U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Optimizing Energy Efficiency for Train Operation Constrained to Scheduling

Agostinho Manuel Afonso da Rocha

Supervisor: Prof. Armando Luís Sousa Araújo

Co-Supervisor: Prof. Manuel João Sepúlveda Mesquita Freitas

Thesis submitted for the degree of Doctor of Philosophy

Doctoral Program in Electrical and Computer Engineering,

Department of Electrical and Computer Engineering

Faculty of Engineering, University of Porto

22nd July, 2020

Resumo

O uso ineficiente de energia elétrica é visto como um problema atual. A sociedade é totalmente dependente desta para o exercício das suas atividades diárias. O seu uso ineficiente provoca um excesso de produção que quando praticado a larga escala pode contribuir para problemas de maior magnitude tais como a escassez de recursos, aumento de custos de produção e manutenção assim como o aumento da poluição resultante da transformação da matéria prima. Uma vez que dela somos tão dependentes, é necessário intervir, implementando medidas que promovam o seu uso racional e eficiente. A atividade ferroviária é um exemplo de um sistema que é fortemente dependente de grandes quantidades de energia. Algumas das linhas existentes, devido à imensa procura, encontram-se a trabalhar perto da sua capacidade máxima, o que impossibilita o aumento da oferta, necessitando assim de intervenção a fim de tornar este sistema mais seguro e confiável. É nesta perspetiva que atualmente aparecem incentivos ao desenvolvimento de soluções que visam a redução da pegada ambiental, assim como a otimização do uso do sistema ferroviário.

A tese aqui apresentada propõe como solução, um algoritmo que pretende contribuir para a redução do consumo da energia de tração em linhas de ferrovia ou metropolitanas. O algoritmo foi desenhado com o objetivo de reduzir o custo energético associado às viagens entre estações. O seu desenvolvimento foi centrado em aplicações de tempo real a fim de poder vir a ser integrado num sistema de apoio à condução. O algoritmo apresentado integra um modelo dinâmico do veículo em conjunto com um algoritmo de otimização. A utilização do modelo dinâmico do veículo serve para o cálculo de perfis de velocidade, assim como a estimação das necessidades energéticas do percurso atual. Por outro lado, o algoritmo de otimização tem como principal função a procura da melhor solução face ao objetivo inicial do sistema, considerando restrições e condições atuais da viagem. Além do desenvolvimento do algoritmo de minimização de consumo de energia, e atendendo às necessidades atuais do sistema ferroviário, um novo objetivo foi adicionado, focado em reduzir os tempos de viagem, a fim de minimizar os atrasos que possam ocorrer.

Em conjunto com os desenvolvimentos anteriores, um segundo algoritmo foi também implementado com o objetivo de determinar os parâmetros para o modelo dinâmico do comboio. Este algoritmo procura, com base na comparação de dados simulados com dados reais, os parâmetros que

mais aproximam o modelo à realidade. Para a implementação deste algoritmo duas abordagens foram utilizadas. A primeira utiliza o método dos mínimos quadrados. A segunda utiliza a meta-heurística chamada Simulated Annealing. Em ambas as abordagens, o algoritmo utiliza dados recolhidos em viagens para determinar os parâmetros pretendidos para o modelo. Ao contrário da implementação anterior, a maior preocupação desde algoritmo não se centra na obtenção de uma resposta em tempo real, mas sim na obtenção de parâmetros capazes de tornar o modelo o mais realista possível.

Após analise dos resultados apresentados, verifica-se que os algoritmos implementados se encontram de acordo com objetivos inicialmente propostos, procurando contribuir para o desenvolvimento do sistema ferroviário, assim como o aumento da sua capacidade de resposta.

Abstract

The inefficient use of electrical energy is seen as an actual problem. Nowadays, society is totally dependent on electrical energy for the exercise of its daily activities. Its inefficient use leads to an excessive production and, when practiced on a large scale, it can contribute to problems of greater magnitude, such as scarcity of resources, increased production and maintenance costs as well as increased pollution caused by the transformation of raw materials. So, it is necessary to interview, in order to implement actions that promote rational and efficient use of energy. Railway activity is an example of a system that is heavily dependent on large amounts of energy. Some of the existing lines, due to immense demand, are working close to their maximum capacity, which makes it impossible to increase its capacity, thus requiring intervention in order to make this system more secure and reliable. It is in this perspective that there are currently incentives for the development of solutions aiming to reducing the environmental footprint, as well as optimizing the use of the railway system. The thesis presented here proposes, as a solution, an algorithm that aims to contribute to the reduction of train traction energy consumption. The algorithm was designed with the objective of reducing the energy costs associated with travels between stations on the railroad. Its development was focused on real-time applications in order to be integrated in a driver assistant system. It integrates a dynamic model of the vehicle together with an optimization algorithm. The dynamic model of the vehicle is used for speed profiles determinations, as well as to estimate the energy needs for the actual journey. On the other hand, the main function of the optimization algorithm is to search for the best solution within system main objective, considering current travel conditions and restrictions. In addition to the implemented algorithm, to reduce the consumption of traction energy, and meeting the current needs of the railway system, a new objective was added, focused on reducing travelling time. This second objective was designed with the purpose of minimizing any delays that may occur. Together with previous developments, other algorithm was also implemented with the purpose of determining parameters for the train dynamic model. This algorithm, based on the comparison between simulated and real data, searches for train model parameters which approximate the most the model to reality. For the implementation of this algorithm, two approaches were used. The first one uses the least squares method and the second one uses Simulated Annealing meta-heuristic. In both approaches,

the algorithm uses data collected on regular journeys to determine the desired model parameters. Unlike the previous implementation, the main concern of this algorithm is not focused on obtaining real time results, but on obtaining the better parameters for the most realistic possible model. After analyzing obtained results, it appears that the implemented algorithms are in accordance with the objectives initially proposed, seeking to contribute to the development of the railway system, as well as to increase its response capacity.

Agradecimentos

Ao longo do meu percurso, como estudante de doutoramento, cruzei-me com várias pessoas que, cada uma à sua maneira, marcaram esta fase. Aproveito para deixar algumas palavras de agradecimento.

Começo pelo meu orientador, o Professor Armando Araújo, que já tem vindo a acompanhar o meu percurso desde longa data. Agradeço-lhe todo o apoio prestado, não só na realização desta tese, como também por todo o conhecimento transmitido ao longo destes anos. Mais do que um professor e mentor tem-se mostrado como um amigo que pretendo conservar. Ao meu co-orientador pela sua disponibilidade, assim como por todo o apoio fornecido no decurso da tese.

Seguidamente, quero deixar também uma palavra de agradecimento ao Professor Adriano de Carvalho. Sendo alguém que também já vem a acompanhar o meu percurso, desde longa data, quero agradecer-lhe pela ajuda, pelas palavras motivadoras na hora do café da manhã assim como pelo contínuo interesse e apreciação do meu trabalho.

À Nomad Tech, pela disponibilidade para me atenderem sempre que necessário, sempre prontos a ajudar assim que uma dúvida surgia. Obrigado pelo tempo comigo dispensado . Sem o vosso apoio a realização deste trabalho teria sido muito mais complicada. Agradeço também ao programa doutoral iRail, que financiou este doutoramento através da bolsa PD/BD/114104/2015, pela oportunidade dada para enfrentar este desafio do sistema ferroviário .

À minha família, pelo apoio dado, não só nesta última etapa, como também por toda a educação à qual tive direito. À minha namorada, por me acompanhar de perto neste percurso com apoio e compreensão, principalmente nos momentos de maior pressão. Não me esqueço das horas "roubadas" para serem investidas no desenvolvimento desta tese.

Aos meus colegas de trabalho, Jorge Pinto, Ricardo Carvalho, Vítor Lopes e Vítor Morais. Todos os momentos de trabalho, companheirismo, discussão e lazer foram importantes na realização desta dissertação. Sem querer retirar importância aos restantes, direciono uma mensagem de maior agradecimento ao Jorge Pinto, colega de luta neste percurso, amigo com quem partilhei todas as conquistas nesta tese de doutoramento. Quero também englobar nestes agradecimentos o Professor Carlos Ramos, o Professor Rui Brito, o Professor Paulo Costa e o Professor António Martins pela

amizade e apoio dados. Todos os conselhos fornecidos foram úteis, grande parte deles nos momentos mais necessários.

Ficam por deixar algumas palavras de agradecimento a muitas outras pessoas com quem me cruzei em todos estes anos na Faculdade de Engenharia. A todos, sem me querer alongar, o meu muito obrigado.

Muito Obrigado

Contents

1 Introduction **1**

1.1 Contextualization 1

1.2 Problem Description 2

1.3 Objectives 3

1.4 System Requirements 4

1.5 Original thesis contributions 5

1.6 Thesis Structure 5

2 State of the Art **7**

2.1 Regenerative Energy Flow 7

2.2 Energy Efficiency in Railways 9

2.2.1 Optimal Control Theory 9

2.2.2 Literature Review 11

2.3 Parameter Estimation for the Train Model 16

2.4 Optimization Algorithms 20

2.4.1 Linear Programming 21

2.4.2 Genetic Algorithms 21

2.4.3 Simulated Annealing 23

2.4.4 Particle Swarm Optimization 25

2.4.5 Ant Colony Optimization 26

3 Optimal Speed Profile Simulator **27**

3.1 Train Model 27

3.2 Train Dynamics 32

3.2.1 Acceleration 32

3.2.2 Cruising 32

3.2.3 Coasting 33

3.2.4 Braking . . . 33
3.3 Train Motion Simulator . . . 34
3.3.1 Formulation . . . 34
3.3.2 Single Trip . . . 35
3.3.3 Model Integration . . . 45
3.3.4 Non constant accelerations . . . 48
3.4 Line Constraints . . . 50
3.4.1 Gradients . . . 51
3.4.2 Velocity Limits . . . 54
3.4.3 Neutral Zones . . . 57
3.5 Train Model State Machine . . . 60
3.6 Train Motion Simulator with Line Constraints . . . 62
3.7 Speed Profiles Generator . . . 65
3.7.1 Speed Profile Phases . . . 68
3.7.2 Solution Construction . . . 85

4 Optimization Algorithm **93**
4.1 Introduction . . . 93
4.2 Simulated Annealing . . . 93
4.2.1 Generation Mechanism . . . 96
4.2.2 Temperature Scheme . . . 97
4.2.3 Cost Function . . . 99
4.2.4 Acceptance Probability . . . 99
4.2.5 Stop Criteria . . . 100
4.3 Driving Assistant Algorithm . . . 101
4.3.1 Energy Consumption . . . 101
4.3.2 Travelling Time . . . 126

5 Parameters Estimation **141**
5.1 Introduction . . . 141
5.2 Methodology . . . 142
5.3 Least Square Methods Approach . . . 145

5.3.1 Theoretical demonstration 145
5.3.2 Method Application 148
5.3.3 Method Implementation 150
5.3.4 Results 153
5.4 Simulated Annealing Approach 157
5.4.1 Generation mechanism 159
5.4.2 Cost Function 161
5.4.3 Results 162
5.5 Methodology Comparison 174

6 Conclusion **177**

6.1 Main Conclusions 177
6.2 Future Work 179

List of Figures

1.1 Train Control Problem. 2

1.2 System Overview. 4

2.1 Regenerative energy flow in railways [5] 8

2.2 Train Trajectory 10

2.3 Genetic algorithm flowchart (adapted from [44]) 22

2.4 Simulated Annealing flowchart (adapted from [46]) 24

3.1 Train acting forces in uphill motion. 28

3.2 Train traction, braking and resistive forces vs. train velocity. 31

3.3 Optimal Speed Profile 34

3.4 Speed profile with three driving regimes - definition of variables. 35

3.5 Speed profile with four driving regimes - definition of variables. 40

3.6 Sequence of operations to determine speed profiles. 45

3.7 Train Model Integration. 46

3.8 Constant acceleration with low and high velocity - algorithm results with train model. 47

3.9 Example of velocity limits - data representation. 55

3.10 TMS algorithm using the train model state machine. 62

3.11 TMS algorithm - driving regimes sequence. 65

3.12 TMS algorithm flowchart. 66

3.13 Braking phase - graphical representation of the phase determination. 68

3.14 Braking phase: comparison between v_{bk}, current and following velocity limit, V_{MAX}. . 70

3.15 Braking phase: acceleration regime estimation. 71

3.16 Results for braking phase determination. 73

3.17 Acceleration phase: comparison between v_{op}, current and following velocity limit. . 75

3.18 Acceleration phase: braking estimation. 75

3.19 Acceleration phase: examples of speed profiles. 77

3.20 Coasting phase - graphical representation of the phase determination. 79
3.21 Coasting phase: comparison between v_{op} and velocity limits. 80
3.22 Coasting phase: examples of speed profiles. 81
3.23 Cruising phase: comparison between v_{op} and velocity limits. 83
3.24 Cruising phase: examples of speed profiles. 84
3.25 Acceleration phase: train velocity and position 88
3.26 Cruising phase: train velocity and position 89
3.27 Coasting phase: train velocity and position 89
3.28 Braking phase: train velocity and position 90
3.29 Train velocity vs. time graphic. 90
3.30 Train position vs. time graphic. 91
3.31 Train velocity vs. distance graphic. 91

4.1 Simulated Annealing flowchart [46]. 96
4.2 Algorithm Flowchart. 103
4.3 Line profile - gradients and velocity limits values. 110
4.4 Algorithm solutions - Speed profiles and cost function values. 111
4.5 Algorithm solutions – Calculated speed profiles and cost function value. 112
4.6 Line profile - gradients and velocity limits values. 113
4.7 Algorithm solutions - Speed profiles determined and cost function value 114
4.8 Algorithm's solutions – Calculated speed profiles and cost function value. 115
4.9 Line profile - gradients and velocity limits values. 116
4.10 Algorithm's solutions – Calculated speed profiles and cost function values. 117
4.11 Coasting phase: train velocity and position 118
4.12 Line profile - gradients and velocity limits values. 119
4.13 Algorithm's solutions - Speed profiles determined and cost function values 120
4.14 Coasting phase: train velocity and position 121
4.15 Coasting phase: train velocity and position 122
4.16 Line profile - gradients and velocity limits values. 123
4.17 Coasting phase: train velocity and position 124
4.18 Coasting phase: train velocity and position 125
4.19 Algorithm Flowchart. 129

4.20 Algorithm results for all scenarios simulated in journey 1. 134
4.21 Algorithm results for all scenarios simulated on journey 2. 137

5.1 Train Motion Simulator (TMS) basic operations. 143
5.2 Parameters estimation methodology block diagram. 145
5.3 Input data - velocity vs. time. 148
5.4 Algorithm flowchart for Least Square Method (LSM). 150
5.5 Velocity records for LSM algorithm tests 153
5.6 Velocity results using parameters from Table 5.3 - Measurements in red dashed line, TMS in blue line 155
5.7 Position results using parameters from Table 5.3 - Measurements in red dashed line, TMS in blue line 156
5.8 Simulated Annealing (SA) algorithm for parameters estimation. 157
5.9 Velocity results using parameters from Table 5.7 - Measurements in red dashed line, TMS in blue line 165
5.10 Position results using parameters from Table 5.7 - Measurements in red dashed line, TMS in blue line 166
5.11 SA cost function (5.19). 167
5.12 Velocity results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line 168
5.13 Position results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line 169
5.14 SA cost function (5.20). 170
5.15 Velocity results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line 172
5.16 Positions results using parameters from Table 5.9 - Measurements in red dashed line, TMS in blue line 173
5.17 SA cost function (5.21). 174
5.18 Results comparison. 174

