A POWER MANAGEMENT STRATEGY FOR A PARALLEL THROUGH-THE-ROAD PLUG-IN HYBRID ELECTRIC VEHICLE USING GENETIC ALGORITHM

A Thesis

Submitted to the Faculty

of

Purdue University

by

Akshay Kasture

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Mechanical Engineering

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Peter H. Meckl, Chair

School of Mechanical Engineering

Dr. Gregory M. Shaver

School of Mechanical Engineering

Dr. Patricia Davies

School of Mechanical Engineering

Approved by:

Dr. Nicole Key

Head of the School Graduate Program

Dedicated to my family.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my advisor, Dr. Peter H. Meckl, for his guidance, support and encouragement throughout the duration of this research project. His deep knowledge, insight and patience was instrumental in the realization of this project. I would also like to thank the committee members Dr. Gregory Shaver and Dr. Patricia Davies for their perceptive comments and feedback.

I also wish to express my gratitude to Sauradeep Samanta who familiarized me with this project in the beginning. I would also like to thank Kaushal Jain and Yun Jui Tu for working together with me on the dynamometer. I am also grateful to Zihao Xu and Da Huo for the weekly meeting sessions which helped me gain new perspective on the project I was working on. I would like to thank Dr. William Crossley from the School of Aeronautics and Astronautics at Purdue University for his lectures on genetic algorithms.

The friends I made at West Lafayette made my graduate school life at Purdue University enjoyable and I would like to thank them.

Last, but not least, I am grateful to my family for their unwavering support and love.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

SYMBOLS

A	Road Loss Coefficient
B	Road Loss Coefficient
C	Road Loss Coefficient
c	penalty coefficient, genetic algorithm
F_{tract}	Total tractive force on vehicle model
F_{load}	Road load force on vehicle model
$F_{inertia}$	Inertia force on vehicle model
g_j	Constraint violation, genetic algorithm
I_{batt}	Battery current
I_m	Motor current
J_π	Cost function, dynamic programming
J_t	Cost function, ECMS
J	Cost function, genetic algorithm
L_k	Instantaneous cost function, dynamic programming
M_{veh}	Mass of the vehicle
N_g	Gear number
P_j	Final state constraint, genetic algorithm
P_e	Engine power
r_p	Penalty multiplier, genetic algorithm
r_w	Radius of the vehicle wheels
τ_e	Engine torque
τ_{ev}	Engine torque at wheels
τ_m	Motor torque
τ_{mv}	Motor torque at wheels
τ_v	Torque required by the vehicle

v	Speed of the vehicle
ω_e	Angular velocity of engine
ω_{ed}	Angular velocity of engine differential
ω_m	Angular velocity of motor
ω_{ed}	Angular velocity of motor differential
ω_w	Angular velocity of wheel
X_{ed}	Engine differential gear ratio
X_g	Gear ratio
ζ	Equivalence factor, ECMS

ABBREVIATIONS

ASM	Automotive Simulation Models
Btu	British Thermal Unit
CAN	Controller Area Network
CBD	Central Business District
DEF	Diesel Particulate Filter
DP	Dynamic Programming
DOC	Diesel Oxidation Catalyst
DPF	Diesel Particulate Filter
ECMS	Equivalent Consumption Minimization Strategy
EGR	Exhaust Gas Recirculation
EPA	Environmental Protection Agency
ESS	Energy Storage System
GA	Genetic Algorithm
GHG	Greenhouse Gases
HEV	Hybrid Electric Vehicle
HIL	Hardware-in-Loop
HWFET	Highway Fuel Economy Test
IBM	International Business Machines
LSTM	Long Short Term Memory
MB	Mega Bytes
PCA	Principal Component Analysis
PHEV	Plug in Hybrid Electric Vehicle
pSOC	Proportional State of Charge
RNN	Recurrent Neural Network
RPM	Revolutions Per Minute

SCR	Selective Catalytic Reduction
SOC	State Of Charge
UDDS	Urban Dynamometer Driving Schedule
WTW	Well to Wheel

ABSTRACT

Kasture, Akshay M.S, Purdue University, May 2020. **A Power Management Strategy for a parallel through-the-road plug-in hybrid electric vehicle using Genetic Algorithm**. Major Professor: Peter Meckl, School of Mechanical Engineering.

With the upsurge of greenhouse gas emissions and rapid depletion of fossil fuels, the pressure on the transportation industry to develop new vehicles with improved fuel economy without sacrificing performance is on the rise. Hybrid Electric Vehicles (HEVs), which employ an internal combustion engine as well as an electric motor as power sources, are becoming increasingly popular alternatives to traditional engine-only vehicles. However, the presence of multiple power sources makes HEVs more complex. A significant task in developing an HEV is designing a power management strategy, defined as a control system tasked with the responsibility of efficiently splitting the power/torque demand from the separate energy sources. Five different types of power management strategies, which were developed previously, are reviewed in this work, including dynamic programming, equivalent consumption minimization strategy, proportional state-of-charge algorithm, regression modeling and long short term memory modeling. The effects of these power management strategies on the vehicle performance are studied using a simplified model of the vehicle. This work also proposes an original power management strategy development using a genetic algorithm. This power management strategy is compared to dynamic programming and several similarities and differences are observed in the results of dynamic programming and genetic algorithm. For a particular drive cycle, the implementation of the genetic algorithm strategy on the vehicle model leads to a vehicle speed profile that almost matches the original speed profile of that drive cycle.

1. INTRODUCTION

1.1 Introduction

The transportation industry, which involves movement of people, goods and services from one location to another, is one of the most critical industry sectors and is intimately linked with other sectors. Due to technological advancements, all physical media on earth like water, air and land are utilized in the transportation sector. Consequently, the energy consumption by the transportation sector is among the largest. In fact, the total energy consumption by the transportation industry in the United States was roughly 28,253 trillion Btu in 2019 [1]. Figure 1.1 shows the distribution of the total energy consumption among major industry sectors in the United States in 2019.

Oil and natural gas are the major types of fossil fuels that power the transportation industry. There are several advantages of using fossil fuels. Firstly, the energy contained per unit quantity of fuel is high as compared to the total quantity of fossil fuels available in the world. Secondly, fossil fuels are cheap due to their abundance and they are also easy to transport. However, the disadvantages of fossil fuels are slowly beginning to outweigh their advantages.

Although fossil fuels are abundantly available in the world right now, they are non-renewable sources of energy; they are going to completely deplete one day. The rate of fossil fuel consumption is increasing per year because of the higher energy demand due to advancements in science and technology. Figure 1.2 shows the energy consumption in the U.S. for different energy sources [1]. It is predicted that oil will run out by 2052 and natural gas by 2060 [2]. Secondly, burning fossil fuels leads to emissions of harmful gases which damage the environment. Some of these gases like carbon monoxide and sulfur dioxide are toxic to humans and animals. Others

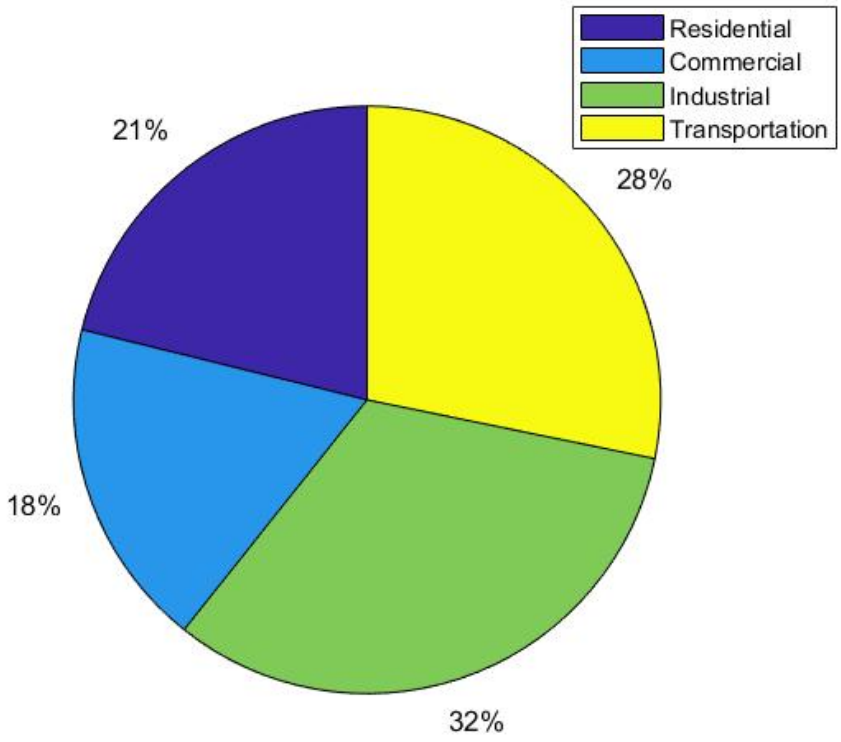

Figure 1.1. Sectorwise energy consumption in the U.S. in 2019 [1].

like greenhouse gases are insidious in nature. While they allow visible light to pass through them, they block infrared radiation from leaving the earth's atmosphere, thus warming the planet. Figure 1.3 shows the increase of earth's average temperature since 1980.

Carbon dioxide, methane, nitrous oxide and fluorinated gases constitute the majority of greenhouse gases. The total greenhouse gas emissions in 2017 were 6457 million metric tons of CO_2 equivalent [4]. Figure 1.4 shows how the total greenhouse gas emissions were distributed among their components.

While greenhouse gases can be generated by a number of sources, burning fossil fuels for transportation, electricity and heat is the largest source [5]. Figure 1.5 shows how total greenhouse gas emission was distributed among its sources.

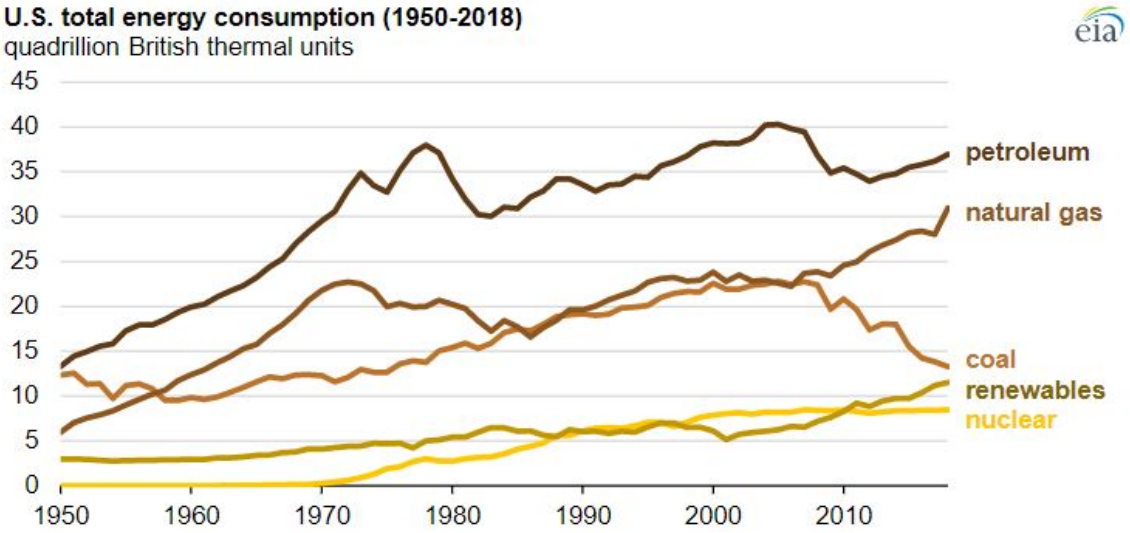

Figure 1.2. Energy consumption in the U.S. for different energy sources [1].

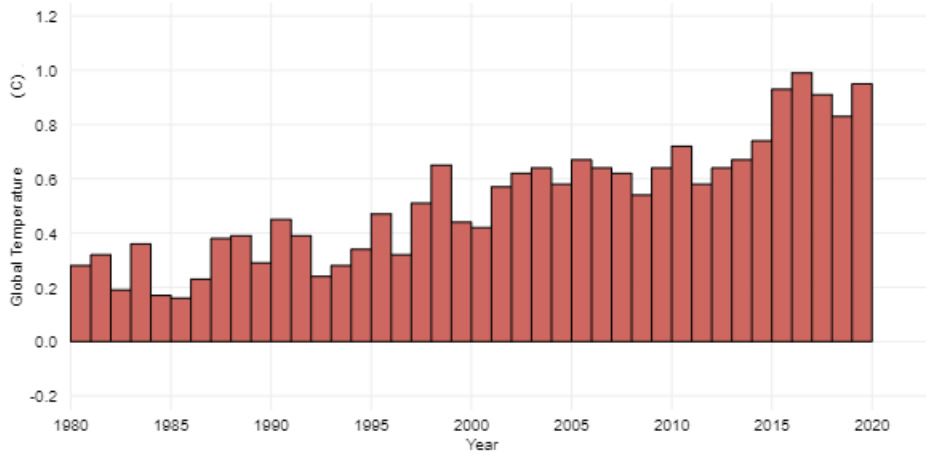

Figure 1.3. Global surface temperature since 1980 to 2020 [3].

It is evident from Figure 1.5 that the transportation industry generates the largest amount of greenhouse gases. Greenhouse gas emissions from transportation primarily come from burning fossil fuel for cars, trucks, ships, trains, and planes. Over 90 percent of the fuel used for transportation is petroleum based, which includes primarily gasoline and diesel [5].

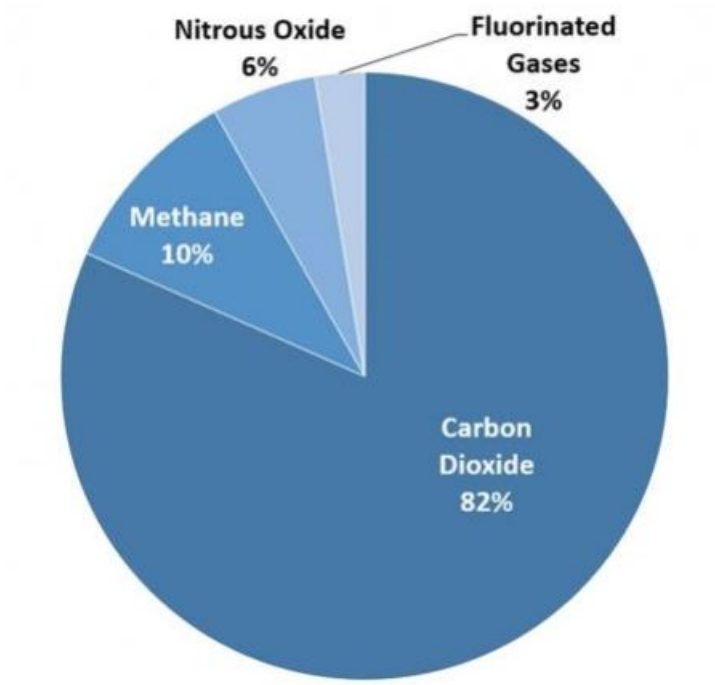

Figure 1.4. U.S. greenhouse gas emissions in 2017 [5].

Fortunately, these detrimental effects of greenhouse gas emissions have been realized and research is being conducted to reduce the dependency and usage of fossil fuels in the transportation industry. A significant component of research is devoted to developing alternative fuels like biodiesel, bioalcohol, hydrogen and others which are renewable. Usage of electricity to power vehicles has also gained traction over the past two decades. Electric vehicles have some major advantages over their fuel-based counterparts. Predominantly, they are powered by electricity which has less carbon production associated with it, higher efficiency, and lower greenhouse gas emissions and toxic gas emissions per kW-h of energy.

Electric vehicles suffer from some well documented disadvantages, though. Currently, charging stations are not as ubiquitous as gas stations. If an electric vehicle runs out of charge, there are high chances that the driver will be stuck for a while. The second major disadvantage of electric vehicles is their short driving range.

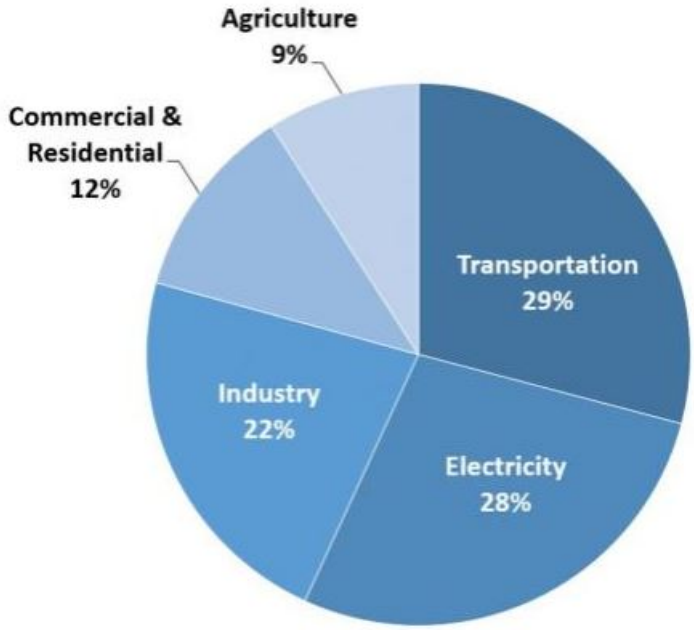

Figure 1.5. U.S. greenhouse gas emissions in 2017 [4].

Hybrid vehicles, which run on both fuel and electricity, combine the best of both worlds. They have relatively lower carbon footprint and emissions as compared to fuel-only vehicles due to usage of electricity and in case of emergency situations where the battery charge is fully depleted, the vehicle can run on fuel.

The hybrid vehicle used for this research project is a 2013 Chevrolet Malibu retrofitted with an Opel Astra turbo diesel engine of 1.7L capacity which can run on both bio-diesel and regular diesel fuel. For the electric drivetrain, a 100kW Magna motor is also installed which is powered by a 16.2 kW-h Li-ion battery pack. This vehicle was developed for the EcoCAR2 competition in 2014.

The goal of this research was to develop a novel power management strategy to control the torque split between the energy sources for reducing greenhouse gas emissions and comparing it with previously developed strategies. The following power management strategies were studied and compared:

1. Dynamic Programming

2. Equivalent Consumption Minimization Strategy
3. Proportional State-of-Charge Algorithm
4. Regression Modeling
5. Long Short Term Memory Modeling
6. Genetic Algorithm

1.2 Hybrid Vehicle Architectures

A hybrid vehicle is powered by both an IC engine and an electric motor. The vehicle can thus operate in different modes based on whether the engine and/or the motor are propelling the vehicle and their relative proportions of supplied powers.

1. *Engine only*: The vehicle is powered by the engine only and the electric drivetrain is not involved in any way.
2. *Motor only*: The vehicle is powered by the electric motor only and the battery charge is depleted.
3. *Combined power*: The engine is run at optimal conditions and any extra power demand is fulfilled by the motor.
4. *Regenerative braking*: Negative torque is applied to the wheels by the electric motor which acts as a generator in this mode, thus converting the kinetic energy of the vehicle into electric energy which is stored in the battery.

The following subsections describe the popular hybrid vehicle architectures available in industry.

1.2.1 Series Architecture

Figure 1.6 shows the series architecture configuration.

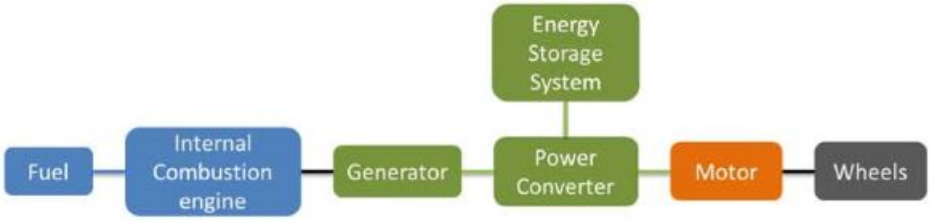

Figure 1.6. Series architecture for a hybrid electric vehicle [6].

In series architecture, the fuel is the main source of energy. The fuel undergoes combustion in the internal combustion engine, thus generating mechanical energy. The mechanical energy is converted into electrical energy by the generator which is then stored in an energy storage system such as a battery pack. The motor, which is powered either directly by the generator or by the energy storage system, subsequently drives the wheels of the hybrid vehicle. The advantage of series architecture is the ability of the vehicle to operate at optimum points as fuel is the main source of energy and this simplifies the process of designing a control system for the vehicle. On the flip side, though, the energy conversion efficiency of this configuration is quite low as the energy goes through two conversions in series: mechanical to electrical and electrical to mechanical. Also, the engine, generator, energy storage system and motor have to be powerful as the motor is the only component that drives the wheels.

1.2.2 Parallel Architecture

Figure 1.7 shows the parallel architecture configuration.

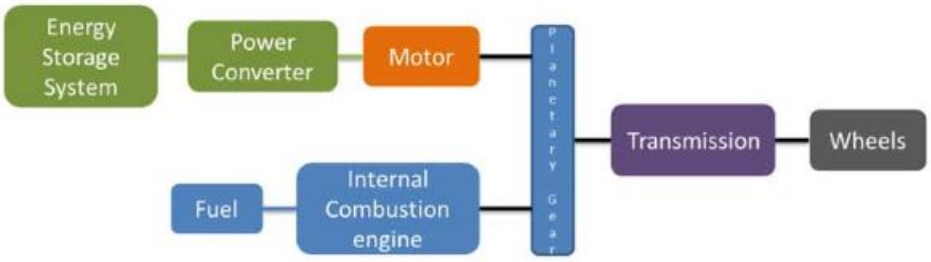

Figure 1.7. Parallel architecture for a hybrid electric vehicle [6].

As the name suggests, the engine and the motor systems are connected in parallel to the transmission that drives the wheels. These systems can be coupled with the transmission using various types of mechanisms like planetary gears, belts and pulleys, clutches, etc. The motor in this configuration can act as a generator for charging the energy storage system during conditions of low power demand. Braking is an instance where the power demand of the vehicle is low. The advantage of parallel architecture over series is that the motor sizing is smaller as compared to the parallel architecture as both motor and engine together can power the vehicle. On the other hand, due to the added complexity in this configuration, the process of designing the control system becomes more complicated.

1.2.3 Parallel Through-The-Road Architecture

Figure 1.8 shows the parallel through-the-road architecture.

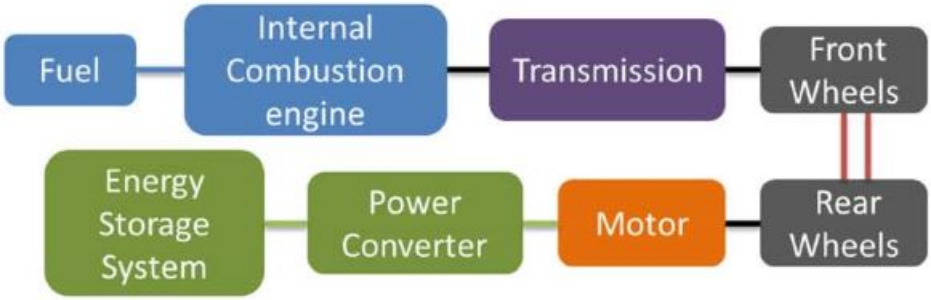

Figure 1.8. Parallel through-the-road architecture for a hybrid electric vehicle [6].

In the parallel through-the-road architecture, the engine is connected to the front axle and the motor is connected to the rear axle. The configuration can also be the other way round, wherein the engine is connected to the rear wheels and the motor is connected to the front wheels. With respect to the road, however, the energy sources are connected in parallel. In terms of retrofitting a new energy source to an existing fuel-only vehicle, this configuration is the easiest to implement. The test vehicle described in the first section has this configuration.

1.2.4 Other Architectures and Approaches to Hybridization

Apart from the three hybrid vehicle configurations described previously, there are other complex configurations too. For instance, there is a series-parallel configuration in which the hybrid vehicle can operate in both series mode or parallel mode. The advantage of this configuration is the ability to optimize the fuel economy and drivability with respect to the state of the vehicle and requirements of the driver. Also, some hybrid vehicles are powered by sources other than a fuel-based internal combustion engine. Some examples of these energy sources are flywheel, compressed air, hydraulic systems, etc. Due to their ability to provide large amounts of torque, hydraulic systems are used in heavy-duty vehicles like trucks.

1.3 Power Management Strategies

The ratio of the powers supplied by the engine and the motor in an HEV is called power split. During an HEV's operation, there are an infinite number of values the power split can assume. Hence, developing a methodology to find the power split values that will make the vehicle operate better with respect to fuel efficiency or exhaust gas emissions is a significant factor in the design process of an HEV. There are several methodologies to solve this problem, and they are collectively called power management strategies. Developing these strategies falls within the realm of control theory.

These power management strategies depend on, among several factors, the architecture of the vehicle and the mode in which the vehicle is desired to be operated. For instance, as the battery can be charged from an external source in a plug-in HEV, an additional dimension exists compared to a normal HEV. Hence, a plug-in HEV can safely operate in charge-depleting mode, defined as the mode in which the primary source of power is the battery and where the energy in the battery keeps on depleting throughout the vehicle trip. In comparison, the battery in a normal HEV can be charged only through regenerative braking or with the help of the engine. Hence, it is important to maintain the energy in the battery of a normal HEV. In other words, a normal HEV is recommended to be operated in a charge-sustaining mode, defined as the mode in which the energy in the battery at the start of the vehicle trip is the same as that at the end of the trip.

1.4 Objective of Current Research

There are two different phases into which the objectives of this project are divided.

Phase 1 focuses on model development and was completed by Gupta [6]. Firstly, the available PHEV was tested on a dynamometer and data was collected from the dynamometer for developing a mathematical model of the PHEV. Secondly, based on the collected data, various components of the PHEV such as engine, motor, battery

and powertrain, were modeled separately as individual components and they were used collectively to form an integrated model of the vehicle.

Phase 2 focuses on power management strategies and is the primary objective of this thesis. This phase involves developing a control algorithm for managing the power demand of the PHEV using a genetic algorithm and testing it on the simplified vehicle model developed in phase 1.

1.5 Distribution of Thesis Content

This thesis is divided into five chapters.

Chapter 1 provides an introduction to HEVs. It explains the current need of HEVs followed by a brief overview of the different configurations into which an HEV can be built. Then, the idea of power management strategies is introduced along with the objectives of this research project.

Chapter 2 reviews previous approaches to solving the power management problem of an HEV. After providing a structured classification of previously developed power management strategies, this chapter explains optimal control, a branch of control theory which is the cornerstone of the best known power management strategies. Additionally, it introduces dynamic programming, whose results are currently considered as the benchmark for all other power management strategies, and genetic algorithm, which is used to develop a new power management strategy.

Chapter 3 describes the vehicle which was used for model development, the type of controller in the vehicle and the testing equipment which was used to collect the data for model development. This chapter also explains how to use the dynamometer to collect data and the concept of a drive cycle is introduced.

Chapter 4 develops the simplified model of the vehicle based on data collected from the dynamometer. This chapter also explains various power management strategies developed previously, such as dynamic programming, equivalent consumption min-

imization strategy, proportional state-of-charge algorithm, regression modeling and long short term memory.

In chapter 5, a new power management strategy is introduced which is based on genetic algorithms. Genetic algorithms are a class of soft computing techniques inspired by Darwin's theory of natural evolution and are a mathematical analog of natural selection and survival of the fittest. They have been used to solve problems related to mathematical and combinatorial optimization, and machine learning before. In this thesis, a genetic algorithm has been used to develop a power management strategy of a hybrid electric vehicle. It can be inferred that the tools used to develop this strategy can also be extended to solve any discrete time optimal control problem using genetic algorithms. The major challenge involved in developing a genetic algorithm solution method for an optimal control problem is representing the solution using a data structure that is compatible with the algorithm, and developing and tuning critical functions based on this data structure that essentially drive the algorithm. These ideas have been explored in chapter 5.

Chapter 6 concludes the thesis, delineates the original contributions and recommends directions for future research.

2. LITERATURE REVIEW

This chapter reviews important literature related to the work in this thesis. The first section classifies various power management strategies that are currently available. The second section discusses optimal control. The third section is dedicated to dynamic programming and the last section describes genetic algorithm, which is a stochastic global search and metaheuristic optimization technique.

2.1 Classification of Power Management Control Strategies

Developing a control algorithm for the power management of an HEV is an optimal control problem, which will be discussed in the next section. Based on the theory of optimal control in general and particular research related to the power management problem of an HEV, the following classification of the various power management control strategies is proposed [7].

Numerical Global Optimization: These methods assume knowledge of the entire drive cycle of the vehicle is known *a priori*. Thus, the controller developed using this technique is non-causal because the torque split output at a particular time depends on the current as well as the future values of the variables associated with the drive cycle. As the future profile of the drive cycle cannot be accurately predicted, this controller is not implementable in a real HEV. Additionally, this controller is computationally very intensive. However, the controller output is optimal in nature as it optimizes the objective (fuel consumption or exhaust emissions) over the entire drive cycle. Thus, the behavior of this type of controller sets a benchmark against which the real-time implementable controllers can be developed. Deterministic Dynamic Programming (DP) [8–11] and Genetic Algorithm (GA) [12] fall under this category.

Numerical Local Optimization: These techniques also attempt to optimize an objective for solving the power management problem. However, unlike the global optimization methods, these techniques do not assume prior knowledge of the entire drive cycle but they predict the drive cycle over a shorter horizon and perform optimization over that horizon. Stochastic Dynamic Programming [13] [14] [15] and Model Predictive Control [16] [17] fall under this category. In Stochastic Dynamic Programming, the drive cycle is modeled as a Markov Decision Process and thus a transition matrix is associated with it which models the probability distribution of the future drive cycle profile based on the current and past values. This matrix is developed by performing maximum likelihood estimation over several available standard drive cycles. Thus, the result of this controller is not optimal for every drive cycle but is optimal in an average sense of the several drive cycles involved in developing the probability model of the drive cycle. Like the previous one, this controller is also computationally intensive.

Analytical Optimization: Solving an optimal control problem depends primarily on the nature of the objective function to be optimized, which in turn depends on the complexity of the model. Simpler models can be solved using analytical techniques. However, the model of a typical HEV is too complex to be solved analytically so numerical techniques are used most of the time. However, if the HEV model is simplified enough, then analytical techniques like the Pontryagrin's Minimum Principle (PMP) [18] [19] [20] [21] [22] can be employed to develop a torque split control law. These analytical techniques are computationally very efficient compared to the numerical optimization techniques. The disadvantages of this controller, however, are the oversimplification of the model and its non-causal nature (these methods also assume prior knowledge of the entire drive cycle).

Instantaneous Optimization: Equivalent Consumption Minimization Strategy or ECMS [23] [24] [25] [26] is an example of an instantaneous optimization technique. The global optimal control problem in this case is converted into a series of local (or instantaneous) optimization problems. Defining the appropriate local minimization

objective in order to be as close to the global objective as possible is the critical step in this technique. The original ECMS cannot be implemented in a real HEV as this technique requires the drive cycle to be known in advance. However, adaptive ECMS has been developed [27] [28] which is based on driving pattern recognition which recognizes the type of driving conditions and accordingly changes the definition of the instantaneous objective function to be minimized.

Heuristic Control: Heuristic techniques do not use optimization techniques to find the control law but use a set of rules and algorithms based on calculated intuition and substantial testing. A popular heuristic control development technique is using the results of dynamic programming, which are considered to be ideal, to extract rules which can then be used to predict the torque split in real time as a function of the external inputs and the vehicle states. The advantage of some of these techniques is that they can be implemented in real time. The disadvantage is that the results of these rule-based techniques are sub optimal in nature as they aren't based on a formal optimization procedure. Regression analysis performed by Gupta [6] and LSTM model developed by Sun [29] are examples of heuristic strategies.

In this thesis, DP (Numerical Global Optimization), ECMS (Instantaneous Optimization), Proportional SOC (Heuristic Control), regression (Heuristic Control), Recurrent Neural Network (Heuristic) and GA (Numerical Global Optimization) techniques are discussed and compared. The rest of this section is dedicated to introducing the key mathematical concepts behind this thesis.

2.2 Dynamic Programming

Dynamic Programming is a numerical method for solving those optimization problems that can be broken down into a series of simpler optimization problems. The main challenge in solving an optimization problem using dynamic programming is recasting it into a dynamic programming problem; many times the recasting is not apparent and significant creativity and engineering insight is required. Dynamic Pro-

gramming is a popular method for solving an optimal control problem because model complexity doesn't hinder its applicability (as long as the computational limitations are kept in mind). Due to its non-causal nature, however, it cannot be used to develop a real-time optimal control algorithm.

Dynamic Programming was introduced by Richard Bellman [30] and is based on Bellman's Principle of Optimality, which can be worded as follows: "An optimal control policy has the property that no matter what the previous decisions (i.e., controls) have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions." [31]

This means that the optimal path from any of its intermediate steps to the end corresponds to the terminal part of the entire optimal solution. In principle, dynamic programming can be used to solve continuous-time optimal control problems. However, as computations involved in a digital computer are discrete in nature, the continuous-time model must be discretized in time, states, and inputs, and the dynamic programming technique must be adapted to this discretized space. This can be done mathematically as follows:

Consider the following discrete-time system:

$$x_{k+1} = f_k(x_k, u_k), \tag{2.1}$$

which states that the future state of a system x_{k+1} depends on the current state x_k, current control input u_k and current time step k. Here, $x_k \in X_k$, $u_k \in U_k$, and $k = 0, 1, 2, ..., N-1$. The control policy can be written as

$$\pi = \{u_0, u_1, ..., u_{N-1}\}. \tag{2.2}$$

Then, the cost or objective of π starting at x_0 is given by

$$J_\pi(x_0) = L_N(x_N) + \sum_{k=0}^{N-1} L_k(x_k, u_k(x_k)) \tag{2.3}$$

where L_k is the instantaneous cost function and L_N is the final cost. The optimal cost function minimizes the total cost,

$$J^*(x_0) = \min_\pi J_\pi(x_0). \tag{2.4}$$

Let the optimal policy be

$$\pi^* = \{u_0^*, u_1^*, ..., u_{N-1}^*\}. \tag{2.5}$$

Thus, the optimal set of control inputs satisfies

$$J_{\pi^*}(x_0) = J^*(x_0). \tag{2.6}$$

It's prudent to now consider the subproblem of minimizing the cost-to-go from time i and state x_i to time N

$$V_i = L_N(x_N) + \sum_{k=i}^{N-1} L_k(x_k, u_k(x_k)), \tag{2.7}$$

where the tail policy $\{u_i^*, u_{i+1}^*, ..., u_{N-1}^*\}$ is the last part of the optimal policy π^*. It can be proved using the theory of mathematical induction that according to Bellman's optimality principle, the tail policy is optimal for this subproblem. Combining with Bellman's optimality principle, the dynamic programming algorithm starts from the final step N and moves backwards using the series of policies

$$u_k^* = \arg\min_{u_k}(L_k(x_k, u_k) + J_{k+1}(x_{k+1})) \tag{2.8}$$

where $k = N-1, N-2, ..., 1, 0$. $J_0(x_0)$ is generated at the last step and is equal to the optimal cost $J^*(x_0)$. Thus, it is possible to determine the optimal control policy moving backwards from the final state while choosing the path that minimizes the cost-to-go at each step. This is how dynamic programming breaks down the global optimization problem into a series of simpler optimization problems and combines the series of results into the solution of the global problem. In chapter 4, we will show how this theory can be used to solve the power management problem of an HEV.

