	$gear3up$	$gear4up$
1	65	60	0.7	0.3	18.1086	37.2463	59.1816	87.833
2	65	60	0.7	0.3	20.6553	40	51.032	70.1022
3	65	60	0.7	0.3	23.0123	40	50.0018	70.8382
4	65	60	0.7	0.3	20.8729	40	57.6209	70.4232
5	65	60	0.7	0.3	24.15	40	57.4784	70.4456
6	65	60	0.7	0.3	21.5093	40	59.07	87.9881
7	65	60	0.7	0.3	21.2481	39.4962	50.9919	70.0278
8	65	60	0.7	0.3	22.6826	39.4849	57.5939	70.1729
9	65	60	0.7	0.3	21.834	40	57.4753	70.6256
10	65	60	0.7	0.3	22.3459	39.4883	57.7218	70.0076

Tabelle 5.16: Paralleles Hybrid Modell, Schaltstrategie - Berechnete Parametersets des PSAGADOs

Run	SOC_{abw}	$fuelcons$	$targetval$	$kubisch/\%$
1	138,746	69,2403	207,9863	0,61
2	144,482	63,8085	208,2905	0,46
3	144,034	63,782	207,816	0,69
4	142,984	64,5796	207,5636	0,81
5	143,263	64,5843	207,8473	0,67
6	138,65	68,9867	207,6367	0,77
7	144,546	63,8009	208,3469	0,43
8	143,044	64,5421	207,5861	0,80
9	143,034	64,6112	207,6452	0,77
10	142,995	64,5361	207,5311	0,82
		Maximum	208,3469	
		Minimum	207,5311	
		Durchschn.	207,82497	
		Std.Abw.	0,297252444	

Tabelle 5.17: Paralleles Hybrid Modell, Schaltstrategie - Zielfunktionswerte des PSAGADO

Opt. Parameter	Verfahren	min	max	avg	$stadev$
8 Parameter	DOE_{quad}	213,1985	215,5082	214,0362	0,7261
	DOE_{cub}	210,1867	210,8729	210,3975	0,2210
	$PSAGADO$	206,6896	207,524	206,9216	0,2326
4 Parameter (Hybrid spez.)	DOE_{cub}	232,344	232,344	232,344	0
	$PSAGADO$	206,744	211,2593	207,2693	1,403
4 Parameter (Schaltstr.)	DOE_{cub}	209,2533	209,2533	209,2533	0
	$PSAGADO$	207,5311	208,3469	207,82497	0,2972

Tabelle 5.18: Paralleles Hybrid Modell, Übersicht

KAPITEL 6

Zusammenfassung

Ziel der Arbeit war es, einen neuen verbesserten Optimierungsalgorithmus für ein HEV-Modell zu entwickeln, mit dem es möglich ist, den Treibstoffverbrauch des Fahrzeuges im NEFZ weiter zu senken. Es sollten in etwa zehn Parameter optimiert werden, wobei eine höhere Anzahl an Parametern mit Weiterentwicklung des Projekts nicht ausgeschlossen wurde. Weiters wurde die maximale Zeit für einen Optimierungsprozess und die Anzahl an parallelen Berechnungen begrenzt.

Als Software für die Simulation des HEVs wurde GT-SUITE verwendet. Das Fahrzeug kann damit im gesamten Fahrzyklus mit entsprechenden Bedingungen simuliert werden. Dieser werden die Werte für die zu optimierenden Parameter übergeben und nach der Beendigung der Simulation erhält man den Treibstoffverbrauch und den SOC.

Um den Treibstoffeinsatz bewerten zu können, durchfährt ein Fahrzeug einen bestimmten Fahrzyklus. Dieser legt fest mit welchen Geschwindigkeiten und unter welchen Bedingungen ein Fahrzeug betrieben wird. Für die Ermittlung des Kraftstoffverbrauchs von Kraftfahrzeugen wird in der Europäischen Union der in Abbildung 1.1 gezeigte NEFZ verwendet. Um den Treibstoffverbrauch sinnvoll bewerten zu können, ist es bei HEVs wichtig, dass zu Beginn und am Ende des Fahrzyklus der SOC möglichst gleich ist (siehe Kapitel 2.2) [Hofmann 09].