List of Tables

3.1 Example of a gradients vector's input 51
3.2 Example of velocity limits input 54
3.3 Example of neutral zones location input 58
3.4 Results for speed profile phases 88

4.1 Algorithm configurations 108
4.2 Results comparison for case 1 113
4.3 Results comparison for case 2 116
4.4 Results comparison for case 3 119
4.5 Results comparison for case 4 123
4.6 Results comparison for case 5 125
4.7 Algorithm initializations. 133
4.8 Results for journey 1 136
4.9 Results for journey 2 138

5.1 LSM function organization 150
5.2 Train model parameters 152
5.3 LSM result 154
5.4 Searching area for each parameter. 159
5.5 SA initial configurations. 163
5.6 Initial points for the SA algorithm using cost function (5.19) 163
5.7 SA results for cost function (5.19) 164
5.8 SA results using cost function (5.20) 167
5.9 SA results using cost function (5.21) 170
5.10 Summary table of results 175

Acronyms and Symbols

Acronyms

ACO	Ants Colony Optimization
ANN	Artificial Neural Network
ATO	Automatic Train Operation
DAS	Driving Assistant System
DE	Differential Evolution
DP	Dynamic Programming
EA	Evolutionary Algorithm
ESS	Energy Storage System
GA	Genetic Algorithms
HGA	Hierarchical Genetic Algorithm
IBEA	Indicator - Based Evolutionary Algorithm
LSM	Least Square Method
MARK	Minimum-Allele-Reserve-Keeper
MLES	Maximum Likelihood Estimation
MMAS	MAX-MIN Ant System
ODE	Ordinary Differential Equation
OSP	Optimal Speed Profile
PMP	Pontryagin's Maximum Principle
PSO	Particle Swarm Optimization
RMSE	Root Mean Square Error
SA	Simulated Annealing
TMS	Train Motion Simulator

Symbols

a	Acceleration
A	Davis equation constant
a_{acc}	Acceleration on acceleration phase
a_{bk}	Acceleration on braking phase
a_{coa}	Acceleration on coasting phase
a_{cru}	Acceleration on cruising phase
a_{max}	Maximum acceleration
B	Davis equation constant
B_1	Maximum braking force
C	Davis equation constant
E_c	Energy consumed
E_{c_r}	Energy consumed measured
E_{c_s}	Energy consumed estimation
F_b	Braking force
Δt	Schedule time
Δt_r	Remaining time to travel
Δx_{acc}	Distance travelled for acceleration
Δx_{bk}	Distance travelled for braking
Δx_{coa}	Distance travelled for coasting
Δx_{cru}	Distance travelled for cruising
Δx_{total}	Total distance to travel
Δx_r	Remaining distance to travel
F_g	Gravitational force
F_r	Resistance force
F_{Total}	Total force
F_t	Traction force
M_e	Effective mass
γ	Mass correction factor

M_p	Passengers mass
M	Train mass
P_a	Maximum traction power
P_b	Maximum braking power
P_m	Mechanical power
v	Velocity
v_F	Final velocity
v_i	Train velocity at instant i
v_I	Initial velocity
v_{r_i}	Train velocity at instant i measured
v_{s_i}	Train velocity at instant i estimated
v_{bk}	Braking velocity
V_{Max}	Velocity limit
v_{op}	Cruising velocity
t_F	Final trip time
t_I	Initial trip time
θ	Line angle
T_1	Maximum traction force
x	Position
x_F	Arrival station position
x_i	Train position at instant i
x_I	Departure station position
x_{r_i}	Train position at instant i measured
x_{s_i}	Train position at instant i estimated

CHAPTER 1

Introduction

This first thesis chapter, the introduction, starts with a brief contextualization of the problem to be solved and the associated motivation. Then, the problem is contextualized and its main issues described, as well as the intended objectives and requirements. Finally, the main original contributions and conclusions are presented.

1.1 Contextualization

It is well known that the railway is one of the most used systems for freight and passenger transportation. Its widely use derives mainly from being a reliable, on time and secure transport system.

Unfortunately, operation and maintenance costs associated to railways have augmented over the years, and so the associated consumption of electrical energy has also increased. On the other hand, due to today high traffic, some infrastructures are near to its maximum capacity, debilitating their ability to respond to all costumers/system needs. So, associated to the iRail Program framework, and with the purpose of accomplishing modern railway system needs, this thesis presents the results of a research in methods capable of improving trains' energy efficiency and at the same time maintaining the defined train time table.

The iRail program - Innovation in Railway Systems and Technologies - has the purpose of trying to answer to the actual concerns related with railway systems, such as achieving a high capacity, cost-efficient and sustainable rail transportation system. The Thesis work covers the innovation program IP3 – Cost-Efficient, Sustainable and Reliable High Capacity Infrastructures, exploring how electronics and computer systems can contribute for the iRail program objectives, more precisely with this innovation program [1].

Pursuing these objectives, this thesis presents an algorithm that solves, in real time, for optimal train velocity in order to minimize energy consumption during train operation. This reduction is, therefore, an opportunity for the increase concerning the railway system capacity.

Thus, this thesis starts presenting a typical train driving cycle. This will enable to identify the train dynamics and, consequently, the amount of energy involved on a typical railway system. The work carried out so far allows to develop a Driving Assistant System (DAS) capable of presenting, in real time, to the train driver, the optimal speed profile between two consecutive stations. The DAS optimization algorithm tries to minimize train energy consumption and to maximize the use of regenerative energy constrained to the system schedules and line velocity limits.

1.2 Problem Description

The main problem that this thesis tries to solve is how to travel from two consecutive stations minimizing the consumed energy. Also, the time table must be accomplished, velocity restrictions in any part of the line should be respected, and maximum acceleration/deceleration, for passenger comfort, must be taken into account. Additionally, lines with some parts without electrical energy are also allowed as problem restrictions.

Fig. 1.1 shows a graphical representation of the problem.

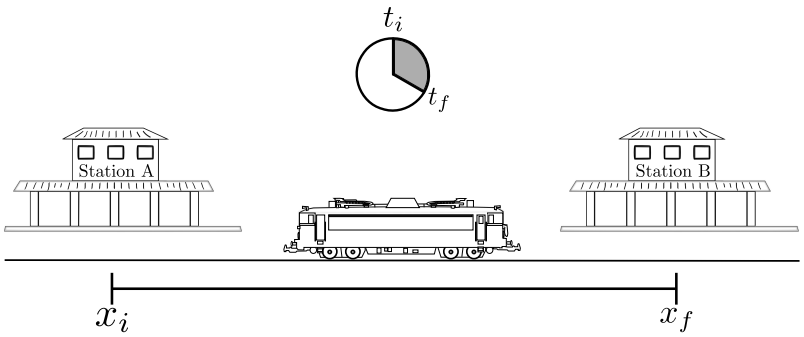


Figure 1.1: Train Control Problem.

It intends to exemplify a common journey between two consecutive stations. Being the sequence of departure and arrival from and into stations, in a journey, controlled by a train operator, it is expected that the train travels from an origin station (Station A) to an arrival one (Station B). It should be noted that, while in other types of transport systems the path may be unknown, on a railway journey the line is always known. Besides, since the railway line is known, the train operator usually has enough information about its characteristics, such as line gradients, velocity limits and any other constraints. More than defining stations sequence, the train operator must accomplish the timetables, which requires being aware of line availability dependent on the type of services performed.

In order to get a solution for the presented problem, this thesis focused on developing a software tool that determines speed profiles to be applied between two consecutive railway stations. Knowing all information about the journey, as the distance to be travelled, total time available and line physical characteristics (e.g. gradients and parts without electrical energy), speed profiles can be determined concomitant to one or more objectives. The ones chosen as the main targets of the developed algorithm was journey energy minimization constrained to the existing time table. In addition, it is also important not to forget that, in commercial services, train load consists mainly of passengers and, for that reason, the algorithm must be aware of maximum accelerations/decelerations allowed in order not to compromise passengers' comfort and security.

1.3 Objectives

As already mentioned, the aim of this work is the development of a tool with the purpose of achieving energy efficiency improvement in the railway system. This way, its objective will be met by developing solutions allowing both minimization of energy consumption and maximization of energy regeneration. To reach these objectives, the development of a DAS is proposed. As the name suggests, a DAS is a system that advices the driver about train operation in order to meet a specific purpose. These systems are typically installed in the cabin and, through a display, information about velocity, acceleration and braking phases are displayed. As velocities profiles generated by DAS are usually created considering energy consumption and/or time delays reduction [2], it is intended that the developed one should just do the same but with real time capabilities.

Based on this thesis proposal, and associated to Shift2Rail program, the following objectives were defined :

1. To make a research on the state of the art concerning train models;
2. To develop and implement a train model for simulation purposes;
3. To do research on state of the art's algorithms considering energy efficiency improvement in railway systems;
4. To implement a software tool that determines the best speed profile constrained to line restrictions;
5. To validate the developed algorithm through real data;

6. To develop a tool for parameter optimization for the developed train model;

1.4 System Requirements

Considering that one of the thesis main objectives, the development of an algorithm suitable to use on a DAS, and before presentation of the state of the art, as well as all thesis's developments, a small description about system requirements will be presented. The DAS has the main purpose of advising the train driver about the speed to use in the actual journey. So, the train driver takes actions over train controls so as to follow a specific speed profile. The determination of the speed profiles is achieved through a TMS, which is the algorithm responsible for the dynamic representation of the vehicle. An overview of the algorithm can be seen in Fig. 1.2

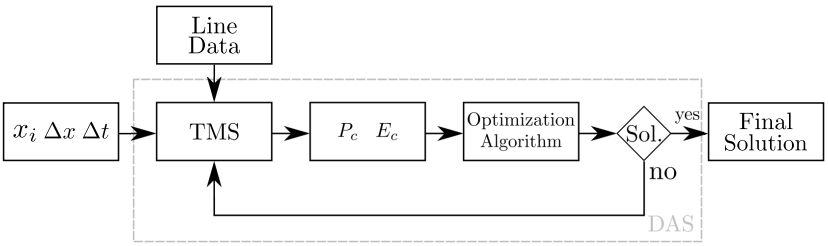


Figure 1.2: System Overview.

Starting with algorithm inputs, it must be ready to receive the position of the departure station (x_i), actual journey distance (Δx), as well as schedule time, (Δt). Total distance is defined by the distance between two consecutive stations, defined as stops, and the available time is defined on a timetable by the train operator. Besides time and distance, line constraints and its characteristics (line data) must also be available and used by the algorithm, once they will constrain energy needs as well as train maximum accelerations/decelerations and, consequently, the respective velocities. After data acquisition or, in other words, after all inputs read, the following step is to make a speed profile (TMS) according to all requirements initially imposed.

Depending on the system architecture, the optimal speed profile can be determined by following a set of rules or being selected from a pool of some options. Following the second hypothesis, the algorithm must include a mechanism responsible for selecting the best solution from a bunch of possible candidates. This mechanism is known as optimization algorithm and its implementation requires the definition of how new solutions are determined as well as how they will be evaluated. From state of the art's examples, DAS could determine velocity profiles with the main purpose of reducing energy

consumption and/or decreasing travelling time.

For the specific algorithm to be developed, another requirement was added: it is intended that the algorithm must run in real-time DAS. This means that a solution must be determined as fast as possible (minutes), preferably during train stop on departure station, being presented to the train driver before its departure. This additional requirement adds some difficulties to algorithm implementation and will be determinant in the decision about its structure.

1.5 Original thesis contributions

Based on the thesis main objectives inserted in the Shift2Rail scope, the contributions identified for this thesis work are:

1. The development of a Simulated Annealing based real time algorithm capable of the increase in railway system's capacity by reducing energy consumption in each train;
2. The fact that the algorithm optimizes timetables so reducing time delays;
3. The development of an algorithm, also based on Simulated Annealing, capable of making train parametric optimization.

1.6 Thesis Structure

This thesis is divided in six chapters. Chapter one presents a small introduction to the problem to be solved. Chapter two exposes the state of the art. This chapter is dedicated to the main objective of the thesis, a brief description about energy flow on a train, the actual state of the art over energy optimization in railway systems, the parameter estimation associated to the Train Motion Simulator and some optimization algorithms being presented. After the state of the art, the third chapter is mainly focused on the Optimal Speed Profile (OSP) generator. The chapter starts presenting the train dynamic model, followed by the demonstration of the way speed profiles can be determined. This is done by following OSP generator algorithm's timeline, starting with initial developments on simple cases, with simple constrains, showing the algorithm evolution and solutions adopted when new constrains appear. The chapter ends with an explanation about all developments for the last algorithm version. Chapter four introduces the optimization algorithm. This chapter starts with a detailed explanation of the developed algorithm, followed by the description of its implementation within OSP. Two versions

were implemented with different cost functions, both being both presented in detail and, for each one, the obtained results being presented. Algorithm validation is done comparing its results with real data results acquired in several journeys. The fifth chapter shows two techniques used to estimate the train's model parameters. The results for each technique are presented after each algorithm presentation and, before the end of the chapter, are compared in order to see its performance. Chapter six presents the main conclusions as well as some possible future developments, which can be taken into consideration so as to improve the current work.

CHAPTER 2

State of the Art

This chapter presents the state of the art related to the main subjects discussed in this thesis. It starts in Section 2.1, with a brief presentation about regenerative energy flow on railway systems. The chapter continues in Section 2.2, showing the research, found in the actual literature, associated to the tools used to obtain the optimum speed profiles for train operation. After covering this first part of the thesis, chapter continues in Section 2.3, with the research related to parameter estimation for dynamic train models. Finally, Section 2.4 presents five algorithms as possible candidates associated to the implementation of the optimization process.

2.1 Regenerative Energy Flow

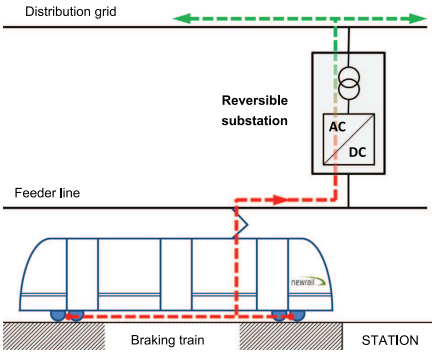
The study associated to the railway system, performed throughout this thesis, started with an analysis of the regenerative energy flow during train operation. Regenerative braking is frequently used as a technique associated to energy regeneration. As in many other systems, this technique uses train kinetic energy associated to decelerations, to generate electrical energy. This resulting recovered energy can be used in the system or stored properly [3].

The recovered energy, in the railway system, has three possible destinations has described in [4–6]:

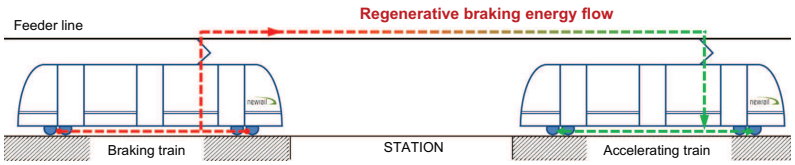
- Auxiliary systems power supply: feeding auxiliary systems installed in the train (air conditioner or door opening system, etc.);
- This typically happens when the amount of recovered energy is higher than the train internal auxiliary systems consumption. However, energy sent back to catenary must be consumed. Usually, there are two different destinations:
    - Grid injection, Fig. 2.1a. This energy injection, into the grid, implies that the substation is equipped with a bidirectional power flow converter;

- Reused, in the catenary, in the same line section, by another train, Fig. 2.1b. This application requires some synchronization between train departures and arrivals, to allow energy exchanges between two lines' "dead zones".

- Dissipated on braking resistors installed in the train.