2.3 Genetic Algorithm

Genetic algorithms were introduced by John Holland in 1975. A genetic algorithm is a probabilistic global search technique that is inspired from Charles Darwin's theory of evolution and is based on the concept of survival of the fittest. A genetic

algorithm, however, wasn't originally meant to be an optimization technique and certainly not a solution method of an optimal control problem. It was David Goldberg who first used a genetic algorithm to perform global optimization [32]. As a genetic algorithm does not involve gradient computations, it can be used as an optimization technique when the objective function is discontinuous, non-linear, stochastic and/or non-differentiable. Genetic algorithms can also be used to solve problems of mixed integer programming, where some variables can take only integer values.

It is important to note that a genetic algorithm differs from a classical gradient-based optimization technique in two ways. A classical algorithm generates a single point at each iteration and the sequence of points approaches an optimal solution. On the other hand, a genetic algorithm generates a population of points at each generation and the best point in the population approaches an optimal solution. Secondly, the point in the next iteration in a classical algorithm is generated by a deterministic approach whereas in a genetic algorithm, the population in the next generation is developed using a stochastic approach.

It is useful to set a standard terminology related to a genetic algorithm in order to facilitate smooth discussion. Some of the key terms are defined as follows:

Fitness Functions: This is the function to be optimized. For unconstrained optimization problems, this is just the objective function. Solving constrained optimization problems is trickier using genetic algorithms as the algorithm doesn't have the facility to include any constraints. As a result, the constraints have to be included in the objective function itself in the form of penalty functions. A typical penalty function consists of an error function, which is multiplied by a factor which depends on the severity of the penalty violation. For instance, if we're attempting to solve the following constrained minimization problem:

$$\min f(\mathbf{x}) \tag{2.9}$$

subject to the following constraint

$$c_i(\mathbf{x}) \leq 0, \forall i \in I. \tag{2.10}$$

The constraint can be included in the objective function in the form of a penalty function to generate an unconstrained optimization problem as follows:

$$\min \Phi_k(\mathbf{x}) = f(\mathbf{x}) + \sigma_k \sum_{i \in I} g(c_i(\mathbf{x})) \tag{2.11}$$

where

$$g(c_i(\mathbf{x})) = \max(0, c_i(\mathbf{x}))^2. \tag{2.12}$$

In the above set of equations, $g(c_i(\mathbf{x}))$ is called the exterior penalty function and σ_k are the penalty coefficients. The penalty functions do not need to be L2 norm. L1 norm can also be used and in some cases, using the L1 norm is encouraged because of its robustness to outliers and the ease of computation. The penalty coefficients do not need to be constants. They can change during each generation by some predefined law as well. As there is no physical interpretation of the penalty functions and the coefficients, appropriately choosing them is a major unsolved challenge in the theory of mathematical optimization. Currently, these factors are selected on the basis of engineering intuition along with some trial-and-error experiments.

Individual: An individual is any point in the input space where the fitness function can be applied. The fitness function's value for an individual is called its score. An individual is sometimes called a genome with reference to the biological aspect of genetic algorithms and the vector entries which define the individual fully are called its genes. The way the genes identify the particular individual is called its genetic representation. Some examples of genetic representation are the coordinates of the point in Euclidean space or its binary representation. Representing the individual in binary form is a popular method because the crossover and mutation functions can be easily applied to this representation. The cornerstone of this thesis work is the unique way the control input was represented in the form of genes and individuals.

Population: A population is a set of individuals, represented in the form of an array. Selecting the number of individuals in a population is a critical factor that decides the performance of a genetic algorithm. As a rule of thumb, the size of a population should be at least four times the number of genes used to define an individual in the population.

Diversity: Diversity of a population is a measure of how the individuals in a population are spread in the input space. A population has high diversity if its variance is high. It is important for the population to be diverse during the initial generations in order to enable a genetic algorithm to cover the entire input space for performing optimization. In case of low diversity, the algorithm has high chances of getting stuck in a local minimum, especially when the fitness function contains several troughs or is difficult to visualize.

Fitness values: The fitness value of an individual is the value of the fitness function for that individual. If the objective function is to be minimized using a genetic algorithm, the lowest fitness value in a population is considered to be the best. Although both minimization and maximization can be performed using a genetic algorithm, minimization is generally recommended because of the nature of the penalty functions.

Parents and children: To create the population in the next generation, a genetic algorithm selects certain individuals in the current population, called parents, and uses them to create individuals in the next population, called children. The selection is performed using a selection function, which selects the individuals with the best fitness values.

With the basic terminology fully established, we can now talk about how a genetic algorithm works. The following set of instructions encapsulates a genetic algorithm [33]:

1. Initialize a population randomly.

2. Compute the fitness value of each individual in the population. These fitness values are called raw fitness scores.

3. Create expectation values by scaling the raw fitness scores.

4. Use a selection function to select parents from the current population based on their expectation values.

5. Produce children from parents by either making random changes to a single parent (also called mutation) or by combining the genes of a pair of parents (also called crossover).

6. Replace the current population with the children to form the next population.

7. Repeat steps 2 to 6 until a stopping criterion is met.

The selection function that selects the strongest candidates and makes the other candidates extinct is analogous to survival of the fittest and the crossover and mutation functions that generate children are analogous to the actual chromosomal processes that occur in genes. Thus, with each generation, the population is said to evolve towards an optimal solution.

The following is a discussion of the various types of functions used in a genetic algorithm [34].

Selection function: A selection function decides how a genetic algorithm chooses parents for producing children for the next generation. The following are some standard types of selection functions that are commonly used:

1. In stochastic uniform selection, a line consisting of sections that correspond to each parent is developed in which the length of every section is proportional to the expectation value of the parent to which that section is assigned. The algorithm is traversed with steps of equal size along this line and a parent is allocated at each step from the section it lands on.

2. Remainder selection assigns parents in a deterministic way from the integer part of each individual's scaled value and then uses roulette selection on the remaining fractional part. For example, if the scaled value of an individual is 2.3, that individual is listed twice as a parent because the integer part is 2. After parents have been assigned according to the integer parts of the scaled values, the rest of the parents are chosen in a stochastic fashion. The probability that a parent is chosen in this step is proportional to the fractional part of its scaled value.

3. Uniform selection chooses parents using the expectations and number of parents. Uniform selection is useful for debugging and testing, but is not a very effective search strategy.

4. Roulette selection chooses parents by simulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation. The algorithm uses a random number to select one of the sections with a probability proportional to its area.

5. Tournament selection is one of the most popular selection functions. In tournament selection, K/L competitions take place in one tournament where L is the number of chromosomes or participants in one competition. In each competition, L chromosomes are chosen at random without replacement from the population, their fitness values are evaluated and the chromosome with the lowest fitness value is declared as the winner and chosen as a parent. One chromosome can participate in only one competition in a tournament. Thus, K/L parents are generated after one tournament. L number of tournaments are conducted independently on this population to generate K parents. Some parents exist more than once in a parent pool, which indicates that those parents have lower fitness values than others and will contribute more genes for reproducing children.

Crossover function: A crossover function combines two parents to form a child of the next generation. In a genetic algorithm, a majority of the children are generated using crossover functions. There are several standard crossover functions, which are described below:

1. The uniform crossover function creates a random binary vector and selects the genes where the vector is one from the first parent and the genes where the vector is zero from the second parent, and combines the genes to form a child. This will be explained more in chapter 5.

2. In single point crossover, two parents are split into two sections each such that for both parents, the proportions of the lengths of the two split sections are random but equal to each other. These split sections of both parents are then swapped to form two children.

3. The two point crossover is a more general case of the single point crossover. In two point crossover, two parents are split into three sections each such that for both parents, the proportions of the lengths of the three split sections are random but equal to each other. Two children are then formed by joining two sections of one parent with one section of the other parent without changing the order of the sections.

4. Intermediate crossover function creates children by taking a weighted average of the parents.

Mutation function: A mutation function specifies how the genetic algorithm makes small changes randomly in the individuals in the population to create mutation children. Mutation leads to genetic diversity and helps the genetic algorithm to cover the entire input space. The following are some types of mutation functions:

1. The Gaussian mutation function adds a random number drawn from a Gaussian distribution with mean 0 to each entry of the parent vector.

2. In uniform mutation, the algorithm first selects a fraction of the vector entries of an individual for mutation, where each entry has a specific probability of being mutated. The algorithm then replaces each selected entry by a random number selected uniformly from the range for that entry.

3. Adaptive feasible mutation function randomly generates directions that are adaptive with respect to the last successful or unsuccessful generation. The mutation chooses a direction and step length that satisfy bounds and linear constraints.

Stopping criterion: As genetic algorithm was originally intended to be a global search method and not an optimization technique, genetic algorithm has no way of knowing whether it has reached the optimal solution. Thus, even though a genetic algorithm might have found the optimal solution, it will continue to run forever if it's left unchecked. Hence, a manual stopping criterion must be set to terminate the algorithm. The criterion must be such that computational power isn't wasted after it has found the optimal solution but computations should not be terminated prematurely, which otherwise would result in solutions that are far from optimal. The following are some common types of stopping criteria for a genetic algorithm:

1. The algorithm stops after a fixed number of iterations or generations.

2. The algorithm stops after running for some fixed amount of time.

3. The algorithm stops when the value of the fitness function for the best point in the current population is less than or equal to a certain value, called the fitness limit.

4. The algorithm stops when the average relative change in the fitness function value over a certain fixed number of generations, called stall generations, is less than a certain value called the function tolerance.

3. SYSTEM ARCHITECTURE

This chapter describes the characteristics of the actual vehicle whose development started the project of the hybrid electric vehicle at Purdue University's School of Mechanical Engineering.

3.1 Vehicle Platform

A 2013 Chevrolet Malibu provided by Purdue University's EcoCAR2 team is used as a vehicle platform. This section describes the specifications of the car. While 2016 was the year the hybrid version of the Chevrolet Malibu was first introduced, Purdue's EcoCAR2 team had manually hybridized the 2013 version for the EcoCAR2 competition.

The original Chevrolet Malibu had a 2.4L gasoline engine with a start-stop capability. This gasoline engine was swapped by Purdue's EcoCAR2 team with a 1.7L diesel engine, which uses 20 percent biodiesel and 80 percent diesel. A 100kW Magna motor is coupled to the rear wheels for the electric drivetrain, which is powered by a 288V DC Energy Storage System. This storage system consists of a DC-DC converter to supply auxiliary devices during the electric-only mode of operation. The plug-in configuration is provided to the vehicle by a Brusa Charger, which charges the Energy Storage System. Figure 3.1 is a schematic diagram of the vehicle platform [6].

As shown in the figure, the vehicle is powered by two drivetrains. The details of the front drivetrain are as follows:

1. IC Engine: 4-cylinder in-line diesel engine of 1.7L displacement with EGR and turbocharger. Rating of 96kW at 2500 RPM. This engine was originally used in an Opel Astra.

2. Fuel system: Common rail fuel system from Denso [35].

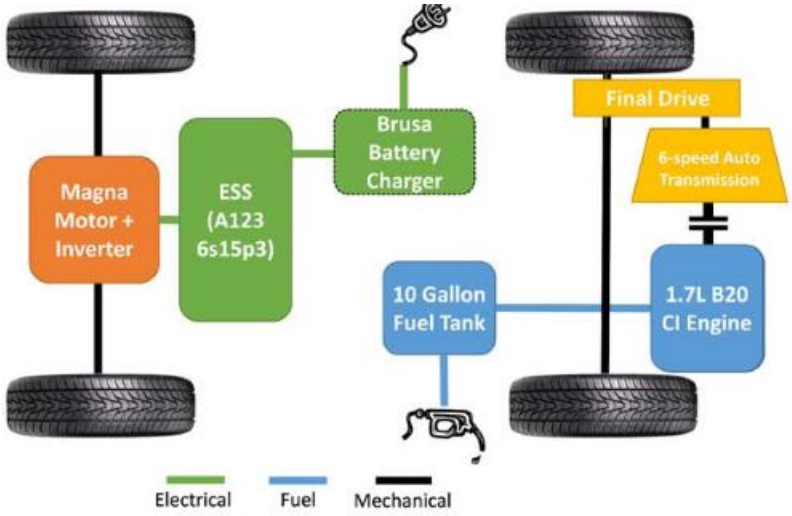

Figure 3.1. Architecture of the parallel-through-the-road hybrid electric vehicle [6].

3. Transmission: Hydra-Matic 6T40 6-speed automatic transmission from General Motors.

4. Fuel storage: 10 gallon capacity tank

5. After-treatment: Under-floor Diesel Oxidation Catalyst (DOC), Diesel Particulate Filter (DPF), Urea Selective Catalytic Reduction (SCR), Onboard Diesel Exhaust Fluid (DEF) storage and delivery

The details of the rear drivetrain are as follows:

1. Electric motor: 100kW Magna motor

2. Transmission: Fixed gear transmission integral to motor

3. Energy storage system: 16.2 kWh A123 Li-ion battery with 6S15P3 configuration

3.2 Supervisory Controller

The MicroAutoBox II, which was donated by dSPACE, is used as the supervisory vehicle controller, which controls the different power sources and actuators in the

vehicle platform. MicroAutoBox II is a real-time system for performing fast function prototyping. Similar to an Electronic Control Unit, it operates without user intervention. The processor used in the controller is an IBM PPC 750 FL, with clock speed of 900 MHz. Additionally, it has a 16 MB main memory, 6 MB memory only meant for communication between the controller and a personal computer, and a 16 MB nonvolatile flash memory. The CAN interface (discussed in the next section) has 4 channels and all are used in this vehicle platform. Furthermore, the controller has 32 16-bit analog input channels, 8 16-bit analog output channels, 24 digital input channels and 24 digital output channels. The controller operates at 12 V provided by the vehicle's battery and appropriate overvoltage, overcurrent and short circuit protections have been provided for uninterrupted operation.

3.3 Controller Area Network (CAN)

The Controller Area Network (CAN) is a serial communication bus designed for robust and flexible performance in harsh environments, and particularly for industrial and automotive applications [36].

The CAN bus was invented by Bosch and officially released in 1986 at the Society of Automotive Engineers conference in Detroit, Michigan. The major advantage of CAN is reduction of cable wiring due to its presence in the vehicle. As a result, the electronic control units inside a vehicle can communicate with each other using only a single pair of wires in theory. Low cost, centralization, robustness, efficiency and flexibility are some other advantages of using a CAN bus in a vehicle.

The original Chevrolet Malibu 2013 used as the vehicle platform contained only one CAN bus. In the present car, however, there are four CAN channels and they are as follows:

1. Engine CAN: In the vehicle platform, the engine is operated by the engine CAN bus and signals from the vehicle CAN are added to it by the MicroAutoBox controller.

2. Vehicle CAN: All the other systems of the vehicle platform are connected to the vehicle CAN bus. When the engine is running, the engine CAN and the vehicle CAN are joined together. When the motor is running and the engine is not, the MicroAutoBox controller fakes the required engine signals for letting the vehicle have the impression that the engine is running and the vehicle is in neutral. The vehicle CAN bus is connected to the after-treatment controller.

3. Motor CAN: The electric motor is operated by the motor CAN bus.

4. Battery CAN: There is a separate CAN bus for the battery. This bus also has the BRUSA battery charger connected to it.

3.4 Drive Cycles

A drive cycle is a time series that represents the speed of a vehicle. There are several drive cycles that depict different driving situations. For example, the Urban Dynamometer Driving Schedule (UDDS) drive cycle in Figure 3.2 [37] simulates urban driving conditions whereas the Highway Fuel Economy Test (HWFET) drive cycle in Figure 3.3 [38] simulates highway driving conditions. Drive cycles are used to analyze the performance of vehicles with respect to fuel consumption and exhaust gas emissions.

In the current work, the UDDS and the HWFET drive cycles have been used for testing and evaluating the performance of the genetic algorithm power management strategy. Some strategies like regression and LSTM network require several drive cycles for their development which have been mentioned in [6] and [29]. The data points for all drive cycles are available at [39].

In this chapter, we reviewed the vehicle platform, the supervisory controller and the CAN channels used in the vehicle and the concept of a drive cycle.

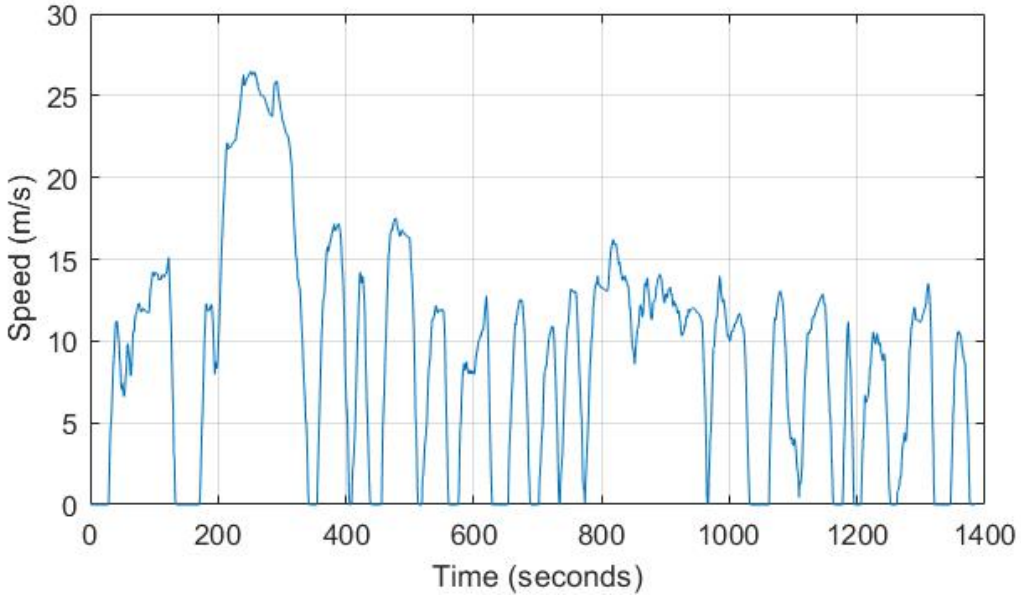

Figure 3.2. Speed profile of a vehicle for the Urban Dynamometer Driving Schedule drive cycle [37].

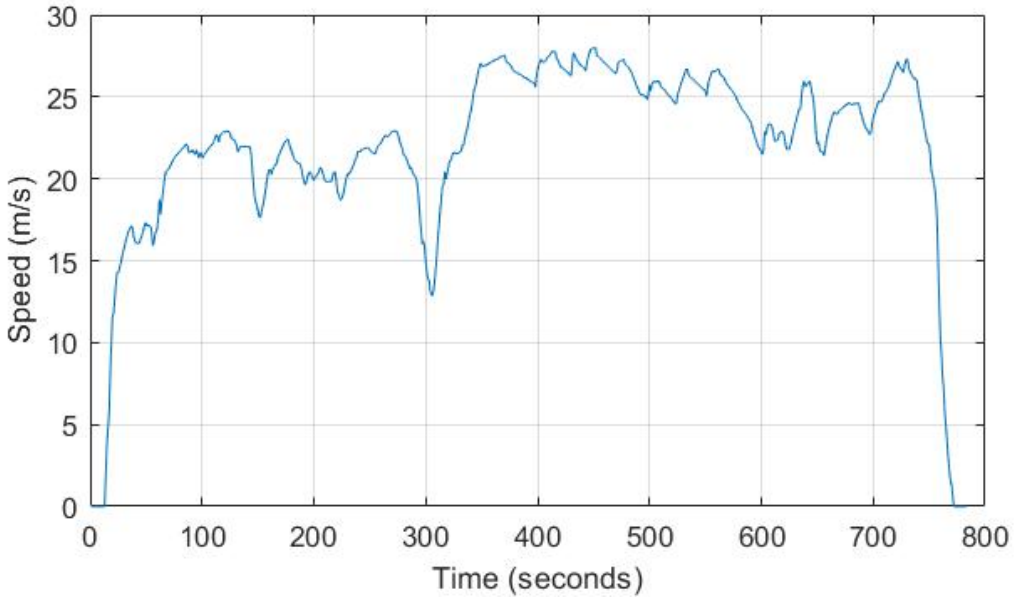

Figure 3.3. Speed profile of a vehicle for the Highway Fuel Economy Test drive cycle [38].

4. PREVIOUSLY DEVELOPED POWER MANAGEMENT STRATEGIES

The purpose of this chapter is to introduce the mathematical model of the vehicle and previously developed power management strategies tested on this model. The vehicle model was developed by Gupta [6]. The dynamic programming (DP) strategy was borrowed from an open-source DP function [40]. The Equivalent Consumption Minimization Strategy was developed by Paganelli [23]. The regression model and pSOC model were developed by Gupta [6]. The LSTM model was developed by Sun [29].

4.1 Simplified Model

A mathematical model of the vehicle relates the input variables associated with the vehicle with its output variables using several equations that depict the vehicle dynamics. Based on the variables considered as inputs (and outputs), there are two types of vehicle models.

In the forward facing model, the torque delivered by the powertrain is an input to the model and the vehicle velocity is the output. In other words, the vehicle velocity is a result of the powertrain torque. To control the vehicle velocity, the actual velocity of the vehicle can be compared with the desired velocity using a negative feedback loop and a PID controller can be implemented. This model is called forward facing because it is in accordance with the principle that an effect (vehicle velocity) cannot occur before its cause (powertrain torque). Implementing the control system with the forward facing model requires finding solutions of differential equations and for complicated vehicle models, the process is computationally intensive.

In contrast, the vehicle velocity is the input in a backward facing model and the powertrain torque is the output. With reference to the causality in a vehicle, it means

that we know the vehicle speed in advance and we hope that it will be matched by the real vehicle. And based on the vehicle dynamics, we calculate the torque to be delivered by the powertrains. In terms of computational cost and simplicity, the backward facing model triumphs over the forward facing model.

The total tractive force F_{tract} required at the wheels of the vehicle can be considered to be the sum of the road load force F_{load} and the inertial force $F_{inertia}$ of the vehicle,

$$F_{tract} = F_{load} + F_{inertia}. \tag{4.1}$$

The load force is given by

$$F_{load} = A + Bv + Cv^2, \tag{4.2}$$

where the values of the loss coefficients A, B and C are taken from the EPA dynamometer testing data [41]. The inertial force is given by Newton's first law of motion,

$$F_{inertia} = M_{veh}\frac{dv}{dt}, \tag{4.3}$$

where M_{veh} is the total mass of the vehicle excluding the wheel and the gear box inertia. The angular velocity of a wheel of the vehicle is given by

$$\omega_w = \frac{v}{r_w}, \tag{4.4}$$

where r_w is the radius of the wheel. The angular velocity of the engine differential is given by

$$\omega_{ed} = X_{ed}\omega_w, \tag{4.5}$$

where X_{ed} is the engine differential gear ratio. The angular velocity of the engine is given by

$$\omega_e = X_g\omega_{ed}, \tag{4.6}$$

where X_g is the transmission gear ratio. The angular velocity of the motor differential is given by

$$\omega_{md} = X_{md}\omega_w \tag{4.7}$$

where X_{md} is the motor differential ratio. As there is a direct coupling between the motor and the motor differential, their velocities are equal. The total wheel torque required at the wheels is given by

$$\tau_v = \frac{F_{tract}}{r_w}. \tag{4.8}$$

The ratio between the motor torque at the wheels τ_{mv} and the total torque at the wheels τ_v is called the torque split,

$$split = \frac{\tau_{mv}}{\tau_v}. \tag{4.9}$$

The split can assume values between -1 and 1. The mode of operation of the HEV depends on the split value. A split value of 0 means that the vehicle is powered purely by the engine and a split value of 1 means that the vehicle is powered purely by the motor. Any value between 0 and 1 means that the vehicle is powered by both sources and the mode is called blended mode. A value between -1 and 0 means that the torque provided by the engine is more than what is required by the vehicle and the rest of the torque is used to charge the battery via the motor which acts like a generator. When all the torque provided by the engine is used to charge the battery, the split value is -1.

The split value is thus a free factor and as long as the total torque requirement is met by the vehicle dynamics, the split value can assume infinitely many values between -1 and 1. The goal of any power management strategy in this backward facing model is attempting to find the time series of the split value that will minimize an objective function based on the vehicle model. This concept is further explored in subsequent sections.

The values of the motor and engine torques are given by

$$\tau_m = \frac{\tau_{mv}}{X_{md}} \tag{4.10}$$

$$\tau_e = \frac{\tau_{ev}}{X_{ed}X_g}. \tag{4.11}$$

Following are the constraints on the torques associated with the vehicle,

$$\min(\tau_m) \leq \tau_m \leq \max(\tau_m) \tag{4.12}$$

$$\min(\tau_m), \max(\tau_m) = f_1(\omega_m) \tag{4.13}$$

$$0 \leq \tau_e \leq \max(\tau_e) \tag{4.14}$$

$$\max(\tau_e) = f_2(\omega_e, N_g). \tag{4.15}$$

The transmission used in the real vehicle is GM 6T40 from General Motors. Its transmission shift map is shown in Figure 4.1 and is used to calculate the gear number N_g. For GM 6T40, X_g can take 6 values depending on the driving conditions. From these 6 values, N_g decides what value X_g should take for the driving conditions at that instant. In other words, all possible values of X_g can be considered as a vector of length 6 and N_g is the index which is used to pick a particular value of X_g from this vector. The transmission map shows the variation of the accelerator pedal position as functions of transmission output speed for various gear shifts. In mathematical terms, these plots for different gear shifts are available as vectors with each vector consisting of different transmission output speeds recorded at equal intervals of accelerator pedal position.

It is assumed here that the shift in gear or the change in gear number is instantaneous and occurs at the end of each time step. The accelerator pedal position is calculated using the engine power at each time step and using the current gear number, the transmission output speed is estimated. The gear number of the next time step being unknown, the engine speeds for the up-shift and the down-shift are located and are compared with the current engine speed. The gear number is incremented if the current engine speed is greater than the engine speed for up-shift and vice versa. When the gear output is zero, the engine is assumed to be in the idling mode and the engine torque and speed are taken as $\omega_e = 810$ RPM and $\tau_e = 0$.

Unless we have the accelerator pedal position, however, the above technique cannot be used to find the gear number. Figure 4.2 shows the variation of the engine

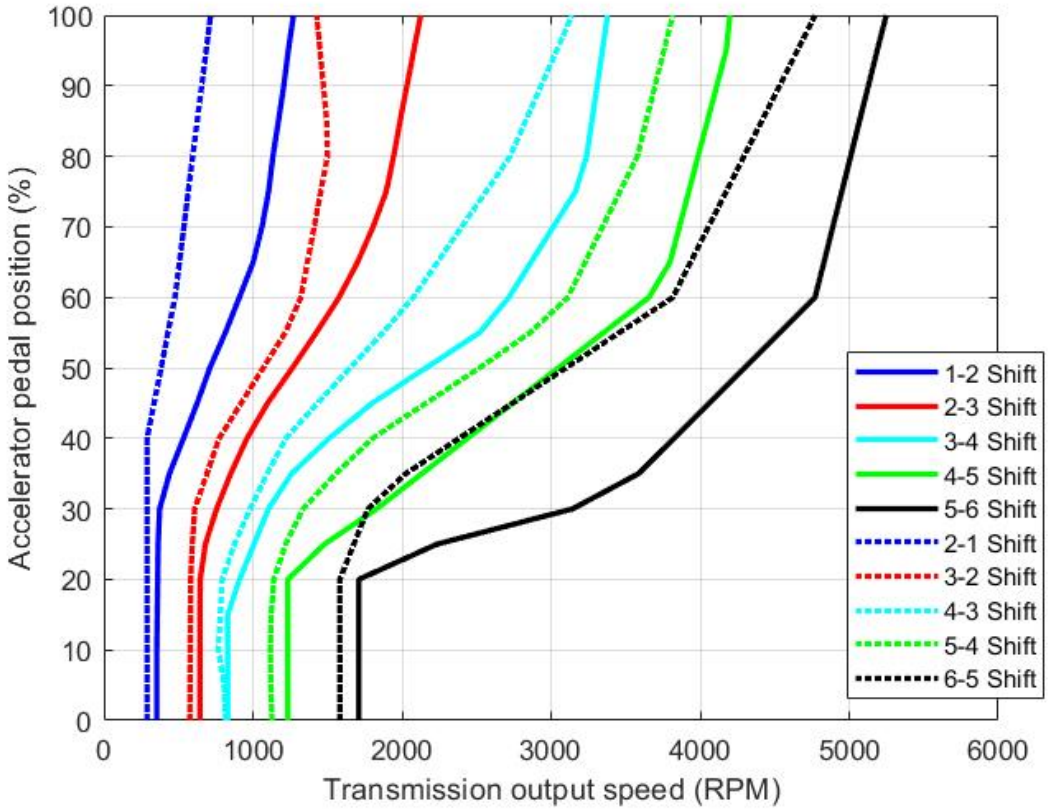

Figure 4.1. Gear transmission map for GM 6T40 transmission [6].

power P_e with the accelerator pedal position obtained from dynamometer testing. The engine power that is used to estimate the accelerator pedal position is given by,

$$P_e = \tau_e \omega_e. \tag{4.16}$$

In mathematical terms, this plot is available as a vector consisting of engine power values recorded at equal intervals of accelerator pedal position. Using the current value of the engine power, interpolation and extrapolation techniques are then used to estimate the accelerator pedal position.

Figure 4.3 shows a surface plot that depicts the variation of injected fuel as a function of the engine speed and the accelerator pedal position. The fuel consumption can be calculated from the surface fit equation of this plot combined with the curve fit equation of the accelerator pedal position and the engine power.

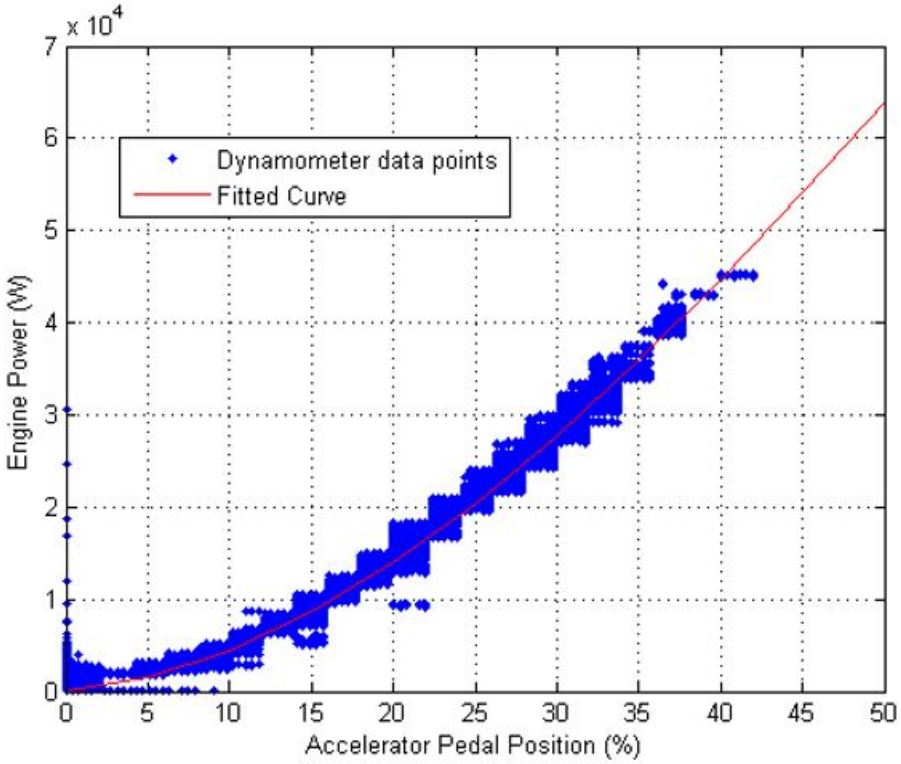

Figure 4.2. Relationship between accelerator pedal position and engine power [6].

The motor current is given by

$$I_m = \frac{\tau_m \omega_m}{V_m} \tag{4.17}$$

where $V_m = 300$ V is the motor voltage. Due to conversion efficiency between the battery and the motor being less than 1, a 10 percent power loss is assumed. The state of charge (SOC) for the next time step can be calculated from the current time step as follows,

$$SOC_{t+1} = SOC_t \frac{I_{batt}/(N_p Q_{cell})}{\delta t}. \tag{4.18}$$

In the above equation, N_p is the number of cells in a battery, Q_{cell} is the capacity of a single battery, I_{batt} is the battery current and δt is the time step. The state of charge (SOC) is a measure of the total amount of charge in a battery. It's a dimensionless

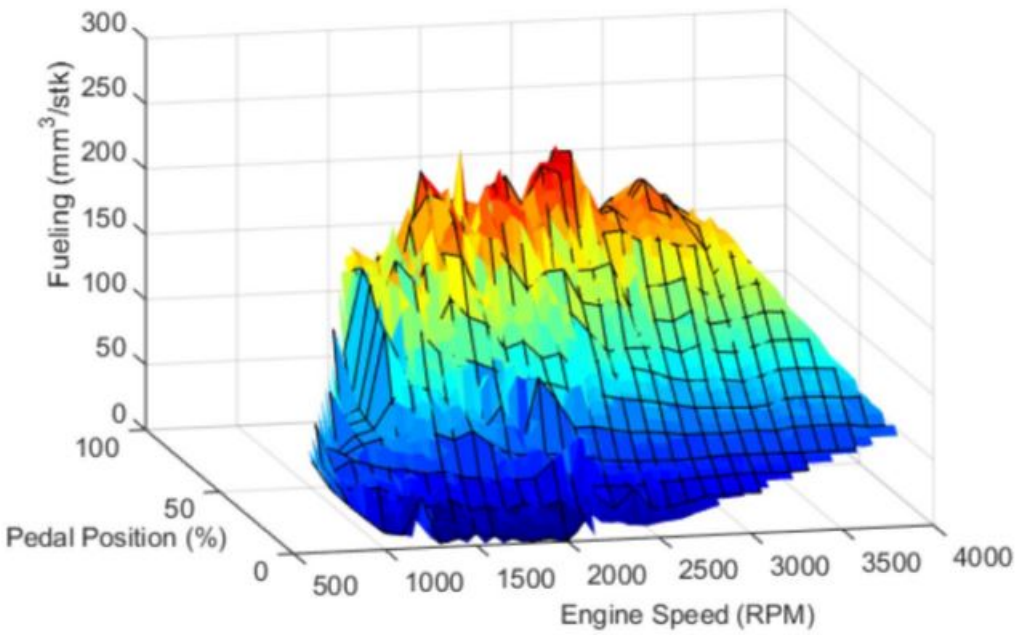

Figure 4.3. Fueling map to predict injected fuel [6].

quantity. An SOC value of 1 means that the battery is fully charged and 0 means that the battery is fully discharged.