Es wurde einerseits der Treibstoffverbrauch und andererseits die quadratische Abweichung zu einem ausgeglichenen SOC berücksichtigt. Der Treibstoffverbrauch wurde weiters angepasst, falls die Batterie am Ende eines Zyklus einen höheren oder geringeren SOC wie zu Beginn aufwies.

Da bei HEVs mehr Freiheitsgrade bei der Energieumwandlung zur Verfügung stehen, haben auch die Modelle erwartungsgemäß eine höhere Anzahl an zu optimierenden Parametern. Für das IFAHEV wurde die Schaltstrategie, die Lastpunktanhebung bzw. das Boosten (siehe Kapitel 2.2) und die Schwellwerte für den rein elektrischen Betrieb optimiert. Ein weiteres Einsparungspotenzial im Vergleich zu einem konventionellen Fahrzeug bieten Start/Stopp Systeme und die teilweise Rückgewinnung und Speicherung der Bremsenergie (Rekuperation). Diese mussten allerdings nicht optimiert werden, da das Start/Stopp System bei jedem Stillstand des Fahrzeugs aktiviert wurde und jeder Bremsvorgang vollständig mithilfe der Rekuperation erfol-

gen konnte [Hofmann 09].

Als erster Schritt wurde die bereits in der Software des Fahrzeugmodells enthaltene Optimierungssoftware analysiert (siehe Kapitel 3). Diese stellt grundsätzlich zwei Verfahren zur Verfügung - Design of Experiments (DOE) und ein direktes Optimierungsverfahren. Bei DOE sind für brauchbare mathematische Modelle eine Vielzahl an Durchläufen durchzuführen. Das direkte Optimierungsverfahren dient hauptsächlich zur Verbesserung einer bereits bestehenden Lösung, da es relativ schnell zu lokalen Optima konvergiert. Um die Auswahl der Parameter für die Optimierung automatisch bestimmen zu können, wurde eine Parameteranalyse entwickelt. Diese berechnet für jeden der möglichen Parameter eine gewisse Priorität, die dann vom eigentlichen Optimierungsprozess genutzt werden kann. Es konnten nur Strategien (siehe Kapitel 4.2) auf das Problem angewendet werden, die keine Ableitungen einer mathematischen Funktion benötigen.

Als erster Algorithmus wurde ein Monte-Carlo Suchverfahren implementiert, bei welchem innerhalb eines bestimmten Bereiches Zufallslösungen erzeugt werden. In Abhängigkeit der besten dieser Lösungen wird der Bereich verändert und um einen gewissen Faktor verkleinert. Im ersten Iterationsschritt wurde der gesamte Wertebereich der Parameter in Betracht gezogen und somit Zufallslösungen innerhalb der erlaubten Parametergrenzen erzeugt. Das Verfahren wurde hauptsächlich dazu verwendet, Ausgangslösungen für die weiteren Algorithmen zu erzeugen.

Weiters wurde ein Downhill-Simplex Algorithmus implementiert, der auf einem Simplex basiert, das in einem v-dimensionalen Raum aus $v + b$ Punkten aufgespannt wird. Jeder Punkt entspricht dabei einer Lösung. In jedem Iterationsschritt werden die b schlechtesten Punkte jeweils durch neue ersetzt. Dazu wird von allen v Lösungen der arithmetische Mittelwert jedes Parameters berechnet. Von den b Lösungen wird dann über Reflexion, Erweiterung oder Kontraktion der berechneten Mittelwertlösung versucht bessere Lösungen zu berechnen. Nach diesem Schritt werden erneut die b schlechtesten Lösungen bestimmt und die Prozedur beginnt erneut. Der Nachteil des Algorithmus ist, dass er schnell zu lokalen Optima konvergieren kann, die nicht das globale Optimum darstellen.

Außerdem wurde ein GA entwickelt. Die Auswahl der Individuen aus der Population zur Rekombination erfolgt per Zufall. Bei der Rekombination wird immer ein Wert einer Lösung übernommen und als Ergebnis erhält man somit nur eine neue Lösung. Die Auswahl, welcher Parameterwert von welcher Lösung übernommen wird, wird in Abhängigkeit der Zielfunktionswerte der jeweiligen Ausgangslösungen bestimmt. Weiters kann noch eine Wahrscheinlichkeit angegeben werden, mit der ein Parameter einen zufälligen Wert innerhalb seiner zulässigen Grenzen zugewiesen bekommt. Per Zufall wird ein Individuum der Population ausgewählt, welches dann, abhängig vom Zielfunktionswert, durch die neu erzeugte Lösung ersetzt wird.