(a) Grid injection



(b) Between trains

Figure 2.1: Regenerative energy flow in railways [5]

Looking at all the possible paths for regenerative energy flow, it can be concluded that there are several focuses of development in order to contribute to the optimization of the rail system. To maximize the regenerative energy recover and efficiency in the railway system, some developments on integration in Energy Storage System (ESS) are being taken into considerations. As already assumed, this thesis will not deal with regenerative energy maximization but with algorithms capable of minimizing the total energy consumption, thus contributing to maximize energy efficiency in the system. This leads to the next section, related to obtaining the optimal speed profile for this energy minimization.

2.2 Energy Efficiency in Railways

2.2.1 Optimal Control Theory

First developments on train speed profiles optimization started in the 70s, as mentioned in [7], with optimal control theory. In his PhD thesis, Milroy uses Pontryagyn's Maximum Principle (PMP) applied to a train running between two consecutive stations. The method is applied with the purpose of finding the OSP allowing minimum energy consumption [8–10]. One of the most important outcomes of Pontryagin's maximum principle, applied to this energy minimization problem was the shape definition and driving regimes sequence to be applied on a train's movement between two consecutive stations, which minimizes energy consumption. Four driving regimes were identified based on traction and braking force values [7, 11]:

- Acceleration — Driving regime applied, typically at the beginning of the journey. It applies maximum force to move the train from starting up to the so-called cruising velocity. This regime uses the maximum available force to maximize acceleration. When cruising velocity is reached, the train driver ends this regime.

- Cruising — After reaching cruising velocity, the next driving regime is characterized by constant force, which means zero acceleration and, consequently, constant velocity. The duration of this regime is a variable to be determined and defined by the problem's objective (consumed energy minimization).

- Coasting — To reduce energy consumption, a coast phase is added to the optimal speed profile. During this regime, no traction force is applied and, consequently, energy consumption is zero. The end of this period is also a variable to be determined (also constrained by problem objectives). In fact, good choices for coasting points shortens traction regimes. However, it tends to increase journey time, which is constrained to the schedule. So, the points in the journey where coasting should start and finish are very important in terms of minimizing energy consumption [12–14].

- Full braking — Driving regime applied at the end of a journey to guarantee the train stops at the scheduled time and correct position. This regime uses the maximum available braking force to maximize deceleration.

Figure 2.2 shows the four driving regimes, applied between two consecutive stations, associated with the OSP. This profile assumes a line with zero slope and no velocity constraints. As can be seen in the figure, the train's movement starts with a phase of maximum acceleration followed by a cruising one. The acceleration phase starts at departure station and ends when the train reaches the pre-defined cruising velocity. Cruising phase starts in the point where the acceleration phase ends, the same velocity value being maintained until the end of this phase. When the cruising phase comes to the end, the coasting one starts. This phase uses zero traction force so energy consumption is null. As expected, minimum energy consumption is reached by longer coasting regimes. Nonetheless, unfortunately, travelling time increases. Therefore, a period of coasting phase is determined as a trade-off between energy consumption and total travelling time. The last phase is the braking, which ensures station arrival at zero speed at the schedule time.

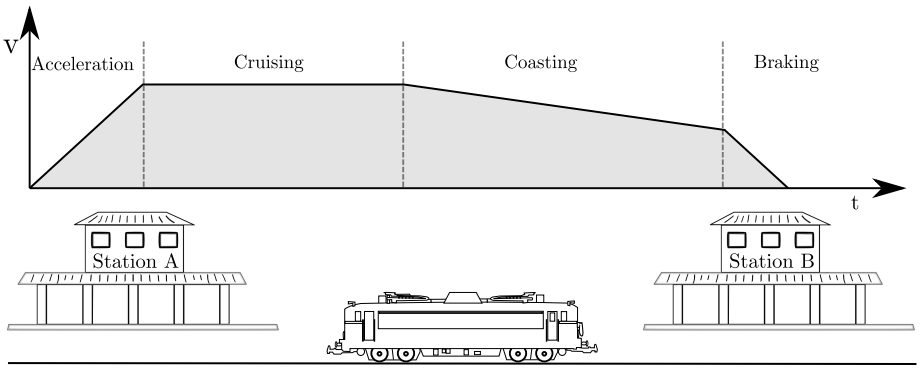


Figure 2.2: Train Trajectory

Although the optimum profile was determined considering a flat line, it can be also applied in the presence of small gradients. Nevertheless, the searching for OSPs for real situations must use line slopes and velocity limits. In fact, the presence of slopes and velocity constraints in the line will cause a break in each traction regime. As an example, the existence of a maximum velocity limit can interrupt a cruising phase and introduce a braking one. A new acceleration phase and another cruising one will then follow. Other changes to OSPs can happen in situations where, for instance, there is no space to apply all driving regimes, or a different sequence must be applied. This usually happens in suburban or metro services [7, 15, 16].

Therefore, using PMP for the solution to the general problem, associated with the identification of the points related to the four identified driving regimes, and to solve the resulting optimization problem, is a hard task and not usually compatible with real-time use. So, other approaches must be

tried in order to obtain OSP. Next, some algorithms capable of solving this problem are presented.

2.2.2 Literature Review

Some search algorithms have been recently used for the OSP problem when applied to real train pathways constrained to its velocity limits, gradients and timetables, with the purpose of energy consumption minimization. The most common applied meta-heuristics uses Artificial Neural Network (ANN), Ants Colony Optimization (ACO), Dynamic Programming (DP), Genetic Algorithms (GA), Particle Swarm Optimization (PSO) or SA.

In [12] the authors present a GA for coast point determination in a train pathway. It generates a new solution at each train stop, and before departure, producing a coast control table. The use of a coast control table is justified by the possibility to use it in an Automatic Train Operation (ATO) system. It searches for the optimum solution considering a multi-objective function. Each coast profile result is constrained to time punctuality, passenger comfort and energy consumption. The algorithm results for two different cases, based on schedules definition, were compared with a fuzzy logic control ATO.

The algorithm does not have a cruising phase, using several cycles of motoring and coasting, making the trip uncomfortable. Also, nothing is said about the used TMS and the possible line constraints. The algorithm is implemented offline with a pre-determined number of coasting points. This is so because simple GA do not have the capability to make this selection. Each point has an associated gene, with 10 bits, for distance discretization. This way, computation time is higher for longer trips if spatial resolution is to be maintained. Nevertheless, a C++ implementation of the algorithm typically runs in half a minute in an IBM 486-compatible PC. Therefore, the authors claim that it has potential for online implementation. The gains in energy, in the two reported cases, are small and about 7% and 3%, respectively.

Another GA is presented in [13]. It also locates coast points in a train pathway. At each stop, a new profile for the travelling distance between two consecutive stations is determined. It uses a binary string, with a variable length, dependent on the total travelling distance, to represent the coast point's position. The use of a binary string intends to reduce the complexity of mutation and crossover operations used during the optimization process. The authors also considered, beyond a single station-to-station optimization, a multiple-station scenario. For this last scenario, a Hierarchical Genetic Algorithm (HGA) was implemented. It uses previous string structure with an extra variable

representing the total coast points needed. The algorithm uses a Minimum-Allele-Reserve-Keeper (MARK) mutation scheme, in order to reduce processing time. Solution results use a cost function based on two parameters: schedule time and energy consumption.

The algorithm uses only one or two cycles of motoring and coasting and does not have the cruising phase. As a simple GA lacks the capability to select the number of coasting points, an HGA is tried, but only for two coasting points, so it does not guarantee a near optimal solution. Also, in order to meet the request for real time, a simpler mutation scheme is used. TMS and associated constraints are not mentioned. Presented energy gains are 30% but at the expense of 30% extra trip time, which is not always possible. The algorithm was implemented in Visual Basic so as to achieve a good human interface. However, Visual Basic is a proprietary programming language not easily transferred to other operating systems.

In [14], the search for an optimum speed profile, with energy consumption minimization, uses a GA together with an ANN. The purpose of the ANN is to substitute the train dynamic model. ANN inputs are a sequence of coast points and its outputs are total time and energy consumption. GA determines the best option of coast point sequence based on a cost function defined as a weighted sum of travelling time with energy consumption. Algorithm tests on a Turkish metro line with five stations and two lines, on a multi-train situation, proved its effectiveness.

In this case, the cruising phase is omitted once more. Train speed profiles are made with the use of proprietary software, namely SimuX. Being aware of the problems related to GA real-time implementation, namely the huge number of times that the TMS must be called (for the considered population size and number of generations 10000 times), the authors, considering its replacement, use an ANN without slope constraints. Nevertheless ANN has limitations as it is trained for a specific line with specific constraints and, therefore, is not flexible enough, e.g., online timetables or line speed limits change. Also, ANN training demands hundreds of offline simulations (made by SimuX) which are extremely time-consuming. Moreover, a search for optimal coasting points is carried out with Matlab GATool, making real-time implementation very difficult. The two simulated results show, respectively, a 30.85% energy saving, with a 4.81% increase in travelling time, and 18.25% less energy, associated with a 4.65% increment in travelling time.

In [17], the authors present another example with GA: speed profile determination based on a multi-population GA. The speed profile determination uses two phases. The first one applies travelling distance between two consecutive stations, in order to find the most economical scenario. The second

one considers the full trip. The searching process makes use of a multi-population scheme, which enables time travel reduction and avoids that the algorithm is stuck in local minima. Real data from a subway line section in Beijing with a total distance of 21 *km* was used to test the algorithm. The tests considered line gradients, curve radios and velocity limits.

The aim of the work is the minimization of total energy consumption between multiple stations. Inter-station trip time may vary but total time should be relatively constant. This is also achieved by finding the positions for switching between acceleration, coasting and braking. As usual in GA the cruising phase is missing. As general GA suffers from premature convergence, when associated with these kinds of problems, a multi-population GA is proposed. This has the problem of generating a considerable number of calls to the used TMS (20000 calls are expected for the 200 generations and 100 individuals proposed in the paper). Therefore, this will make its real-time operation hard. Additionally, the method assumes that times between consecutive stations can vary as much as 16% and, this is usually not the case. Nevertheless, results for the six simulated inter-stations show that the algorithm enables 6.16% energy reduction, keeping the total trip time.

Evolutionary Algorithm (EA) is another example of algorithms applied in railways. In [18, 19] the authors present a multi-objective one for velocity profile determination. The proposed algorithm is of Indicator - Based Evolutionary Algorithm (IBEA) type. It focuses on minimizing energy consumption and travelling time, on the one hand, and energy consumption minimization, total travelling time and delays reduction, on the other hand. It divides train pathway, limited by the distance between two consecutive stations, into several sections defined by velocity limits. For each section, it is defined a group of values for velocities profile solution (algorithm uses input and output velocities, as well as, the maximum and desired ones. It also uses velocity limits of next section). Two different lines in France served for algorithm validation.

Again, the main objective is associated with minimizing energy and travel duration. It uses a complex TMS, dividing the travel into sections according to line speed profiles. For each one, five variables have to be determined. The solution encoding needs a vector with triple the number of sections. The algorithm needs the ParadisEO framework and uses as termination criteria simulation time (60 seconds). Therefore, its real-time implementation outside the referred framework will be a challenge. However, simulation results show that energy decreases up to 54% can be achieved with 15% of increase in trip time.

A comparison between three methods is presented in [20]. The proposed problem is the search

for optimum speed values along the journey. Points with zero velocity and with speed limits changes are the ones where there is a need for velocity calculation. The studied algorithms, for performance comparison, were GA, ACO and DP, which were used in three situations with different trip times. In both situations, DP showed a better performance at the expense of an enormous computing complexity, so it cannot be used in real-time applications. Travel smoothness is poor for both the ACO and the GA algorithms mainly for longer travelling times. Also, in some of the studied cases, both the ACO and the GA algorithms were unable to find a solution.

Another example that uses ACO algorithm for speed profile optimization is presented in [21]. The generated speed profiles use a cycle of acceleration, cruising, coasting and braking. The solution is searched by means of a MAX-MIN Ant System (MMAS) in two optimization stages (the first stage is offline and its results are the reference for second-stage optimization). For its assessment, the algorithm used real train data acquired from a metro line in Beijing.

Algorithm simulation results show a total computing time of about one minute, being 40 s for the first stage optimization and the remaining 20 s for the second one. Energy saving rate is 14%. Therefore, for dwell times bigger than the second stage optimization time, the algorithm can be used in real-time applications.

The application of other nature-inspired searching algorithms, besides GA and ACO, was also recently investigated. In [22] a PSO-based algorithm, with a multi-objective function, was implemented in order to find the Pareto front for energy and trip time. It uses a train equipped with an ATO. The objective is the minimization of both energy consumption and running time. The algorithm generates a population of random command sequences and searches for a set of solutions, making a Pareto front of minimum energy and possible running times. Its parameter tuning and validation used tests on a flat track, in combination with a simple speed limit profile, in order to reduce computation time. Algorithm validation uses a line section of the Madrid's subway.

Considering the 20860 possible command combinations for speed profile generation, it needs 50 minutes of computation using a TMS previously developed by the authors. The obtained Pareto front with the multi-objective PSO-based algorithm has a computation time of about ten minutes. Therefore, it cannot be used for real-time applications. Also, the algorithm searches for pairs of energy/trip time so cannot be constrained to existing timetables. However, in some cases, it can reach energy savings of 20%.

In [23], a SA-based algorithm is presented, intends to reduce energy consumption in the metro

line between New York and Connecticut. The algorithm considers line gradients and velocity limits during the searching process. Energy consumption calculation, for each generated solution, uses a dynamic model of the train, previously developed by the authors. The TMS considers four motion regimes, accelerating, cruising, coasting, and braking. However, cruising is only applied if needed. The SA algorithm uses a cost function based on minimization of energy consumption and schedule travelling times. Moreover, it enables re-annealing and uses Metropolis criterion as the rule for solution acceptance. The initial value of temperature is 100 and temperature updates use an exponential schedule. Results of the algorithm are, for each travelling regime, maximum velocity, and time and position of each coast point.

SA-based algorithms need to define a good starting temperature but the paper does not refer to it. Furthermore, the initial solution and the method for generation of new solutions are not presented. A cooling schedule factor is also missing. These are very important factors required for good algorithm output. Regarding results, nothing is stated about computation time and obtained energy savings.

In [24], the authors present an algorithm that is a combination between GA and SA, a so-called Genetic Simulated Annealing Algorithm. The objective of joining both algorithms is to eliminate its individual weaknesses and to enhance its advantages. So, in a first stage GA, is used to create a good initial value solution for SA. In a second stage SA, is initiated. This is to avoid local minima that could occur if GA was only used. Matlab is used for running the TMS. It outputs speed, position and time vectors, and energy consumption of the obtained speed profile. Some stations of the Eskisehir light rail served for algorithm performance tests.

As already mentioned, GA and SA algorithms need well-chosen parameter sets. This way, for satisfactory results, a method has to be used for parameter tuning. Therefore, the authors used for this purpose a single 2000 *m* straight track with gradients, and made six test runs varying GA crossover and mutation rates, as well as selection and crossover function. The SA algorithm's annealing and temperature functions were also varied. Nevertheless, nothing is stated about the criteria used for these variations. The paper concludes that GSA can provide very good solutions but it has convergence problems, so the tuning of its parameters is crucial. Unfortunately, once again, there is no reference the gains in energy and total computational time.