Equation 4.18 is essentially the discrete time model of our vehicle as it resembles Equation 2.1, which is a general form of a discrete time model. Here, the SOC is the state x of our vehicle model and the input u is the vehicle speed which, as a result of the complicated vehicle dynamics explained previously, affects the battery current I_{batt}. Hence, for comparing various power management strategies of an HEV, comparing the variation of SOC with time is a good strategy.

4.2 Dynamic Programming

As explained in chapter 2, dynamic programming is an optimization technique that solves a problem by transforming it into a series of simpler subproblems. The authors of [40] have developed a function in MATLAB that can solve any optimization problem, in which the time and states are discretized, using dynamic programming.

Reference [40] itself shows how the function can be used to employ dynamic programming with a couple of examples. The open-source MATLAB function has been used for obtaining the results of dynamic programming.

Dynamic programming cannot be employed in a vehicle controller as the control algorithm is not causal in nature. However, dynamic programming serves as a benchmark to which other power management controllers can be compared. So studying the results of dynamic programming is a worthwhile effort.

4.2.1 Implementation

Solving the dynamic programming problem using the open source function provided in [40] involves two major steps. The first is to discretize and initialize the states and inputs. As explained in the previous section, the state of charge (SOC) is a state of our mathematical model of the vehicle and it can take any value between 0 and 100 as SOC is expressed as a percentage of the total battery capacity. So we must specify the possible values between 0 and 100 which the SOC can take. In the case of the optimal control problem of power management of a hybrid electric vehicle, the torque split is the control law as the goal of the optimization problem is finding the torque split time series that will minimize an objective function. Hence, the torque split must also be discretized (between -1 and 1). There is a trade-off between computational efficiency and accuracy in the discretization process. A coarser discretization is more efficient computationally but might not result in an accurate control law due to the finitely smaller number of values the states and the inputs can assume. On the other hand, a finer discretization results in better accuracy due to the larger number of values the states and the inputs can assume but as the algorithm has to search through a larger number of values, the computations become more involved. Furthermore, the initial and final state constraints should also be specified. In our case, we wish to operate our vehicle in the charge sustaining mode. Hence, the initial and final SOC should be the same. If we do not provide any constraints on the state of charge, the

dynamic programming will minimize the greenhouse gas emissions from the fuel by consuming all charge from the battery. This is essentially the charge-depleting mode. The dynamic programming function itself doesn't recognize that the user wants to employ the charge sustaining mode as the function just performs mathematical computations. The user needs to set the charge sustaining mode by setting the initial and final SOC values to be equal. All of the above initializations are performed in the main MATLAB code.

The second major step involves computations so that the objective function is minimized. A function is developed which takes the model inputs like speed, acceleration and gear number time series and based on the vehicle model developed in the last section, calculates the objective function. In the charge sustaining mode, we set the total greenhouse gas emissions as the objective function.

After the initializations are set in the main function, we pass these initial parameters and the objective function (as a function handle) to the dpm function and it calculates the control law to be followed so as to minimize greenhouse gas emissions over the drive cycle as well as the variation of SOC over the drive cycle.

4.2.2 Results

The results of dynamic programming for the UDDS and the Highway FET drive cycles are shown in Figure 4.4 to Figure 4.9. For each drive cycle, a total of three plots are shown which depict the variations of the torque split, the SOC and the total power over time [29].

The motor energy consumption for the UDDS drive cycle is 2.0490×10^6 J, the engine energy consumption is 7.5592×10^6 J and the total energy consumption is 9.6082×10^6 J.

The motor energy consumption for the HWFET drive cycle is 4.9428×10^5 J, the engine energy consumption is 1.1636×10^7 J and the total energy consumption is 1.2130×10^7 J.

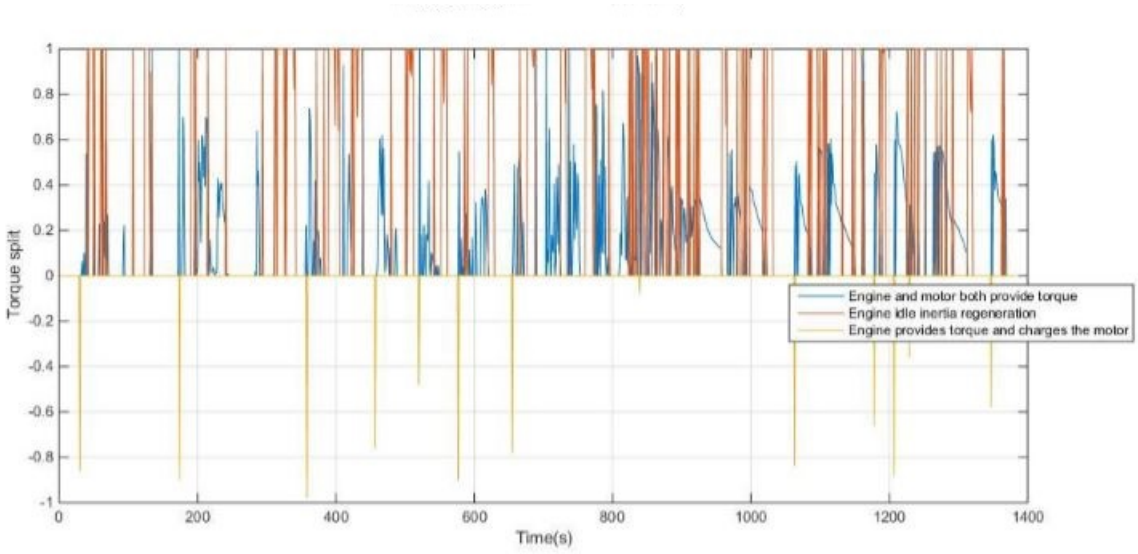

Figure 4.4. Torque split generated by dynamic programming for the Urban Dynamometer Driving Schedule drive cycle [29].

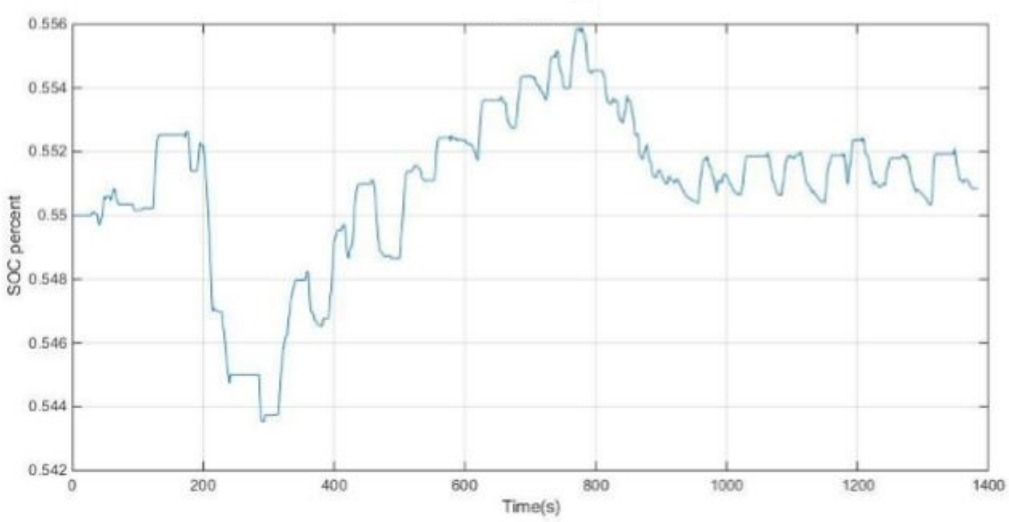

Figure 4.5. SOC variation generated by dynamic programming for the Urban Dynamometer Driving Schedule drive cycle [29].

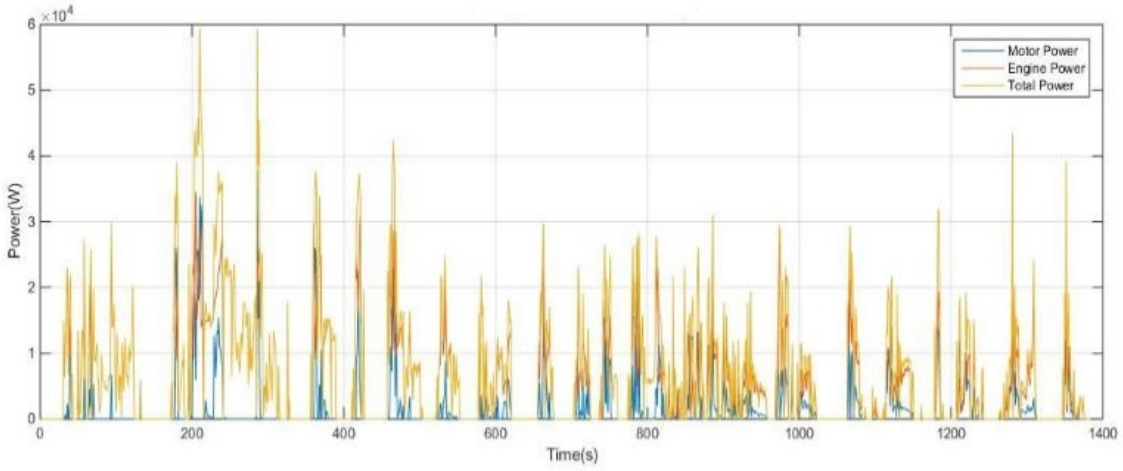

Figure 4.6. Dynamic programming power required for the Urban Dynamometer Driving Schedule drive cycle [29].

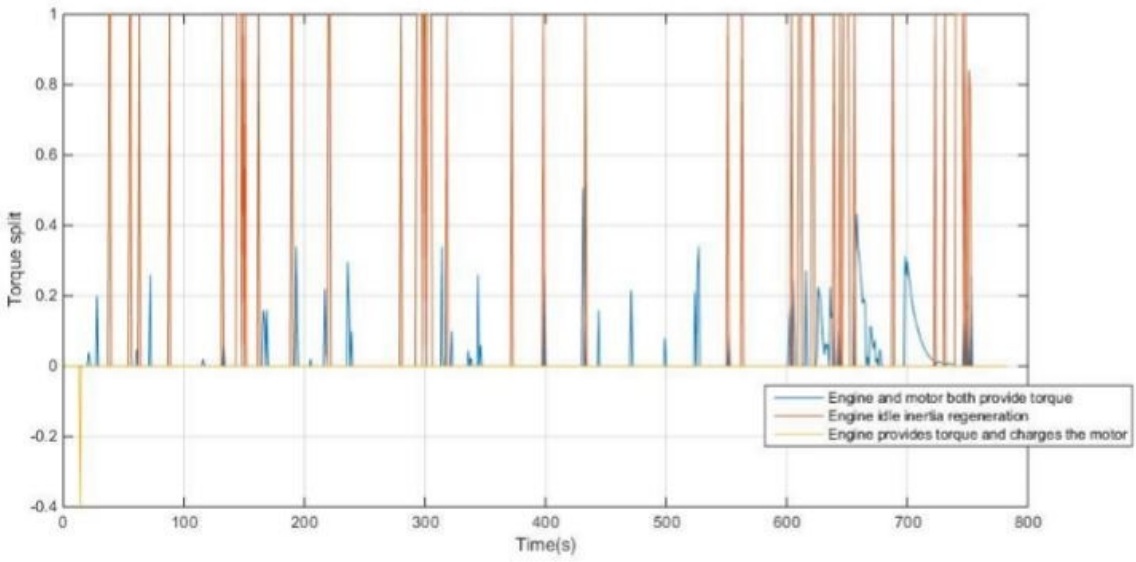

Figure 4.7. Torque split generated by dynamic programming for the Highway Fuel Economy Test drive cycle [29].

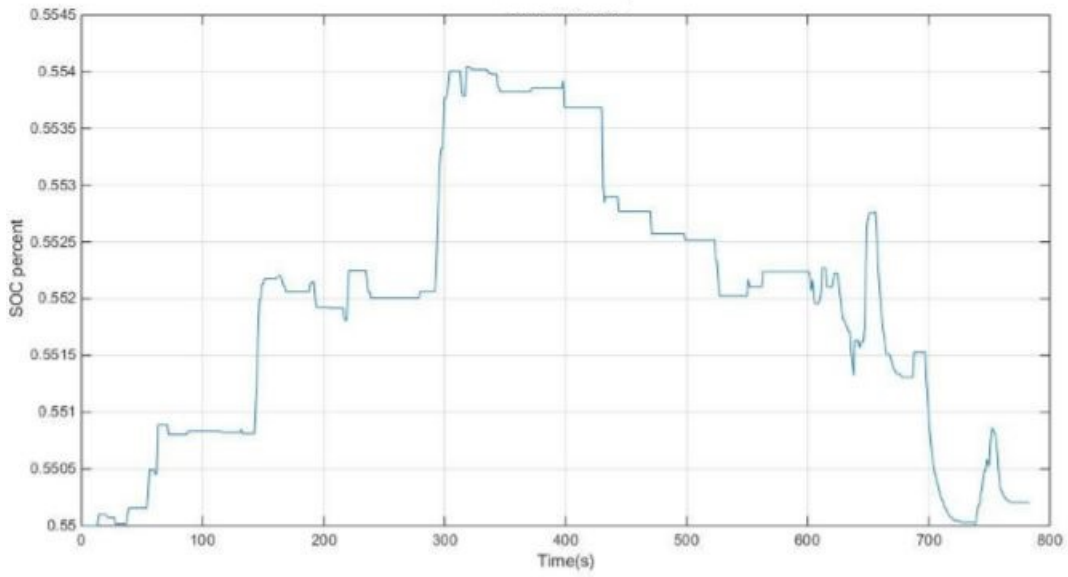

Figure 4.8. SOC variation generated by dynamic programming for the Highway Fuel Economy Test drive cycle [29].

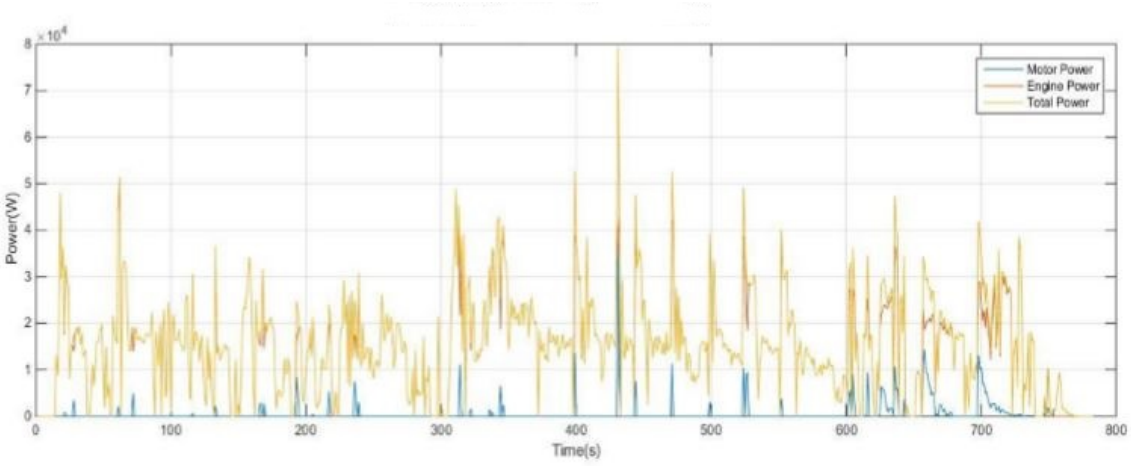

Figure 4.9. Dynamic programming power required for the Highway Fuel Economy Test drive cycle [29].

The torque split is, in theory, a continuous function of time. In Figure 4.4 and Figure 4.7, the torque split plots are divided into 3 regions each. The values indicated

by blue lines are representative of the blended mode, in which the total power is supplied by both the engine and the motor. The values indicated by red lines are representative of the regeneration mode in which the total torque demand is negative and thus is used to charge the battery. In this mode, the engine is idle and the motor acts as a generator. Lastly, the yellow line represents the charging mode, in which the torque provided by the vehicle is more than what is required by the vehicle and the surplus torque is used to charge the battery through the motor which acts as a generator.

The SOC is also, in theory, a continuous function of time. Figure 4.5 and Figure 4.8 show the SOC variation with time for the UDDS and HWFET drive cycles, respectively. In both drive cycles, the SOC is maintained over the entire duration and the maximum deviation of the SOC from the target value of 0.55 does not exceed 0.006. Every drop in an SOC plot indicates that the battery charge is being consumed for powering the vehicle. The battery charge should be replenished later by either operating the vehicle in the regeneration mode or in the charging mode, which is reflected in the SOC plot by a rise in the SOC value.

Like the torque split and the SOC, the required power is also a continuous function of time. In Figure 4.6 and Figure 4.9, the power plots are divided into 3 regions each. The values indicated by blue lines represent the total power demand of the vehicle fulfilled by the motor and the values indicated by red lines represent the total power demand of the vehicle fulfilled by the engine. The plots are indicative of the fact that most of the power demand is fulfilled by the engine and any additional power demand is fulfilled by the motor. This can also be corroborated by the total motor energy consumption and the total engine energy consumption values for both drive cycles mentioned previously.

4.3 Equivalent Consumption Minimization Strategy (ECMS)

While dynamic programming attempts to optimize the torque split for minimizing an objective function (fuel consumption or net emissions), the Equivalent Consumption Minimization Strategy (ECMS) performs instantaneous optimization. Hence, the optimization is performed with respect to vehicle features that are measured in real time like engine torque and motor torque, among others. The following equation is a general representation of the ECMS strategy,

$$\tau_e^{opt}, \tau_m^{opt} = \arg\min J_t. \tag{4.19}$$

Here, the term J_t is the instantaneous cost and not the cost accumulated over the time period up to the current time.

4.3.1 Implementation

For power management of a hybrid electric vehicle, there are several choices for J_t based on the driving mode of the vehicle and/or the goal of the optimization. The following equation describes a suitable cost function that minimizes the net greenhouse gas emissions due to the engine and the motor operation,

$$J_t = GHG_{WTW,fuel} + \zeta * pen * GHG_{WTW,electricity} \tag{4.20}$$

$$GHG_{WTW,fuel} = f(\tau_e, \omega_e) \tag{4.21}$$

$$GHG_{WTW,electricity} = f(\tau_m, \omega_m), \tag{4.22}$$

where $GHG_{WTW,fuel}$ and $GHG_{WTW,electricity}$ are the amount of emissions at that particular time instant due to engine and motor operation, respectively. Reference [6] describes how these emissions are calculated.

The equivalence factor ζ plays an important role in the ECMS and is used to convert electrical power into equivalent fuel consumption. ζ is a measure of the future efficiency of the engine and the energy storage device. A higher value of the

equivalence factor makes the controller reduce battery usage to power the vehicle as it implies that using electrical energy will result in higher value of J_t and vice versa.

Since the vehicle is aimed to be operated in the charge sustaining mode, the initial and final SOC are equal. Hence, the battery discharge that results during lower ζ should be compensated by recharging the battery using the fuel in the engine during lower ζ. In this work, ζ is taken to be the same for both charging and discharging modes.

The equivalence factor ζ is different for different drive cycles. If the drive cycle is known in advance, ζ can be calculated by a numerical optimization procedure. The approach used by Gupta [6] for finding ζ for the UDDS drive cycle was running a Design of Experiments (DOE) with varying equivalence factors and choosing the value that satisfies the constraint of maintaining the SOC. Based on this DOE, an equivalence factor of 2.24 was found to maintain the SOC over the UDDS drive cycle.

The penalty function *pen* depends on the deviation from the target SOC and is plotted in Figure 4.10 [6].

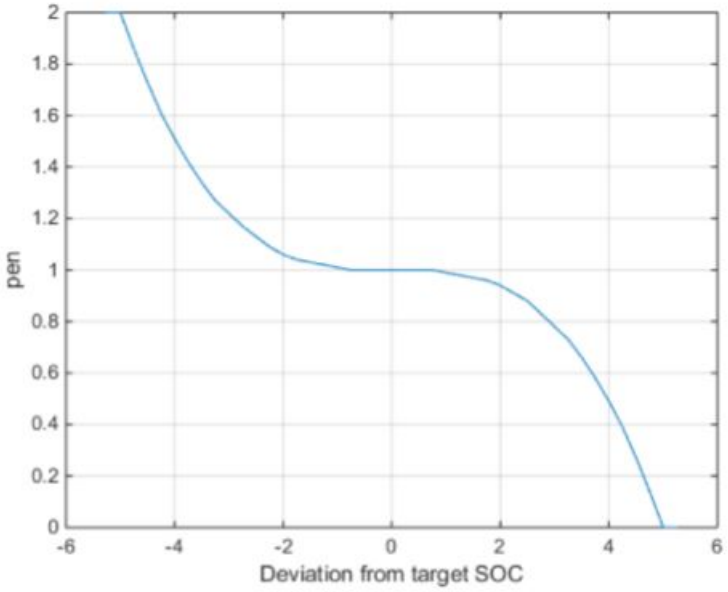

Figure 4.10. The plot of the penalty function.

Following is the ECMS algorithm employed to solve the local minimization problem:

1. Find the maximum and minimum motor torques available at the current motor speed.
2. Find all the corresponding possible motor and engine torques.
3. For all combinations of motor and engine torques, calculate J_t.
4. Find the combination with the minimum value of J_t that satisfies all vehicle constraints.

4.3.2 Results

The results of ECMS algorithm for the UDDS and the Highway FET drive cycle are shown in Figure 4.11 - Figure 4.16. For each drive cycle, a total of three plots are shown that depict the variations of the torque split, the SOC and the total power over time [29].

The motor energy consumption for the UDDS drive cycle is $5.3856 \times 10^6(J)$, the engine energy consumption is $7.8520 \times 10^6(J)$ and the total energy consumption is $1.3238 \times 10^7(J)$.

The motor energy consumption for the HWFET drive cycle is $3.5721 \times 10^6(J)$, the engine energy consumption is $1.1636 \times 10^7(J)$ and the total energy consumption is $1.2130 \times 10^7(J)$.

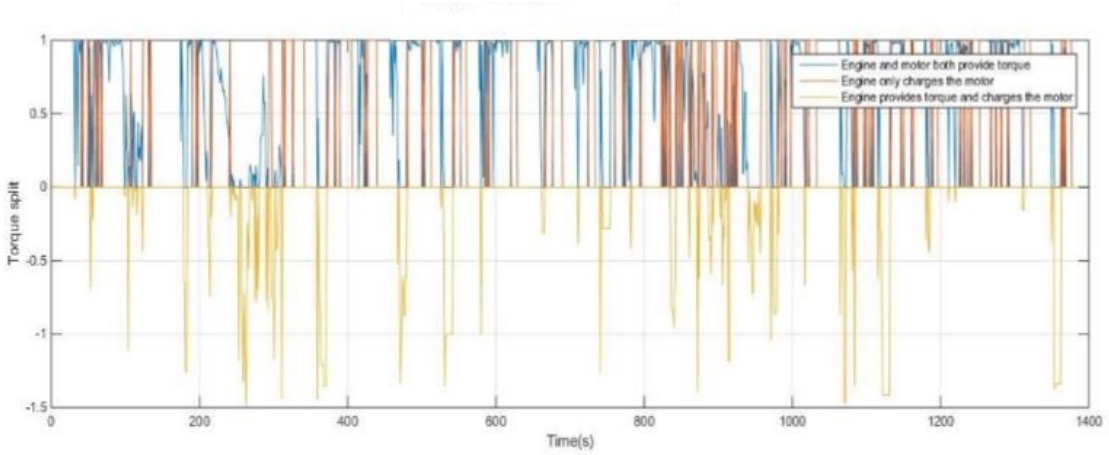

Figure 4.11. Torque split generated by Equivalent Consumption Minimization Strategy for the Urban Dynamometer Driving Schedule drive cycle [29].

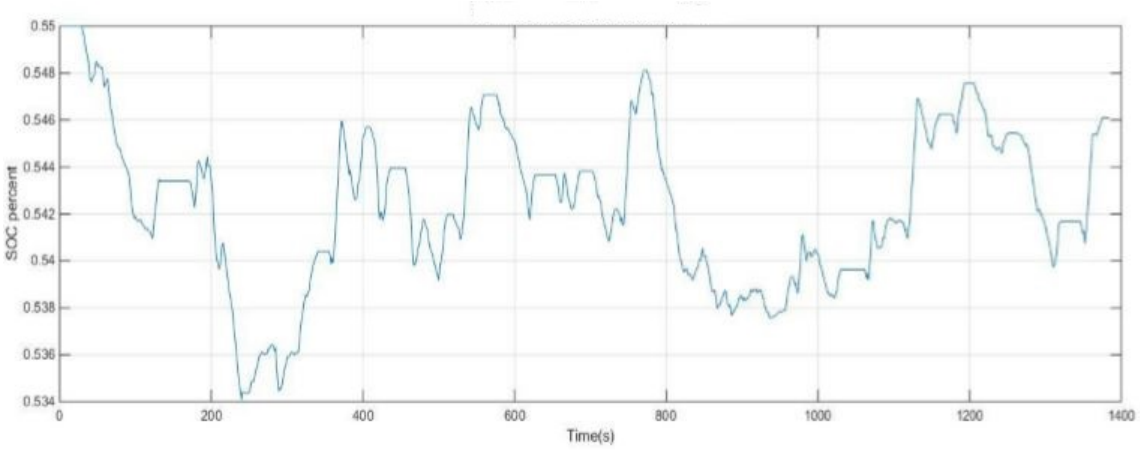

Figure 4.12. SOC variation generated by Equivalent Consumption Minimization Strategy for the Urban Dynamometer Driving Schedule drive cycle [29].

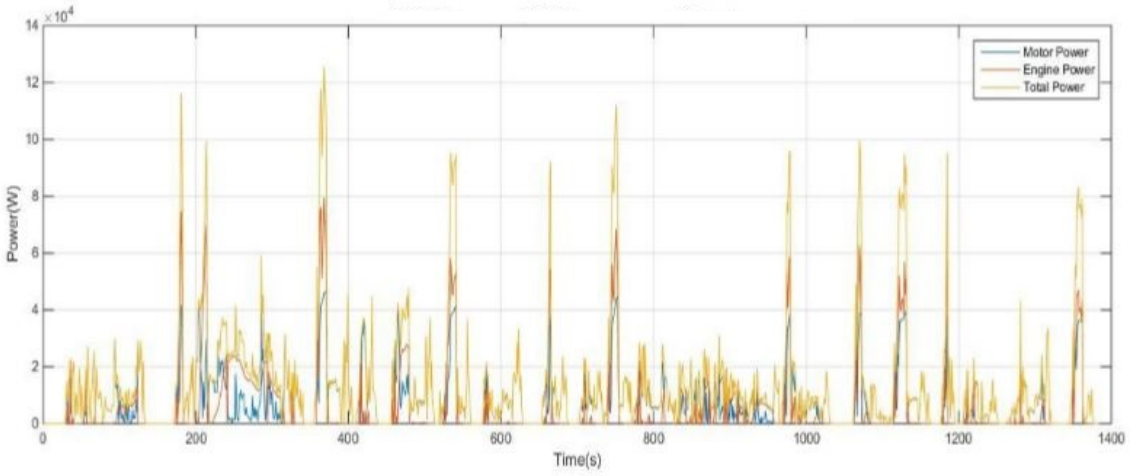

Figure 4.13. Equivalent Consumption Minimization Strategy power required for the Urban Dynamometer Driving Schedule drive cycle [29].

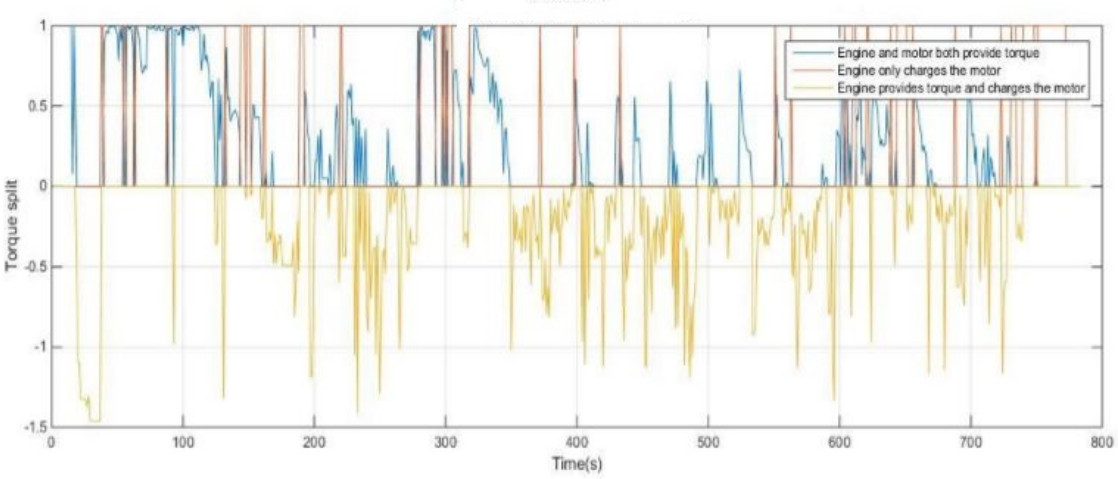

Figure 4.14. Torque split generated by Equivalent Consumption Minimization Strategy for the Highway Fuel Economy Test drive cycle [29].

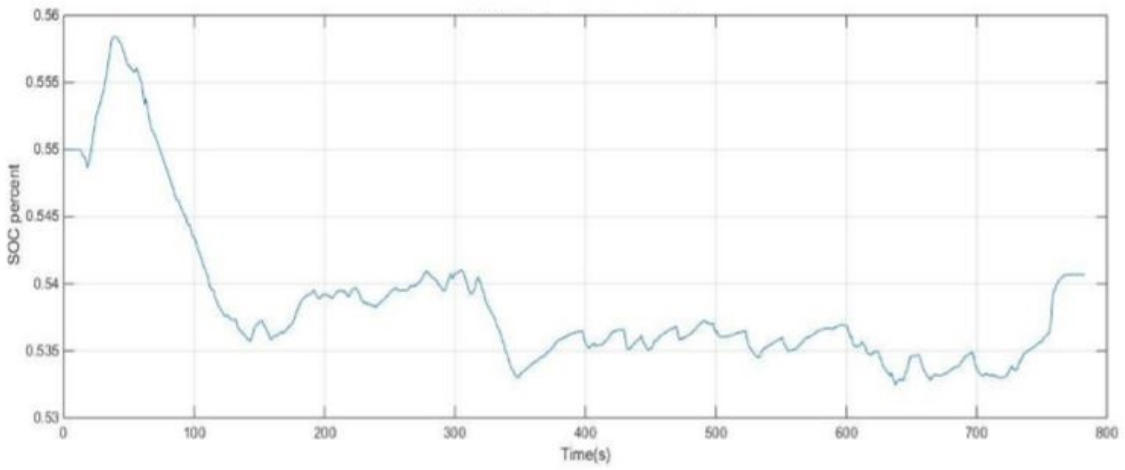

Figure 4.15. SOC variation generated by Equivalent Consumption Minimization Strategy for the Highway Fuel Economy Test drive cycle [29].

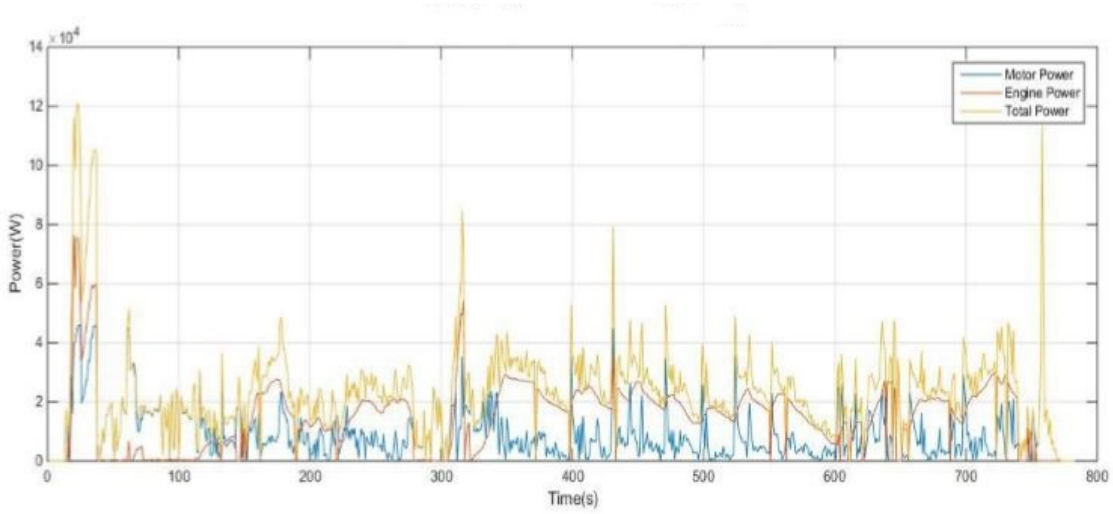

Figure 4.16. Equivalent Consumption Minimization Strategy power required for the Highway Fuel Economy Test drive cycle [29].

The torque split is, in theory, a continuous function of time. In Figure 4.11 and Figure 4.14, the torque split plots are divided into 3 regions each. The values indicated by blue lines are representative of the blended mode, in which the total power is supplied by both the engine and the motor. The values indicated by red

lines are representative of the regeneration mode in which the total torque demand is negative and thus is used to charge the battery. In this mode, the engine is idle and the motor acts as a generator. Lastly, the yellow line represents the charging mode, in which the torque provided by the vehicle is more than what is required by the vehicle and the surplus torque is used to charge the battery through the motor which acts as a generator. As compared to the DP strategy, the number of times the vehicle operates in the charging mode is higher in the ECMS strategy as indicated by the higher number of yellow lines.

The SOC is also, in theory, a continuous function of time. Figure 4.12 and Figure 4.15 show the SOC variation with time for the UDDS and HWFET drive cycles, respectively. These plots show that the ECMS algorithm prefers depleting the battery state of charge as indicated by the fact that the SOC values hardly rise above the target value of 0.55 and the final SOC values are less than the initial SOC values for both drive cycles. Proper tuning of the equivalence factor by more extensive numerical optimization or DOE will make the vehicle operate as close to the charge-sustaining mode as possible.

Like the torque split and the SOC, the required power is also a continuous function of time. In Figure 4.13 and Figure 4.16, the power plots are divided into 3 regions each. The values indicated by blue lines represent the total power demand of the vehicle fulfilled by the motor and the values indicated by red lines represent the total power demand of the vehicle fulfilled by the engine. The plots are indicative of the fact that most of the power demand is fulfilled by the engine and any additional power demand is fulfilled by the motor. This can also be corroborated by the total motor energy consumption and the total engine energy consumption values for both drive cycles mentioned previously.

4.4 Proportional State of Charge (pSOC) Algorithm

The proportional State of Charge (pSOC) algorithm [6] is another power management strategy. Equations 4.23 and 4.24 encapsulate the pSOC algorithm well,

$$Split = K(SOC_{present} - SOC_{target}), \tag{4.23}$$

$$K = \frac{1}{100(SOC_{UB} - SOC_{LB})}. \tag{4.24}$$

The highest value of split (1) is associated with SOC_{UB} and the lowest value of split (-1) is associated with SOC_{LB}.