Als nächstes wurde ein PSO Suchverfahren implementiert, das auf Schwarmintelligenz beruht. Jede Lösung entspricht einem Individuum des Schwarms, das sich innerhalb des Suchraums bewegen kann. Die Bewegung hängt von der besten bekannten Lösung des Individuums und des gesamten Schwarms ab. Bei der Initialisierung werden alle Individuen möglichst zufällig im Suchraum positioniert. Für jede Lösung wird ein Vektor definiert. Dieser gibt die Suchrichtung und Geschwindigkeit an, mit der sich ein Individuum bewegt.

Weiters wurde ein Surface-Fitting Algorithmus, aufbauend auf das in [Meywerk 07] vorgestellte Jacob-Verfahren, entwickelt. In jedem Iterationsschritt werden immer zwei zufällig

gewählte Parameter verändert und mithilfe von wenigen berechneten Parametersets wird eine quadratische Funktion approximiert. In weiterer Folge muss von der erzeugten Ersatzfunktion das Optimum berechnet werden. Beide Problemstellungen wurden mithilfe der GNU Scientific Library (GSL) realisiert. Um zu verhindern, dass nicht öfters hintereinander dieselben Parameter gewählt werden, werden diese in einer Tabu-Liste gespeichert.

Aufgrund dessen, dass sich bei den einzelnen Verfahren unterschiedliche Schwächen zeigten, wurden in einem hybriden Ansatz (PSAGADO) die zuvor vorgestellten Algorithmen miteinander kombiniert. Zu Beginn werden mithilfe des Monte-Carlo Suchverfahren Lösungen erzeugt, die in weiterer Folge als Ausgangslösungen dienen. Als zentraler Algorithmus dient der PSO. Er ist gut geeignet das Zentrum des gesamten Algorithmus zu bilden, da er ein robustes Verfahren ist welches anfangs Lösungen mit hoher Diversität betrachtet. Nach einer bestimmten Anzahl an Iterationen wird die beste Lösung des PSO Algorithmus mit dem Surface-Fitting Verfahren versucht weiter zu verbessern. Um den Aufwand einzuschränken wird das Surface-Fitting nur auf die beste Lösung angewendet. Anschließend wird auf die letzten berechneten Lösungen des PSOs der GA angewendet. Zu Beginn des GAs entspricht somit der Pool den Lösungen des PSOs. Durch den GA kann durch Rekombination und Mutation eventuell schneller eine bessere Lösung gefunden werden. Sind alle Individuen im Suchraum nahe beieinander, so kann dies trotzdem durch Mutation zu besseren Lösungen führen. Gibt es mehrere Lösungen verteilt im Suchraum, so kann dies auch durch Rekombination zu einer besseren Lösung führen. Kann durch den GA zumindest eine Lösung verbessert werden, so werden die Hälfte der Lösungen, die der besten Lösung, bezogen auf das Parameterset, am ähnlichsten sind, durch Zufallslösungen aus der Datenbank ersetzt. Sollte der GA keine Verbesserung erzielen, so wird das Simplex Verfahren angewendet. Führt das Simplex-Verfahren zu einer Verbesserung, so wird mit dem PSO fortgefahren. Falls keine Verbesserung möglich ist, werden alle Lösungen durch neue Lösungen aus der Datenbank ersetzt, wobei die beste bisher gefundene Lösung erhalten bleibt. Die zu implementierende Optimierungssoftware und die dazugehörige grafische Oberfläche wurden in C++ für die Linux Distribution CentOS und Windows entwickelt (siehe Kapitel 4.3). Um auf bereits berechnete Lösungen zurückgreifen zu können, wurden alle berechneten Lösungen in einer Datenbank abgespeichert. Weiters wurde auf einen möglichst hohen Parallelisierungsgrad geachtet, da eine Modellsimulation relativ aufwendig ist und mehrere GT-SUITE Lizenzen und Rechnerkerne zur Verfügung stehen. Die Optimierungsalgorithmen werden dementsprechend angepasst, sodass möglichst viele Lizenzen genutzt werden können. Die Anzahl der zur Verfügung stehenden Prozessoren ist dabei unerheblich, da wesentlich mehr Prozessorkerne als Lizenzen zur Verfügung stehen. Die grafische Oberfläche dient der Konfiguration und Steuerung der Algorithmen. Alle getätigten Einstellungen werden in eine Konfigurationsdatei geschrieben, die von der Optimierungssoftware ausgelesen wird. Es können Informationen über die bisher beste Lösung und die gesamte Laufzeit der Optimierungssoftware angezeigt werden. Weiters ist es möglich den Optimierungsprozess zu pausieren oder abzubrechen.