From the previous paragraphs it can be conclude that, regardless of being scarcely used for the train energy minimization problem and only for offline solutions, the SA algorithm has showed its effectiveness in solving minimization problems. In fact, the most common limitations of current works

can be overcome by the use of the SA algorithm, which can solve problems with multi-constraints (comfort, speed limits, gradients, scheduling), can use of the optimal driving strategies for a train (acceleration, cruising, coasting and braking), is of easy implementation in a non-proprietary environment, has a reduced number of parameters, without need for any tuning, escapes from local minima, and can converge to an optimal solution in a small number of iterations. Also, fast calculation time enables its online use. Therefore, from the literature review, a solution for the train energy minimization problem with a new SA-based algorithm is developed in this thesis. Following, the state of the art in what concerns parameter estimation for TMS, needed for OSP generation, will be presented.

2.3 Parameter Estimation for the Train Model

OSP generation, as well as the estimation of total journey energy, requires the use of a train model, responsible for train movement description, which implies the determination of train mass and all acting forces. It also calculates the amount of energy involved during train regular operation. Determination of forces and energy calculation depend on the train dynamic model accuracy and the level of detail associated to the dynamic representation. This level of detail may be achieved by a model which is able to represent train behavior as similar as its reality. The dynamic representation is done by a set of equations, some with a physical meaning, and others that are empirical and parameter dependent. So, a few of them use coefficients to represent some kind of behavior and/or property. As expected, the train dynamic model results are dependent on how well the model is capable of representing the physical process, as well as to what extent the used parameters represent the chosen vehicle.

After analyzing the state of the art, I can conclude that dynamic models currently in use to represent train motion use a reasonable number of parameters, which can be divided in two groups: known and unknown parameters. Examples of known parameters are the ones given by train manufacturers, as train mass and traction and braking characteristic tables or graphs. On the other hand, coefficients associated to train resistance forces estimation are still subject of most studies associated to their determination. The model used in this thesis represents train resistance force using a quadratic function dependent on train velocity. This quadratic function is known in the literature as Davis equation and uses three coefficients to represent train resistance force. Two coefficients are related with train mass, having more influence on low velocities, and the third one is related with aerodynamic characteristics, so it has a higher weight for higher velocities. Looking at each parameter physical meaning, it is

expected that the value of each one is dependent on train physical characteristics. So, its determination is necessary when a change on vehicle characteristics occurs. Train modelling and its parameter estimation will be presented in Chapters 3 and Chapter 5.

Coefficients used on train model to represent train resistance force must be determined. According to CEN standards, their its determination must follow a set of methods (all requiring experimental tests and, consequently, measurements [25–27]). These methods can be summarized in three categories:

- Tractive effort methods: These are carried out when the train is moving. Resistance to the movement is then correlated to the energy spent on the traction.
- Coasting methods: This method uses a line with constant gradient. The train is accelerated until a pre-defined velocity. Then traction effort is set to zero and the resistance to movement is determined by estimating train deceleration;
- Dynamometer or draw-bar methods: These methods make use of a dynamometer to estimate train resistance in low velocities. They are typically used on constant gradient lines.

To perform all the tests associated to these methods, imposed by CEN standards, it is necessary to have adequate measurement equipment and a dedicated line. As so, this approach, for train parameter determination, being resource-dependent, introduces some practical difficulties. Nevertheless, estimation of these parameters is essential for developing the algorithms associated with ATO and DAS, as already seen. The train speed control must be precise in order to optimize energy consumption as well as to maintain passenger safety and comfort. Therefore, since the methods imposed by CEN standards required additional material and equipment, they are costly and so some alternatives, to these standards procedures, have been proposed. These alternative approaches are focused in techniques capable of reducing the costs associated to the typical methods coefficients determination. Also they intend to be as simple as possible, so reducing the associated complexity related to standard CEN methods. Thus, a brief review, of these methodologies, is presented in the following paragraphs.

A good overview to the discussion over methods to calculate train resistance force (Davis equation coefficients), is presented in [28]. The technique presented in this paper uses empirical relations for determination of Davis coefficients, based on physical train characteristics (such as masses, number of bogies, area and perimeter, etc.). The used rolling stock was the British Rail at the time. The method is compared with other approaches used by the French National Railways, SNCF, German railway and Japan. A comparison with measure data is also presented.

Another approach for train running resistance determination is made in [25]. The author claims for method simplicity, using the coasting energy technique. Nevertheless, it needs a rolling train and measurement equipment. The presented method can be used in tracks with variable gradients being suitable for trains operating in mountain areas.

Following the same purpose, in [26, 29] a method to determine the mass factor, representing the contribution of the inertia of the rotating parts, as well as coefficients for Davis equation, are presented. The technique used in the paper uses a full-scale coasting test. Authors present, as its main advantage, the independence on railway line characteristics as well as its accuracy. The method has only, on-board, time and position measurements. Train acceleration and velocity are determined from these data.

In [30, 31] a new method to determine the train model parameters is presented, determining coefficient values for as Davis equation and for the contribution of the rotating parts inertia. A set of tests, in more than one hundred train routes, were done in order to get experimental data. Parameters were determined using a technique known as IDIM-LS (Identification Dynamic Inverse Model – Least-Squares).

Another technique is exemplified in [32]. The method uses an Unscented Kalman Filter to estimate train resistance force. The filter was applied on a train coasting on a flat track. Parameters associated to the resistive force were determined only for the first and last term of Davis equation (Authors claim that the additional term, which multiplies the velocity, is small when compared to the remaining ones). The filter was tested using the train model with known parameters. Noise is then added to the generated speed profiles and this velocity sequence is used as an input to the algorithm. The results were compared with real measurements and the maximum relative error in the resistance force equation was determined.

Another process is presented in [33] in order to find train resistive force. Main focus of the method is the determination of the coefficients for the resistive force. These will then be used in an ATO system. Differing from the works previously presented, in this case, data used for coefficients estimation was acquired by the ATO system itself. So, the use of additional measurement systems, as well as any experiment in the train, were avoided. Parameter estimation was achieved using LSM in the acquired data. According to the authors, the use of LSM was not appropriate due to the fact that there is no opportunity to constrain the values of the coefficients. Thus, a modified version of the method has been implemented called multi-innovation least squares algorithm. This method is a

recursive one, which can be applied for on-line coefficients determination.

In [27], a new coefficients estimation method is shown using on-board telemetric system data. The method implemented determines the resistance force coefficients in an iterative process. At each iteration, a new solution for the coefficients is evaluated and a train model is used to estimate train energy consumption. Then, this energy is compared with measurements from a telemetric system. The algorithm implemented uses a formulation based on a minimization problem. The objective function is defined so as to minimize the sum of the square differences between estimated and measured energy.

Another technique to determine the train model parameters is presented in [34]. This paper uses a Maximum Likelihood Estimation (MLE) to determine train mass on inter stations journeys. Train resistance force is considered as known. The method is applied only to determine train mass. The algorithm output is analyzed by comparing its results with line measurements. Train operation data can be successfully used to estimate parameters values for train dynamic model.

In [35], the authors suggest a method to estimate train resistance force in tunnels, using data acquired during railway operation. Two different approaches, in order to find for train resistance parameters, were considered. The first one makes use of train acceleration. For this first approach, two hypotheses for acceleration determination were considered: one measured and the second one determined by velocity records. The second approach uses train velocity measurements and determines train resistive force by fitting the model output with the real measurements. In the end, comparing both approaches, it is concluded that the second one, using minimization of the error between estimated and measured velocities, for resistive force parameter determination, is the one that shows better results.

In [36], authors use acceleration measurements, from high speed trains, in order to find resistive force parameters. The study took place on Korean high-velocity lines. The velocity measurements were taken and, posteriorly, train acceleration was determined. Velocity measurements were carried out during regular operation. Some specific track positions were selected as well as coast commands between defined velocities. Due to the requirement to meet schedules, coasting phases, and consequently, measurements were taken in different line sections. After collecting all measurements, train resistance to motion was determined by fitting train acceleration to a polynomial function, in open air as well as tunnels.

A study on resistance force in Korean high-speed trains, similar to the one presented in [36], is found in [37]. In this study, some coasting tests are performed and resistance force coefficients

are determined based on a polynomial fitting curve. Results from coasting tests are compared with coefficients determined in real conditions and some empirical methods.

Currently, the coefficient estimation for train models is an unexplored area, since the amount of published work is not as high as expected. The problem itself can be defined generally as parameter estimation on Ordinary Differential Equation (ODE) system. The estimation can be done by fitting experimental data to the model. Data can be obtained during train regular operation or experimental tests in dedicated tracks. This estimation be achieved applying some mathematical tools, as presented previously, or by applying an optimization algorithm. The main idea is to reformulate the estimation problem as an optimization one. Then parameters are estimated by comparing real data with the train model output. Resulting error is then used for, iteratively, try to converge for a near optimal solution.

A review of optimization methods applied to parameters estimation for ODE models is offered in [38]. Authors present a survey on optimization algorithms for systems biology, presenting a list of the possible ones that can be applied to these problems. The list includes SA, GA, EA and Differential Evolution (DE). The article describes each one of these algorithms and shows possible algorithm definition and implementation. In the end, a comparison between all methods is made. Another example of some algorithm as SA, PSO GA are presented in [39–43]. In all these papers, it is visible that input data is needed and, independently to what algorithm is selected, the problem must be well defined in concerning the way the new space of solutions is generated as well as how the cost function is built. As this thesis uses one of the mentioned algorithms, namely SA, the next section is dedicated to a brief review of some of these algorithms.

2.4 Optimization Algorithms

Optimization algorithms can be used to solve problems that are usually defined by a mathematical model subjected to a set of constraints. The algorithm starts by choosing the so called design variables that will be changed during the process of optimization. Then it must formulate the associated constraints. These describe functional associations between design variables and/or other design parameters imposed by any physical or resource limitation. The next step involves the formulation of an objective or cost function, depending on the design variables. This objective function is then iteratively minimized, or maximized, with the purpose of finding a solution that is optimal, in some sense, to the problem requirements. The optimal solution, for an optimization algorithm, is considered found when a set of values for the design variables, is determined within algorithm constraints.

This solution must satisfy some termination criteria, such as minimum/maximum values for design variables or maximum number or iterations [44].

From the state of the art's analysis, and since the thesis's objectives are related with energy efficiency optimization, some algorithms were considered, as possible candidates, to be applied in the solution of the proposed problem. Bearing this in mind, in this section some of them will be presented. There are two distinct types of optimization algorithms: deterministic and stochastic. Deterministic ones use specific rules when generating new solutions. Stochastic algorithms use probabilistic rules when moving between solutions. Most of them try to imitate some rules associated to nature. In the following section, some of these algorithms are briefly presented, namely linear programming, GA, SA, PSO and ACO.

2.4.1 Linear Programming

Linear programming, as the name suggests, is a technique used to find optimal solutions for linear problems. Linear problems are defined by linear objective functions and constraints dependent on design variables.

When the problem is defined by a low number of design variables (between 2 and 4), the optimal solution can be determined through a graphic method. The optimal solution is obtained by a graphical visualization of the problem, using a 2-D or 3-D reference frame.

To increase the range of applications of linear programming, a standard form can be used in order to convert a non-linear problem into a linear one. In the standard form, the restrictions are reformulated as equalities and the problem is converted into a minimization one. To rewrite the problem in standard form, some transformations must be carried out. In [44], it can be found a good explanation of the necessary steps that must be followed during problem transformation.

One well-known linear programming technique is the simplex method. This method is applied in problems with high number of design variables. The method searches for the optimal solution by moving, iteratively, from each feasible solution to another one. Feasible solutions are the solutions that accomplish all the constraints imposed by the problem formulation [44].

2.4.2 Genetic Algorithms

Based on evolutionary theory, GA are searching algorithms that look for the optimal solution through a space of solutions. The optimal solution searching process uses a group of points, instead

of a single one, and so GA are also known as global optimal searching algorithms. The group of points used for the solution searching is called population. For each individual of this population, a fitness value is iteratively determined in order to understand how the latter adjusts to the former. This is sometimes regarded as a disadvantage since algorithm processing time increases with the number of individuals. Furthermore, a group search does not prevent the algorithm from being trapped in a local solution instead of searching for the optimal one [45].

Even before starting algorithm description, some definitions must be considered. The algorithm search, as referred, runs based on a population. Each population is composed by several individuals and each individual is structured by genes. Genes are considered the most basic elements in the algorithm. In a GA, and before it runs, an assumption for the population must be reached. This assumption defines the population codification, a process which happens during its construction. Each individual is codified, being converted into a string of bits. This makes it easier to do some operations associated to the searching process, such as mutation and gene crossover. The GA algorithm flowchart is presented in Fig. 2.3.

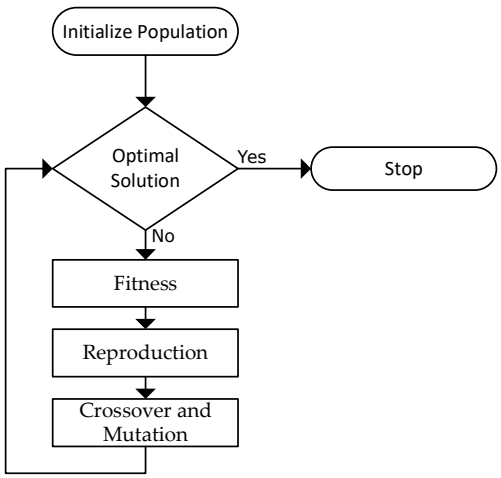


Figure 2.3: Genetic algorithm flowchart (adapted from [44])

As can be seen, a GA is a hierarchical algorithm that tries to follow the rules of evolution, just like in nature, thus, the best information, selected at each iteration, passes through generations. As so, during the searching process, the population suffers some mutations based on best individuals, defined by the best fitness value. The purpose is to generate the next population, used in the following iteration, based on the best individual characteristics.

The algorithm starts defining initial values for the population, which is followed by the determination of its fitness value. Based on the obtained fitness values, the best individuals are identified and selected, the remaining ones being dropped away. The next step is reproduction, where a new population is created using the best genes from each individual. To allow for changes in new generations, some genes exchanges must occur. This happens in the step called "crossover", where individuals are randomly selected to cross genes. As result of these operations, a new population is generated and it will be used at the next algorithm iteration.

The GA algorithm admits that the best population is generated by the best individuals. So, the algorithm evolves by continuously mutation over best individuals. The optimal solution is found when there are almost no variations among several individuals of the population [44, 45].

2.4.3 Simulated Annealing

The SA algorithm tries to imitate the metallurgical process of annealing, more specifically the thermodynamics associated to the process of heating a material, above the point of recrystallization, and then cooling it down slowly, in order to obtain perfect crystalline structures. Fig. 2.4 shows the associated flowchart.

The SA algorithm searches for the optimal solution in an iterative process. It starts with parameter initialization. Then, iteratively, a new solution of the problem is tried, starting with this initial set of values. The new solutions are accepted or not, based on the value they obtain, associated to the defined objective or cost function. If the current cost function value is better than the last one, the new solution is accepted and stored. If not, the current solution can still be accepted based on the value of a probabilistic function, usually associated to the Boltzmann distribution [46]. This probabilistic function uses a parameter, called temperature, T, analog to the physical temperature of the annealing metallurgical process. In each iteration, temperature is decreased, according to some scheme, normally a geometric progression of the type $T_{i+1} = \alpha T_i$, where α is the so-called cooling factor, which typically assumes a value somewhere between 0.8 and 0.9. This temperature reduction makes the probability to accept a worst solution, this is to say a solution with higher cost value than the previous one, higher at the beginning. This is the main feature of the algorithm that enables it to escape from local minimum. As temperature decreases, the probability of accepting worst solutions also decreases.