The pSOC algorithm shares some characteristics with a feedback control strategy and the different range of values that $SOC_{present}$ can take with respect to SOC_{target}, SOC_{UB} and SOC_{LB} decides the mode of operation of the vehicle. These different modes of operation are described below:

1. $SOC_{present} > SOC_{target}$ implies that the vehicle will be powered by the motor.
2. $SOC_{present} < SOC_{target}$ implies that the battery in the vehicle will be charged by the engine.
3. $SOC_{present} < SOC_{UB}$ implies that the vehicle will operate in electric-motor-only mode.
4. $SOC_{present} < SOC_{LB}$ implies that the vehicle will operate in IC-engine-only mode.

4.4.1 Results

The results of pSOC algorithm for the UDDS and the Highway FET drive cycle are shown in Figure 4.17 - Figure 4.22. For each drive cycle, a total of three plots are shown that depict the variations of the torque split, the SOC and the total power over time [29].

The motor energy consumption for the UDDS drive cycle is $1.8109 \times 10^6(J)$, the engine energy consumption is $7.9455 \times 10^6(J)$ and the total energy consumption is $9.7474 \times 10^6(J)$.

The motor energy consumption for the HWFET drive cycle is $4.0145 \times 10^5(J)$, the engine energy consumption is $1.1703 \times 10^7(J)$ and the total energy consumption is $1.2104 \times 10^7(J)$.

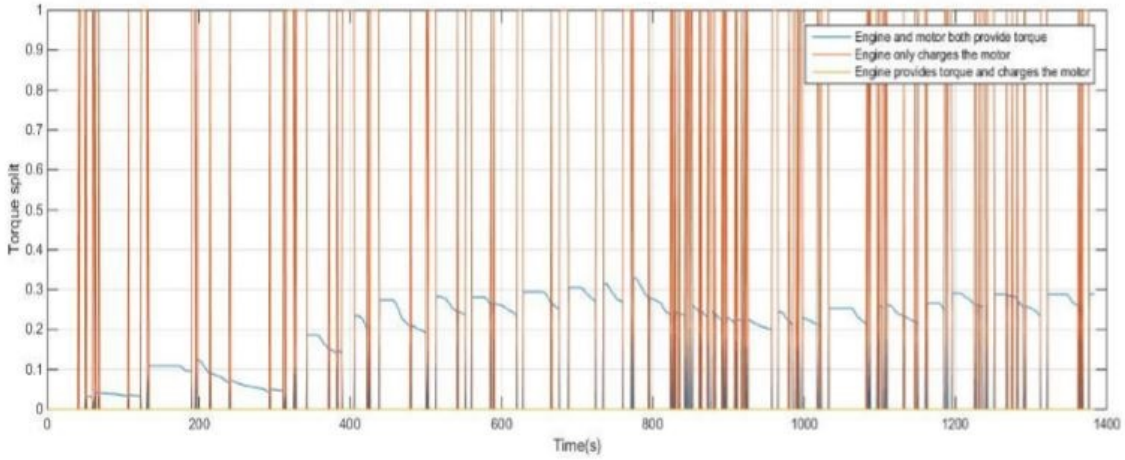

Figure 4.17. Torque split generated by the proportional SOC algorithm for the Urban Dynamometer Driving Schedule drive cycle [29].

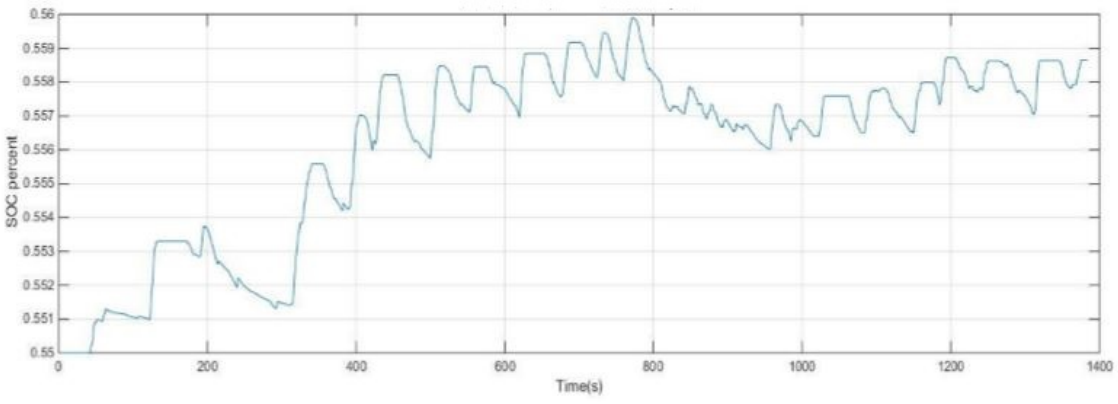

Figure 4.18. SOC variation generated by the proportional SOC algorithm for the Urban Dynamometer Driving Schedule drive cycle [29].

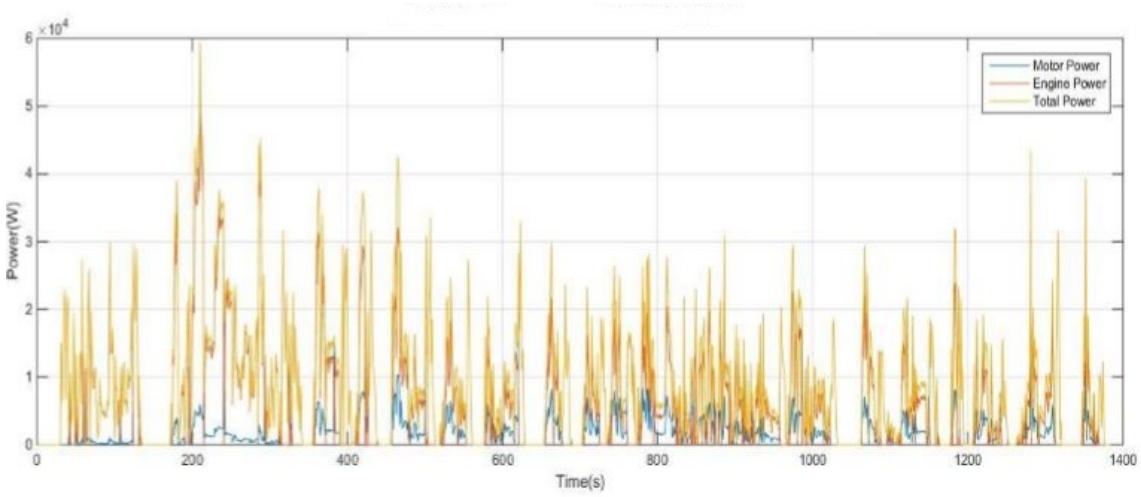

Figure 4.19. The proportional SOC algorithm power required for the Urban Dynamometer Driving Schedule drive cycle [29].

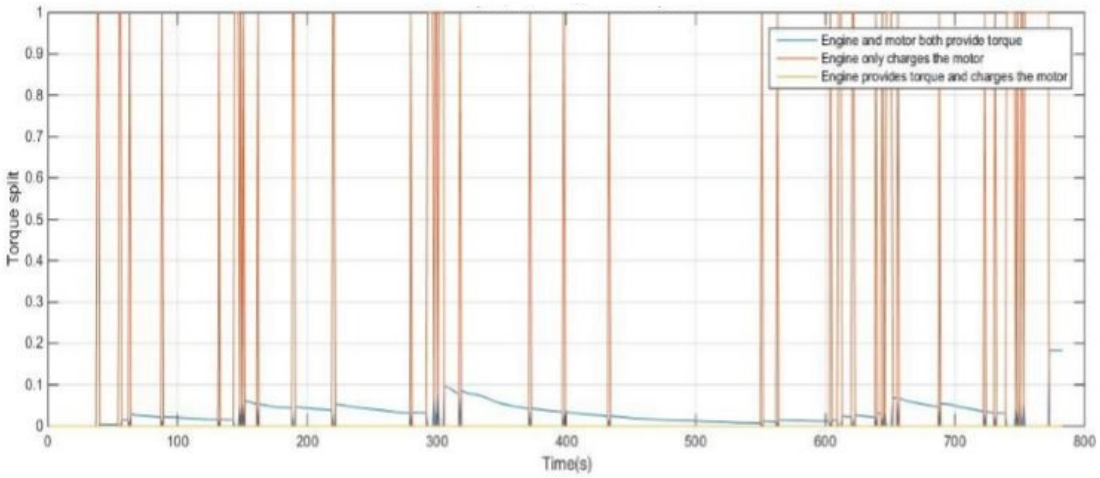

Figure 4.20. Torque split generated by the proportional SOC algorithm for the Highway Fuel Economy Test drive cycle [29].

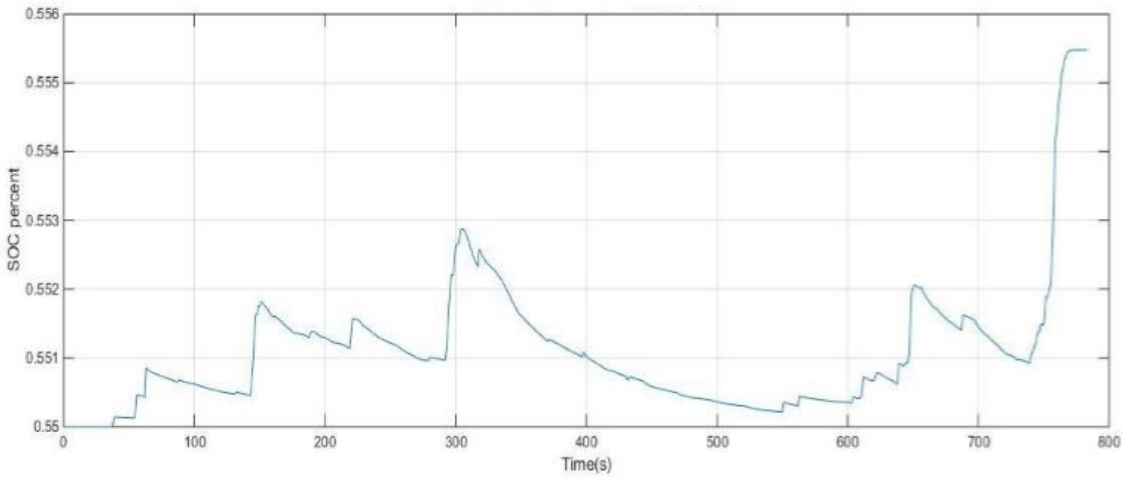

Figure 4.24. pSOC battery SOC for HWFET drive cycle

Figure 4.21. SOC variation generated by the proportional SOC algorithm for the Highway Fuel Economy Test drive cycle [29].

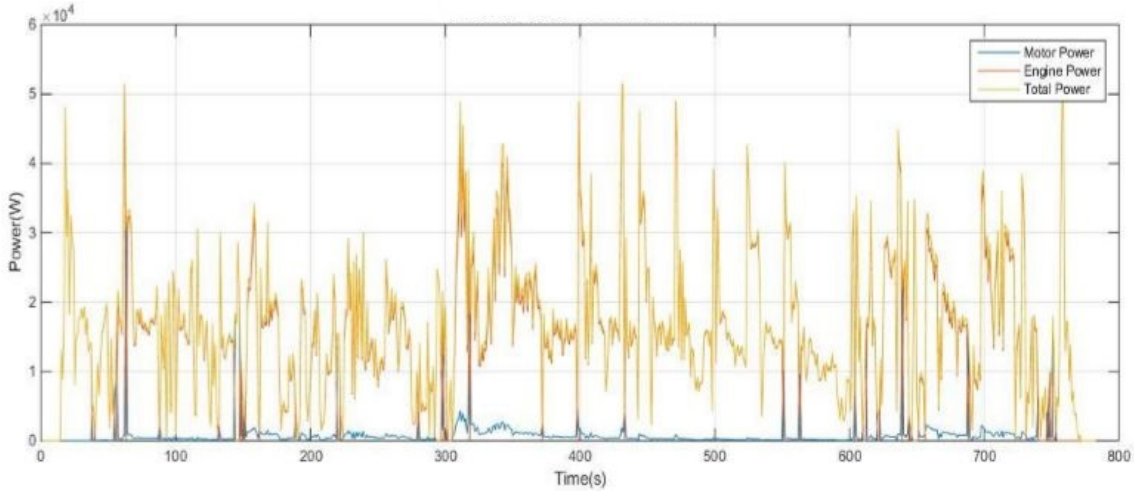

Figure 4.22. The proportional SOC algorithm power required for the Highway Fuel Economy Test drive cycle [29].

The torque split is, in theory, a continuous function of time. In Figure 4.17 and Figure 4.20, the torque split plots are divided into 3 regions each. The values indicated by blue lines are representative of the blended mode, in which the total

power is supplied by both the engine and the motor. The values indicated by red lines are representative of the regeneration mode in which the total torque demand is negative and thus is used to charge the battery. In this model, the engine is idle and the motor acts as a generator. Lastly, the yellow line represents the charging mode, in which the torque provided by the vehicle is more than what is required by the vehicle and the surplus torque is used to charge the battery through the motor which acts as a generator. With pSOC algorithm, the vehicle never operates in the charging mode; the battery gets charged through the regeneration mode.

The SOC is also, in theory, a continuous function of time. Figure 4.18 and Figure 4.21 show the SOC variation with time for the UDDS and HWFET drive cycles, respectively. Compared with the ECMS algorithm, the pSOC algorithm prefers charging the battery as indicated by the fact that the SOC values never go below the target value of 0.55 and the final SOC values are higher than the initial SOC values for both drive cycles.

Like the torque split and the SOC, the required power is also a continuous function of time. In Figure 4.19 and Figure 4.22, the power plots are divided into 3 regions each. The values indicated by blue lines represent the total power demand of the vehicle fulfilled by the motor and the values indicated by red lines represent the total power demand of the vehicle fulfilled by the engine. The plots are indicative of the fact that most of the power demand is fulfilled by the engine and any additional power demand is fulfilled by the motor. This can also be corroborated by the total motor energy consumption and the total engine energy consumption values for both drive cycles mentioned previously.

4.5 Regression Modeling

Regression is a rule-based power management strategy rather than an optimization-based strategy. As the name suggests, regression modeling involves estimating the relationship between the input and the output variables associated with the problem

of power management using data. In the context of this problem, the torque split is the output variable as we're trying to estimate it for minimum fuel consumption. The aforementioned data is generated by running simulations of dynamic programming for several drive cycles for the charge sustaining mode for reasons explained in the dynamic programming section. Then, formal regression techniques are applied to find out the equation of the curve that estimates the relationship between the input and the output most accurately.

The advantages of regression modeling over traditional optimization techniques are twofold. Firstly, the regression control algorithm is causal in nature and thus can be implemented in a real vehicle controller as we do not require the drive cycle of the vehicle to be known in advance. Secondly, this technique is computationally more efficient than optimization-based techniques. On the flip side, though, the torque split results obtained from regression techniques are not as optimal as optimization-based techniques. The primary reason for the suboptimal nature of regression-based controllers is that the trend lines are generated over all drive cycles considered in the obtained data and not just for a particular drive cycle.

4.5.1 Implementation

The feature variables in this regression modeling are the ones that can be measured and/or estimated by the vehicle's controller and are listed below.

Before directly applying linear regression on these variables, it's important to determine whether we require all of the above listed variables for our analysis or whether a subset of these variables would give better results. If some of these features are related to each other, modeling with all features would give us low values of R-squared due to overfitting and hence is undesirable. Feature selection is a branch of statistics and machine learning that attempts to reduce the dimensions of a data set by selecting a subset of the features or transforming the set of features into a set of features with lower dimensions. Popular methods of feature transformation

are Principal Component Analysis (PCA), Factor Analysis and Nonnegative Matrix Factorization. Popular methods of feature selection techniques are regularization and correlation analysis. The regression modeling performed by Gupta [6] uses correlation analysis.

Using the data obtained by running dynamic programming simulations on 36 drive cycles mentioned in [6], a correlation matrix is developed that showcases the degree of linear relationship between the features mentioned in Table 4.1.

Table 4.1. The correlation coefficients between different observable features in the vehicle's controller.

	τ_v	τ_e	ω_e	τ_m	ω_m	v	a	SOC	N_g
τ_v	1	0.34	0.37	0.26	0.13	0.13	0.99	-0.01	0.07
τ_e	0.34	1	0.67	0.20	0.73	0.73	0.20	-0.10	0.66
ω_e	0.37	0.67	1	0.27	0.74	0.74	0.39	-0.08	0.72
τ_m	0.26	0.20	0.27	1	0.25	0.25	0.24	-0.02	0.27
ω_m	0.13	0.73	0.74	0.25	1	1	0.02	-0.17	0.96
a	0.13	0.73	0.74	0.25	1	1	0.02	-0.17	0.96
a	0.99	0.20	0.39	0.24	0.02	0.02	1	0.01	-0.02
SOC	-0.01	-0.10	-0.08	-0.02	-0.17	-0.17	0.01	1	-0.17
N_g	0.07	0.66	0.72	0.27	0.96	0.96	-0.02	-0.17	1

We can see in Table 4.1 that some features have a very high degree of correlation and hence not all of those features should be included in our regression analysis. The features that have a low degree of correlation are relatively independent of each other as far as linear relationships is concerned and only those features should be selected for better model performance. Hence, the variables chosen on the basis of the correlation matrix for the regression model are τ_v, τ_e, ω_e, τ_m, v and SOC.

Due to the complex nonlinear relationships between the selected features, a model that is just linear in the features would have a low R-squared value. Hence, nonlinear

relationships between the selected features should also be considered in the regression model. The nonlinear relationships between the features that were used in the regression model developed by Gupta [6] are listed in Table 4.2.

Table 4.2. The nonlinear terms used in the regression model.

Nonlinear relationship used	**Symbolic representation**
Quadratic	X_i^2
Cross-quadratic	$X_i X_j$
Exponential	e^{X_i} and $\frac{1}{1+e^{-X_i}}$
Square root	$\sqrt{\lvert X_i \rvert}$

Using a least-squares fit, a relationship between the features and the output (torque split) was obtained. Due to the large number of linear and non-linear relationships involved, the coefficients of the regression model have been included in Table 4.3 instead of including the whole equation here.

Table 4.3. Coefficients of all terms used in the regression model.

Variable	**Coefficient**
1	-8619.33
ω_e	-4007.67
SOC	68.97
τ_v	480.74
v	-1032.45
τ_e	-0.3584
τ_m	-443.66
$\omega_e SOC$	-2.80
$\omega_e \tau_v$	-0.489
$\omega_e \tau_m$	5.63

Table 4.3. Coefficients of all terms used in the regression model.

Variable	**Coefficient**
1	-8619.33
$SOC\tau_e$	3.60
$SOCv$	0.3273
$SOC\tau_m$	0.8816
$\tau_v v$	2.10
$\tau_v \tau_e$	1.36
$\tau_v \tau_m$	-1.93
$v\tau_e$	1.25
$v\tau_m$	1.84
$\tau_e \tau_m$	-1.77
$\omega_e \omega_m$	100.59
$SOCSOC$	31.68
vv	-37.66
$\tau_e \tau_e$	-0.7911
$\tau_m \tau_m$	-68.38
e^{ω_e}	428.42
e^{SOC}	-68.62
e^{τ_v}	-92.33
e^{v}	163.71
e^{τ_m}	132.59
$\sqrt{\lvert\omega_e\rvert}$	226.83
$\sqrt{\lvert v\rvert}$	9.6
$\sqrt{\lvert\tau_e\rvert}$	-0.7679
$\sqrt{\lvert\tau_v\rvert}$	-1.69
$\sqrt{\lvert\tau_m\rvert}$	0.5784

Table 4.3. Coefficients of all terms used in the regression model.

Variable	**Coefficient**
1	-8619.33
$\frac{1}{1+e^{-\omega_e}}$	12936.26
$\frac{1}{1+e^{-\tau_e}}$	-1532.453
$\frac{1}{1+e^{-v}}$	3376.72
$\frac{1}{1+e^{-\tau_m}}$	1241.87

The disadvantage of using correlation analysis for linear regression is that it only gauges the linear relationship between the involved variables. If two variables have a nonlinear relationship, then their correlation coefficient will be low. Yet, for better model performance, only one of them should be included in the list of predictor variables if the regression model is nonlinear in the features. Hence, advanced techniques like regularization will give better results than a simple correlation analysis.

4.5.2 Results

The results of regression modeling for the UDDS and the Highway FET drive cycle are shown in Figure 4.23 - Figure 4.28. For each drive cycle, a total of three plots are shown that depict the variations of the torque split, the SOC and the total power over time [29].

The motor energy consumption for the UDDS drive cycle is $2.2270 \times 10^6(J)$, the engine energy consumption is $7.4058 \times 10^6(J)$ and the total energy consumption is $9.6828 \times 10^6(J)$.

The motor energy consumption for the HWFET drive cycle is $7.5568 \times 10^7(J)$, the engine energy consumption is $1.1247 \times 10^7(J)$ and the total energy consumption is $1.2003 \times 10^7(J)$.

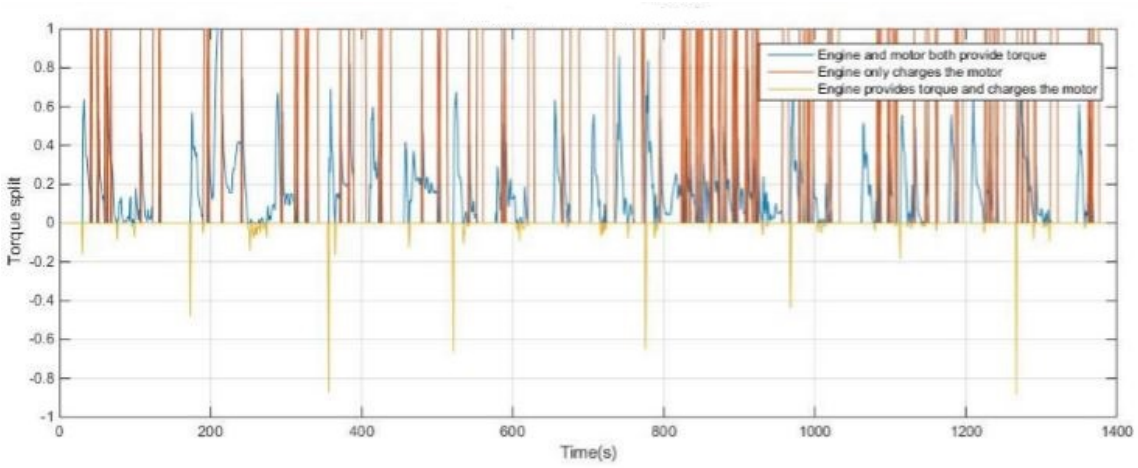

Figure 4.23. Torque split generated by the regression model for the Urban Dynamometer Driving Schedule drive cycle [29].

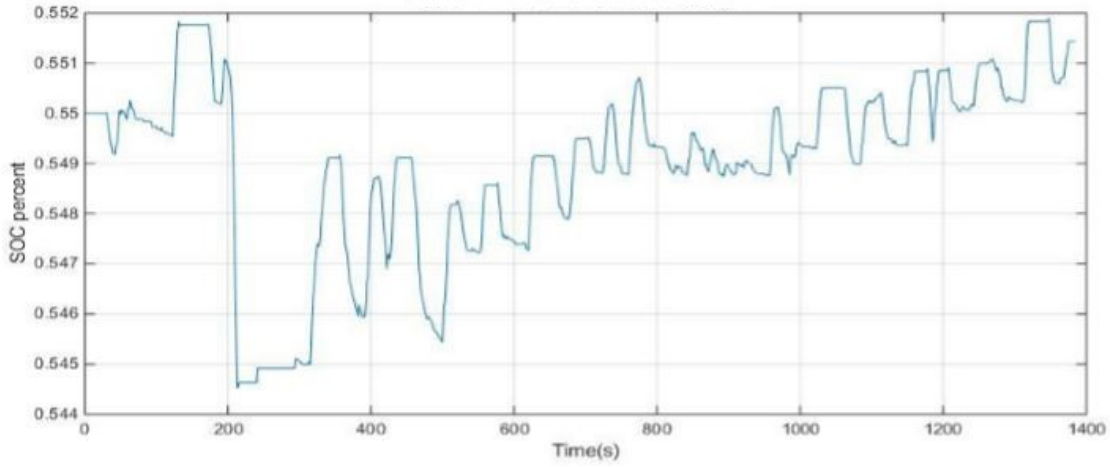

Figure 4.24. SOC variation generated by the regression model for the Urban Dynamometer Driving Schedule drive cycle [29].

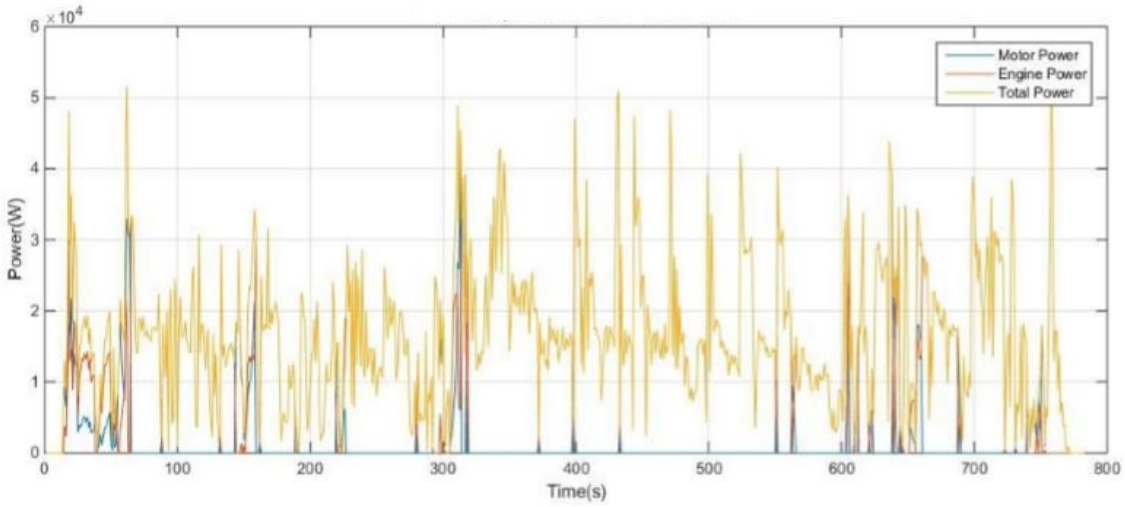

Figure 4.25. Regression model power required for the Urban Dynamometer Driving Schedule drive cycle [29].

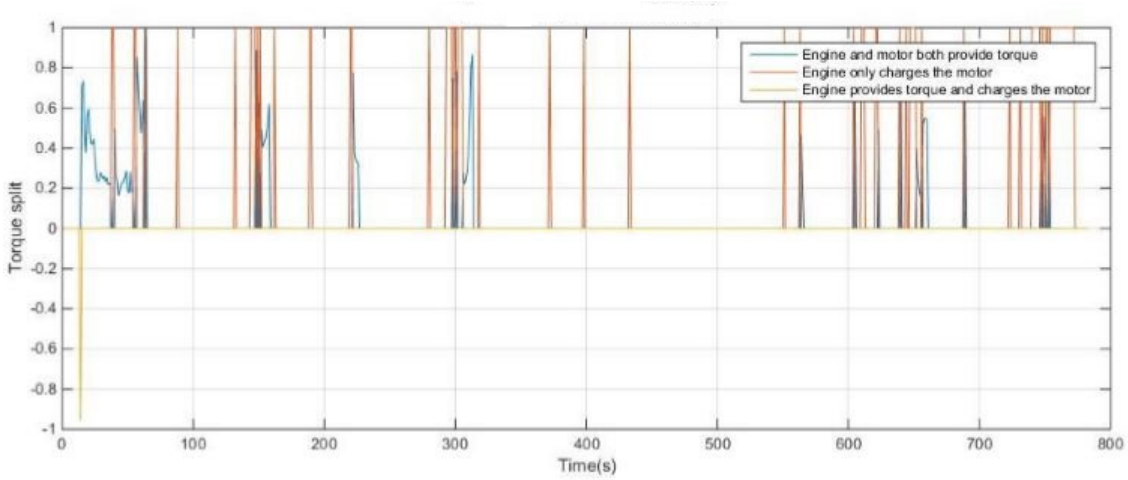

Figure 4.26. Torque split generated by the regression model for the Highway Fuel Economy Test drive cycle [29].

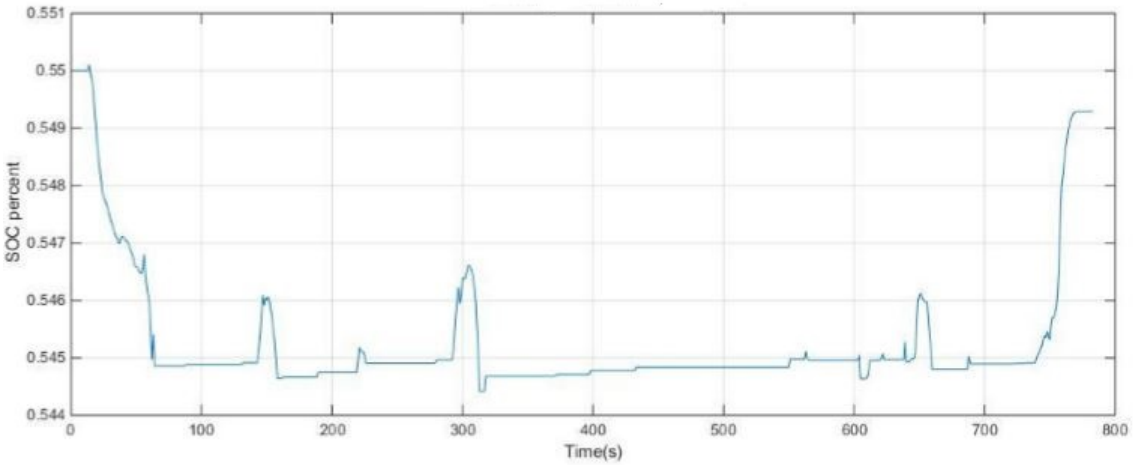

Figure 4.27. SOC variation generated by the regression model for the Highway Fuel Economy Test drive cycle [29].

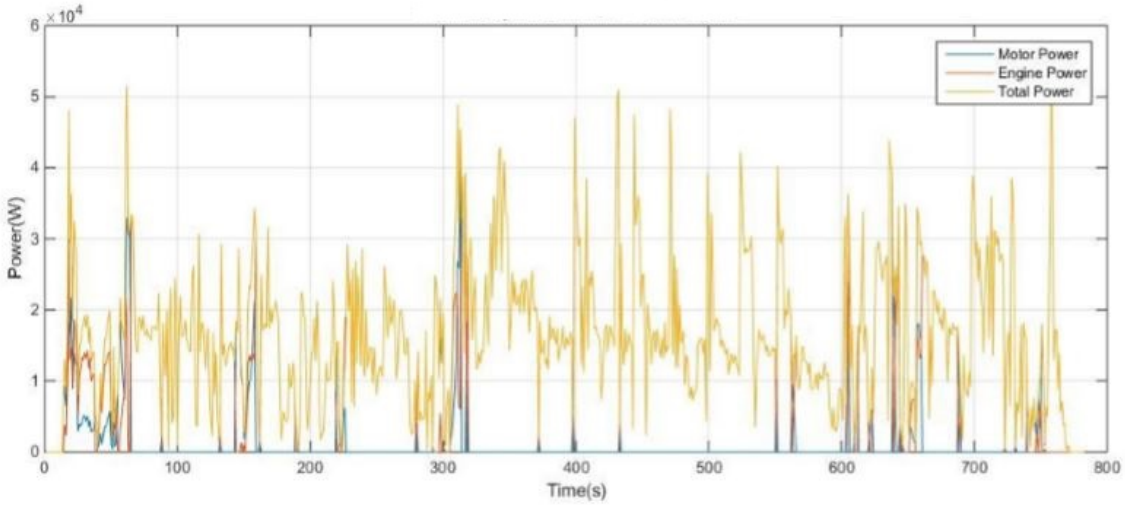

Figure 4.28. Regression model power required for the Highway Fuel Economy Test drive cycle [29].

The torque split is, in theory, a continuous function of time. In Figure 4.23 and Figure 4.26, the torque split plots are divided into 3 regions each. The values indicated by blue lines are representative of the blended mode, in which the total

power is supplied by both the engine and the motor. The values indicated by red lines are representative of the regeneration mode in which the total torque demand is negative and thus is used to charge the battery. In this mode, the engine is idle and the motor acts as a generator. Lastly, the yellow line represents the charging mode, in which the torque provided by the vehicle is more than what is required by the vehicle and the surplus torque is used to charge the battery through the motor which acts as a generator.

The SOC is also, in theory, a continuous function of time. Figure 4.24 and Figure 4.27 show the SOC variation with time for the UDDS and HWFET drive cycles, respectively. In both drive cycles, the SOC is maintained over the entire duration and the maximum deviation of the SOC from the target value of 0.55 does not exceed 0.006. Every drop in an SOC plot indicates that the battery charge is being consumed for powering the vehicle. The battery charge should be replenished later by either operating the vehicle in the regeneration mode or in the charging mode, which is reflected in the SOC plot by a rise in the SOC value.

Like the torque split and the SOC, the required power is also a continuous function of time. In Figure 4.25 and Figure 4.28, the power plots are divided into 3 regions each. The values indicated by blue lines represent the total power demand of the vehicle fulfilled by the motor and the values indicated by red lines represent the total power demand of the vehicle fulfilled by the engine. The plots are indicative of the fact that most of the power demand is fulfilled by the engine and any additional power demand is fulfilled by the motor. This can also be corroborated by the total motor energy consumption and the total engine energy consumption values for both drive cycles mentioned previously.

The plots described in this section depend exclusively on the regression equation. The coefficients of the regression equation depend on the number and nature of the drive cycles used for generating the equation. With reference to the theory of statistical learning, the DP results of these drive cycles are referred to as the training data. As mentioned before, the trend lines generated by regression analysis attempt

to make a least squares fit for all drive cycles used for its training and hence the results of testing the regression equation on any drive cycle will not exactly match the DP results of the same drive cycle. This is quite evident by comparing the plots in this section with the corresponding plots in the DP section. For the UDDS drive cycle, the shapes of the SOC plots for DP and regression are somewhat similar up to the first major dip at around 200 seconds, following which the SOC plots are significantly different. For the HWFET drive cycle, the shapes of the SOC plots for DP and regression are significantly different. The SOC plot for DP consists of a big crest whereas the SOC plot for regression consists of a big trough.

The plots in this section will change if the number of drive cycles used for training the regression model changes as doing that will change the regression equation. The plots will also change with the way in which the model is trained. A popular method of training a regression model is using k-fold cross validation, in which the training data is split into k groups. The model is trained k times. For each time, one group is considered as the test data and the remaining $k-1$ groups are considered as the test data. This way, each group works as testing data once and training data $k-1$ times. The regression analysis described in [6] doesn't describe whether the model was trained using k-fold cross validation, though.