Um eine Lösung sinnvoll bewerten zu können, müssen für jede Optimierung Initialwerte der zu optimierenden Parameter angegeben werden. Mithilfe dieser Werte wird eine Referenzlösung berechnet.

Die Ergebnisse für das zu optimierende IFAHEV-Modell, sowie weitere Modelle, die zu Vergleichszwecken mit der bereits bestehenden Optimierungssoftware entwickelt wurden, wur-

den analysiert. Für das IFAHEV wurden mehrere Betriebsstrategien verfolgt. Das komplexeste Modell brachte durch empirische Konfiguration und partielle DOE Optimierung eine Treibstoffeinsparung von etwa 28 Prozent im Vergleich zu einem Modell eines konventionellen Kraftfahrzeuges. Durch die Anwendung des PSAGADOs wurde eine Treibstoffeinsparung von etwa 33 Prozent erreicht. Bei beiden Ergebnissen war der SOC nahezu ausgeglichen. Da die Veröffentlichung der Ergebnisse des IFAHEVs seitens des Herstellers bzw. Auftraggebers untersagt ist, und DOE in vertretbarer Zeit nicht ausführbar ist, wurde der PSAGADO auch auf ein bereits in GT-SUITE mitgeliefertes Modell angewendet. Der Vergleich zeigte, dass unser Ansatz immer bessere Ergebnisse liefert, wenn bei DOE die Zielfunktion quadratisch oder kubisch approximiert wird. Es gibt zwar die Möglichkeit eines komplexeren Ersatzmodells, allerdings konnte dieses ab einer gewissen Anzahl an zu optimierenden Parametern und damit verbundenen Anzahl an zu berechnenden Lösungen nicht mehr verwendet werden. Weiters zeigte sich, dass der PSAGADO bessere Ergebnisse erzielt als die einzelnen Heuristiken.

Zusätzlich zur Anwendung des PSAGADOs auf Computermodelle von Kraftfahrzeugen könnte generell jede lineare und nicht lineare mathematische Funktion damit optimiert werden. Beispiele hierfür wären die Optimierung von Gewinn- und Kostenfunktionen sowie die Berechnung von vereinfachten physikalischen Modellen.

KAPITEL 7

Ausblick

Bei den angewandten Algorithmen werden alle Parameter unabhängig voneinander verändert. Durch die Anforderung, dass der SOC immer ausgeglichen sein sollte und eine stärkere Abweichung so stark bestraft wird, dass die besten gefundenen Lösungen immer einen nahezu ausgeglichenen SOC haben, ergibt sich die Möglichkeit Parametereinstellungen abhängig voneinander zu betrachten. Wird z.B. die Lastpunktanhebung verstärkt, so führt das zu einem höheren SOC. Dies kann wiederum durch Erhöhung des Schwellwertes für rein elektrisches Fahren ausgeglichenen werden. Das Problem dabei ist nur, dass sich diese Abhängigkeiten mit großer Wahrscheinlichkeit nicht durch einfach beschreibbare z.B. lineare Zusammenhänge definieren lassen, sondern komplexer betrachtet werden müssen. Gründe dafür sind u.a. die unterschiedlichen Wirkungsgrade bei unterschiedlichen Parametereinstellungen. Solche Abhängigkeiten könnten u.a. mit neuronalen Netzen [Hertz 91] abgebildet werden. Diese können in den eigentlichen Optimierungsprozess eingegliedert werden. Dabei könnte das neuronale Netz theoretisch nach jeder neuen Lösungsberechnung trainiert werden. Der Aufwand dafür ist im Vergleich zur Berechnung eines Modells wesentlich geringer. Nach genügend berechneten Lösungen kann aus dem neuronalen Netz eine Ersatzfunktion abgeleitet werden. Diese kann dann entweder wie bei DOE das eigentliche Modell ersetzen oder dazu verwendet werden, um abzuschätzen ob die Berechnung eines bestimmten Parameter Sets sinnvoll ist. Ein Kriterium dafür ist z.B. ein möglichst ausgeglichener SOC.