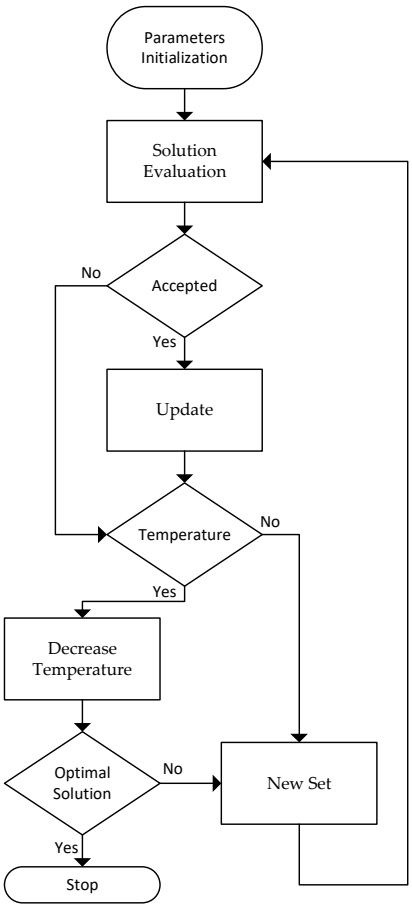


Figure 2.4: Simulated Annealing flowchart (adapted from [46])

Finally, the algorithm must have a stopping criteria. Some of the possible options are:

- Maximum number of iterations;
- Minimum value of temperature;
- Optimal solution found.

The first two options are easy to define. The last one it can use some measuring criteria associated to the evolution or value of the cost function. If, after a pre-determined number of iterations, the cost function fails to accomplish this criterion, the algorithm stops.

2.4.4 Particle Swarm Optimization

Particle Swarm Optimization, PSO, is an optimization algorithm that searches for an optimal solution in a candidate space of possible ones. This algorithm is inspired in the movement of a group of animals, such as bird flock or fish schooling. This way, searching for solutions, inside the space of candidates happen as a swarm behavior. Thus, the algorithm keeps tracking the best individual positions in the population of particles. So, updating a particle position is influenced not only by its actual position but also by the best known in the candidate solution space, of all other particles. This will likely moves the particles in the direction of the optimal solution.

As the already presented algorithms, PSO starts with the initialization of some variables. In this case, the definition of the number of the population particles and its individual positions are the initial tasks to be done. The individual positions are initialized using a uniformly distributed random vector:

$$x_{i,k} = x_{i,min} + (x_{i,max} - x_{i,min})\, u_i \tag{2.1}$$

Since the PSO algorithm is an iterative searching process, the evaluation of new solutions and new particles definition is repeated until the cost function minimization is accomplished. So, at each iteration, the position of each individual changes and is updated by:

$$x_{i+1,k} = x_{i,k} + v_{i+1,k} \tag{2.2}$$

It should be highlighted that this new position is constrained to the search space boundaries. In (2.2), $v_{i+1,k}$ represents the velocity of the individual that must be updated in each iteration. The individual velocity is characterized by three components: inertia, personal and social influence. These parameters must be chosen by the user and are fundamental for a good behavior and efficacy of the PSO method. Thus, the formula for velocity update is:

$$v_{i+1,k} = w_1 v_{i,k} + \phi_1 (p_{xik} - x_{i,k})\, u_i + \phi_2 (g_{xi} - x_{i,k})\, u_i \tag{2.3}$$

In (2.3), p_{xik} is the best individual and g_{xik} the global fitness that are found using [44]:

$$p_{i+1,k} = f(x_{i+1,k}) \tag{2.4}$$

$$g_{i+1} = min(p_{i+1,k}) \tag{2.5}$$

The next step is the update of the particle and swarm's best-known positions. The termination criterion is usually associated to a maximum number of iterations or to a pre-defined value for the objective function.

2.4.5 Ant Colony Optimization

Just like the previous algorithm, ACO also has a nature analogy: pheromone-based trail-following ant's behavior in searching for food. The algorithm starts by defining the number of ants, layers and nodes. The number of layers is associated to the design variables while nodes are related with its values. As it happens in an ant colony, when ants are searching for food, they must pass through all the nodes, searching for the best solution, defined by the objective function. During the searching process, the best solution tends to get more ants. As it happens in nature, each ant leaves a trail with the purpose to guide other ants. So, in the algorithm, the same occurs by updating a variable called pheromone with the purpose to guide all the ants to the best solution. Defining N as the number of ants, the probability of node j to be selected is determined by:

$$p_{ij}^{k} = \frac{\tau_{ij}}{\sum_{j \varepsilon N_i^k} \tau_{ij}} \tag{2.6}$$

In (2.6), τ_{ij} represents the trail. It can be determined by:

$$\tau_{ij} = \tau_{ij} + \Delta\tau^{k} \tag{2.7}$$

The trail is updated during the searching process by:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{N} \frac{\phi f_b}{f_w} \tag{2.8}$$

In (2.8), ρ is known as the evaporation rate (typically equal to 0.5). f_b and f_w are, respectively, the best and the worst values of the objective function.

ACO has several applications in the field of large combinatorial optimization problems, as optimal vehicle routing and project scheduling [20, 44].

CHAPTER 3

Optimal Speed Profile Simulator

This chapter will present the developed OSP algorithm. This procedure is responsible for the creation of the speed profiles that will be evaluated by the optimization algorithm. The algorithm uses, as its inputs, given values for cruising and braking velocities, v_{op} and v_{bk}, respectively. The chapter starts in section 3.1, by explaining the adopted train model, followed by the presentation of the four different dynamic driving regimes, established in section 2.2.1. Next, TMS formulation is carried out, followed by the introduction of the three considered line constraints: gradients, velocity limits and neutral zones. Finally, in sections 3.5 and 3.7, the TMS state machine and the construction of the speed profile are exposed.

3.1 Train Model

The implementation of an algorithm capable of creating speed profiles requires a dynamic model of the train. This train dynamic model has the purpose of simulating, using a mathematical model, the vehicle behavior, concerning its speed along time. So, in agreement with the thesis objectives, a dynamic model will be presented, allowing to compute velocity profiles, as well as, time, travelled distance and energy needs.

In this thesis, train dynamics is modelled using the mass-point model, as presented in [47, 48]. This model assumes the motion of a train with distributed mass, as the motion of a point coincident with the train center of mass. Despite its simplicity, this model was selected for usage because it can be reproduced with precision train behavior. Moreover, being a less complex model, less processing requirements, as well as time, are required, which is an advantage since it is intended to be applied in real time simulations. The forces acting on the train are presented in Fig. 3.1. An uphill situation is considered.

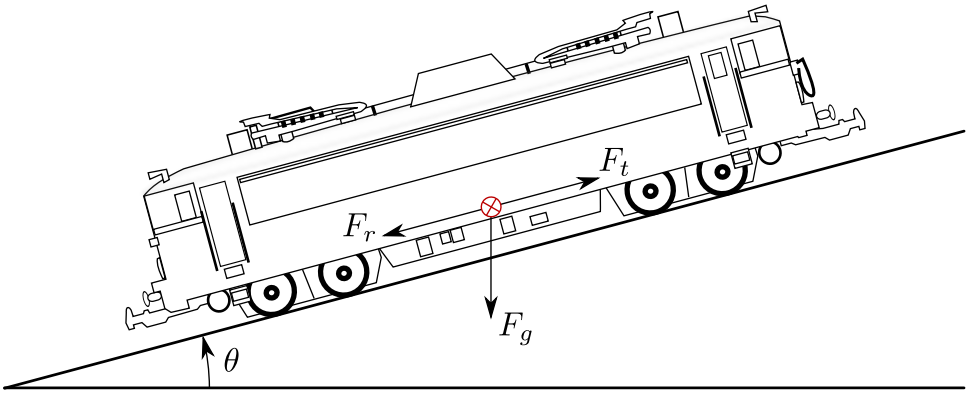


Figure 3.1: Train acting forces in uphill motion.

These acting forces, associated to its movement, are:

- Traction Force - $F_t(v)$
- Resistance Force - $F_r(v)$
- Gravitational Force - $F_g(s)$.

Knowing all forces acting on train, total force can be defined as (3.1).

$$F_{Total} = F_t(v) - F_r(v) - Fg(s) \tag{3.1}$$

Force $F_t(v)$, in Equation (3.1), can represent both a traction or a braking force. When a traction force is needed, $F_t(v)$ assumes a positive value. When in a braking situation, its value is negative. Three situations can occur, influencing the type of train movement:

1. $F_t(v) - F_r(v) - F_g(s) > 0$
2. $F_t(v) - F_r(v) - F_g(s) = 0$
3. $F_t(v) - F_r(v) - F_g(s) < 0$

Case 1, commonly happens when the traction force value exceeds resistance and gravitational ones. The result is a positive acceleration and, consequently, the train increases its velocity. In the second case, resulting total force is zero. This case results in a zero acceleration, causing a constant

velocity. Last case typically occurs when resistant forces are bigger than traction or when a negative traction force is applied (normally associated to regenerative braking). The result is a negative acceleration and, accordingly, train reduces its velocity. Having all acting forces defined, it is possible, by Newton's second law, (3.2), to establish vehicle acceleration:

$$F_{Total} = Ma \tag{3.2}$$

As it is well known, equation (3.2) states that acceleration, a, of an object, with mass M, is caused by the sum of all applied forces it, F_{Total}. In the studied case, which uses the mass-point model, train mass, in equation (3.2), is usually replaced by the so-called effective mass, M_e, which allows to take into account the mass of the train's rotating parts. As an exact value of the influence of the rotating parts is of difficult calculation, train effective mass is established considering total mass and a correction factor, γ, as shown in (3.3), M_e being the equivalent mass.

$$M_e = M(1+\gamma) \tag{3.3}$$

The γ factor typically takes values between 0.06 and 0.11, accordingly to the type of train [49]. To properly determine train velocity, time and distance travelled, passengers' mass, M_p, must also be considered in the dynamic model. Bearing in mind passenger and/or load masses, the total mass to be used in equation (3.2), is:

$$M_{total} = M(1+\gamma) + M_p \tag{3.4}$$

Forces opposing the train movement are namely resistance and gravitational ones, respectively F_r and F_g, and we will first analyze the former. Resistance force, F_r, is commonly represented by Davis equation (3.5). This states that this force is the sum of three terms: a constant and two others, depending on train velocity:

$$F_r = A + Bv + Cv^2 \tag{3.5}$$

Davis equation is widely used because it simplifies the calculus of resistive forces, describing it as a polynomial function. So, this force will change nonlinearly with train velocity. Each parameter, A, B and C, have a different physical meaning and they are specific to each train. Parameter A is independent of train velocity and represents the resistance at start-up. Second parameter, B represents

resistance dependence on velocity, influenced by the mechanical transmissions. It can be stated that A and B are parameters which dependent directly on the trains mass. The last parameter, C, multiplies the speed square and depends on the train's shape. This one is due to the aerodynamic resistance. [50]. Fig. 3.2 shows representative curves of a train resistive force (black), as well as, its traction (blue) and braking (red) ones. Finally, the gravitational force will be discussed. This is due to tracks with nonzero slopes and will also clearly influence the train dynamics. This influence is represented by a gravitational force, F_g, which depends on the train mass and track angle, θ. Equation (3.6) is used for its calculus:

$$F_g = Mgsin(\theta) \tag{3.6}$$

On railways lines, slopes gradients are frequently small, and, in that case, $sin(\theta)$ is normally estimated as θ. This approximation simplifies the train model since it avoids the nonlinearity due to the trigonometric function. In this document, and in model implementation, a climb was defined as a positive gradient while a descent as a negative one. In order to finish train modelling, the forces involved in traction and braking, respectively, F_t and F_b, are now discussed. Traction and braking forces, F_t and F_b, are also dependent on the train's velocity. In the developed model, these forces are both represented by F_t, since both are connected with the train motors. The only difference to consider is that F_t is positive when has to accelerate, and F_t is negative when a deceleration is needed, and the train's braking occurs. If the associated braking converts the kinetic into electrical energy, this is known as regenerative braking and contributes to energy minimization. Both forces follow the traction and braking characteristics curve of the motors installed on the train. Traction and braking curves can be divided in two different zones: constant force and constant power. Constant force happens for low velocities and, in this case, the train's traction force is limited to a maximum constant value for any velocity. On constant power zone, as the name suggests, the train's motors maintain a constant power, which results on a maximum force available reduction, with increasing velocity. Traction and braking characteristic, of the train considered in this work can be seen in Figure 3.2

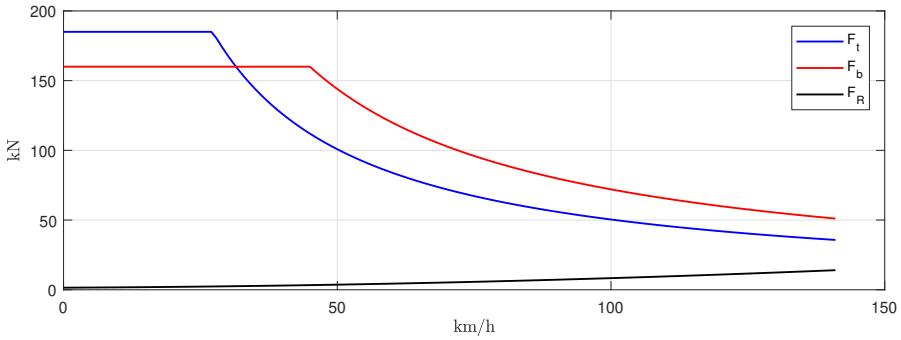


Figure 3.2: Train traction, braking and resistive forces vs. train velocity.

These curves are given by the train maker. For its implementation into the model, equations (3.7) and (3.8) were used:

$$F_t = \begin{cases} T_1, & \text{if} \quad v \leq V_a, \\ \\ \dfrac{P_a}{v}, & \text{if} \quad v > V_a. \end{cases} \tag{3.7}$$

$$F_b = \begin{cases} B_1, & \text{if} \quad v \leq V_b, \\ \\ \dfrac{P_b}{v}, & \text{if} \quad v > V_b. \end{cases} \tag{3.8}$$

T_1, in equation (3.7), it the motor maximum available traction force applied during constant force regime. P_a is the maximum power traction available limiting traction force, during constant power region. V_a is the boundary velocity associated to the change between both regimes. Equation (3.8) has B_1 as the maximum braking force, P_b consists of the maximum braking power, and V_b is the boundary velocity between both regimes. Finally, mechanical power, equation (3.9), is obtained as the product between traction force and train velocity:

$$P_m = F_t \times v \tag{3.9}$$

Mechanical energy consumption is achieved applying the usual relationship, namely equation(3.10):

$$E_c = \int_0^T P_m(t)\partial t \tag{3.10}$$

Having finished model presentation, train motion simulation, determination of velocity profiles and estimates related to the amount of energy needed in each trip will follow. First, model formulation

that enables calculus of acceleration, for each of the four regimes typically associated to speed profiles will be presented, as already seen in 2.2.1.

3.2 Train Dynamics

For an optimal speed profile, four different driving regimes must be considered [8–10], [7, 11]. So, this section intends to present how acceleration may be determined in each regime. It starts with the so-called acceleration phase.