4.6 Long Short Term Memory (LSTM) Modeling

A Recurrent Neural Network (RNN) is an artificial neural network that has the ability to classify, process and/or make predictions of sequential data. RNNs, however, suffer from the vanishing gradient problem, which makes it pretty challenging and time-consuming to train them [42] [43]. LSTM networks solve the vanishing gradient problem because they contain special units in their architecture that implement constant error flow due to which the networks can learn to bridge minimal time lags greater than 1000 time steps [44]. As time series data can contain lags of unknown duration, LSTM networks are useful in modeling them.

Figure 4.29 shows the structure of an LSTM cell [45].

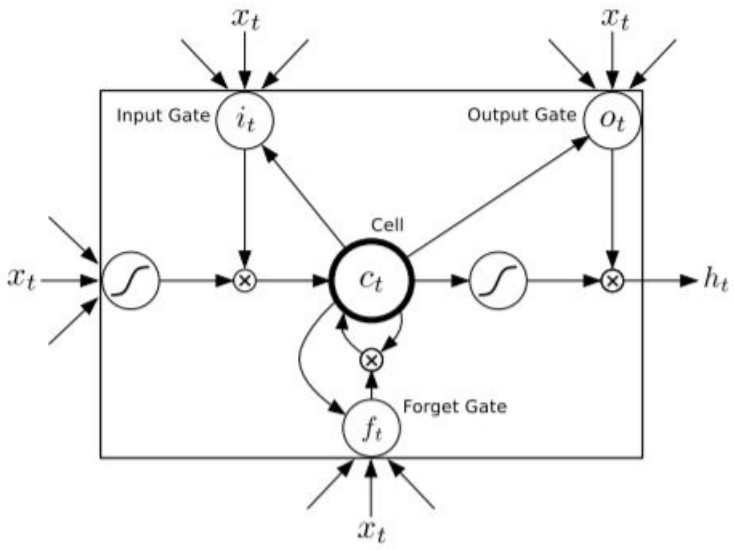

Figure 4.29. Structure of a long memory term memory cell [45].

The function of each gate in the LSTM cell works as follows [46]:

1. The cell gate c_t is the memory of the network and it transfers relative information all the way down to the sequence chain.

2. The job of the sigmoid activation function, which squishes values between 0 and 1, is to quantify the amount of information that should be discarded or forgotten. 0 implies that all the information should be forgotten and 1 implies that all the information should be retained.

3. The forget gate f_t decides what information should be thrown away or kept.

4. The job of the input gate is to update the state of the LSTM cell by processing the previous hidden state and current input.

5. The job of the output state is to decide what the next hidden state should be.

4.6.1 Implementation

The PHEV power management deep learning model was developed by Sun [29] and implemented using PyTorch, an open-source machine learning library developed by Facebook. The inputs to this model are the same vehicle features that were used in regression modeling based on their correlation analysis and the output of this network is a scalar of torque split. As these inputs themselves can be considered as features, there is no need for using convolutional neural networks to extract features from input data. Hence, the deep learning model consisted of only LSTM and fully connected layers as shown in Figure 4.30 [29]. The LSTM layer accepts the inputs and the fully connected (FC) layer outputs the torque split.

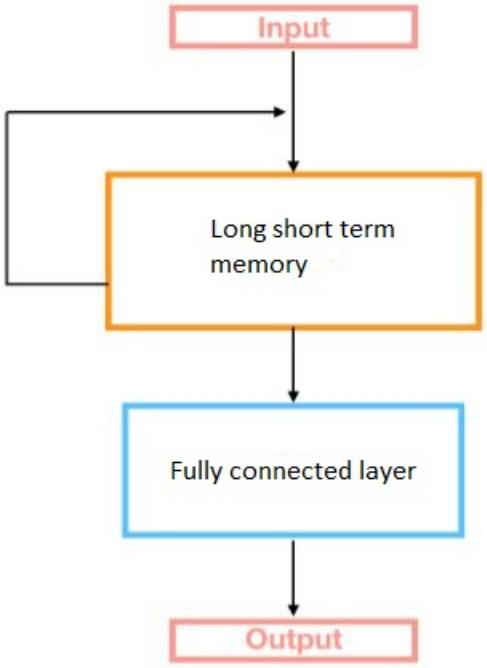

Figure 4.30. Deep learning model used for torque split prediction [29].

The data set for training and testing the deep learning model was generated by running the dynamic programming code [40] on 33 drive cycles mentioned in the Appendix. The Japan 10 Mode, Japan 15 Mode and Japan 1015 Mode drive cycles

did not have enough data points for the dynamic programming code to run. Of the remaining 33, the Economic Commission of Europe Drive Cycle and the West Virginia University Suburban Drive Cycle were used for model validation and the rest were used for training the model.

The results of training and validating any deep learning model depend on how its hyperparameters are initialized. Significant research is being conducted all over the world on effective algorithmic tuning of hyperparameters. Currently, however, the most common way of tuning a model's hyperparameters is trial-and-error. The hyperparameters of the PHEV power management deep learning model are the hidden dimensions, learning rate, momentum and epoch number. Also, a central part of training a deep learning model is the choice of a loss function whose gradient is used to tune the model's features such as weights and thresholds. The mean squared error loss and the L1 norm loss are the most commonly used loss functions. Several experiments were run for different values of these hyperparameters and different loss function [29]. Based on the results of these experiments, the values of the hyperparameters were initialized as shown in Table 4.4.

Table 4.4. Hyperparameter initializations in the deep learning model.

Hyperparameter	**Initialized value**
Hidden Dimension	64
Loss function	Mean squared error loss
Learning rate	0.01
Momentum	0.9
Epoch number	160

4.6.2 Results

The results of the LSTM model for the UDDS and the Highway FET drive cycle are shown in Figure 4.31 - Figure 4.36. For each drive cycle, a total of three plots are

shown which depict the variations of the torque split, the SOC and the total power over time [29].

The total energy consumption for the UDDS drive cycle is $7.6167 \times 10^6(J)$ and that for the HWFET drive cycle is $1.2192 \times 10^7(J)$.

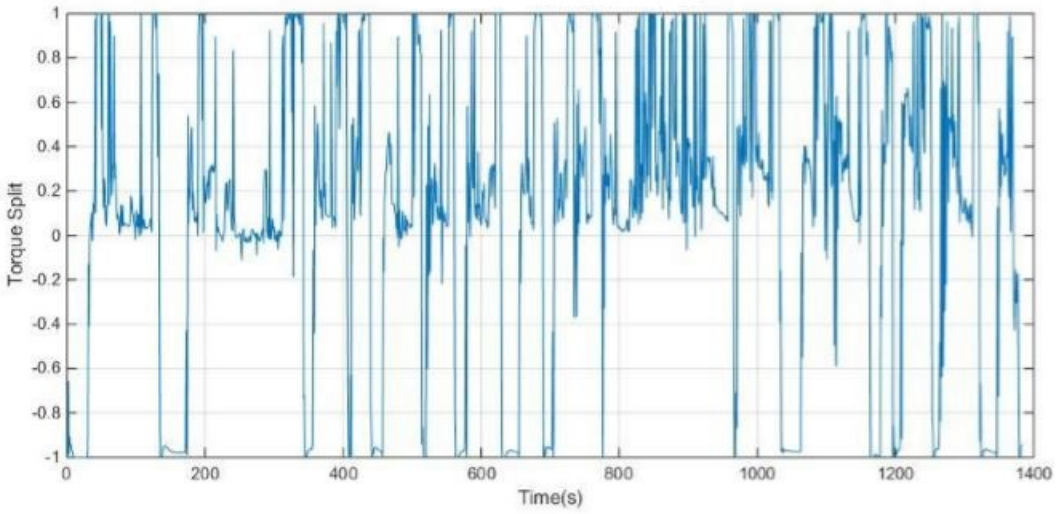

Figure 4.31. Torque split generated by the neural network model for the Urban Dynamometer Driving Schedule drive cycle [29].

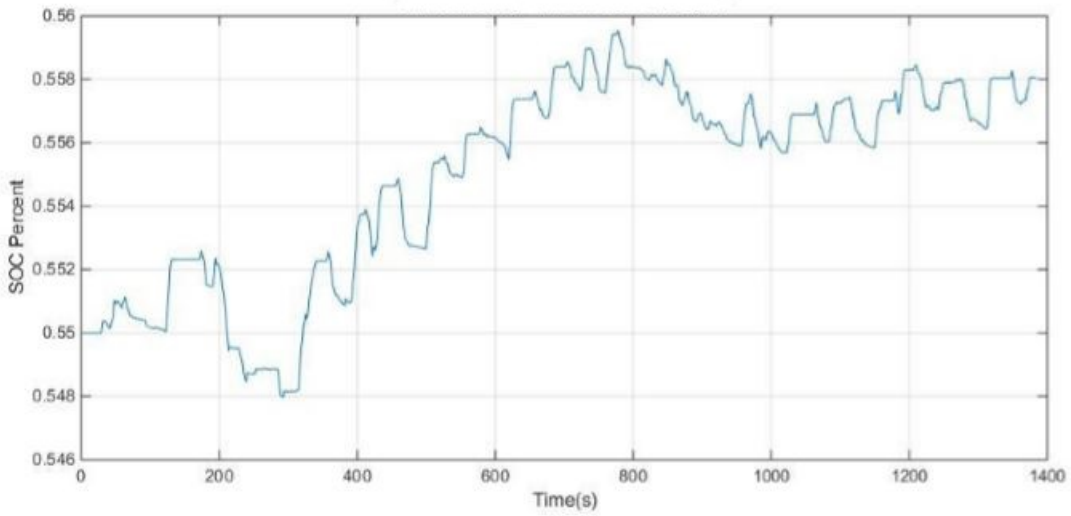

Figure 4.32. SOC variation generated by the neural network model for the Urban Dynamometer Driving Schedule drive cycle [29].

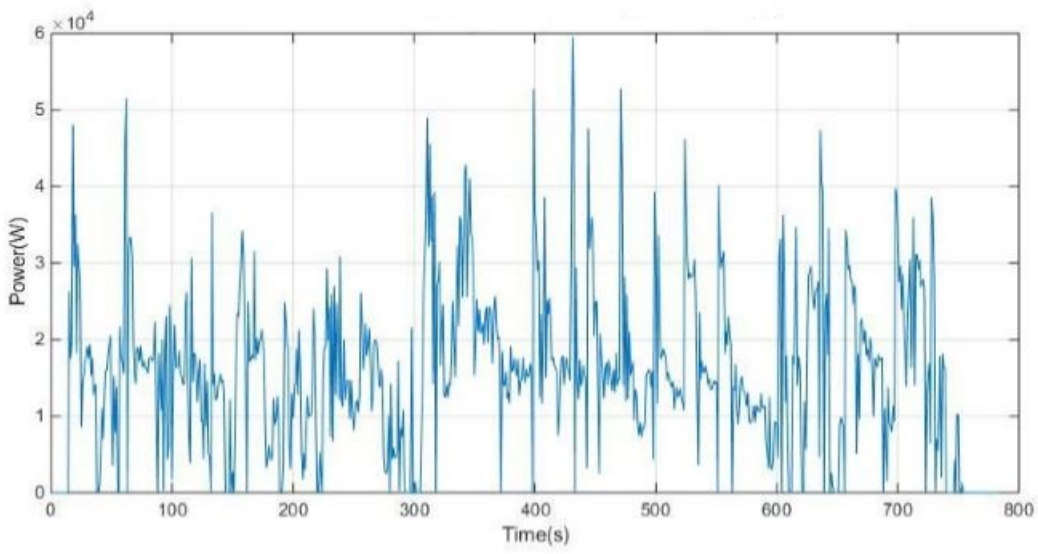

Figure 4.33. Neural network model power required for the Urban Dynamometer Driving Schedule drive cycle [29].

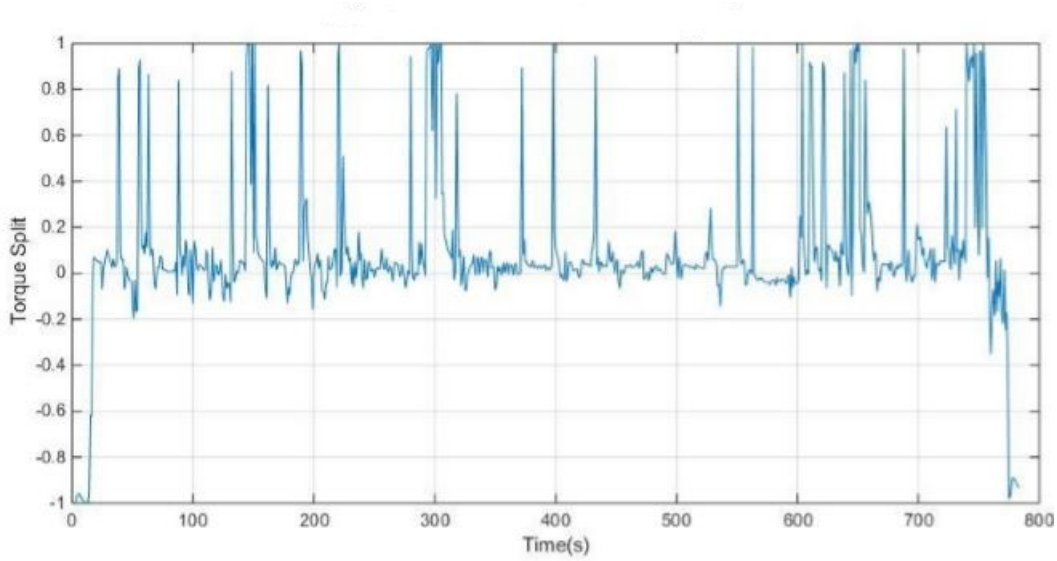

Figure 4.34. Torque split generated by the neural network model for the Highway Fuel Economy Test drive cycle [29].

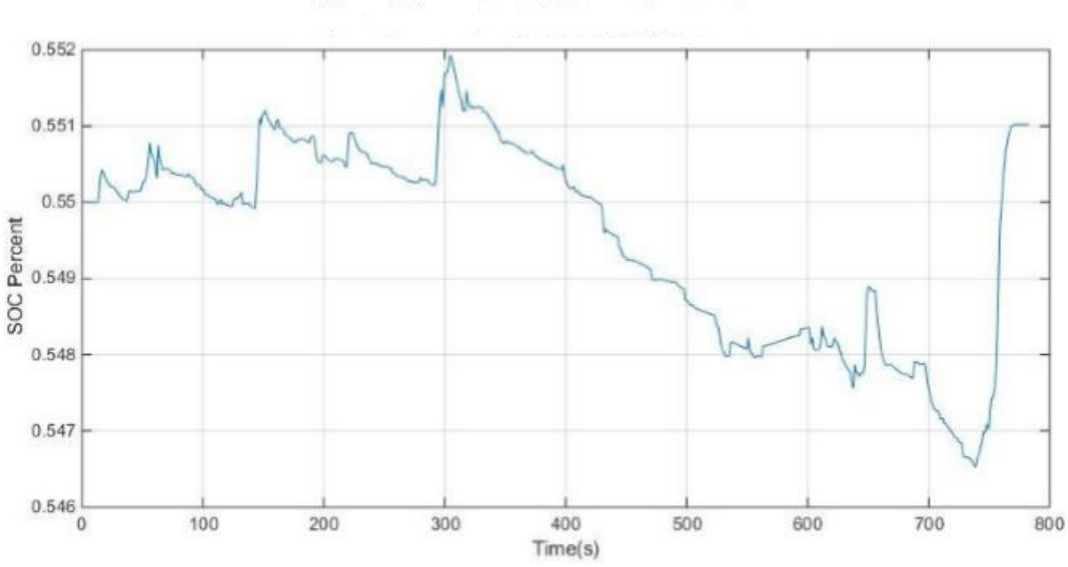

Figure 4.35. SOC variation generated by the neural network model for the Highway Fuel Economy Test drive cycle [29].

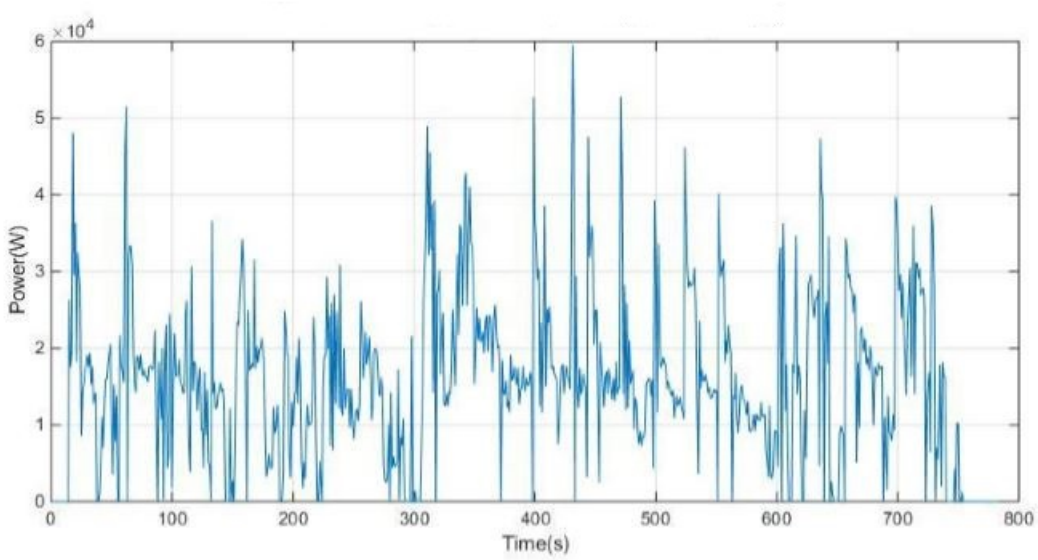

Figure 4.36. Neural network model power required for the Highway Fuel Economy Test drive cycle [29].

Similar to regression analysis, the results of the LSTM model depend on the drive cycles used for training the model. Due to the larger number of parameters involved, the LSTM can model nonlinear relationships between the input variables and the torque split values better than regression analysis, in general. This fact is evident from the plots described in this section. The SOC plot for the UDDS drive cycle in Figure 4.32 resembles the DP plot in Figure 4.5 more than the regression plot in Figure 4.24 does. These results also depend on the hyperparameters mentioned in Table 4.4. Different values of the hyperparameters will lead to different plots. Training deep learning models takes a lot of time. If time is not a constraint, the performance of this model can be improved by stopping the training process when the average change in the value of the objective function over certain epochs is less than some tolerance value, rather than manually stopping training after some fixed number of epochs. Better performance means closer resemblance to DP results.

Furthermore, reference [29] states that the LSTM model was tested on EPA LA92 drive cycle and West Virginia University Suburban drive cycle and the rest of the

33 drive cycles were used for training. The UDDS and the HWFET drive cycles are also a part of these 33 drive cycles. Therefore, the results of the LSTM model for the UDDS and HWFET drive cycles as described in [29] and plotted in this section are not true indicators of the model's performance as comparing the results of LSTM model with the DP model for these drive cycles is merely akin to validating the model. For finding out how the model actually performs, either of the following two methods should have been employed. Firstly, the UDDS and HWFET drive cycle data should have been a part of the testing data and not the training data. Or secondly, with the training data used in [29], the DP results of the EPA LA92 drive cycle and West Virginia University Suburban drive cycle should be compared with the LSTM results of the same drive cycles.

5. GENETIC ALGORITHM CONTROL STRATEGY

This chapter describes the Genetic Algorithm (GA) control strategy, which is the subject of this thesis. Some of the initialization strategies in genetic algorithms, like the types of penalty functions and the choice of population size, were borrowed from the AAE 550 course offered by the School of Aeronautics and Astronautics, Purdue University, and taught by Professor William Crossley. The theory of genetic algorithms has been used by the author of this thesis to develop a power management strategy for a PHEV.

5.1 Development and Implementation

The Global Optimization Toolbox in MATLAB provides a set of functions for selection, initialization, crossover and mutation to solve standard optimization problems using genetic algorithms. The Toolbox also has a provision for defining custom data type optimization problems where the population can be of any data structure such as an array or a cell. MATLAB does not provide the essential functions for solving such custom optimization problems; the user has to develop these functions from scratch. MATLAB uses these functions to solve custom genetic algorithm optimization problems using the *ga* function. The rest of this section explains how the theory of genetic algorithms was used for developing a power management strategy for a PHEV.

The most important aspect of solving an optimization problem using genetic algorithms is the way the input variables are represented so that the corresponding mutation and crossover functions can be developed accordingly. The binary equiva-

lent is one of the most common representations. For instance, suppose we're trying to solve the following one-dimensional optimization problem using genetic algorithms,

$$x^* = \min f(x). \tag{5.1}$$

If the binary representation has been chosen for this problem and one of the genomes in a certain population is, say, 45, then this genome is represented as follows,

$$x = 45_{10} = 00101101_2. \tag{5.2}$$

The number of zeros preceding the most significant 1 in the binary representation depends on the genomes in earlier generations. In general, larger number of zeros preceding the most significant 1 implies that a larger input space is being searched for finding the optimal solution. This is beneficial for the algorithm's performance as it promotes diversity in the population.

With reference to genetic algorithms, the usage of binary representation essentially means that a potential solution is transformed into a vector whose entries can only be 0 or 1. Each entry of this vector is called a gene and the crossover and mutation operations are performed on these genes. Due to the nature of these crossover and mutation operations, it's convenient to represent each potential solution in terms of a vector.

In the case of developing a power management strategy for a PHEV, the goal is to find the time variation of torque split for minimizing an objective function. As the optimal control problem is to be solved on a computer, it has been discretized in time, states and inputs. Thus, for a particular drive cycle, a potential solution π of this optimal control problem is a vector of torque split values u whose length is equal to the number of time steps of that drive cycle. For instance, the UDDS drive cycle lasts for 1384 seconds. If the time discretization is 1 second, the UDDS drive cycle data set is an array of 1384 entries of speed. Thus, a potential solution of the optimal control problem for the UDDS drive cycle can be represented as follows,

$$\pi = \{u_0, u_1, u_2, ..., u_{1384}\}, \tag{5.3}$$

where $u_0, u_1, u_2, ..., u_{1384} \in [-1, 1]$.

Comparing with the binary representation in Equation 5.2 π is a genome and $u_0, u_1, u_2, ..., u_{1384}$ are its genes which can take any value between -1 and +1 (as opposed to the binary representation where the array entries can only take 0 or 1). In MATLAB, the genome can be represented using a vector.

Now that we have established how genomes will be represented in this optimization problem, the next step is establishing a population of candidate genomes. There is a trade-off between the number of genomes in a population, also called population size, and the run time of the genetic algorithm. If the population size is high, a larger input space is covered in each generation which leads to better potential solutions in each generation. However, the run time of the algorithm also increases as it has to calculate the objective function for each genome in the population. As a rule of thumb, the population size should be at least four times the number of genes in a genome of that population. In the case of UDDS drive cycle, for instance, the population size should be at least $1384 \times 4 = 5536$. Thus, the population consists of 5536 vectors of torque split values each of length 1384.

In MATLAB, an intuitive way of representating a population is by using a matrix of size $4N \times N$ where each row represents a genome of length N and the number of rows $4N$ represents the number of genomes in the population. For instance, the population matrix in case of the UDDS drive cycle will be of size 5536×1384. However, developing the essential functions that drive the genetic algorithm becomes complex due to the simple but cumbersome indexing scheme of a matrix. A cell array is a better data structure for representing the population. A cell array is a data type with indexed data containers called cells, where each cell can hold any type of data. A population in MATLAB can be a cell array of size $4N$, where each of these $4N$ cells holds a genome/vector of N torque split values.

While the genetic algorithm will generate a population of genomes every generation, it is important to initialize a population for the first generation. The algorithm for initializing a population is simple. Firstly, we create a vector a of values that

the torque split can take. This vector is simply a discretization of input space of the optimal control problem and thus contains values between -1 and 1. Secondly, we create an empty cell array of size $4N$. In each cell of this array, we enter a vector of length N that randomly takes values from a. The MATLAB function *datasample* can randomly sample N values from a with replacement. Among other factors, the number of values between -1 and +1 that a contains decides the performance of the genetic algorithm. Larger number of values implies finer discretization of the gene space and better performance of the algorithm in terms of accuracy. On the other hand, the number of generations required for the algorithm to converge also increases with finer discretization. Hence, there is a trade-off between accuracy and run time of the genetic algorithm.

Another important step in designing this genetic algorithm is developing an appropriate fitness function to be optimized. In MATLAB, this fitness function is developed as a separate MATLAB function and passed to the *ga* function as a function handle. A function handle is a data type that stores an association to a MATLAB function. Typical uses of function handles are passing mathematical functions to operation functions like differentiation, integration or optimization (like *ga*).

This technique works well if the optimization problem is unconstrained as a genetic algorithm is primarily meant for solving unconstrained optimization problems. Constraints in an optimization problem, if any, should be handled separately using some novel techniques. For standard optimization problems, the *ga* function in MATLAB automatically takes care of these constraints by passing a separate function handle that contains these constraints. For custom data type optimization problems, however, MATLAB doesn't handle input constraints and it's the responsibility of the user to handle them. The task of handling constraints in an optimization problem is what makes the process of developing the fitness function challenging. While some progress has been made in this area, there is still a long way to go.

The most popular method of handling constraints in an optimization problem is adding them to the objective function as penalty functions. By doing this, the

constrained optimization problem is effectively converted into an unconstrained one. Hence, the pseudo-unconstrained objective function $\phi(x)$ can be expressed as follows,

$$\phi(x) = f(x) + r_p \sum_{j=1}^{n_{con}} P_j(x), \tag{5.4}$$

where $f(x)$ is the unconstrained objective function, $P_j(x)$, $j = 1, 2, 3, ..., n_{con}$ are the constraints on the input and r_p is the penalty multiplier.

Calculus-based optimization techniques use exterior quadratic penalty functions of the form

$$P_j(x) = c_j(\max[0, g_j])^2 \tag{5.5}$$

because the quadratic form is first-order continuous which makes it differentiable. Here, $g_j(x)$ is the value of the constraint violation. However, as there are no derivatives involved in a genetic algorithm, we have freedom to use other forms of penalty functions like linear or piecewise linear. The advantage of these penalty functions is the ease of computation, especially when the constraint violations are extremely large or extremely small.

The most commonly used penalty functions in a genetic algorithm-based optimization are exterior quadratic as shown in Equation 5.5, exterior linear and exterior step-linear. The exterior linear penalty function is of the form

$$P_j(x) = c_j(\max[0, g_j]) \tag{5.6}$$

and the exterior step-linear penalty function is of the form

$$P_j(x) = \begin{cases} 0, & g_j(x) \leq 0 \\ c_j[1 + g_j(x)], & \text{else.} \end{cases} \tag{5.7}$$

Figure 5.1 shows the different types of penalties as functions of constraint violation, defined as the departure from a limiting value. For instance, if a variable x is supposed to be less than 5, then the constraint violation is $x - 5$ if x is greater than 5 and zero otherwise. The slopes of all curves depend on the constraint coefficient c_j. Higher c_j

leads to more penalty for a particular value of constraint violation. It is clear from the figure that exterior step-linear is the best of the three because of its robustness in penalizing very small constraint violations. For very small values of $g(x)$ (or very small values of constraint violation), the corresponding penalty function values are greater than 2. The value of the step depends on how much we want to penalize constraint violations. For critical optimization problems where even a small constraint violation would be detrimental to the design, the step size should be larger.

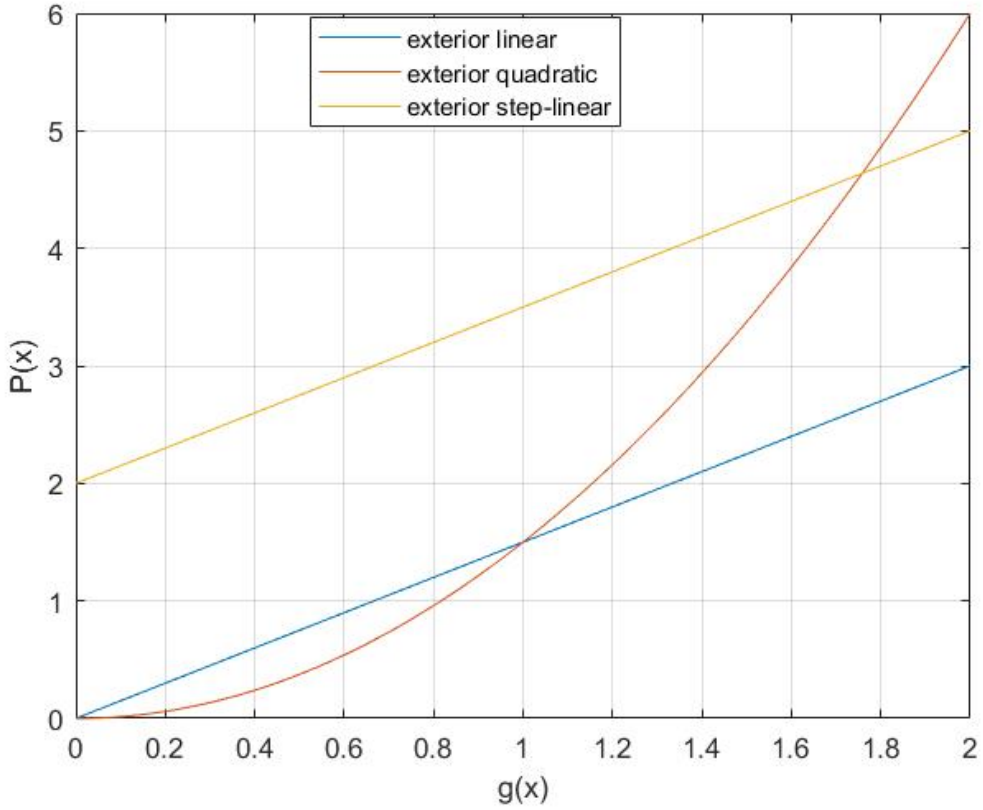

Figure 5.1. Different types of penalty functions.

Setting the values of the penalty multiplier r_p and the constraint coefficients c_j is a challenging task. There is no algorithmic way of setting the values of these parameters; it's a trial-and-error process and intimately dependent on the problem domain.

Having engineering experience and intuition can help in making a calculated guess but it can only take the user so far.

For developing a power management strategy of a PHEV, the total quantity of greenhouse gas emissions due to burning the fuel is the function to be minimized if the vehicle is operating in the charge-sustaining mode. Unlike dynamic programming, however, some constraints in the form of penalty functions should be added to the objective function. Dynamic programming attempts to minimize the objective function in a reverse sequential fashion using Bellman's principle of optimality. There is provision in the algorithm for setting the initial and the final SOC, and constraints on the SOC over the entire drive cycle. Genetic algorithm, on the other hand, attempts to minimize the objective function by considering the entire drive cycle all at once and there is no provision to set any sort of constraints in the algorithm. As far as genetic algorithm is concerned, the goal is to minimize the total greenhouse gas emissions which is possible only if the battery SOC is depleted. Hence, the SOC constraints must be manually set in the form of penalty functions in genetic algorithm. The initial SOC can be included in the objective function itself. Penalty functions should be externally added to the objective function for penalizing the deviation of SOC from the target value during the duration of the drive cycle and deviation from the final SOC. Thus, the general form of the fitness function of the genetic algorithm is given by

$$f(\pi) = \sum_{i=1}^{n} (J(u_i) + c_i |SOC_i - SOC_{target}|) + P(SOC_n), \tag{5.8}$$

$$P(SOC_n) = \begin{cases} 0, & SOC_n = SOC_{target} \\ r_p(1 + |SOC_n - SOC_{target}|), & SOC_n \neq SOC_{target} \end{cases} \tag{5.9}$$

where π is given by Equation 5.3, $J(u_i)$ is the greenhouse gas emission at time step i and $\sum_{i=1}^{n}(J(u_i))$ is the total greenhouse gas emission over the entire drive cycle of time duration n. SOC_i is the value of the SOC at time step i, SOC_{target} is the target value of the SOC for the charge-sustaining mode and $c_i|SOC_i - SOC_{target}|$ is the penalty function which penalizes the deviation of the SOC from the target value

at time step i. As we wish to maintain the value of the SOC near the target value at all time steps, we add the deviation penalty function for all time steps.

It was found after extensive experimentation with many penalty functions and constraint coefficients c_j that the genetic algorithm tries very hard to deplete the battery SOC, especially during the later stages of the drive cycle. A possible way to counter this situation, which is implemented in this thesis, is to make c_i an increasing function of the time step instead of a constant value so that the penalty for SOC deviation in the later stages of the drive cycle also increases in value. Additionally, as the final SOC value should be equal to the initial SOC value, it is also prudent to heavily penalize the deviation of the final SOC value from the target value using the exterior step-linear penalty function $P(SOC_n)$. These ideas are explored in Section 5.2.

After a suitable fitness function is defined for the optimization problem, the next step is choosing a selection function, which selects parents from the current population based on the fitness values of the population's genomes. Of the several selection functions available in the literature and described in Section 2.3, the tournament selection is considered to be the best selection function. In this thesis, the tournament selection function has been used for selecting parents. Any type of selection function depends only on the fitness values of a population's genomes which, irrespective of the data structure of the genome, is a real number. Hence, the tournament selection function (or any selection function, for that matter) available in MATLAB can be used for custom data type optimization.

Child genomes are produced from parent genomes by mutation and crossover functions. The crossover phenomenon involves combining the genes of a pair of parents. Of the various crossover functions mentioned in section 2.3, the uniform crossover and single point crossover are popular. For creating a uniform crossover torque split child, we initialize the child torque split vector of gene length equal to the gene lengths of the parent torque split vectors. Let's name the parent genomes as p1 and p2. We iterate over each element of this child vector. In every iteration, we generate a random

number between 0 and 1. If the random number generated is greater than 0.5, the gene from p1 is passed to the child at the current index. If it's less than 0.5, the gene from p2 is passed to the child at the current index. This random number generation is akin to flipping a fair coin where any value greater than 0.5 can be considered as heads and any value less than 0.5 can be considered as tails (or the other way round). Thus, we can see that in uniform generation, there is randomness involved in generation of every gene of the child genome.

For creating a single point crossover torque split child, on the other hand, we generate a random number M between 1 and the length of the parent genomes. Then, we select the first M genes from p1 and the last $N - M$ genes from p2 and combine them sequentially to form a child genome.

For creating a population of child genomes using crossover in MATLAB, we initialize a cell array of size equal to the number of children we wish to generate using crossover. We iterate over each cell of this cell array. For every iteration of the cell array, we select two parents from the parent pool generated using the tournament selection function, apply the uniform or single point crossover function described in the last paragraph to generate the crossover genome, and store it in the cell.

To promote diversity in the search process and to avoid local minima, it's also recommended to use a mutation function that changes a parent's gene at random to generate a mutated child genome. Two ways of mutating a genome are discussed in this thesis. To create a mutated torque split child, we first choose a torque split parent to be mutated from the parent pool generated by the tournament selection function. The first technique involves using the MATLAB function *randi* to generate a random integer between 1 and the length of the parent genome. The generated random integer is the index of the parent genome that will be mutated. The gene at this index is replaced by a random torque split value. From the discretization vector a described during the development of the population initialization function, the MATLAB function *datasample* can be used to sample a torque split value that replaces the original number at the randomly generated index. The second technique

involves generating a random number M between 1 and the length of the genome to be mutated. Then, we swap the first M genes and last $N - M$ genes to generate the mutated child.