Eine weitere Möglichkeit den Optimierungsprozess zu verbessern, wäre eine individuelle Angabe der Genauigkeit der einzelnen Parameter. Es gibt Parameter, wie etwa die Lastpunktanhebung, bei dem eine Berechnung auf mehrere Kommastellen genau Sinn macht. Bei der Schaltstrategie, in Abhängigkeit der Drehzahl, ist es wiederum fraglich, ob es notwendig ist, jeden möglichen ganzzahligen Wert innerhalb der erlaubten Grenzen zu betrachten, oder ob man die Auflösung eventuell verkleinern kann. Dies hängt weiters davon ab, welche Toleranzen am Prüfstand bei einem realen Fahrzeug vorhanden sind. Somit ist nicht garantiert, dass auch alle berechneten Werte tatsächlich überprüft werden können. Weiters könnte man auch die Diskretisierung dynamisch während des Optimierungsprozesses verändern.

Im derzeitigen PSAGADO wird zuerst das Monte-Carlo Suchverfahren hauptsächlich für

die Erzeugung der Zufallslösungen für den eigentlichen Optimierungsprozess genutzt. Dies garantiert natürlich keine gute Verteilung der Parametersets im Lösungsraum für den PSO Algorithmus. Eine völlig zufällige Generierung kann aber ebenfalls zu Problemen führen, wenn zum Beispiel der Anteil der Lösungen mit einem ausgeglichenen SOC sehr gering ist. Eine Verbesserungsmöglichkeit wäre möglichst gleichverteilte Lösungen für den PSO zu erzeugen, oder die Ausgangslösungen für das Monte-Carlo Suchverfahren nicht gänzlich zufällig zu erzeugen, sondern in jedem Iterationsschritt innerhalb der Grenzen möglichst gut zu verteilen. Weiters wäre es auch denkbar mit mehreren Lösungsmengen zu arbeiten. Diese könnten dann aus zufälligen Lösungen, Lösungen mit ausgeglichenem SOC und den besten Lösungen aus der Datenbank bestehen.

Die implementierte Optimierungssoftware kann prinzipiell für alle GT-SUITE Modelle eingesetzt werden, bei denen es darum geht ein möglichst gutes Parameterset zu finden. Die Änderungen, die speziell für die HEV-Modelloptimierung gemacht wurden, sollten keinen negativen Einfluss auf das Optimieren anderer Modelle haben. Beispiele dafür wäre etwa die Auslegung einer Turbo-Komponente, Optimierung der Einspritzung, der Ventilöffnungszeiten und des Zündzeitpunkts für die Leistungsmaximierung, Verbrauchs- und Abgasminimierung sowie jegliche Fahrzeuge für beliebige Fahrzyklen. Es gibt allerdings auch Modelle bei denen ein Parameter diskrete Werte zugewiesen bekommt, und dann ein Parameterset gefunden werden muss, das im Schnitt für alle Fälle der diskreten Werte das beste Ergebnis erzielt. Ein Beispiel wäre das durchschnittliche Drehmoment eines Motors zu maximieren und dabei fixe Drehzahlen vorzugeben. Um auch solche Modelle optimieren zu können, müsste die Schnittstelle zu GT-SUITE etwas angepasst werden. Dies ist aber mit akzeptablem Aufwand möglich.

Prinzipiell könnte die Optimierungssoftware auch an eine beliebige Modellierungssoftware angepasst werden, sofern es diese durch z.B. einen Kommandozeilen-basierten Solver oder eine direkte Schnittstelle zulässt.

Literatur

[Broyden 70] C. Broyden, R. Fletcher, D. Goldfarb, DF Shanno. Bfgs method. Journal of the Institute of Mathematics and Its Applications, 6:76–90, 1970.

[Buhmann 00] MD Buhmann. Radial basis functions. Acta Numerica, 9(1):1–38, 2000.

[Deuflhard 04] P. Deuflhard. Newton methods for nonlinear problems: affine invariance and adaptive algorithms, Band 35. Springer Verlag, 2004.

[Gao 05] W. Gao, S.K. Porandla. Design optimization of a parallel hybrid electric powertrain. Tagungsband: IEEE Conference on Vehicle Power and Propulsion, Seiten 6–12, 2005.