3.2.1 Acceleration

The acceleration phase is used to speed up the train from an initial velocity to a final one. This way, in this regime, acceleration must be positive. During this regime, the train driver shall apply enough tractive force, $F_t(v)$, to overcome all forces opposed to the movement. As a result, train accelerates. Acceleration in this phase is given by equation (3.11):

$$a(t) = \frac{F_t(v) - F_g(s) - F_r(v)}{M(1+\gamma) + M_p} \tag{3.11}$$

It should be noted that equation (3.11) only results in a positive acceleration if total force, sum of all individual forces, is greater than zero. So, $F_t(v)$, should satisfy:

$$F_t(v) > F_g(s) + F_r(v) \tag{3.12}$$

3.2.2 Cruising

The cruising driving regime is also known as holding speed phase. This speed maintenance is the result of a zero train acceleration, (3.13). Zero acceleration is achieved when the train driver applies a traction force equal to the sum of all resistive forces (3.14).

$$a(t) = 0 \tag{3.13}$$

$$F_t(v) = F_g(s) + F_r(v) \tag{3.14}$$

In fact, during this driving regime, both traction or braking force can be applied. If the sum of all forces acting on the train, $F_g(s)$+$F_r(v)$, is a negative number, a tractive force of equal magnitude shall be applied. Instead, if the sum is positive, the driver must press the brake.

3.2.3 Coasting

During this driving regime no tractive or braking forces are applied by train motors (3.17). So, the only applied forces are opposing to the movement and, as result, train decelerates. Since no traction force is applied during this driving regime, it is commonly used with the purpose to reduce energy consumption. Defining traction force equal to zero, during coasting driving regime, train acceleration can be determined as presented in equation (3.16).

$$F_t(v) = 0 \tag{3.15}$$

$$a(t) = \frac{-F_g(s) - F_r(v)}{M(1+\gamma) + M_p} \tag{3.16}$$

During coasting driving regime, the train is usually subjected to a slow deceleration, consequence of a negative sign on equation (3.16). Nevertheless, although the train resistance force will always provoke a train deceleration, track slope can change this result. In fact, although on a flat or uphill situation train deceleration will increase, on a downhill track, when gravitational force is higher than resistive one, the train will accelerate.

3.2.4 Braking

The last regime presented is braking, commonly applied when the train is arriving at destination, in order to stop it safely. In this regime, train acceleration shall be negative. This is achieved applying a braking force. Train deceleration, is determined by (3.18). During the braking regime, passengers' safety must be guaranteed as well as the on-site halt.

$$F_t(v) < 0 \tag{3.17}$$

$$a(t) = \frac{F_t(v) - F_g(s) - F_r(v)}{M(1+\gamma) + M_p} \tag{3.18}$$

Besides train stop, braking can also be used to reduce velocity, in case the track-imposed velocity should be lower than the train's actual velocity.

3.3 Train Motion Simulator

Having been presented and analyzed, the train model it is needs the development of a TMS algorithm. As the name suggests, a TMS algorithm has the purpose to simulate the vehicle motion by implementing and combining all model equations associated to its dynamics. Thus, the TMS algorithm must be capable of determining the velocity profiles, as well as, the respective distance travelled and time needed to do so. Moreover, the algorithm must also estimate the amount of energy needed for train operation. Besides all these, in order to be used in real railway lines speed profiles, the algorithm must have some mechanisms to consider lines constraints. Therefore, this section will present TMS algorithm development, showing all the various iterations, that have taken place, until its final version.

3.3.1 Formulation

The TMS algorithm implementation uses, as its base reference, for velocity profile determination, the OSP shape. This is so because, as could be seen in the literature review, it is the one that is capable of providing minimum energy consumptions. The OSP, and respective variables defined for algorithm implementation purposes, are presented in Fig. 3.3.

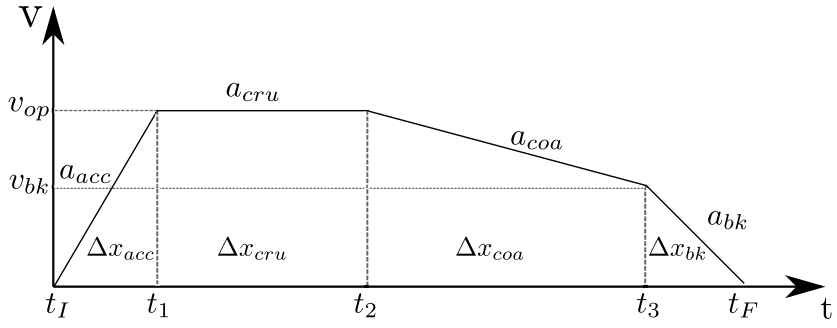


Figure 3.3: Optimal Speed Profile

Using the OSP outline as a reference, TMS algorithm must find the duration of each driving regime in order to be able to make speed profiles. Also, besides defining cruising and braking velocities, the algorithm must determine, for each driving phase, the initial and the final position, as well as the time of their occurrence.

3.3.2 Single Trip

The implementation of the TMS algorithm was first tried using simple cases. Having this fact in consideration, the first development started by considering constant accelerations in all of its phases. For this first try, and to make speed profiles determination easier, no line constraints were considered. Thus, the algorithm just determined time duration and space needed for each driving regime. In what concerns driving regimes, the algorithm complexity has also grown as new advances were achieved. First, only three driving regimes were considered for the new speed profiles, namely acceleration at beginning, followed by cruising, and, at the end of the journey, the braking regime, was applied to stop the train. After having an algorithm capable of creating speed profiles, with these three driving regimes, a second algorithm was implemented by introduction of a fourth one, namely the coasting regime. This section will be dedicated to show how the algorithm for single trips speed profiles generation was implemented. It will start with the first option, Section 3.3.2.1, with three driving regimes, followed by the four driving regimes one, in Section 3.3.2.2.

3.3.2.1 Speed Profiles Without Coasting

First developments, as referred before, started with a speed profile that uses only three driving regimes and railway lines without any constraint. Considering only three driving regime, namely acceleration, cruising and braking and, in each one, constant train acceleration, the speed profile is similar to the one presented in Fig. 3.4.

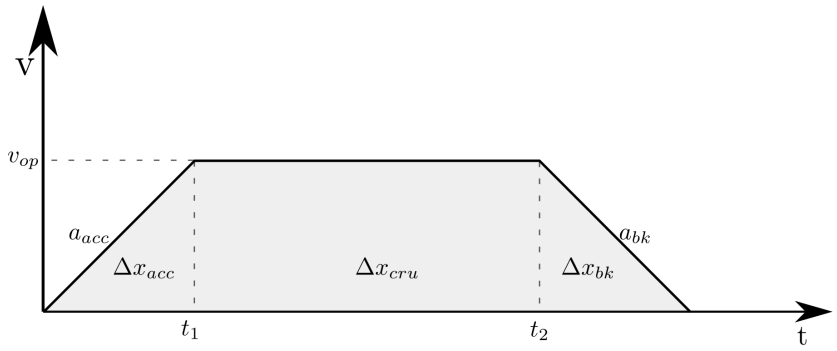


Figure 3.4: Speed profile with three driving regimes - definition of variables.

So, this speed profile uses the definition of all three driving regimes. The definition of a single driving regime requires determination of spent time and distance travelled. Besides, train acceleration and top velocity, need to be defined. Having obtained the speed profile, it is possible to advise the train

driver, concerning the moments he must start to accelerate or press the brake, and also the velocity he must maintain in each instant of the journey. To start defining the speed profile, it is necessary, in the first place, to write the equations associated to the motion of an object subjected to constant acceleration. These equations are (3.19) and (3.20), allowing to relate vehicle acceleration, speed and position.

$$a(t) = \frac{\partial v}{\partial t} \leftrightarrow a(t)\ \partial t = \partial v \leftrightarrow \int_{t_I}^{t_F} a(t)\ \partial t = \int_{v_I}^{v_F} \partial v \leftrightarrow \Delta v = \int_{t_I}^{t_F} a(t)\ \partial t \tag{3.19}$$

$$v(t) = \frac{\partial x}{\partial t} \leftrightarrow v(t)\ \partial t = \partial x \leftrightarrow \int_{t_I}^{t_F} v(t)\ \partial t = \int_{x_I}^{x_F} \partial x \leftrightarrow \Delta x = \int_{t_I}^{t_F} v(t)\ \partial t \tag{3.20}$$

Using equations (3.19) and (3.20), considering a constant acceleration, equation (3.21) and (3.22) are applied to determine train velocity and position.

$$\Delta v = \int_{t_I}^{t_F} a(t)\ \partial t \leftrightarrow v(t) = v(t-1) + at \tag{3.21}$$

$$\Delta x = \int_{t_I}^{t_F} v(t)\ \partial t \leftrightarrow \Delta x = \int_{t_I}^{t_F} v(t-1) + at\ \partial t \leftrightarrow x(t) = x(t-1) + v(t)t + \frac{1}{2}at^2 \tag{3.22}$$

With the equations that allow to get train velocity and position, versus time, it is necessary to outline a strategy capable of achieving a complete speed profile generation. Keeping in mind that the algorithm to be implemented for the DAS has the objective of energy consumption minimization, its bottom line must consider the OSP shape and its driving regimes sequence as a reference.

The speed profile determination is divided in three zones: acceleration, speed maintenance and braking phases. To determine each phase, for the complete speed profile, it is necessary to define the following variables:

- t_1 and t_2: the time for train driver change from acceleration to cruising;
- t_2: the time to start train braking;
- Δx_{acc}, Δx_{cru}, Δx_{bk}: distance travelled respectively for acceleration, cruising and braking driving regimes;
- v_{op}: cruising velocity, to maintain during the whole cruising phase;

- a_{acc} and a_{bk} : acceleration considered for acceleration and braking driving regimes.

To start speed profiles determination, some of these variables must be previously allocated. So, accelerations values for acceleration and braking, a_{acc} and a_{bk}, were selected to be previously assigned. Not to be forgotten that the TMS algorithm is used by the OSP for generating several profiles that will be used latter by an optimization algorithm, in order to choose the one that will minimize energy consumption. After starting the speed profile determination the TMS must receive information about the actual journey. So, the algorithm inputs are:

- Total time available for the journey: $\Delta t = t_F - t_I$;
- Total distance to travel: $\Delta x_{total} = x_F - x_I$.

This inputs list, for single trip speed profiles determination, is reduced since it is not taking into account the information about line constraints. Knowing all inputs available, as well as being defined which variables will be assigned and which will be determined, next step is related to determination of the equations that relate the variables to each other and that describes each of the phases. The speed profile determination starts with the acceleration phase. This phase begins at station departure with an initial velocity. For calculations purposes, departure station is considered at position and time zero, it means, x_I and t_I are equal to zero. Acceleration phase ends at cruising velocity, v_{op}, at a certain unknown time instant, t_1, that must be determined. Besides this time instant, distance travelled must be determined. Given equations (3.21) and (3.22), time spent to go from initial velocity to cruising velocity, t_1 can be determined by (3.23) and respective distance travelled, Δx_{acc}, by equation (3.24).

$$\frac{v_{op} - v_I}{t_1 - t_I} = a_{acc} \tag{3.23}$$

$$\Delta x_{acc} = \frac{1}{2}(t_1 - t_I)(v_{op} - v_I) \tag{3.24}$$

To finish the determination of acceleration phase it is only needed to know cruising velocity, v_{op}.

Second speed profile phase to be determined is braking. This phase can be analyzed following a similar process as the one used for acceleration. Braking phase starts with cruising velocity and ends with a final velocity, v_F, at arriving station. The arrival station, in terms of calculations, is defined as being in a position equal to the total travelled distance, x_F. In addition, it is also considered that the

train should reach this point in the total travelling time, t_4. Once more, considering equations (3.21) and (3.22), it is possible to determine t_2 and Δx_{bk} using equations (3.25) and (3.26) respectively.

$$\frac{v_F - v_{op}}{t_F - t_2} = a_{bk} \tag{3.25}$$

$$\Delta x_{bk} = \frac{1}{2}(t_F - t_2)(v_{op} - v_F) \tag{3.26}$$

Once more, to complete all variables determination, it is necessary to know the cruising velocity.

Last phase to be considered is cruising. During this phase, the train driver is expected to maintain acceleration equal to zero, which results on velocity maintenance. Once acceleration is known, by cruising phase definition, the distance travelled during this phase is determined. Using equation (3.22), it is possible to determine the travelling distance with (3.27). Time instants where cruising velocity starts and finishes are already known, since they are coincident with the end of the acceleration phase and the braking starting point, respectively.

$$\Delta x_{cru} = v_{op}(t_2 - t_1) \tag{3.27}$$

Considering these equations, the last one provides the total travelled distance. There are two options for this equation, which will lead to the same result, as expected. The first one is the sum of the whole distance travelled in each driving regime, in equations (3.24), (3.26) and (3.27). The second option considers all areas inside of each driving regime. On a time versus velocity figure, as Fig. 3.4, the area is the travelling distance. If acceleration and braking phases are considered as triangles, and the cruising one as a rectangle, the total travelling distance can be obtained by the sum of the three areas. This can be confirmed by analyzing the expressions obtained for each travelling distance, which can be determined by the following equation (3.28).

$$\begin{aligned} \Delta x_{total} &= \Delta x_{acc} + \Delta x_{cru} + \Delta x_{bk} \\ \Delta x_{total} &= \tfrac{1}{2}(t_1 - t_I)(v_{op} - v_I) + \tfrac{1}{2}(t_F - t_2)(v_{op} - v_F) + v_{op}(t_2 - t_1) \end{aligned} \tag{3.28}$$

Having all equations that allow each phase characterization, a strategy was defined for the sequence of operations, in order to get all variables. This strategy was also implemented thinking of the

future integration of this algorithm, with the optimization algorithm that iteratively searches for an optimal solution. The sequence of operations is the following:

1. Defining values for a_{acc} and a_{bk};
2. Reading algorithm inputs: total travelling time Δt and distance to travel Δx_{total};
3. Determining time instants to switch between each phase, dependent on acceleration values:
   (a) Solving (3.23) in order to t_1, (3.29), and (3.25) in order to t_2, (3.30);

$$t_1 = \frac{v_{op}}{a_{acc}} \tag{3.29}$$

$$t_2 = t_3 - \frac{a_{acc}}{a_{bk}} t_1 \tag{3.30}$$

   (b) Substituting (3.29) and (3.30) in (3.28), a second order polynomial is obtained, the independent variable being t_1, (3.31);

$$\begin{aligned} x_{total} &= \Delta x_{acc} + \Delta x_{cru} + \Delta x_{bk} \quad \leftrightarrow \\ 0 &= \left[\tfrac{1}{2}a_{acc} - \tfrac{a_{acc}^2}{a_{bk}} - a_{acc} + \tfrac{1}{2}\tfrac{a_{acc}^2}{a_{bk}}\right] t_1^2 + \left[a_{acc}t_3 + \tfrac{1}{2}t_2 a_{acc} - \tfrac{1}{2}t_3 a_{acc}\right] t_1 - x_{total} \end{aligned} \tag{3.31}$$

   (c) Determining t_1 using (3.31) followed by t_2 determination, (3.30), and finally v_{op}, (3.29).
4. Once all variables are known, the whole speed profile can be established from the departure to the arriving stations.

Having defined the order of operations in this first algorithm, several alternatives have been tested in order to realize its potential. This algorithm started with searches for speed profiles with equal accelerations and evolved to the search for the best acceleration values, in acceleration and braking phases, which results in minimum energy consumption.

Therefore, the algorithm described during this section is the beginning of several ones and had the purpose of introducing some base concepts and present the complexity and dynamics of the problem to be solved. As expected, this algorithm rapidly evolved, as it is next presented.

3.3.2.2 Speed Profiles With Coasting

Focused on the energy consumption minimization, a fourth driving regime was added to the algorithm previously developed. As it is expected, adding a new phase to the driving regime, will increase algorithm complexity and the level of difficulty to its solution. The fourth traction regime added to speed profiles determination, was coasting, resulting on a shape involving all four phases. The use of a coasting regime on a railway speed profile has the purpose of a further reduction of energy consumption. As already stated, this regime is characterized by not applying any traction force. This results in train deceleration due to the resistive forces, which causes a slow braking. Once more, since the algorithm was first applied on ideal railway lines, there are no line constraints in this analysis. Besides, train acceleration was taken constant in all speed profile phases.