5.2 Results

The genetic algorithm power management control strategy of the vehicle model developed in Section 4.1 was implemented in MATLAB. Separate functions for population initialization, crossover and mutation were developed as described in the last section, which were then passed as function handles to the main *ga* function. Additionally, a function was developed that plots the best SOC results for a given population. These plots change with each generation and the user sees the population evolve towards the optimal solution like an animated video. All experiments were conducted on a standard Windows computer that contains an 8th Generation Intel Core i5 Processor. This processor has 4 cores and only one core is employed for computations by default. The Parallel Computing Toolbox from MATLAB was used for speeding up computations. In the standard method of computation, only one genome's fitness value is computed at a time. Using parallel processing, several fitness value computations can be performed simultaneously. For a processor containing 4 cores, a maximum of 12 workers can be generated in a parallel pool, thus decreasing the computation time by 12 times.

Due to the high computing power required by the genetic algorithm, the power management strategy was first tested on the first four cycles of the Central Business District (CBD) drive cycle shown in Figure 5.2.

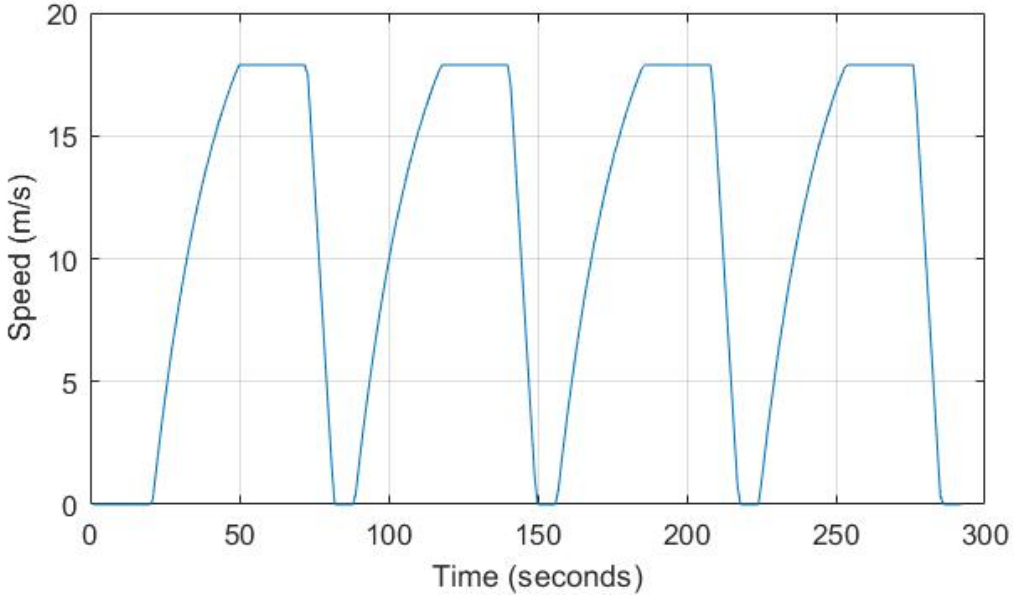

Figure 5.2. Speed profile of a vehicle for the central business district drive cycle [47].

Figure 5.3 shows the SOC variation for the CBD drive cycle if no constraints pertaining to the charge-sustaining mode are used in the genetic algorithm. The initial SOC was set to be 0.55.

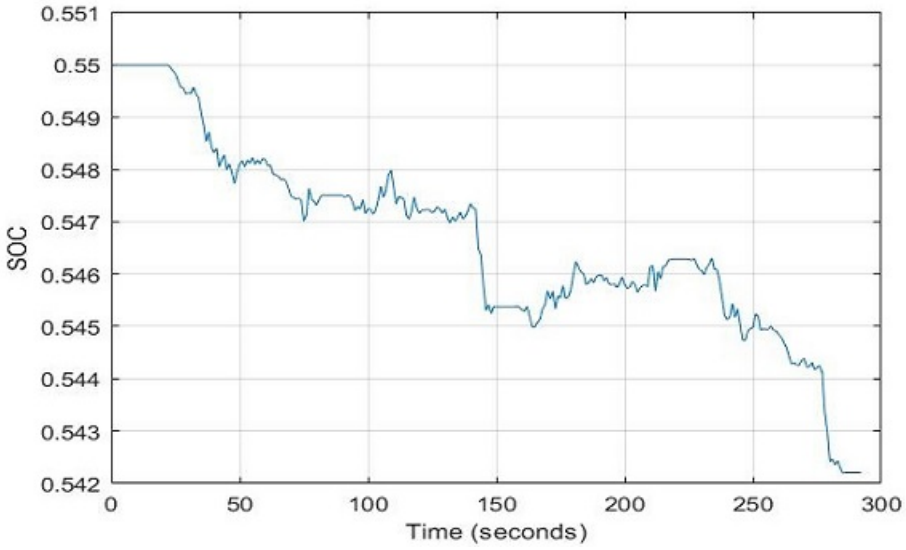

Figure 5.3. SOC plot for genetic algorithm without charge-sustaining constraints.

Without the charge-sustaining constraints on the objective function, the battery SOC is depleted for minimum greenhouse gas emissions. The decrease in SOC is not significant in this drive cycle because of its smaller time duration, but it is significant for larger duration drive cycles like UDDS.

Figure 5.4 shows the SOC variation for the CBD drive cycle with the constraints as defined in Equation 5.8. To counteract the *strength* of genetic algorithm to deplete the SOC in the later stages of the drive cycle, several non-decreasing functions for c_i were used. Figure 5.4 shows the effect of these c_i's on the SOC. In this experiment, the value of r_p was $5,000,000$.

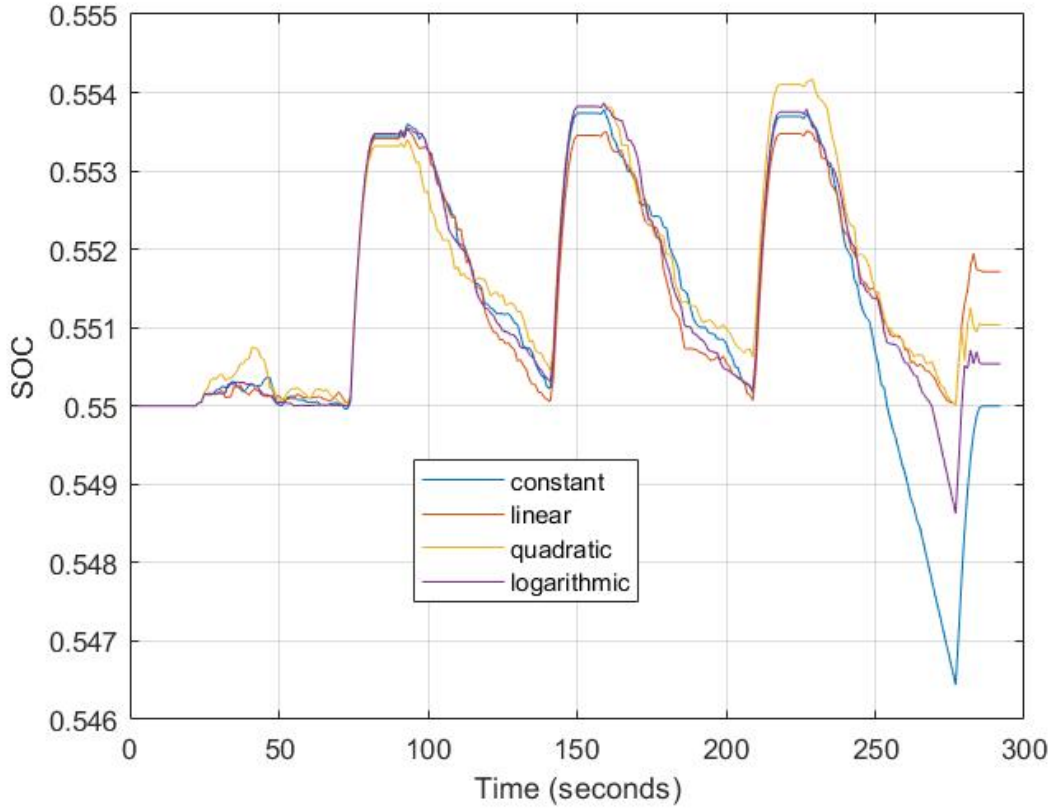

Figure 5.4. SOC plot for genetic algorithm with charge-sustaining constraints.

The following set of equations describe the different c_i's used for this analysis:

$$c_{i,constant} = 100, \tag{5.10}$$

$$c_{i,linear} = 100i, \tag{5.11}$$

$$c_{i,quadratic} = 100i^2, \tag{5.12}$$

$$c_{i,logarithmic} = 100log_e(i). \tag{5.13}$$

From the results shown in Figure 5.4, it is hypothesized that using a constant value of c_i is the worst option among the lot because of the large amount of dip in the SOC near the end of the drive cycle. Using an increasing function helps to improve the algorithm's performance. However, the rate of change of the increasing function also plays an important role in the performance of the algorithm. A simple linear or quadratic function doesn't lead to any dip in the SOC but the final SOC ends up being higher than the initial SOC, which means that more fuel is used to increase the final SOC. The natural logarithmic function is a trade-off between the SOC dip at the end of the drive cycle and the difference between the final and initial values of the SOC. The rest of the experiments performed in this thesis used the natural logarithmic function for c_i.

The next set of experiments was performed for studying the effect of varying r_p on the SOC results. Figure 5.5 shows the results of this set. Here, the natural logarithmic function was used for c_i, the crossover function used was uniform crossover, the mutation function used was single point mutation and the crossover ratio, defined as the ratio of the number of genomes in a parent pool that undergo crossover and the number of genomes that undergo mutation, was 0.9.

Figure 5.5 shows that the plots for different sets of final value constraints are almost the same, except for some shifts in the second crest and the final SOC value. It seems that both of these plot characteristics slide down up to $r_p = 50,000$ and then start sliding upwards for $r_p > 50,000$. For the rest of the experiments, though, the value of r_p being used was 5,000,000.

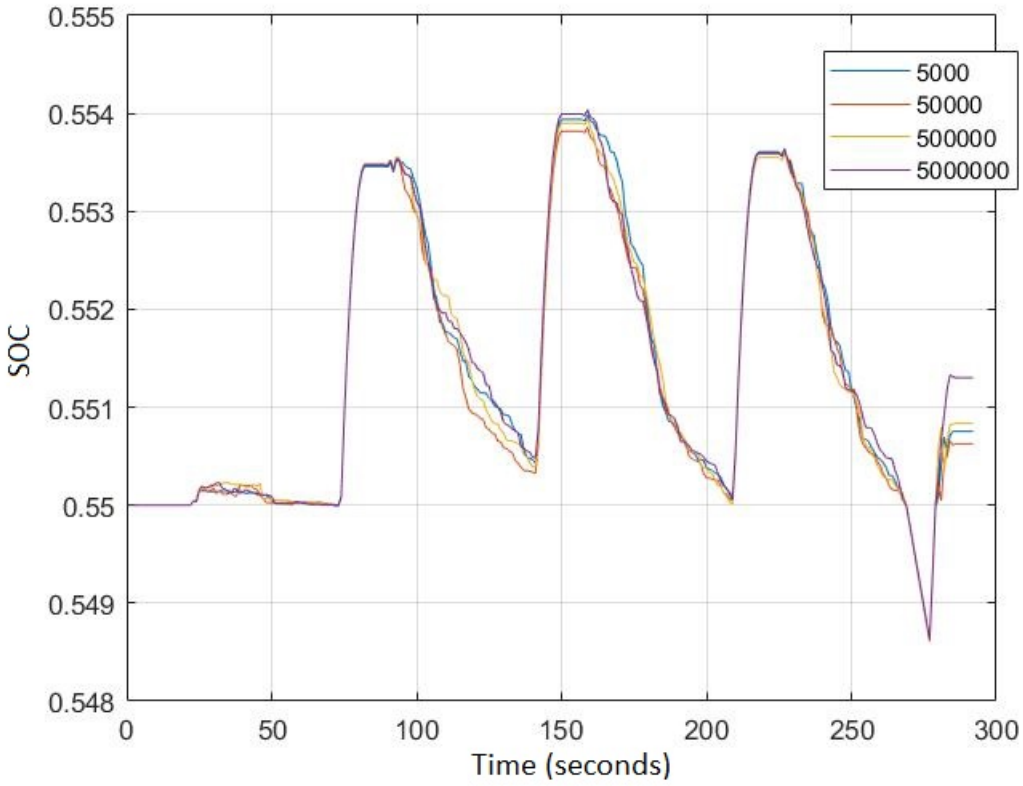

Figure 5.5. SOC plot for genetic algorithm with different final SOC constraints.

The next set of experiments was performed on the choice of the crossover function. Figure 5.6 shows the effect of the crossover function on the SOC. Here, the value of the crossover ratio was 0.9, the mutation function used was single point mutation and the value of r_p was $5,000,000$.

The choice of the crossover function doesn't seem to make a major difference in the SOC plot. In fact, the shapes of both plots are the same, with some minor vertical shifts at some places. The difference in the plots is more noticeable in a relative sense at the second crest of the general shape. Theoretically speaking, as the shape of the drive cycle as shown in Figure 5.2 is the same at all crests, the shape of the SOC plot should also be the same at all crests. By looking at this plot, it seems that single point crossover is better than uniform crossover for this power management strategy

because the crests in the SOC plot for the single point crossover are more similar than those for the uniform crossover.

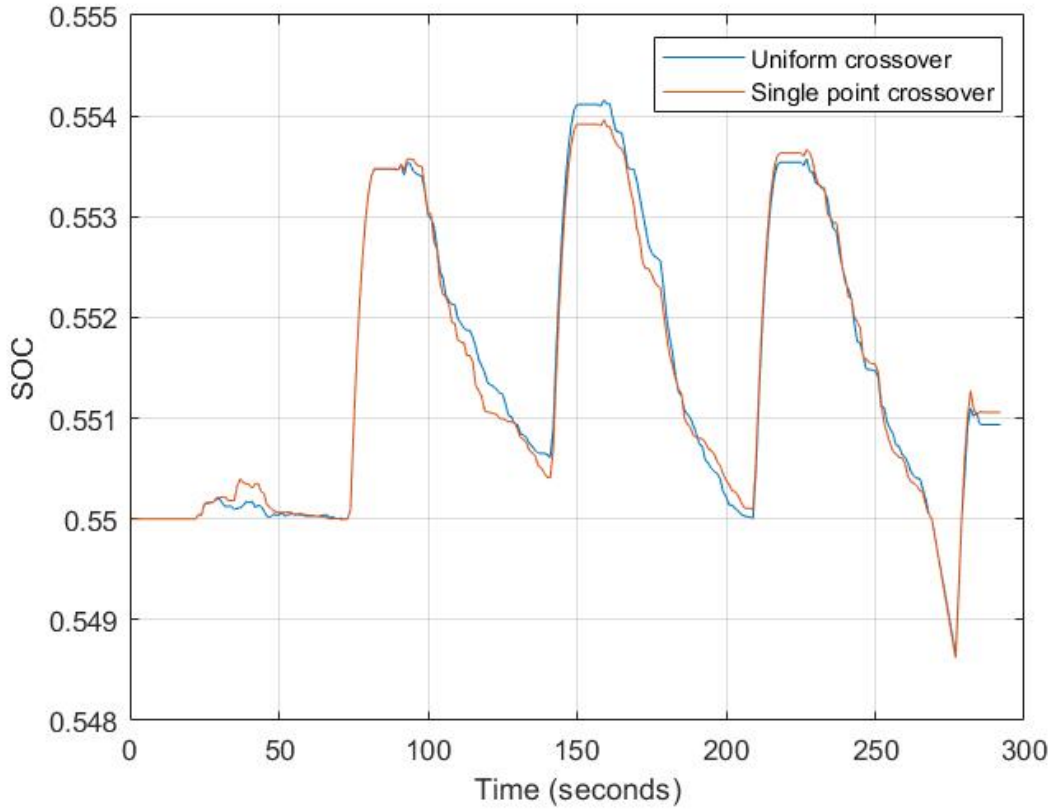

Figure 5.6. SOC plot for genetic algorithm with different crossover functions.

However, the difference in both plots is not large enough to discard uniform crossover entirely. The rest of the experiments in this section were performed using uniform crossover.

Figure 5.7 shows the effect of the choice of mutation function on the SOC. Here, the value of the crossover ratio was 0.9 and the crossover function used was uniform crossover.

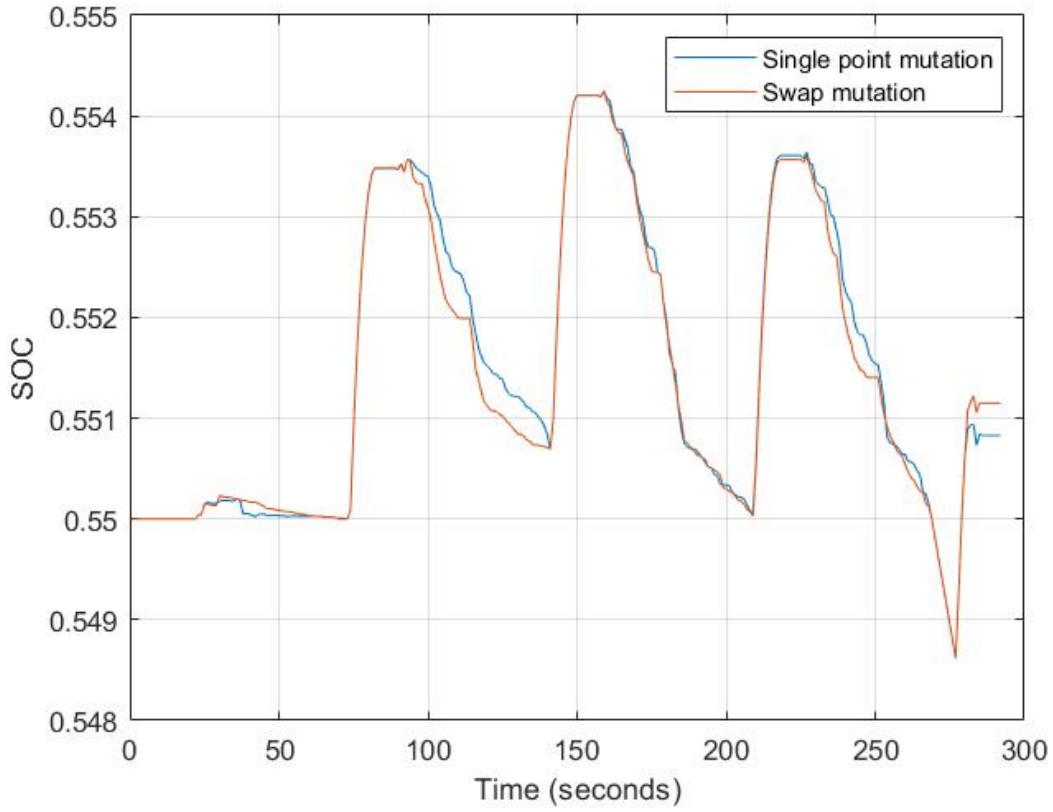

Figure 5.7. SOC plot for genetic algorithm with different mutation functions.

Similar to the crossover function, the choice of mutation function doesn't seem to make a major difference in the SOC plot. For the swap mutation function, however, the final SOC ends up being higher than that for the single point mutation. As the final SOC isn't below the initial SOC and the difference in the final SOCs for both plots is very small, either of the mutation function can work. The rest of the experiments in this section were performed using single point mutation.

Figure 5.8 shows the effect of the choice of the crossover ratio on the SOC plots. In these set of experiments, the crossover function used was uniform crossover and the mutation function used was single point mutation.

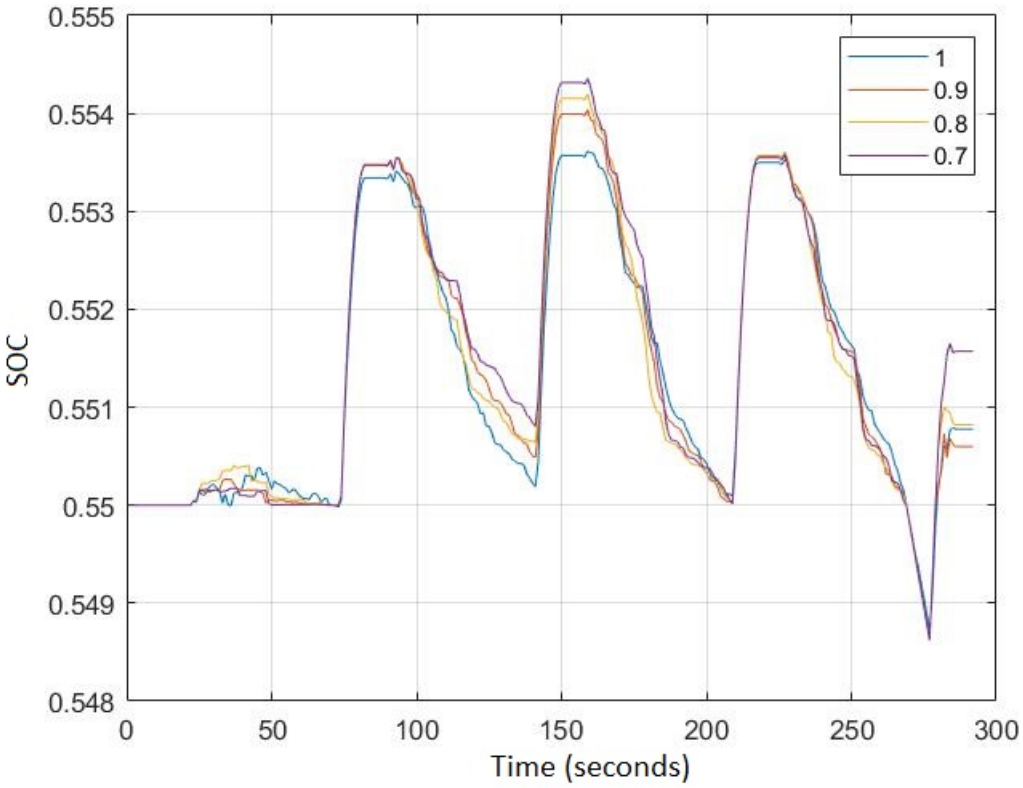

Figure 5.8. SOC plot for genetic algorithm with different crossover ratios.

There are several interesting observations in this plot. Firstly, the value of the crossover ratio doesn't change the overall shape of the plot. Secondly, the most apparent differences in the plots are at the second crest and at the end of the drive cycle. It seems that decreasing the crossover ratio leads to a higher second crest. Theoretically speaking, as the shape of the drive cycle as shown in Figure 5.2 is the same at all crests, the shape of the SOC plot should also be the same at all crests. Also, for crossover ratio of 0.7, the final SOC is relatively much higher than the initial SOC. This phenomenon is not a problem technically but we would prefer the initial and final SOC values to be closer to each other. Hence, we can infer from this plot that mutation associated with crossover ratios less than 1 is detrimental to the results of the genetic algorithm. This inference is in contrast with standard genetic algorithm literature, which encourages mutation for promoting diversity in the pop-

ulation. However, all tunable factors and parameters depend on the problem itself and for this particular problem of PHEV power management strategy, it's better not to introduce mutation in the algorithm.

Based on the above sets of experiments which studied the effects of various tunable parameters and functions, the following were selected for testing the power management strategy for several drive cycles:

$$c_i = 100 \times log_e(i), \tag{5.14}$$

$$r_p = 50,000, \tag{5.15}$$

$$crossoverratio = 0.9. \tag{5.16}$$

Other sets of parameters include using the single point crossover function and single point mutation.

Of the several stopping criteria described in section 2.3, the simplest one is stopping the algorithm after a fixed number of generations. However, it is not a really good criterion as it doesn't guarantee convergence. The one in which the algorithm stops when the average relative change in the fitness function value over a certain fixed number of generations is less than the function tolerance can guarantee convergence if diversity is maintained in the algorithm (so that the algorithm doesn't get stuck in a local minimum). In this project, the latter was chosen as the stopping criterion.

Figure 5.9 shows the SOC results of the genetic algorithm using the aforementioned functions and parameters for the CBD drive cycle and Figure 5.10 shows the torque split results.

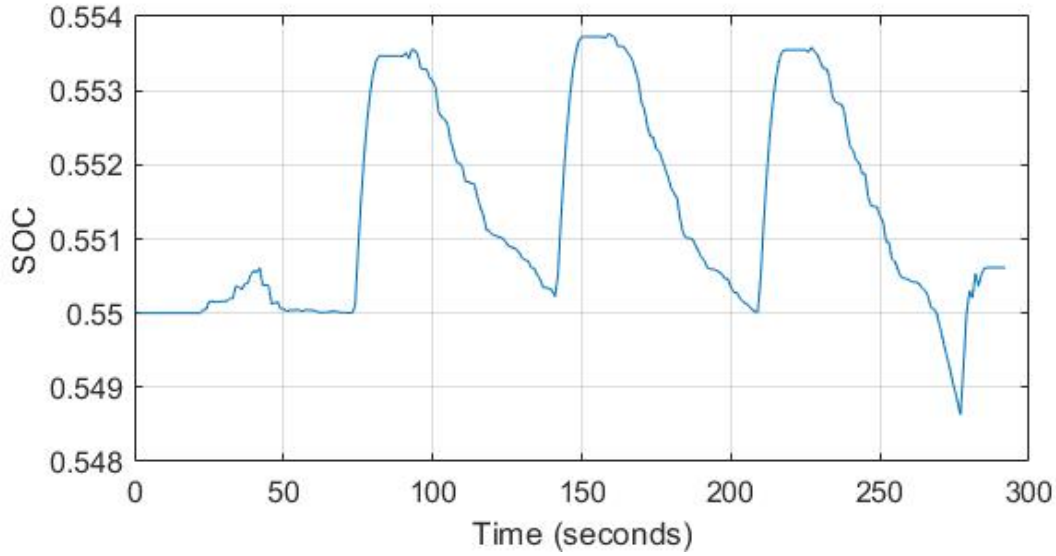

Figure 5.9. SOC plot for genetic algorithm for the Central Business District drive cycle.

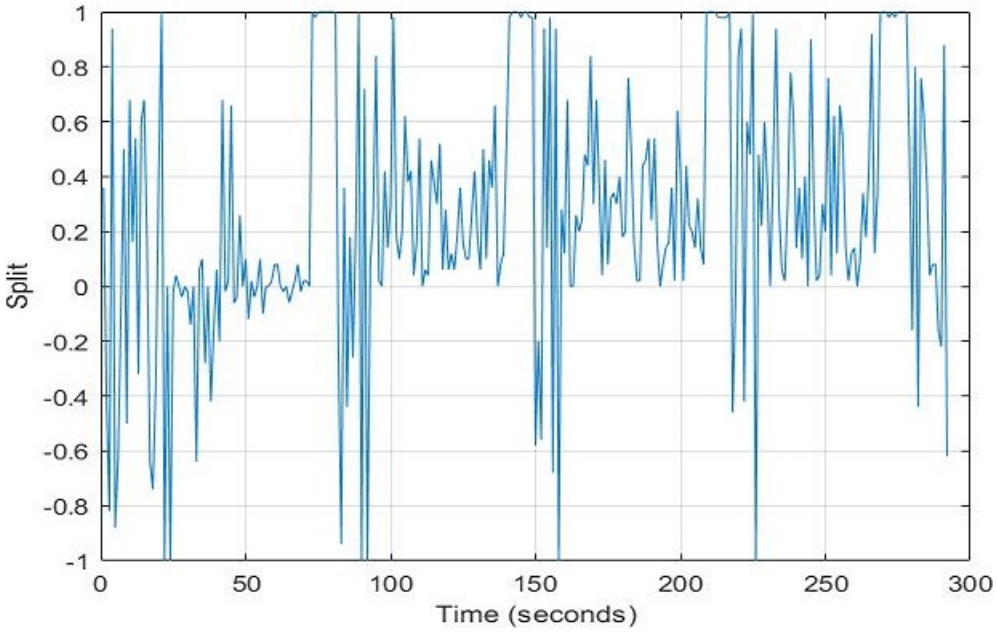

Figure 5.10. Torque split plot for genetic algorithm for the Central Business District drive cycle.

The salient points of these plots are discussed in the next section.

5.3 Comparison and Inferences

Dynamic Programming (DP) and Genetic Algorithm (GA) both fall under the category of Numerical Global Optimization Strategy. Hence, the results of both algorithms have been compared for several drive cycles.

5.3.1 Central Business District Drive Cycle

Torque split results for dynamic programming and genetic algorithm for the Central Business District drive cycle are shown in Figure 5.11 and their difference is plotted in Figure 5.12.

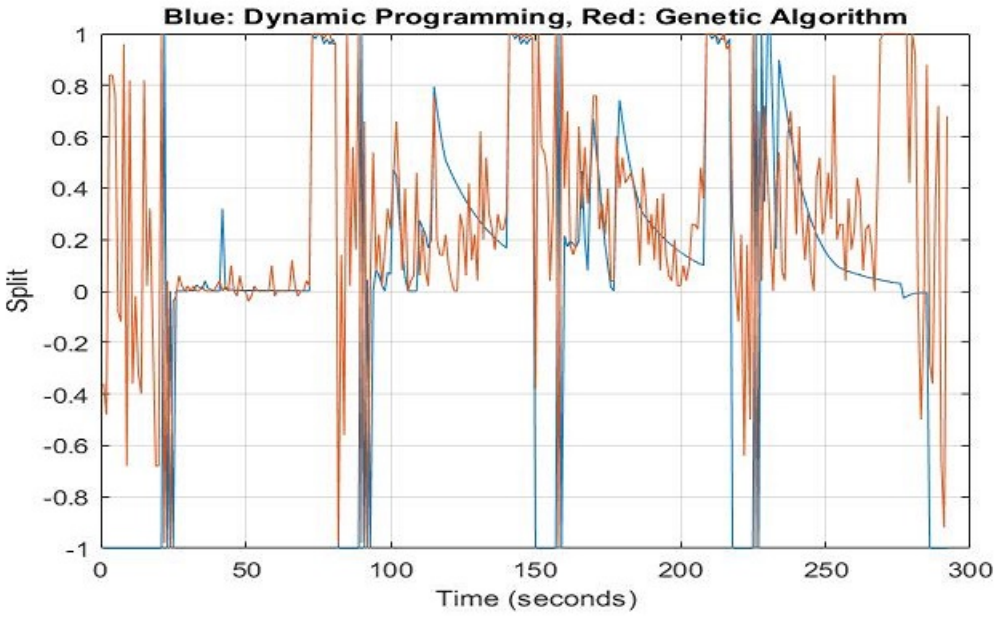

Figure 5.11. Torque split plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

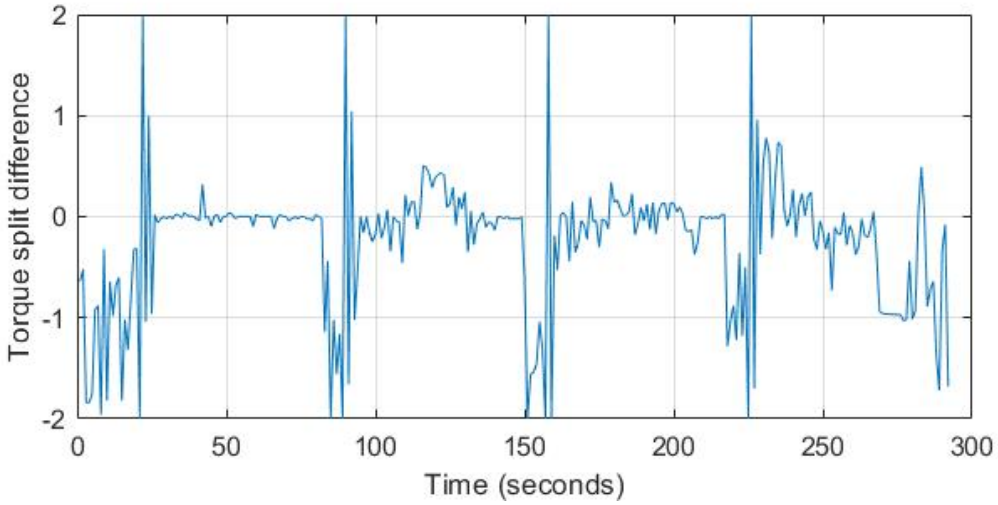

Figure 5.12. Difference between torque split for dynamic programming and genetic algorithm for Central Business District drive cycle.

While there are some time durations where the dynamic programming and genetic algorithm plots for torque split coincide with time, the plots are mostly different elsewhere. The torque split results for genetic algorithm are very oscillatory as compared to dynamic programming. The genetic algorithm torque split oscillations are especially more prominent when there is a gradual decrease in the torque split value in the dynamic programming plot. However, as the genetic algorithm oscillations take place on both sides of the dynamic programming plot, the net effects of both torque split values are similar as can be seen by looking at other plots in this section. This is not a hard fact and some additional experiments should be conducted on this drive cycle to yield more insightful results.

Figure 5.13 and Figure 5.14 show the comparison between the different powers involved in the vehicle for both algorithms.