[Gao 07] D.W. Gao, C. Mi, A. Emadi. Modeling and simulation of electric and hybrid vehicles. Proceedings of the IEEE, 95(4):729–745, 2007.

[Glover 98] F. Glover, M. Laguna. Tabu search. Kluwer Academic Pub, 1998.

[Goldberg 89] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, 1989.

[Hertz 91] J. Hertz, A. Krogh, R.G. Palmer. Introduction to the theory of neural computation, Band 1. Westview press, 1991.

[Hofmann 09] P. Hofmann. Hybridfahrzeuge: Ein alternatives Antriebskonzept für die Zukunft. Springer, 2009.

[Hooke 61] R. Hooke, TA Jeeves. Direct Search Solution of Numerical and Statistical Problems. Journal of the ACM (JACM), 8(2):212–229, 1961.

[Huang 06] B. Huang, Z. Wang, Y. Xu. Multi-objective genetic algorithm for hybrid electric vehicle parameter optimization. Tagungsband: IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 5177–5182, 2006.

[Kennedy 95] J. Kennedy, R. Eberhart. Particle swarm optimization. Tagungsband: Proceedings of the IEEE International Conference on Neural Networks, Band 4, Seiten 1942–1948, 1995.

[Kirkpatrick 83] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. Optimization by simulated annealing. science, 220(4598):671, 1983.

[Markel 01] T. Markel, K. Wipke. Optimization techniques for hybrid electric vehicle analysis using advisor. Tagungsband: Proceedings of the ASME International Mechanical Engineering Congress and Exposition, Seiten 11–16, 2001.

[Meywerk 07] M. Meywerk. CAE-Methoden in der Fahrzeugtechnik. Springer-Verlag, Heidelberg, 2007.

[Michalewicz 96] Z. Michalewicz. Heuristic methods for evolutionary computation techniques. Journal of Heuristics, 1(2):177–206, 1996.

[Montazeri-Gh 06] M. Montazeri-Gh, A. Poursamad, B. Ghalichi. Application of genetic algorithm for optimization of control strategy in parallel hybrid electric vehicles. Journal of the Franklin Institute, 343(4-5):420–435, 2006.

[Myers 09] R.H. Myers, D.C. Montgomery, C.M. Anderson-Cook. Response surface methodology: process and product optimization using designed experiments. John Wiley & Sons Inc, 2009.

[Nelder 65] J. A. und Mead R. Nelder. A Simplex Method for Function Minimization. Oxford Journals - The Computer Journal, British Computer Society, 7(4):308–313, 1965.

[Nocedal 99] J. Nocedal, S.J. Wright. Numerical optimization. Springer verlag, 1999.

[Poli 08] R. Poli, W.B. Langdon, N.F. McPhee. A field guide to genetic programming. Lulu Enterprises Uk Ltd, 2008.

[Schicker 02] R. Schicker, G. Wegener. Drehmoment richtig messen. Hottinger-Baldwin-Messtechnik, 2002.

[van Basshuysen 11] R. van Basshuysen, F. Schäfer. Motorlexikon, Hybridfahrzeuge - Leistungseinteilung. `http://www.motorlexikon.de/?I=9487`, 26.06.2011.

[Wipke 01] K. Wipke, T. Markel, D. Nelson. Optimizing energy management strategy and degree of hybridization for a hydrogen fuel cell suv. Tagungsband: Proceedings of 18th Electric Vehicle Symposium, Berlin, 2001.

[Wirtschaftsgemeinschaft 07] Europäische Wirtschaftsgemeinschaft. Zur Angleichung der Rechtsvorschriften der Mitgliedstaaten über Maßnahmen gegen die Verunreinigung der Luft durch Emissionen von Kraftfahrzeugen. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri= CONSLEG:1970L0220:20070101:DE:PDF, 08.06.2011.

[Wu 08a] J. Wu, C.H. Zhang, N.X. Cui. Pso algorithm-based parameter optimization for hev powertrain and its control strategy. International Journal of Automotive Technology, 9(1):53–59, 2008.

[Wu 08b] X. Wu, B. Cao, J. Wen, Y. Bian. Particle swarm optimization for plug-in hybrid electric vehicle control strategy parameter. Tagungsband: IEEE Conference on Vehicle Power and Propulsion, Seiten 1–5, 2008.