Considering all four traction regimes, the final speed profile will be similar to the one presented in Fig. 3.5.

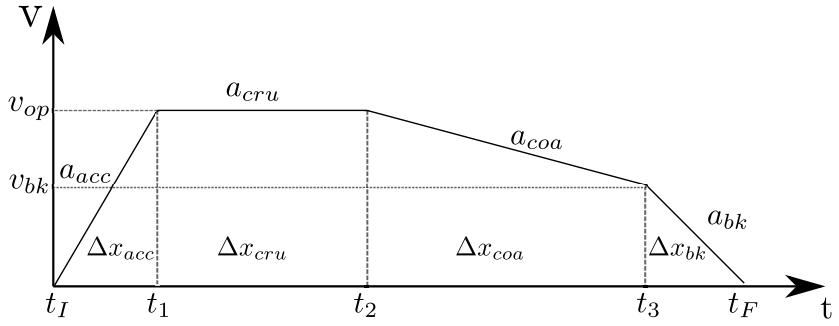


Figure 3.5: Speed profile with four driving regimes - definition of variables.

Getting speed profiles with these four phases requires the determination of a higher number of variables when compared with the previous case. These variables are:

- t_1, t_2 and t_3: time instants to switch between speed profile phases;
- Δx_{acc}, Δx_{cru}, Δx_{coa} and Δx_{bk} : distances travelled during acceleration, cruising, coasting and braking phases, respectively;
- v_{op} and v_{bk}: cruising and braking velocities;
- a_{acc}, a_{bk} and a_{coa}: acceleration and braking values for acceleration and braking phases. Coasting acceleration is also included.

Due to the large number of variables to be determined, it was first necessary to define which ones would be assigned to the initial condition values. So, to determine speed profiles, with these four driving regimes, it was chosen to assign initial values for the acceleration, a_{acc}, and braking phases, a_{bk}. Since the number of variables increased, cruising velocity, v_{op}, was also selected as one of the variables for which an initial value is assigned. With these three variables allocated, the algorithm was implemented in order to determine the values referring to the rest of the variables. As it will be seen, the initial values to be assigned to these three variables will be, on the final version, one of the tasks of the DAS algorithm, as a part of the generation mechanism included it the optimization algorithm.

Before starting the generation of a speed profile, the algorithm must read the available inputs. At this stage of implementation, the available inputs are:

- Time available for journey: $T = t_F - t_I$;
- Distance to travel: Δx_{total}.

The first phase on the speed profile determination is acceleration, which starts at the departure station and goes until the train reaches cruising speed. The initial point of the journey, coincident with the departure station, is defined as position zero, x_I, at initial time instant, t_I, also defined as zero. On this algorithm version, the acceleration phase can be immediately determined, since acceleration and cruising velocity are both known. Considering a constant acceleration, and having already presented how the train's velocity and position can be determined, (3.21) and (3.22), it is possible to write the equations to determine the time instant to change between acceleration and cruising phases, t_1 (3.32), as well as the travelled distance from the initial, v_I, up to cruising v_{op} velocity, (3.33).

$$\frac{v_{op} - v_I}{t_1 - t_I} = a_{acc} \tag{3.32}$$

$$\Delta x_{acc} = \frac{1}{2}(t_1 - t_I)(v_{op} - v_I) \tag{3.33}$$

The second phase is braking. Comparing this second implementation, composed of four driving phases, with the previous one, there are some changes to be mentioned. The braking phase, in this situation, starts at a different point, characterized by the braking velocity. The braking velocity was added to the speed profile, with the purpose of defining when the train driver must start braking. This phase goes on until the train reaches its final destination, the arrival station. The station, in the

algorithm, is defined as being at a position equal to total travelling distance, x_F, and it is supposed that the trip ends in accordance with schedule, t_F. Once again, and similar to the acceleration phase, it is possible to write the equations that determine this time instant, when the braking phase starts, t_3 (3.34), as well as, the space needed to completely stop the train, Δx_{bk} (3.35).

$$\frac{v_F - v_{bk}}{t_F - t_3} = a_{bk} \tag{3.34}$$

$$\Delta x_{bk} = \frac{1}{2}(t_F - t_3)(v_{bk} - v_F) \tag{3.35}$$

The coasting phase, the new driving regime added to the speed profile, must also be defined. This driving regime is applied between cruising and braking phases, providing a smooth train braking. Train movement during this phase is due to its inertia, and should be determined. If the acceleration is approximated by a constant value, equations can be written to determine the time to start coasting, t_2 (3.36), as well as the distance to travel, Δx_{coa} (3.37).

$$\frac{v_{bk} - v_{op}}{t_3 - t_2} = a_{coa} \tag{3.36}$$

$$\Delta x_{coa} = \frac{1}{2}(t_3 - t_2)(v_{op} - v_{bk}) \tag{3.37}$$

The last phase concerning the driving regime is cruising. In this phase train acceleration is maintained constant and equal to zero. At the moment that the algorithm makes calculations for this phase, all important time instants, main velocities values (v_{op} and v_{bk}) and accelerations are known. It is only necessary to determine the travelling distance, knowing cruising velocity and the available time, Δx_{cru} (3.38).

$$\Delta x_{cru} = v_{op}(t_2 - t_1) \tag{3.38}$$

Finally, an equation is written in order to verify if the distance between departure and arrival stations is covered by the achieved speed profile. This equation sums all the travelling distances associated to the driving regimes. They can also be determined by the area of the speed profile shape.

$$\begin{aligned}
\Delta x_{total} &= \Delta x_{acc} + \Delta x_{cru} + \Delta x_{coa} + \Delta x_{bk} \quad \leftrightarrow \\
&\quad \tfrac{1}{2}(t_1 - t_I)(v_{op} - v_I) \\
\Delta x_{total} &= +v_{op}(t_2 - t_1) + \tfrac{1}{2}(v_{bk} - v_{op})(t_3 - t_2) \\
&\quad +v_{bk}(t_3 - t_2) + \tfrac{1}{2}(v_{bk} - v_F)(t_F - t_3)
\end{aligned} \tag{3.39}$$

In possession of the equations that allow to achieve each one of the variables, associated to the speed profile, a methodology has to be defined in order to implement an algorithm that is able to automatically make its determination. This algorithm will only have as inputs the current train trip. As before, the methodology was designed considering the possibility that, at a later phase, it will be integrated into a structure with an optimization algorithm that iteratively determines speed profiles. The considered methodology was:

1. Defining initial values for a_{acc}, a_{bk} and v_{op};
2. Reading inputs: information about distance to travel, $x_I = 0$, $x_{total} = x_F$, and available time, $\Delta t = t_F$;
3. Determineing coasting phase acceleration using equation (3.16):
   (a) Gravitational force is set to zero, since it is not considered, $F_g = 0$;
   (b) Resistance force changes with velocity. For this purpose it is considered constant acceleration determined with velocity equal to v_{op}, $F_r(v_{op})$, resulting on the maximum value of deceleration on this phase.
4. Determination of time instant between acceleration and cruising, t_1 equation (3.40) and Δx_{acc} (3.33);

$$t_1 = \frac{a_{acc} t_I - v_{op} + v_I}{a_{acc}} \tag{3.40}$$

5. Determination of time instant at which the train driver must finish the cruising phase and must start the coasting one, t_2, by considering the equation (3.39) with some substitutions of variables

   (a) Starting by equation (3.34), and rewriting it in order to v_{bk};

$$v_{bk} = -a_{bk}t_F + a_{bk}t_3 + v_F \tag{3.41}$$

   (b) In the following an expression is written in order to t_3. For this, equation (3.36) is used, together with equation (3.41), resulting in:

$$t_3 = \frac{a_{coa}t_2 - a_{bk}t_F + v_F - v_{op}}{a_{coa} - a_{bk}} \tag{3.42}$$

   (c) Considering equation (3.39), which relates each distance travelled per phase to the total distance, and using the expressions (3.41) and (3.42), a second order polynomial, using t_2 as independent variable, is written.

$$d = \frac{1}{2}\frac{1}{a_{coa}-a_{bk}} \begin{bmatrix} -v_{op}^2 + v_F^2 - v_{op}t_1a_{coa} + v_{op}t_1a_{bk} - v_{op}t_Ia_{coa} \\ +v_{op}t_Ia_{bk} - v_It_1a_{coa} + v_It_1a_{bk} + v_It_Ia_{coa} - v_It_0a_{bk} + 2v_{op}t_2a_{coa} \\ -a_{bk}a_{coa}t_2^2 - 2v_{op}a_{bk}t_F - a_{bk}t_F^2a_{coa} + 2t_2a_{bk}t_Fa_{coa} \end{bmatrix} \tag{3.43}$$

6. Solving properly (3.43), the value of t_2 is known, it follows t_3, known using (3.42), and finally, braking velocity is determined (3.41);

7. Having all variables determined, a speed profile starting at the departure station until the arrival one is determined, respecting all time instants as well as distances travelled in each driving regime.

The methodology just described allows for getting a speed profile, starting by the estimation of time and space needed in each driving regime. The algorithm, at this moment, does not dynamically represent a train, neither estimates traction force needs or the energy consumption in one trip. It actually only determines speed profiles based in some values of acceleration and cruising velocity.

In fact, coasting phase considers train characteristics to determine train deceleration, but the other driving phases do not consider the train dynamic model. So, an integration with a dynamic model is needed, as next presented.

3.3.3 Model Integration

Having a structure able to generate possible solutions for train speed profiles, an integration with a dynamic model is needed. This integration has two implementation objectives:

- To validate an executable speed profile;
- To estimate energy needs.

The first objective aims to establish if the proposed speed profile can be met considering train physical limitations. These limitations refer to the maximum available tractive and braking forces. The second objective has an important role in the algorithm used, which is to determine the speed profile associated to the minimization of energy consumption. The integration of train dynamic model in the previous algorithm, leading to each speed profile phase, uses the equations shown on section 3.1. The algorithm implemented to determine speed profiles, as presented before, was maintained, being the train model equations implemented as shown in Fig. 3.6.

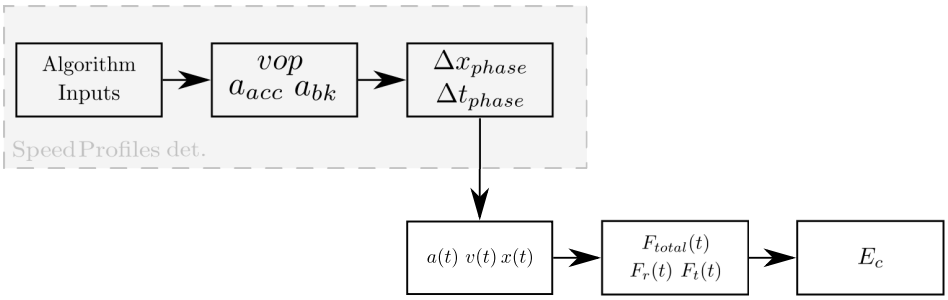


Figure 3.6: Sequence of operations to determine speed profiles.

The algorithm implemented to determine speed profiles is responsible for the calculation of the time instants when the train driver must exchange between driving regimes. It also gets the line position where each change happens. In order to accomplish this goal, the algorithm receives trip information, values for train acceleration on acceleration and braking phases, a_{acc} and a_{bk} respectively, as well as a cruising velocity value, v_{op}. Having all this information, the algorithm determines the

time and space needed to run each speed profile phase. Following Fig. 3.6, the speed profile determination step ends, and knowing this first result, the algorithm considers journey available time. Then, discretizing numerical calculations with a sample time of 1 *s*, the resultant acceleration, velocity and position are determined. As a consequence, total traction and resistance forces are also determined and the energy consumption is estimated, defining the end of this operations' sequence. Since the vectors of accelerations, velocities and positions, as well as all determined forces, only consider values received in the algorithm, it is important to know how feasible those speed profiles are. At the moment, it is significant to understand how far can train acceleration be approximated by constant values. So, in order to validate the obtained speed profiles some tests were made. This validation came out after analyses of train traction and braking curves. It should be reminded that train traction and braking force were always considered as constant, the train's real path being ignored. To verify the accuracy of these approximations, the following test, represented in Fig 3.7 was implemented.

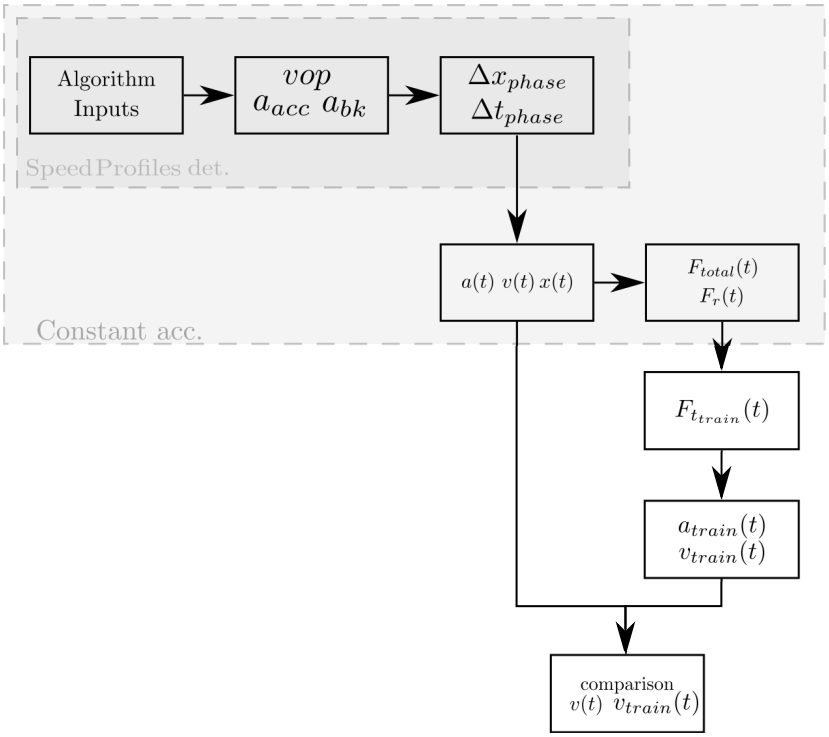
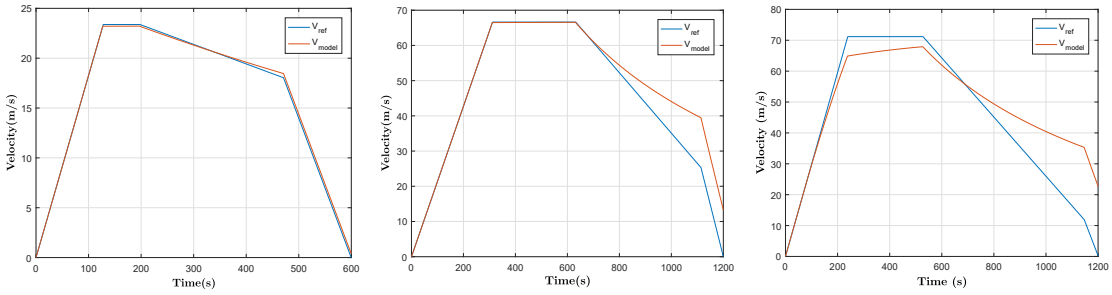


Figure 3.7: Train Model Integration.