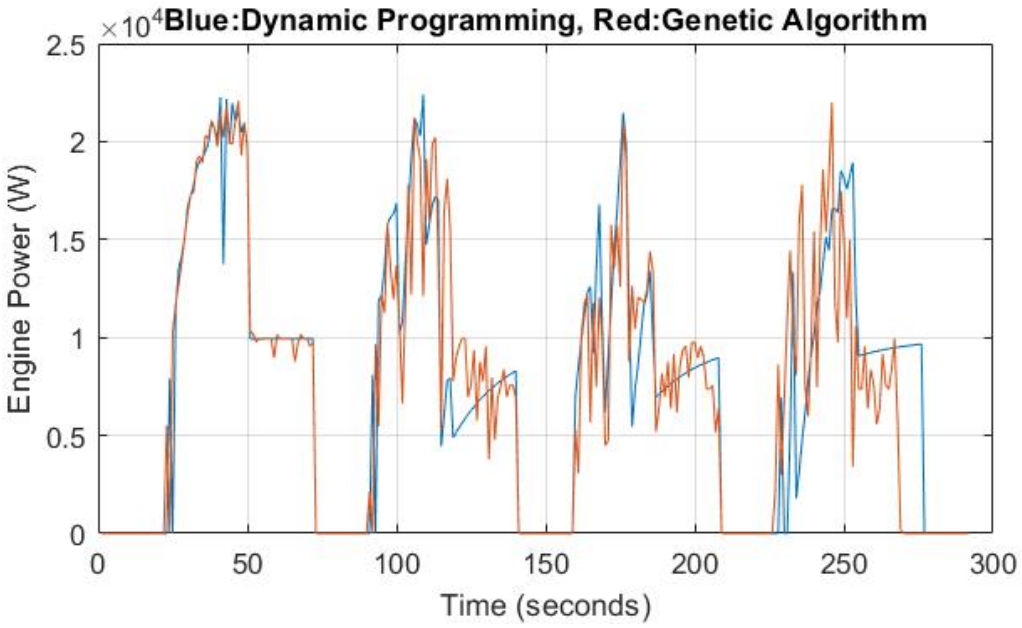

Figure 5.13. Engine power plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

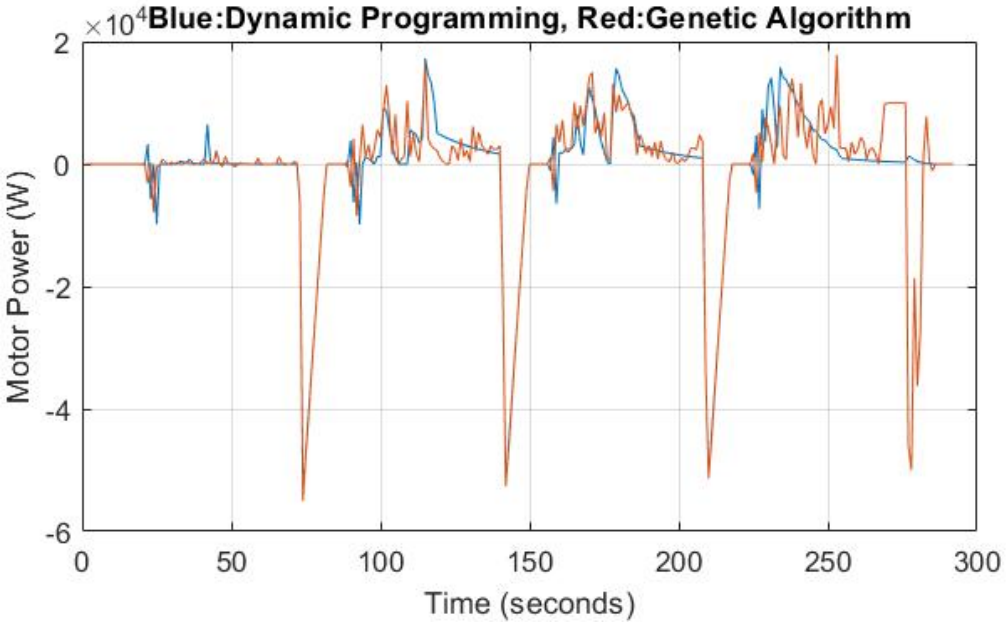

Figure 5.14. Motor power plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

Table 5.1 depicts the different energies supplied by both sources, along with the total energy consumption, for both algorithms.

Table 5.1. Energy consumption comparison for Central Business District drive cycle.

	Engine	Motor	Total
DPM	2.2140×10^6 (J)	6.4283×10^5 (J)	2.8569×10^6 (J)
GA	2.1558×10^6 (J)	7.2463×10^5 (J)	2.8804×10^6 (J)

The engine as well as motor power plots of dynamic programming are similar to those of genetic algorithm. The similarities are probably indicative of the claim made previously that the net effect of the oscillatory genetic algorithm plots and the non-oscillatory dynamic programming plots are close to each other.

The maximum engine power in the drive cycle is 60.753 kW whereas the engine in the real vehicle platform has a rating of 96 kW. The maximum motor power in the drive cycle is 28.441 kW whereas the motor in the real vehicle platform has a rating of 100 kW. Thus we can notice that the maximum powers from both engine sources for this drive cycle are well below their peak values and thus the engine and the motor won't get damaged due to overloading. In fact, we could say that the energy sources underperform for this drive cycle. One of the possible reasons for this underperformance of the vehicle model is the implementation of the constraints mentioned in Equations 4.12-4.15. In the vehicle model, the nature of functions f_1 and f_2 might be too restrictive and those constraints could be relaxed a bit for achieving peak performance.

Figure 5.15 shows the SOC results for both algorithms.

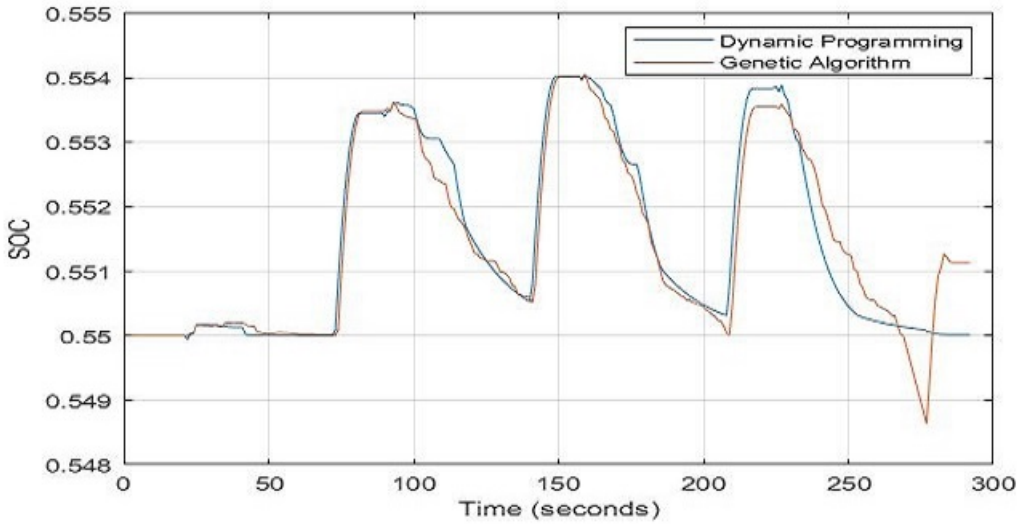

Figure 5.15. SOC plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

The SOC results for both algorithms almost match each other except at the end of the drive cycle. The high degree of similarity probably infers that the SOC is robust to sudden changes in the torque split.

Figure 5.16 shows the fuel consumption results and Figure 5.17 shows the GHG_{WTW} results for both algorithms. The total fuel consumption and GHG_{WTW} results are summarized in Table 5.2.

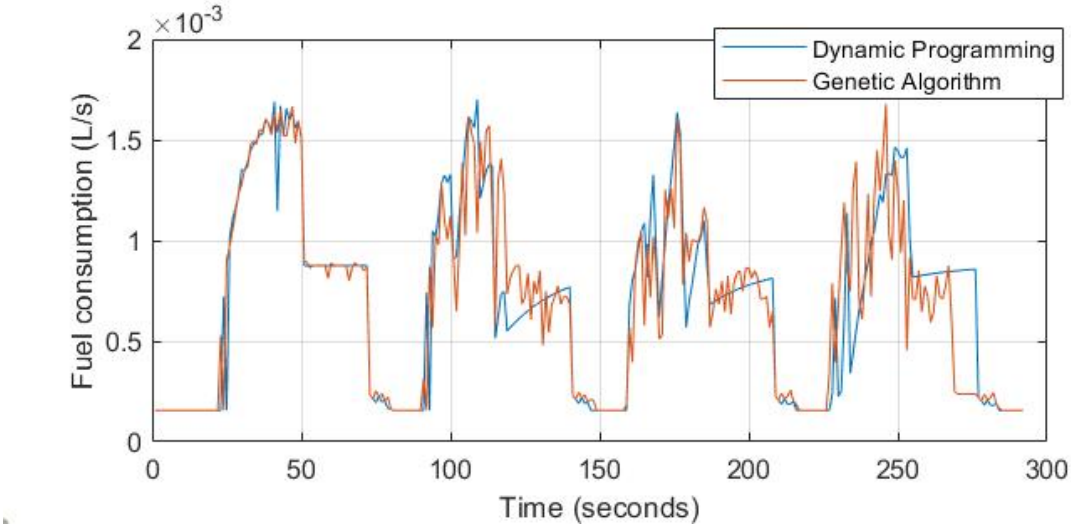

Figure 5.16. Fuel consumption plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

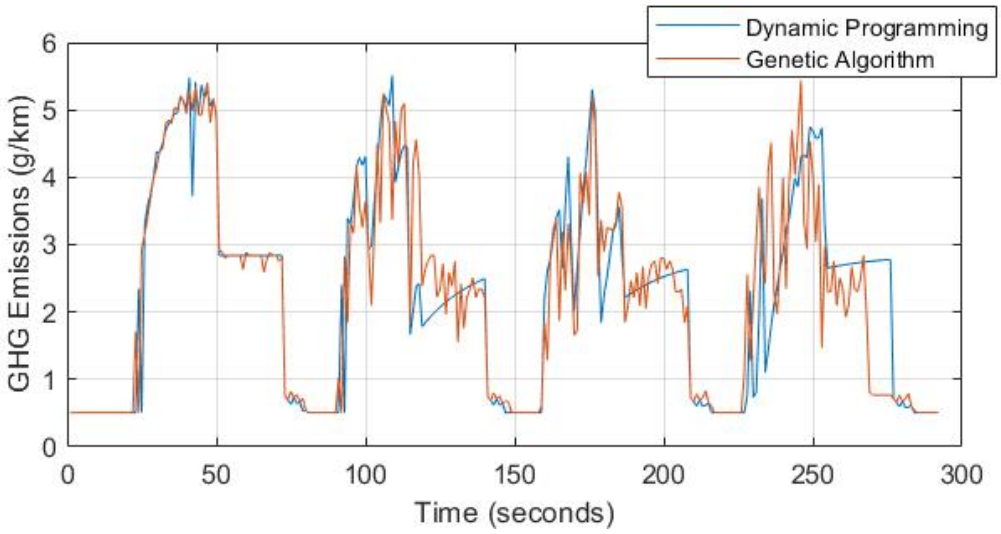

Figure 5.17. Greenhouse gas emissions plot for dynamic programming and genetic algorithm for Central Business District drive cycle.

Table 5.2. Fuel consumption and greenhouse gas emissions for Central Business District drive cycle.

	Fuel consumption	GHG Emissions
DPM	0.2060 (L)	668.1023 (g/km)
GA	0.2024 (L)	656.3394 (g/km)

The shapes and the values of the fuel consumption and greenhouse gas emissions plots for genetic algorithm are very similar to those for dynamic programming. This is also evident in Table 5.2, where we can see that the total fuel consumption and total greenhouse gas emissions values for dynamic programming and genetic algorithm are very close to each other. In fact, genetic algorithm seems to perform slightly better than DP as far as minimizing the total greenhouse gas emissions is concerned. The closeness of these values can be attributed to the fact that the genetic algorithm parameters were tuned with respect to this drive cycle. If the parameters were to be tuned with respect to the UDDS or HWFET drive cycle, the genetic algorithm results would have been different than the current results, although they would probably not depart significantly from the DP results.

5.3.2 Urban Dynamometer Driving Schedule Drive Cycle

Torque split results for the dynamic programming and genetic algorithm for the Urban Dynamometer Driving Schedule drive cycle are shown in Figure 5.18 and their difference is plotted in Figure 5.19.

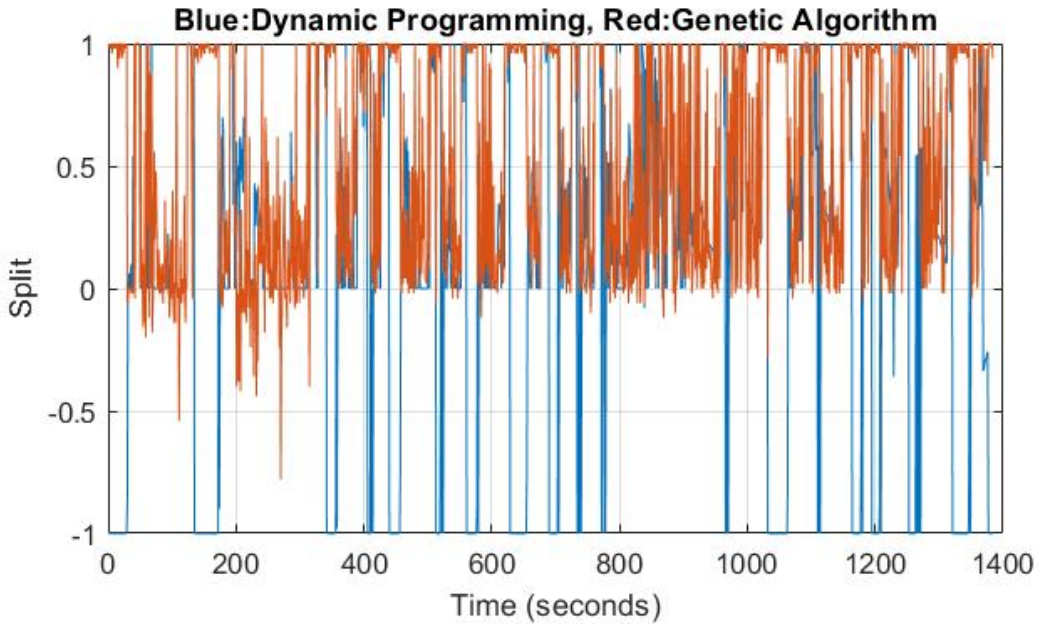

Figure 5.18. Torque split plots for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

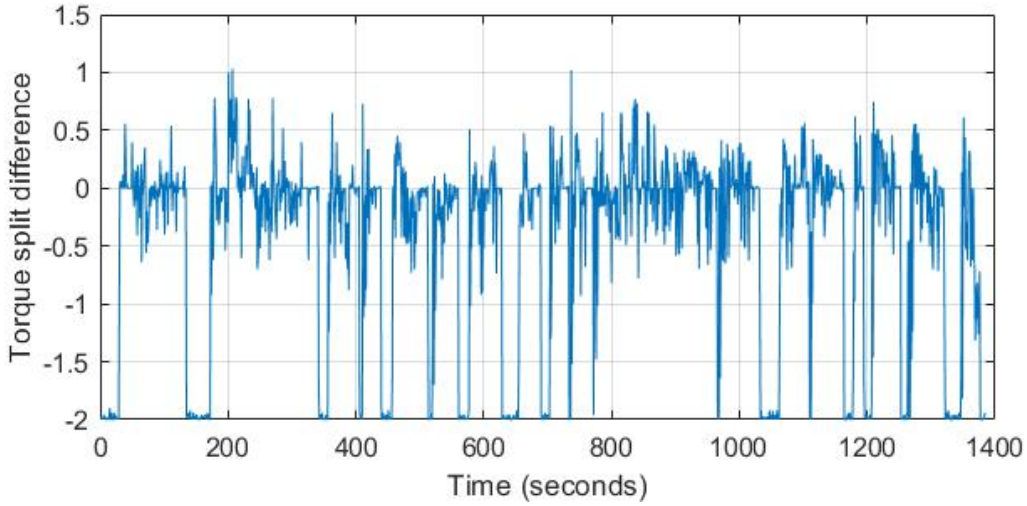

Figure 5.19. Difference between torque split for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

The torque split values are quite different for dynamic programming and genetic algorithm at several points. A strikingly noticeable difference is that the vehicle hardly operates in the charging mode for genetic algorithm. The battery is charged mostly in the regeneration mode.

The comparisons between the different powers involved in the vehicle for both algorithms are shown in Figure 5.20 and Figure 5.21.

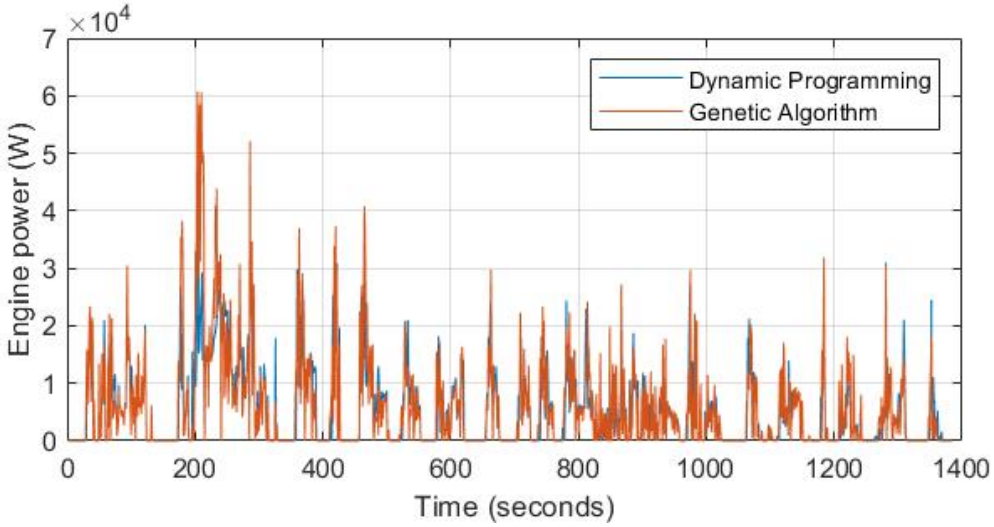

Figure 5.20. Engine power plot for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

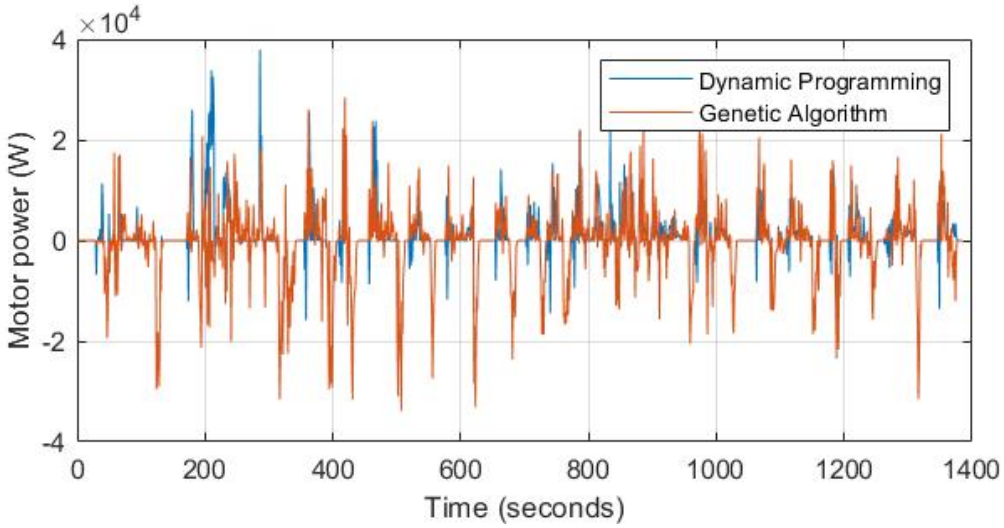

Figure 5.21. Motor power plot for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

Table 5.3 depicts the different energies supplied by both sources, along with the total energy consumption, for both algorithms.

Table 5.3. Energy consumption comparison for Urban Dynamometer Driving Schedule drive cycle.

	Engine	Motor	Total
DPM	$7.2082 \times 10^6(J)$	$2.4105 \times 10^6(J)$	$9.6187 \times 10^6(J)$
GA	$7.6221 \times 10^6(J)$	$2.3999 \times 10^6(J)$	$10.0220 \times 10^6(J)$

From Figure 5.20, Figure 5.21 and Table 5.3, it seems that the total energy consumption by the engine for dynamic programming is less than that for genetic algorithm and the total energy consumption by the motor is higher for genetic algorithm than for dynamic programming. However, the shapes of both plots in Figure 5.20 and Figure 5.21 are similar.

The maximum engine power in the drive cycle is 22.077 kW whereas the engine in the real vehicle platform has a rating of 96 kW. The maximum motor power in

the drive cycle is 17.845 kW whereas the motor in the real vehicle platform has a rating of 100 kW. Thus we can notice that the maximum powers from both engine sources for this drive cycle are well below their peak values and thus the engine and the motor won't get damaged due to overloading. In fact, we could say that the energy sources underperform for this drive cycle. One of the possible reasons for this underperformance of the vehicle model is the implementation of the constraints mentioned in Equations 4.12-4.15. In the vehicle model, the nature of functions f_1 and f_2 might be too restrictive and those constraints could be relaxed a bit for achieving peak performance.

Figure 5.22 shows the SOC results for both algorithms.

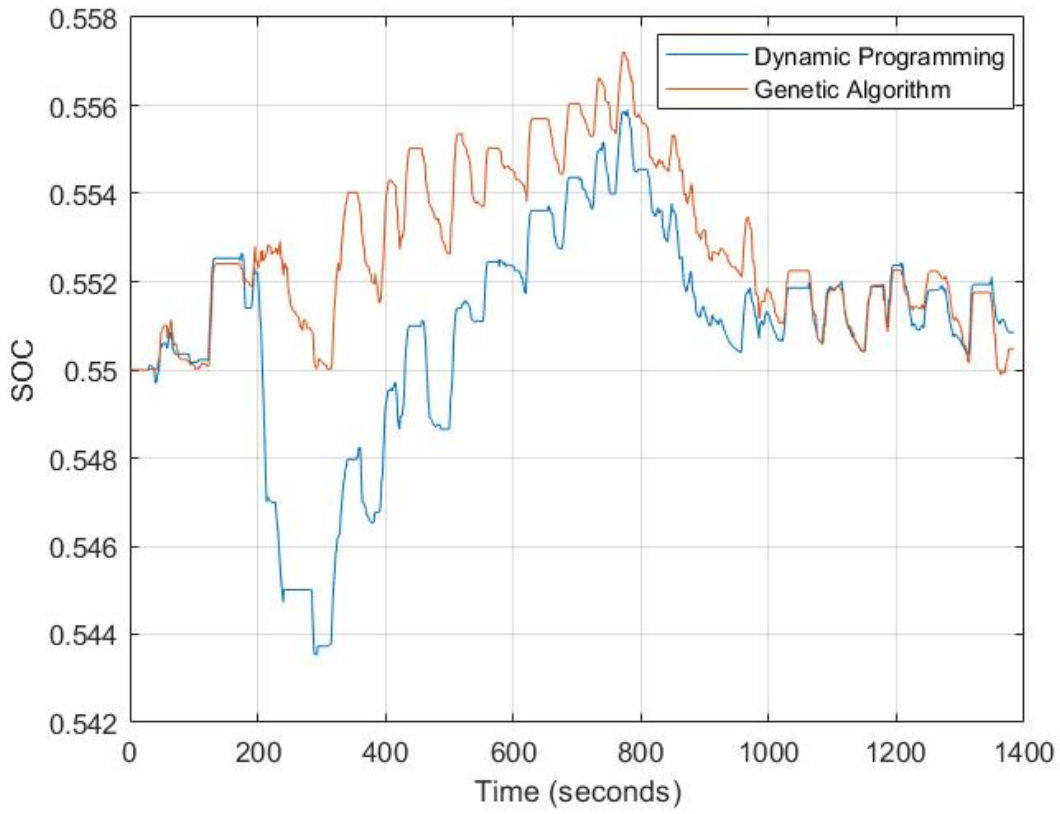

Figure 5.22. SOC plot for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

The shape of the SOC plots, especially the local crests and troughs, for both algorithms are very similar. However, the SOC values never dip below the target value for genetic algorithm. This is probably due to the fact that the tunable parameters for genetic algorithm were set with respect to the CBD drive cycle. Unlike ECMS or regression, though, the optimal tunable parameters for CBD drive cycle also give plots for the UDDS drive cycle that are very similar to their dynamic programming counterparts. This suggests the fact that a single set of tunable parameters can exist which can give better results for all drive cycles.

Figure 5.23 shows the fuel consumption results and Figure 5.24 shows the GHG_{WTW} results for both algorithms. The total fuel consumption and GHG_{WTW} results are summarized in Table 5.4.

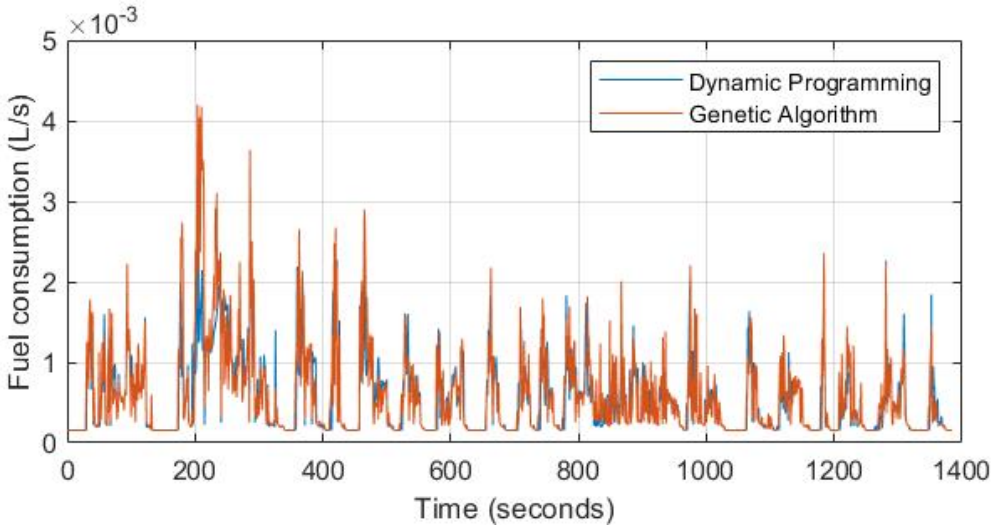

Figure 5.23. Fuel consumption plot for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

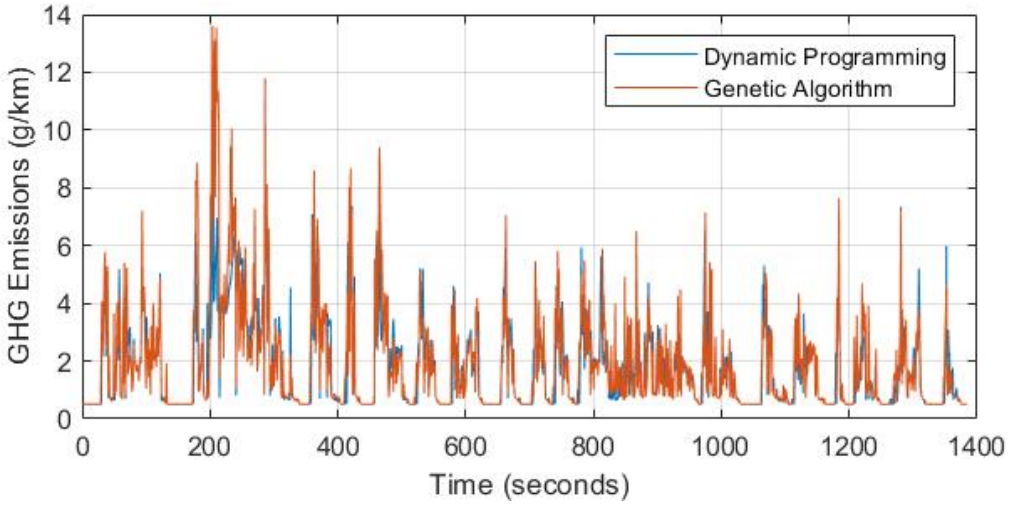

Figure 5.24. Greenhouse emissions plot for dynamic programming and genetic algorithm for Urban Dynamometer Driving Schedule drive cycle.

Table 5.4. Fuel consumption and greenhouse gas emissions for Urban Dynamometer Driving Schedule drive cycle.

	Fuel consumption	GHG Emissions
DPM	0.7558 (L)	2.4510×10^3 (g/km)
GA	0.7945 (L)	2.5765×10^3 (g/km)

The shapes of the fuel consumption plots and greenhouse gas emissions plots for both genetic algorithm and dynamic programming are similar. The genetic algorithm plots are kind of elongated vertically with respect to the dynamic programming plots. This is also evident in Table 5.4, which shows that the total fuel consumption and greenhouse gas emissions over the drive cycle for genetic algorithm are higher than their dynamic programming counterparts. However, the corresponding values are very close to each other, although the genetic algorithm parameters were tuned with respect to the CBD drive cycle. This also corroborates the claim mentioned previously

that there is probably a universal set of parameters for genetic algorithms which work for all drive cycles.

5.3.3 Highway Fuel Economy Test Drive Cycle

Torque split results for the dynamic programming and genetic algorithm for the Highway Fuel Economy Test drive cycle are shown in Figure 5.25 and their difference is plotted in Figure 5.26.

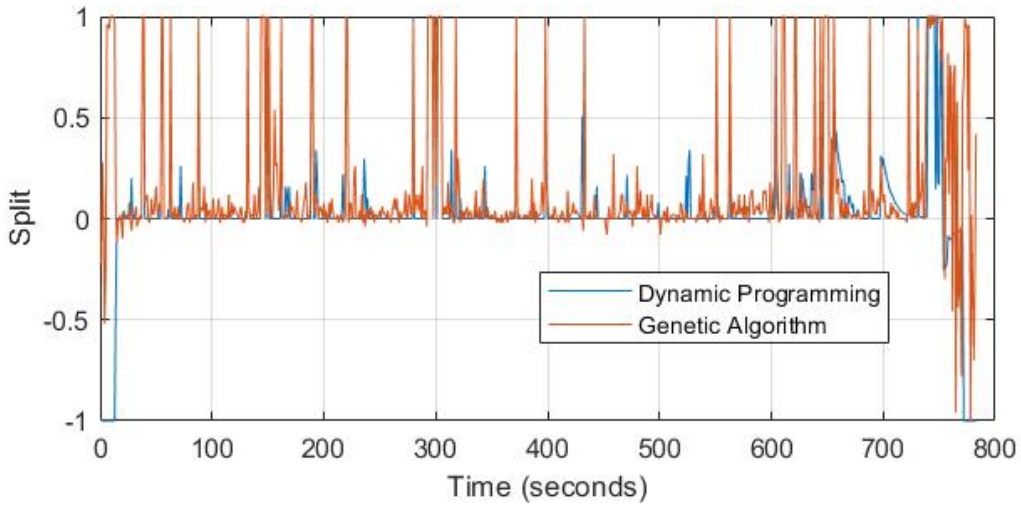

Figure 5.25. Torque split plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

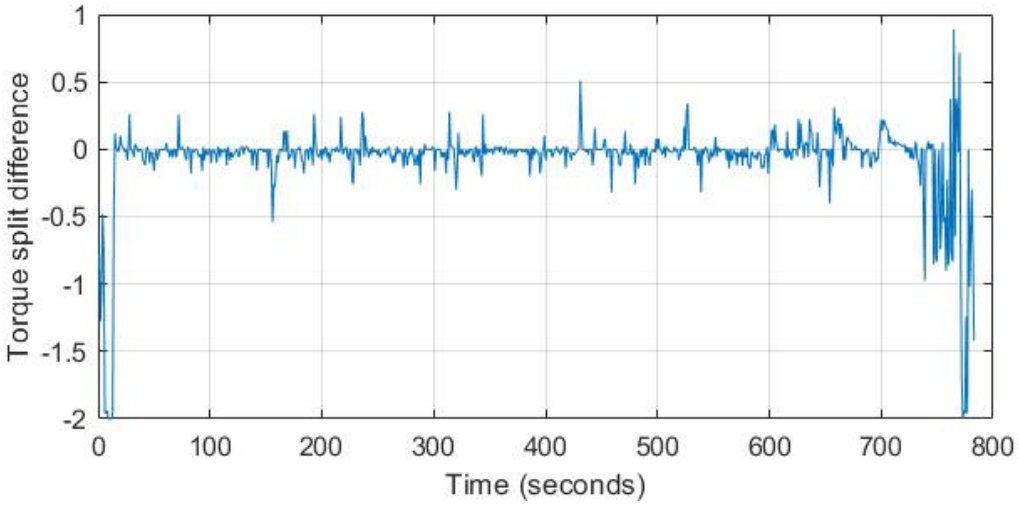

Figure 5.26. Difference between torque split for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

Figure 5.27 and Figure 5.28 plot the comparison between the different powers involved in the vehicle for both algorithms.

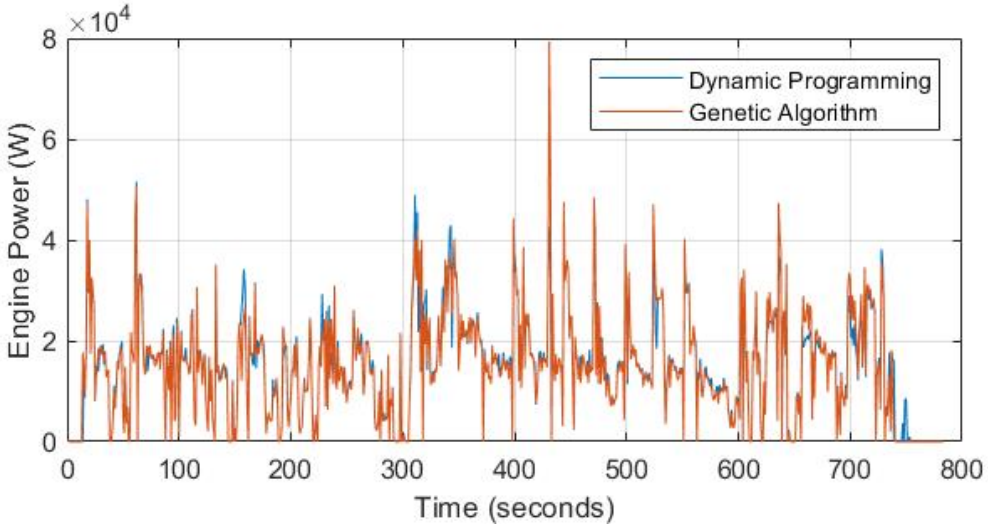

Figure 5.27. Engine power plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

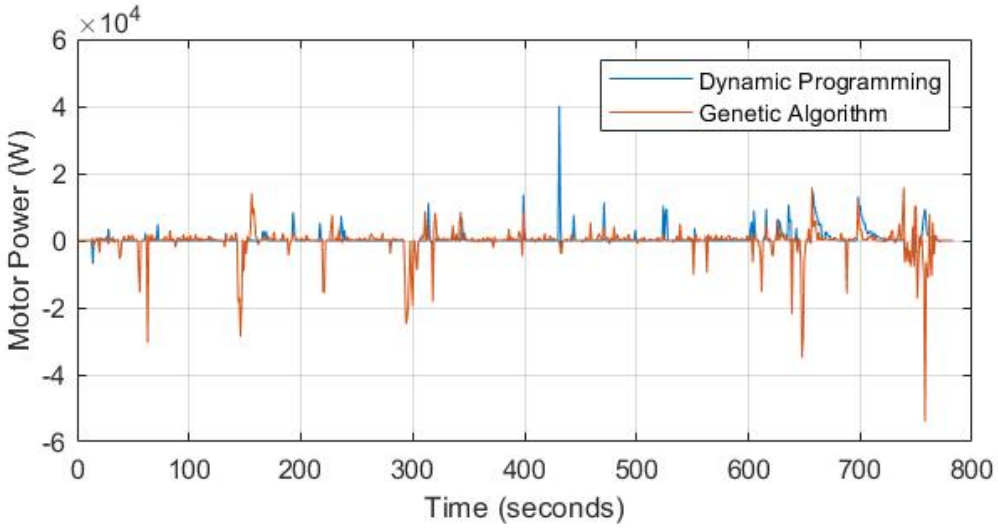

Figure 5.28. Motor power plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

Table 5.5 depicts the different energies supplied by both sources, along with the total energy consumption, for both algorithms.

Table 5.5. Energy consumption comparison for Highway Fuel Economy Test drive cycle.

	Engine	Motor	Total
DPM	1.1612×10^7 (J)	5×10^5 (J)	1.2112×10^7 (J)
GA	1.1598×10^7 (J)	5.4134×10^5	1.2139×10^7 (J)

The maximum engine power in the drive cycle is 79.34 kW whereas the engine in the real vehicle platform has a rating of 96 kW. The maximum motor power in the drive cycle is 15.881 kW whereas the motor in the real vehicle platform has a rating of 100 kW. Thus we can notice that the maximum powers from both engine sources for this drive cycle are well below their peak values and thus the engine and the motor won't get damaged due to overloading. In fact, we could say that the energy sources underperform for this drive cycle. One of the possible reasons for

this underperformance of the vehicle model is the implementation of the constraints mentioned in Equations 4.12-4.15. In the vehicle model, the nature of functions f_1 and f_2 might be too restrictive and those constraints could be relaxed a bit for achieving peak performance.

Figure 5.29 shows the SOC results for both algorithms.

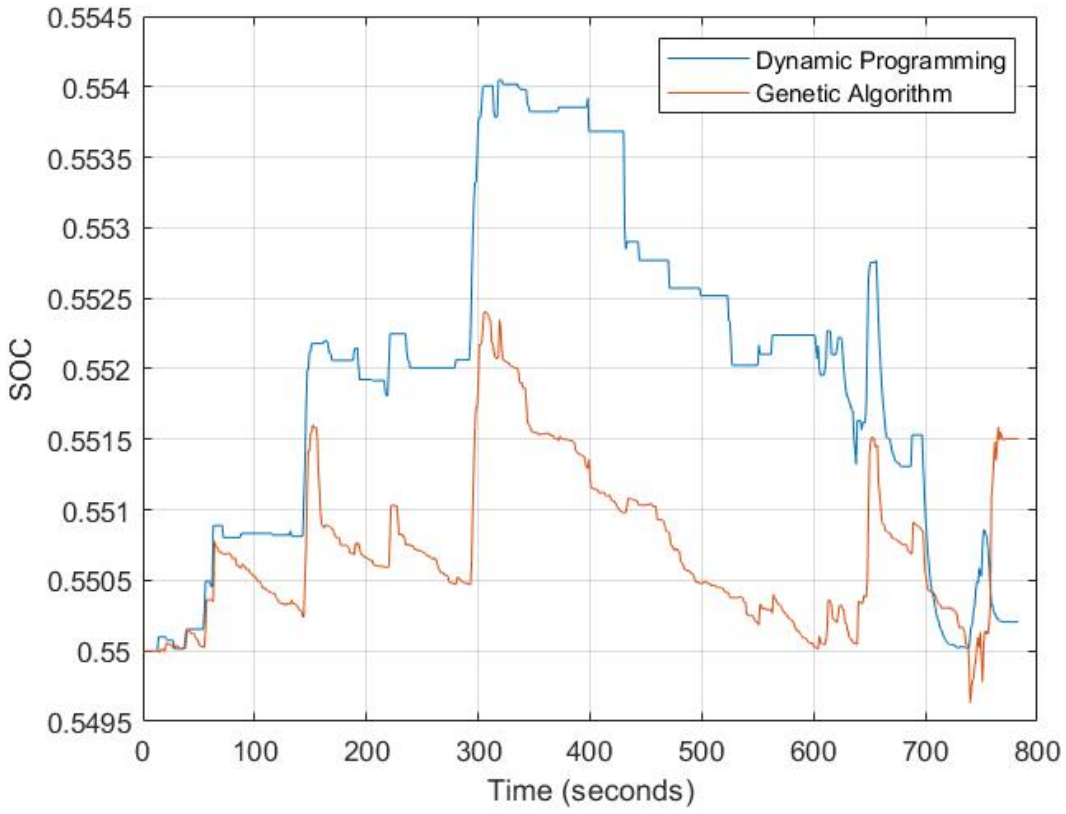

Figure 5.29. SOC plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

Figure 5.30 shows the fuel consumption results and Figure 5.31 shows the GHG_{WTW} results for both algorithms. The total fuel consumption and GHG_{WTW} results are summarized in Table 5.6.

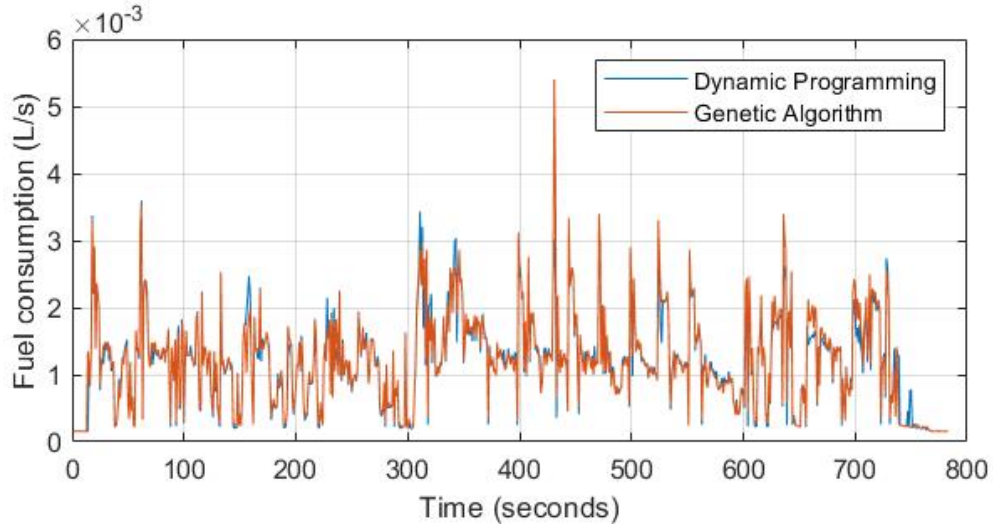

Figure 5.30. Fuel consumption plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

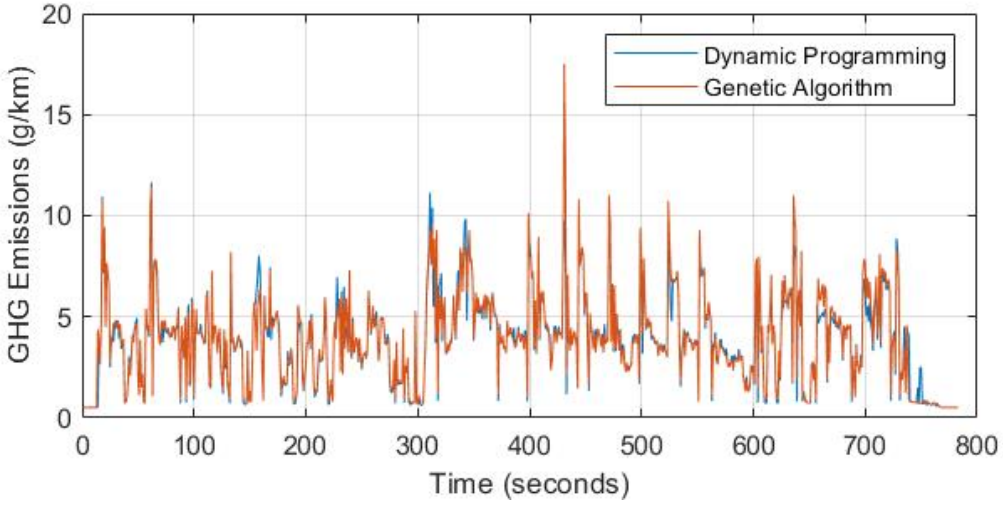

Figure 5.31. Greenhouse gas emissions plot for dynamic programming and genetic algorithm for Highway Fuel Economy Test drive cycle.

Table 5.6. Fuel consumption and greenhouse gas emissions for Highway Fuel Economy Test drive cycle.

	Fuel consumption	GHG Emissions
DPM	0.9347 (L)	3.0312×10^3 (g/km)
GA	0.9505 (L)	3.0827×10^3 (g/km)

All dynamic programming plots for the HWFET drive cycle included in this section are very similar to their genetic algorithm counterparts. The similarities are also evident in Table 5.5 and Table 5.6. In Table 5.6, we can see that the differences between the total fuel consumption results as well as the total greenhouse gas emission results for dynamic programming and genetic algorithm are very small. This further corroborates the claim of the existence of a universal set of genetic algorithm parameters. These universal parameters would most probably shift the genetic algorithm SOC plot in Figure 5.29 upwards towards the dynamic programming SOC plot.

5.3.4 Strength of constraints

One peculiar observation about the results of both dynamic programming and genetic algorithm is that the SOC values hardly deviate from their target values for the charge-sustaining mode of the vehicle for both algorithms. This seems to suggest that the algorithms are probably over-constraining the SOC deviation. This fact was tested by running the vehicle model for all drive cycles with the condition that the vehicle is powered purely by the engine (or in other words, the torque split value is 0) and comparing the total fuel consumption and greenhouse gas emissions for both the charge-sustaining mode and the engine-only mode. The results are summarized in Table 5.7, Table 5.8 and Table 5.9.

Table 5.7. Comparison of engine-only (EO) mode and charge-sustaining (CS) mode for the Central Business District drive cycle.

	Fuel consumption	GHG Emissions
DPM (CS)	0.2060 (L)	668.1023 (g/km)
GA (CS)	0.2024 (L)	656.3394 (g/km)
EO	0.2493 (L)	808.1023 (g/km)

Table 5.8. Comparison of engine-only (EO) mode and charge-sustaining (CS) mode for the Urban Dynamometer Driving Schedule drive cycle.

	Fuel consumption	GHG Emissions
DPM (CS)	0.7558 (L)	2.4510×10^3 (g/km)
GA (CS)	0.7945 (L)	2.5765×10^3 (g/km)
EO	0.9559 (L)	3.1×10^3 (g/km)

Table 5.9. Comparison of engine-only (EO) mode and charge-sustaining (CS) mode for the Highway Fuel Economy Test drive cycle.

	Fuel consumption	GHG Emissions
DPM (CS)	0.9347 (L)	3.0312×10^3 (g/km)
GA (CS)	0.9505 (L)	3.0827×10^3 (g/km)
EO	0.9831 (L)	3.1884×10^3 (g/km)

We notice that the fuel consumption and greenhouse gas emissions for the engine-only mode for all three drive cycles are higher than their charge-sustaining counterparts, which is obvious because no battery energy is being used in the engine-only mode. For the Central Business District and the Highway Fuel Economy Test drive cycle,

the increase in fuel consumption is not as significant as that for the Urban Dynamometer Driving Schedule drive cycle. However, the increase in the greenhouse gas emissions for the engine-only mode for all drive cycles is significant compared to the fuel consumption increase. This seems to suggest that a slight decrease in the fuel consumption leads to significant reduction of greenhouse gas emissions. The larger (and more realistic) the drive cycle is, the higher the reduction in fuel consumption and consequently the greenhouse gas emissions will be. As the power management strategy in this project is aimed to reduce the greenhouse gas emissions (and not the fuel consumption, although the two are have some degree of positive correlation), the charge-sustaining constraints on the vehicle are seemingly appropriately strong. If the goal of the strategy is to reduce the fuel consumption, however, the results might differ in terms of the proportional differences between the fuel consumption and greenhouse gas emissions.

In other words, while the constraints on the SOC for implementing the charge-sustaining mode can be relaxed, the constraints used in this thesis also result in significant reduction of greenhouse gas emissions. In fact, the original constraints are perfect for the blended mode. In this mode, the battery can be fully charged before a trip. During the trip, the vehicle is operated in the charge-depleting mode up to a certain pre-determined SOC level. Once the SOC reaches this target level, the vehicle mode is switched to charge-sustaining mode until a charging station is found. The seemingly strong constraints on the SOC for the charge-sustaining mode will ensure that the SOC doesn't fall below critical levels, the vehicle operates smoothly without any issues until a charging station is found, and greenhouse gas emissions are still significantly reduced compared to the engine-only mode.

5.3.5 Forward implementation

In this thesis, the objective function of the genetic algorithm included the vehicle model, which is used to calculate the total greenhouse gas emissions over the drive

cycle. The form in which the vehicle model was implemented assumed that we already know the vehicle speed a *priori* and accordingly calculated other vehicle variables. It is equally important to find out whether the optimal torque split control law obtained using genetic algorithm results in a vehicle velocity that matches the original speed profile of the drive cycle used. In other words, we need to find out whether the forward model leads to a speed profile that matches the original speed profile used as an input to the backward model. Figure 5.32, Figure 5.33, and Figure 5.34 show the comparison between the original speed profile and the speed profile generated by the forward vehicle model on the Central Business District, Urban Dynamometer Driving Schedule and the Highway Fuel Economy Test drive cycles.

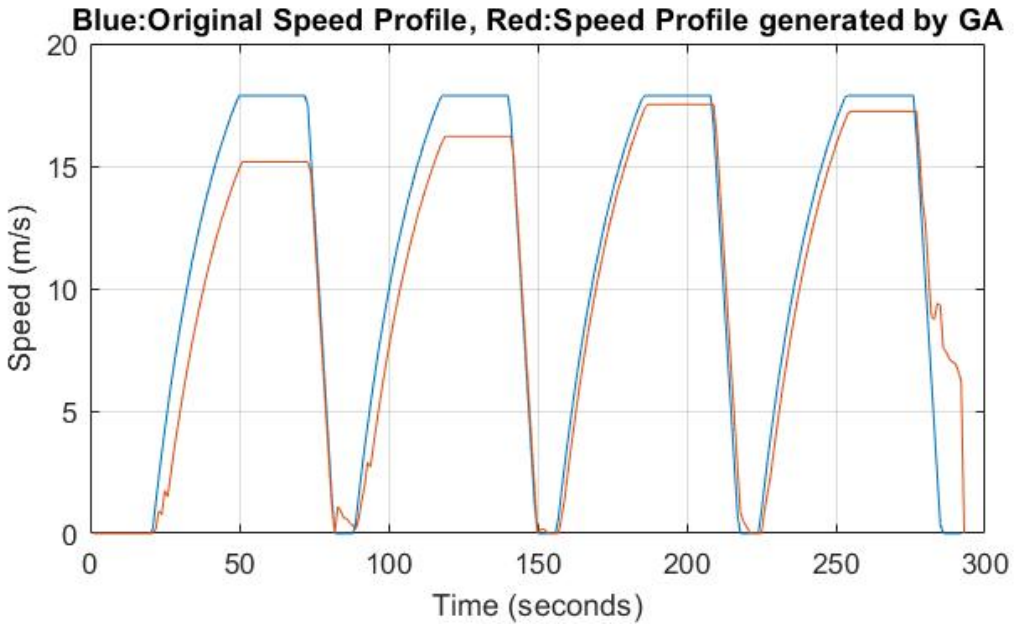

Figure 5.32. Speed profile of vehicle generated by genetic algorithm strategy for the Central Business District drive cycle.

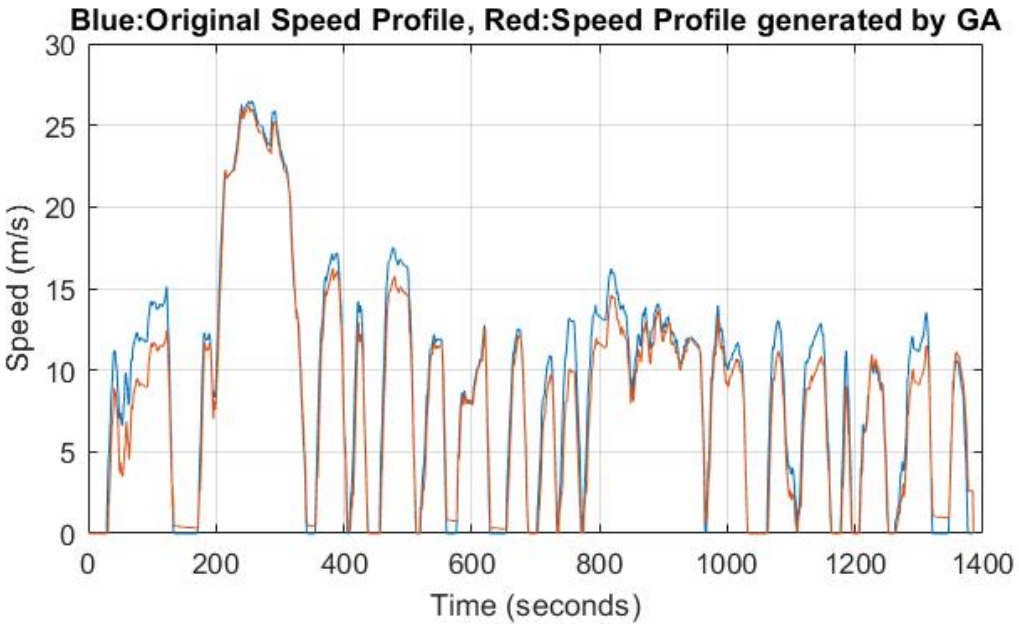

Figure 5.33. Speed profile of vehicle generated by genetic algorithm strategy for the Urban Dynamometer Driving Schedule drive cycle.

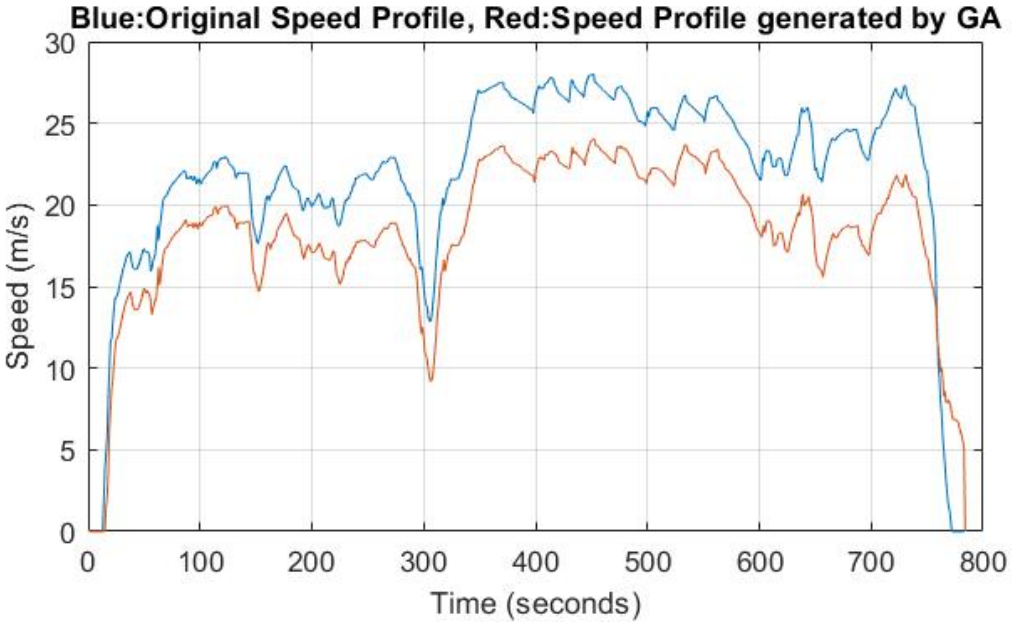

Figure 5.34. Speed profile of vehicle generated by genetic algorithm strategy for the Highway Fuel Economy Test drive cycle.

We notice that there are some differences between both speed profiles. Specifically, the profile generated by the genetic algorithm has lower speeds than the original speed profile. However, the difference between the speeds is not too large and also the shapes of both speed profiles are the same for all drive cycles considered here.

5.3.6 Inferences

By comparing all plots and tables described in subsections 5.3.1-5.3.3, we can observe that there are some similarities and differences between the results of dynamic programming (DP) and genetic algorithm (GA). A possible reason for the discrepancies is the stochastic nature of a genetic algorithm. As there is randomness involved in a genetic algorithm, some departure from dynamic programming is bound to happen. Secondly, the charge-sustaining SOC constraints on the vehicle model are implemented in the form of penalty functions in genetic algorithm. These penalty functions and their coefficients were chosen based on a combination of trial-and-error processes and some engineering intuition. As deciding the penalty functions is not purely a deterministic process, there might be some other choice of penalty functions that would make the results of genetic algorithm more similar to the results of DP. Additionally, Figure 5.5 shows that the final state constraints influence the rest of the plots in a manner that is not exactly known currently.

All discrepancies aside, there are striking similarities between the results of dynamic programming and genetic algorithm that can be attributed to the type of problem that both algorithms attempt to solve (global optimization). Specifically, the greenhouse gas emission and fuel consumption results are very close to each other. These results are important as the goal of both algorithms is minimizing greenhouse gas emissions. As long as the objective function is being minimized, the performances of both algorithms are comparable to each other.

The genetic algorithm power management controller is non-causal in nature as it assumes the vehicle drive cycle knowledge to be known in advance and hence it cannot

be implemented in a real vehicle controller. However, as the control law obtained using this technique is optimal over the drive cycle used for running the simulation, its results can be used as a benchmark against which other real-time and/or rule based controllers like regression or deep learning controllers can be compared.

6. CONCLUSIONS AND FUTURE WORK

Genetic algorithm (GA) is a global search technique that can be used to solve optimization problems. By showing that the idea of transforming each input variable of an optimization problem into vectors of numbers can be extended to variables that are themselves vectors in nature, a new genetic algorithm based power management strategy was developed. The results of genetic algorithm were compared with dynamic programming for several drive cycles and many similarities were observed.

This genetic algorithm-based global optimization strategy can be extended to any optimal control problem; it's not restricted to the power management problem of a hybrid electric vehicle. For solving an optimal control problem using genetic algorithm, we should have a mathematical model of the dynamic system we wish to control, the objective or cost function to be minimized, the constraints on the inputs, outputs, and states, and the way they can be incorporated in the objective function. The population initialization, crossover and mutation functions developed in this thesis can be used for any optimal control problem. In a way, this thesis also presents an original technique of solving an optimal control problem using genetic algorithms.

Genetic algorithms, however, suffer from some disadvantages. Similar to dynamic programming, the genetic algorithm controller is non-causal in nature as it requires prior knowledge of how all input variables are going to change with time. In the context of the power management strategy of a hybrid electric vehicle, the genetic algorithm controller requires the knowledge of the vehicle drive cycle *a priori*. Thus, the genetic algorithm controller cannot be implemented in a real vehicle. However, similar to dynamic programming, the results of the genetic algorithm controller serve as a benchmark against which other causal and real-time controllers can be compared. Additionally, compared to dynamic programming, a genetic algorithm takes a very

long time to run. The major reason for the long run time of genetic algorithm is the vast number of fitness function computations performed per generation, which is defined by the population size, which in turn depends on the number of genes in each genome. As discussed previously in the thesis, there is a trade-off between the population size and performance of genetic algorithm. As we prefer setting the population size to be at least four times the genome size, an optimal control problem spanning a longer time duration will take a longer time to execute. While significant progress has been made in the field of computational technology like computer clusters, cloud computing, and graphics processing units, dynamic programming takes a significantly shorter time for execution. Thirdly, solving a constrained optimal control problem using genetic algorithm requires significantly more engineering intuition than dynamic programming as there is provision in dynamic programming for explicitly including constraints.

This naturally begs the question of the utility of genetic algorithms as dynamic programming provides similar results in less amount of time. There are several reasons why genetic algorithms shouldn't be discarded as a workable power management strategy. Firstly, it was observed that the performance of dynamic programming suffers for short-time optimal control problems. For instance, Japan 10, 15 and 1015 Mode drive cycles [39] are very short as far as time duration is concerned (13, 17 and 54 seconds, respectively). For such short-time problems, the open source dynamic programming function that was used in this thesis gives errors. On the other hand, genetic algorithm works for such short-time optimal control problems. Figure 6.1 shows the SOC results of genetic algorithm for Japan 10 Mode drive cycle.

Secondly, genetic algorithm can work as a sanity check to system modeling and control. Developing models of dynamic systems that emulate real-life conditions is a complicated process and there are several ways for committing syntactical and logical errors. These errors are often not evident because all system constraints are being followed. To check the accuracy of a model, both genetic algorithm and dynamic programming simulations can be run on the model and the results can be compared.

If there are significant discrepancies in the results, we can conclude that there are errors in the model.

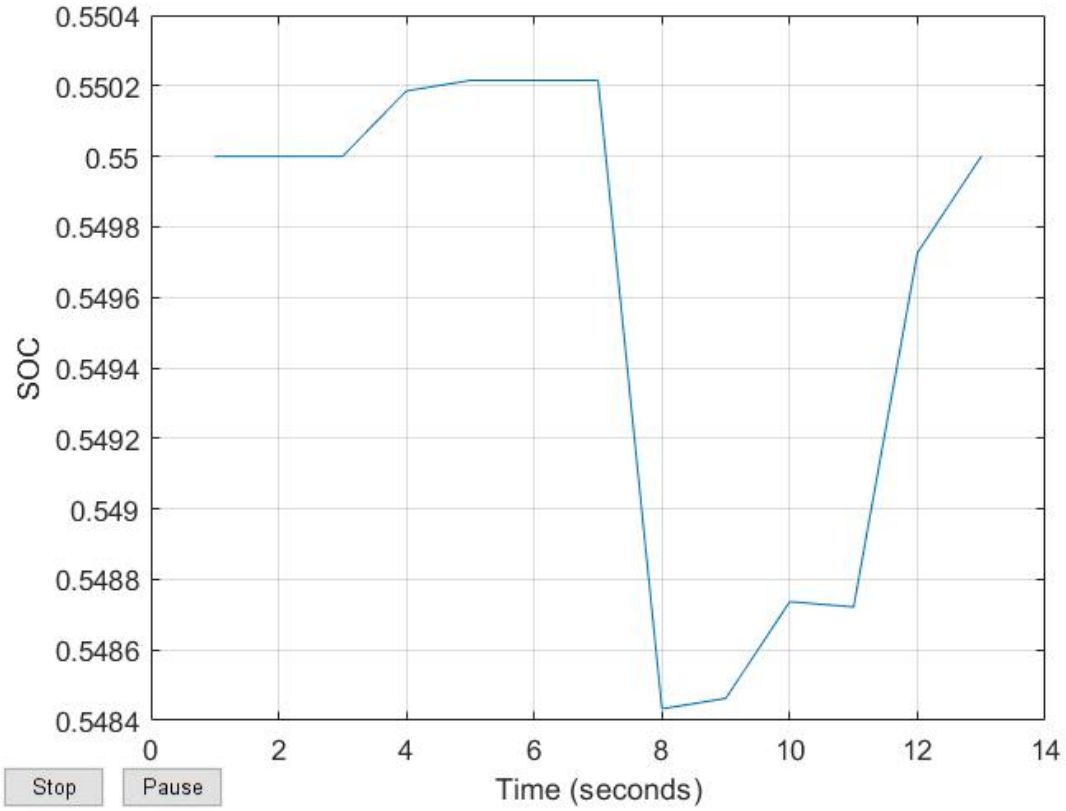

Figure 6.1. SOC results for genetic algorithm for Japan 10 drive cycle.

There are several avenues for future research in this area. Currently, the only known method of solving constrained optimization problems using genetic algorithm is including the constraints in the fitness function in the form of penalty functions. It was claimed in the previous chapter that a universal set of GA parameters exist for all drive cycles. One possible path of future research involves finding this universal set of parameters using more extensive experimenting with different types of penalty functions and their coefficients. Also, deciding the form of penalty functions is a problem-dependent process and requires some intuition whereas tuning the penalty coefficients is a trial-and-error process. Future work involves finding out techniques

to automate these processes and applying them to develop a problem-independent optimal control solution strategy. Alternatively, some modifications can be made in the algorithm itself so as to introduce provisions for explicitly including model constraints.

REFERENCES

REFERENCES

[1] US Energy Information Administration. Total energy monthly data - US energy information administration. https://www.eia.gov/totalenergy/data/monthly/.

[2] The Millennium Alliance for Humanity and the Biosphere. When fossil fuels run out, what then? https://mahb.stanford.edu/library-item/fossil-fuels-run/.

[3] United States National Oceanic and Atmospheric Administration. Climate change: Global temperature. https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature.

[4] Overview of greenhouse gases. https://www.epa.gov/ghgemissions/overview-greenhouse-gases.

[5] United States Environmental Protection Agency. Sources of greenhouse gas emissions. https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions.

[6] Rohinish Gupta. Modelling and control of a parallel through-the-road plug-in hybrid vehicle. *Open Access Theses*, 2016.

[7] Lorenzo Serrao, Simona Onori, and Giorgio Rizzoni. A Comparative Analysis of Energy Management Strategies for Hybrid Electric Vehicles. *Journal of Dynamic Systems, Measurement, and Control*, 133(3), 03 2011. 031012.

[8] Yuan Zhu, Yaobin Chen, Guangyu Tian, Hao Wu, and Quanshi Chen. A four-step method to design an energy management strategy for hybrid vehicles. In *Proceedings of the 2004 American Control Conference*, volume 1, pages 156–161, Boston, MA, USA, July 2004. IEEE.

[9] Chan-Chiao Lin, Huei Peng, Jessy W Grizzle, and Jun-Mo Kang. Power management strategy for a parallel hybrid electric truck. *IEEE Transactions on Control Systems Technology*, 11(6):839–849, 2003.

[10] Olle Sundström, Daniel Ambühl, and Lino Guzzella. On implementation of dynamic programming for optimal control problems with final state constraints. *Oil & Gas Science and Technology–Revue de l'Institut Français du Pétrole*, 65(1):91–102, 2010.

[11] Avra Brahma, Yann Guezennec, and Giorgio Rizzoni. Optimal energy management in series hybrid electric vehicles. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334)*, volume 1, pages 60–64, Chicago, IL, USA, August 2000. IEEE.

[12] Antonio Piccolo, Lucio Ippolito, Vincen zo Galdi, and Alfredo Vaccaro. Optimisation of energy flow management in hybrid electric vehicles via genetic algorithms. In *2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics. Proceedings (Cat. No. 01TH8556)*, volume 1, pages 434–439, Como, Italy, July 2001. IEEE.

[13] Ilya Kolmanovsky, Irina Siverguina, and Bob Lygoe. Optimization of powertrain operating policy for feasibility assessment and calibration: Stochastic dynamic programming approach. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 2, pages 1425–1430, Anchorage, AK, USA, November 2002. IEEE.

[14] Lars Johannesson, Mattias Asbogard, and Bo Egardt. Assessing the potential of predictive control for hybrid vehicle powertrains using stochastic dynamic programming. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):71–83, 2007.

[15] Edward Dean Tate Jr, Jessy W Grizzle, and Huei Peng. Shortest path stochastic control for hybrid electric vehicles. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 18(14):1409–1429, 2008.

[16] MJ West, CM Bingham, and N Schofield. Predictive control for energy management in all/more electric vehicles with multiple energy storage units. In *IEEE International Electric Machines and Drives Conference, 2003. IEMDC'03.*, volume 1, pages 222–228, Madison, WI, USA, July 2003. IEEE.

[17] Balaji Sampathnarayanan, Lorenzo Serrao, Simona Onori, Giorgio Rizzoni, and Steve Yurkovich. Model predictive control as an energy management strategy for hybrid electric vehicles. In *ASME 2009 Dynamic Systems and Control Conference*, pages 249–256, Hollywood, California, USA, October 2009. American Society of Mechanical Engineers Digital Collection.

[18] Roberto Cipollone and Antonio Sciarretta. Analysis of the potential performance of a combined hybrid vehicle with optimal supervisory control. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 2802–2807, Munich, Germany, February 2006. IEEE.

[19] Lorenzo Serrao and Giorgio Rizzoni. Optimal control of power split for a hybrid electric refuse vehicle. In *2008 American Control Conference*, pages 4498–4503, Seattle, WA, USA, June 2008. IEEE.

[20] Lev Semenovich Pontryagin. *Mathematical Theory of Optimal Processes*. Routledge, 2018.

[21] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena scientific Belmont, MA, 1995.

[22] Hans P Geering. *Optimal Control with Engineering Applications*. Springer, 2007.

[23] Gino Paganelli, Thierry Marie Guerra, Sébastien Delprat, Jean-Jacques Santin, Michel Delhom, and Emmanuel Combes. Simulation and assessment of power control strategies for a parallel hybrid car. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 214(7):705–717, 2000.

[24] Gino Paganelli, Gabriele Ercole, Avra Brahma, Yann Guezennec, and Giorgio Rizzoni. General supervisory control policy for the energy optimization of charge-sustaining hybrid electric vehicles. *JSAE review*, 22(4):511–518, 2001.

[25] Pierluigi Pisu and Giorgio Rizzoni. A comparative study of supervisory control strategies for hybrid electric vehicles. *IEEE Transactions on Control Systems Technology*, 15(3):506–518, 2007.

[26] Antonio Sciarretta, Michael Back, and Lino Guzzella. Optimal control of parallel hybrid electric vehicles. *IEEE Transactions on Control Systems Technology*, 12(3):352–363, 2004.

[27] Bo Gu and Giorgio Rizzoni. An adaptive algorithm for hybrid electric vehicle energy management based on driving pattern recognition. In *ASME 2006 International Mechanical Engineering Congress and Exposition*, pages 249–258, Chicago, IL, USA, November 2006. American Society of Mechanical Engineers Digital Collection.

[28] Cristian Musardo, Giorgio Rizzoni, Yann Guezennec, and Benedetto Staccia. A-ecms: An adaptive algorithm for hybrid electric vehicle energy management. *European Journal of Control*, 11(4-5):509–524, 2005.

[29] Mingyu Sun. Artificial neural networks control strategy of a parallel through-the-road plug-in hybrid vehicle, 2019.

[30] Stuart Dreyfus. Richard bellman on the birth of dynamic programming. *Operations Research*, 50(1):48–51, 2002.

[31] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[32] David Goldberg. Computer-aided pipeline operation using genetic algorithms and rule learning. part ii: Rule learning control of a pipeline under normal and abnormal conditions. *Engineering with Computers*, 3(1):47–58, 1987.

[33] MathWorks. How the genetic algorithm works. https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html.

[34] MathWorks. Genetic algorithm options. https://www.mathworks.com/help/gads/genetic-algorithm-options.html.

[35] Denso. Power-train systems diesel. https://www.denso.com/global/en/products-and-services/powertrain/diesel/.

[36] All About Circuits. Introduction to CAN (controller area network). https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/.

[37] Emission Test Cycles DieselNet. FTP-72 (UDDS). https://dieselnet.com/standards/cycles/ftp72.php.

[38] Emission Test Cycles DieselNet. EPA highway fuel economy test cycle (HWFET). https://dieselnet.com/standards/cycles/hwfet.php.

[39] Emission Test Cycles DieselNet. Dieselnet. https://dieselnet.com/.

[40] Olle Sundstrom and Lino Guzzella. A generic dynamic programming matlab function. In *2009 IEEE control applications,(CCA) & intelligent control,(ISIC)*, pages 1625–1630, Saint Petersburg, Russia, July 2009. IEEE.

[41] SJ Pachernegg. A closer look at the willans-line. Technical report, SAE Technical Paper, 1969.

[42] Sepp Hochreiter. Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[43] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[45] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, Vancouver, Canada, May 2013. IEEE.

[46] Towards Data Science. Illustrated guide to LSTMs and GRUs: A step by step explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[47] Emission Test Cycles DieselNet. Central business district (CBD). https://dieselnet.com/standards/cycles/cbd.php.