The test is quite simple. Basically, it uses the algorithm associated to the speed profiles as implemented, and the obtained result is introduced into the train model. The algorithm determines time and space associated to each speed profile phase. Next, train acceleration in each time instant, velocity and

position is determined. With train acceleration, and knowing the mass of the train, the algorithm gets train forces. It starts by resistive forces, F_r, followed by the total force of the train, F_{total}, the last one being traction/braking force, F_t. At this moment, the algorithm changed. Until now, the train's traction force was only compared with its maximum force, being considered constant. After an analysis of traction curves, it was quite obvious that the maximum force reduces with train acceleration, being necessary to include this in the algorithm. This way, as a test, after getting the train's total force, the traction force was the next to be achieved, however, the maximum value determined was compared with train traction curve. After finding out the train's traction force, the reverse process was done. It means, having a new traction force vector, the train's total force was again determined, as well as train acceleration. Finally, looking for a better estimation of the train's acceleration, its velocity was again calculated and posteriorly compared with the one determined by considering constant acceleration. The results of these tests are presented in Fig. 3.8.



(a) Low cruising velocity. (b) High cruising velocity. (c) High cruising velocity.

Figure 3.8: Constant acceleration with low and high velocity - algorithm results with train model.

Figure 3.8 shows an example of a speed profile. The journey used in the test is not a real one. In fact, it was only defined with the purpose of generating one situation with low cruising velocity, Fig. 3.8a, and two with high cruising velocities, Fig. 3.8b and Fig. 3.8c. It can be seen that the approximation of train acceleration by a constant value is valid only for low velocities. As the velocity increases, the model output is moving further and further from the intended velocity profile. Looking at the traction force characteristic, Fig. 3.2, this result was already expected.

So, the solution to this problem involves changing the way velocity profiles are calculated. The constant acceleration approach was abandoned to include the actual train acceleration curves. In the next section, these algorithm changes will be further explained.

3.3.4 Non constant accelerations

Aiming for the correction of the speed profile algorithm so that it can be used with the dynamic train model for higher velocities, its update with non-constant accelerations was carried out, which required new equations to determine time and space allocated to each driving phase. The result will be a more complex and computationally demanding algorithm.

Starting with the acceleration phase, the traction force is limited by dynamic characteristics presented in equation (3.7). During this phase, the train is under influence of traction, gravitational and resistive forces. The total train acting force is the sum of each individual force. Acceleration is calculated as previously presented, in equation (3.11). Equations to determine space and time, needed on this phase, to drive the train from an initial velocity v_I up to cruising v_{op} are:

$$\Delta t_{acc} = \int_{v_I}^{v_{op}} \frac{1}{a_{acc}} \, \partial v \tag{3.44}$$

$$\Delta x_{acc} = \int_{v_I}^{v_{op}} \frac{v}{a_{acc}} \, \partial v \tag{3.45}$$

In equations (3.44) and (3.45) a_{acc} represents train acceleration, characterized by two equations, due to the fact that the train's traction curve is divided into a zone of constant force and a zone of constant power.

The second phase of speed profile determination after acceleration, is the braking phase. The train dynamics is represented in a similar way as acceleration. This phase, as already explained before, it is used in the train's braking process from an initial velocity, defined as braking velocity, v_{bk}, until the final velocity, v_F, usually when the train stops at one destination. Changing from constant acceleration approximation to train real dynamics, time and space needed to stop the train can be determined by equations (3.46) and (3.47).

$$\Delta t_{bk} = \int_{v_{bk}}^{v_F} \frac{1}{a_{bk}} \, \partial v \tag{3.46}$$

$$\Delta x_{bk} = \int_{v_{vk}}^{v_F} \frac{v}{a_{bk}} \, \partial v \tag{3.47}$$

Once more, to determine time and space needed for the braking phase, the operation must be divided in two phases, since the train's braking characteristic is also divided in constant force and constant power regions.

After the braking phase, the algorithm calculates the coasting phase. In this phase, traction force is not applied and acceleration is determined by (3.16). On the velocity profile, the coasting phase appears between the cruising and braking ones, which means, that it starts at cruising velocity and finishes with braking velocity.

$$\Delta t_{coa} = \int_{v_{op}}^{v_{bk}} \frac{1}{a_{coa}} \, \partial v \tag{3.48}$$

$$\Delta x_{coa} = \int_{v_{op}}^{v_{bk}} \frac{v}{a_{coa}} \, \partial v \tag{3.49}$$

The last phase to be considered is cruising. In this one, train acceleration is constant and equal to zero, being possible to apply the equations already presented. It was only added an equation for the space needed in cruising phase, (3.50), since all important time instants on speed profile already have an alternative way to be calculated.

$$\Delta x_{cru} = v_{op} \left(t_2 - t_1 \right) \tag{3.50}$$

Given the equations to determine time and space needed in each driving regime, a methodology was designed to calculate the velocity profiles. The algorithm is supposed to be running iteratively and some variables must be determined externally, with values assigned to them. In this version of the algorithm, a more realistic behavior of the train is considered, concerning the time and space needed for each phase, and being represented by non-constant acceleration due to train traction and braking characteristics. Because of that, train acceleration for acceleration and braking phases is no longer assigned to a constant value as previously determined. Instead, cruising and braking velocities, v_{op} and v_{bk} respectively, are assigned to externally generated values. The algorithm for the main operations and the latter's respective order are:

1. Value assignment to v_{op} and v_{bk};
2. Speed profile determination:
   (a) Determination of space needed for acceleration (3.45), coasting (3.49), braking (3.47) and cruising phase (3.50);
   (b) Time spent in each phase, equations (3.44), (3.48) and (3.46);

(c) Speed profile determination, using the result of 2a to change between driving phases. Time and position are determined each second, using (3.19) and (3.20), respectively;

3. Train model:

   (a) Resistive and traction force determination;

   (b) Energy estimation.

4. Results presentation:

   (a) Full speed profile;

   (b) Time and position to change between driving regimes;

   (c) Energy consumption estimation.

After evaluation of the algorithm results, and having now the possibility to determine velocity profiles considering non-constant accelerations, it was decided to add more information to it, namely line constraints. This extra information is related to line velocity limits and line gradients. The next section will show how these line constraints are used as algorithm inputs, and the way they are integrated into it.

3.4 Line Constraints

On real railway lines, there are some constraints which may interfere with normal train driving. This study, as described so far, considered ideal railway lines, that is to say flat lines without any velocity limits. Since it is intended to use the DAS on real railways lines, it is very important to consider these restrictions associated to velocity profiles determination. So, the following line constraints will have be considered:

- Line gradients;
- Velocity limits;
- Neutral zones.

3.4.1 Gradients

The first considered additional constraint is line gradients. Line gradients influence energy consumption on train operations and, for this reason, they must be used for the speed profiles determination. Therefore, gradients must be available, as to be used as an algorithm inputs, together with the journey information (schedule time and trip distance).

In terms of structure, information on line gradients information is given as a two-column table. The first column comprises the kilometer points and the second one equals grade angles. As an example, Table 3.1 presents a sample of a gradient vector.

Table 3.1: Example of a gradients vector's input

pk(m)	**grad (mils)**
0.00	0
39.10	-0.0105
444.10	0
799.10	-0.014
884.10	-0.013386
1046.96	-0.0134
1201.44	-0.012199
1337.51	0
1700.00	0.015579
2071.99	0
2232.21	0.000158

Starting with first column, containing kilometer points, it should be mentioned that they are not uniformly distributed. Indeed, they are presented where a gradient change appears. The values are recorded using the meter as the unit, and using as reference the kilometer point of the departure station. It means that, for each journey, between two consecutive stops, the algorithm must receive an input vector associated to the considered line section.

The second column has gradient values. These gradient values represent the angle measured between a horizontal line and the train surface. The value is presented in mils. Besides the angle value, it also presents the angles signal. Bearing this in mind, a positive angle is associated to an

uphill path, whereas a negative angle corresponds to a downhill one.

Slopes effect on speed profiles determination is accounted for using gravitational force on train model. This is achieved by using the right angle value on equation (3.6). In order to know the gravitational force to which train is subjected, a search over input data should be carried out concerning the correct value. This is accomplished using the train's position and a lookup on the corresponding column of the table.

To determine actual value for gradient, based on previous train position, two searching algorithms were implemented. In the end, looking at the algorithm's final version, it is concluded that only one function would be enough, but at this time of implementation, this was the best solution found. The use of two functions is related to how actual train position is searched on the kilometers table column. One function was implemented to search using the departure station as reference, and a second one for a search on the kilometers column, considering the arrival station as the reference. Later on, the use of both functions will be clarified as the new version of the algorithm is explained.

Algorithm 1 shows the structure of the function implemented for gradient determination. This function was the first one implemented and is used to search for the gradient value, given the actual train position, and considering the departure station as reference. To simplify the function's objective, this searching method was called as forward search, since it starts at the journey's initial position and finishes at the arrival station.

Algorithm 1: Gradient forward determination

grad_vector = $[pk,\ \theta]_{K\ \times\ 2}$; ▷ Gradient vector - dimension k variable

Grad_determination ($grad\_vector$, x_{i-1}, x_F)

for *k=1:1:K* **do**

  if $x_{i-1} > pk_i[k,1]$ **then**

    return k;

  else

    continues searching

  end

end

$\theta_i = grad\_vector[k,2]$;

$\theta_{i+1} = grad\_vector[k+1,2]$;

$pk_{i+1} = grad\_vector[k+1,1]$;

returns θ_i, θ_{i+1}, pk_{i+1}

The second function implemented is presented in Algorithm 2. The function is similar to the first one, the major difference being the way the search is carried out. The search uses the departure station as the reference point and the algorithm starts the kilometers points column, from the last up to the first line. That is why this function is called backwards function.

The idea to implement two functions, to search for kilometer points, is fundamentally due to the use of only one input file. This means that when a backwards search is needed, instead of changing the kilometers column, the algorithm launches the search by the end of the column and subtracts the arrival station position to the train's actual position. This way, the algorithm keeps input data which can used again.

Algorithm 2: Gradient backwards determination

grad_vector = $[pk, \theta]_{K \times 2}$; ▷ Gradients vector - dimension k variable

Grad_determination_backwards ($grad\_vector$, x_{i-1}, x_F)

for *k=1:1:K* **do**

  if $x_{i-1} > x_F - pk_i[k,1]$ **then**

    return k;

  else

    continues searching

  end

end

$\theta_i = grad\_vector[k,2]$;

$\theta_{i+1} = grad\_vector[k+1,2]$;

$pk_{i+1} = grad\_vector[k+1,1]$;

returns θ_i, θ_{i+1}, pk_{i+1}

3.4.2 Velocity Limits

The second line constraint considered the speed profile determination algorithm, is velocity limits. As it happened with gradients, velocity limits also correspond to an algorithm input presented as a two-column table. The first column contains kilometer points where a velocity limit change happens and the second column shows the associated limit value. Once more, kilometer points are not uniformly distributed. Only points where there are changes to be registered are presented. An example of the input vector is shown in Table 3.2.

Table 3.2: Example of velocity limits input

pk(m)	V_{max} **(km/h)**
0	50
334	45
1017	70
2030	120

Input table interpretation is quite simple. Each journey to be made has a different table where velocity limits are presented. Kilometer points measurements use the departure station as the reference. Looking at the table, the first kilometer point is coincident with the location of the departure station, and first velocity limit is valid from the beginning of the journey up to the kilometer point of the following consecutive change. Profiling the maximum speeds on the line, represented in Table 3.2, gives the scheme presented in Fig. 3.9.

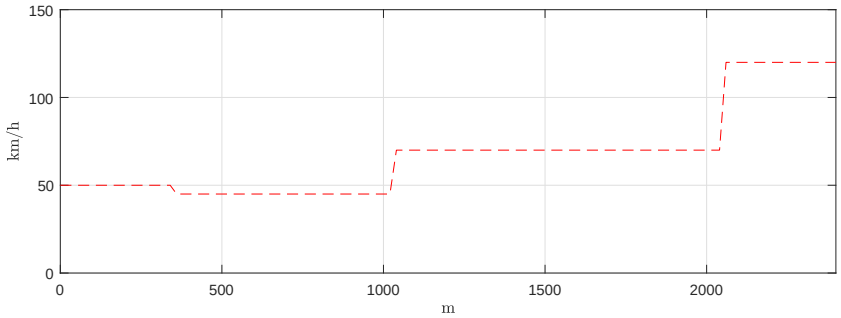


Figure 3.9: Example of velocity limits - data representation.

It is important to mention that the input table with velocity limit information is unique for each line section considered on speed profile determination. Kilometer points are presented in meters, and maximum velocities values in km/h. The function used to read velocity limits before returning the result applies the conversion from km/h to m/s. This is done with the purpose of using only SI units inside the algorithm, avoiding any error that may be caused by the use of improper ones.

Following the same logic applied to the calculation of gradients, two searching functions were implemented with the purpose of determining line velocity limits. These functions have been implemented whenever necessary, returning the current speed limit to the train position as well as the following one. The major difference between both routines is the way the input vector is analyzed. The first one starts at the first position of the vector and makes a forward search, while the second function makes a backwards search from the last position up to the first one.

Algorithm 3 presents the first function implemented. This one implements a forward search, that is to say, in the same direction as the train's movement.

Algorithm 3: Velocity limit determination

V_vector = $[pk, V_{max}]_{K \times 2}$; ▷ Velocity limits- dimension k variable

$kmh2ms = 0.2778$; ▷ km/h to m/s conversion

Vmax_determination (V_vector, x_{i-1}, x_F)

for *k=K:1:1* **do**

  if $x_{i-1} > pk_i[k,1]$ **then**

    return k;

  else

    continues searching

  end

end

$V_{max_i} = V\_vector[k,2] \times kmh2ms$;

$V_{max_{i+1}} = V\_vector[k+1,2] \times kmh2ms$;

$pk_{i+1} = grad\_vector[k+1,1]$;

returns V_{max_i}, $V_{max_{i+1}}$, pk_{i+1}

Given train actual position, the function implemented starts by verifying the kilometer column from the last position up to the first one. In each routine call, train position is compared to vector kilometer points in order to find the imposed speed limit. Besides actual velocity limit, information about next limit and the position at which that change happens are also taken into consideration. In the end, routine *Vmax_determination* returns the actual and next velocity limit as output, as well as the kilometer point where the velocity limits change occurs.

The second function implemented determines velocity limit using a backwards search. The implemented function does the same but, instead of starting the search from the end of the table, it starts at its first position. Algorithm 4 presents the routine outline.

Algorithm 4: Velocity limits determination

V_vector = $[pk, V_{max}]_{K \times 2}$; ▷ Velocity limits- dimension k variable

$kmh2ms = 0.2778$; ▷ km/h to m/s conversion

Vmax_determination_backwards (V_vector, x_{i-1}, x_F)

for *k=1:1:K* **do**

  if $x_{i-1} > x_F - pk_i[k,1]$ **then**

    return k;

  else

    continues searching

  end

end

$V_{max_i} = V\_vector[k,2] \times kmh2ms$;

$V_{max_{i+1}} = V\_vector[k+1,2] \times kmh2ms$;

$pk_{i+1} = grad\_vector[k+1,1]$;

returns V_{max_i}, $V_{max_{i+1}}$, pk_{i+1}

The first conclusion achieved is the similarity with the function presented in Algorithm 3, concluding that one function should be enough. As it happens with gradients, at the moment when the algorithm was developed, both routines were implemented and are in use nowadays. In fact, both could be replaced by an unique function, changing at least, inputs values. *Vmax_determination_backwards* function works following same rules of previous function and gives same outputs to the main algorithm.

3.4.3 Neutral Zones

Neutral zones are common on electrified railway lines due to power line topology. The extent of these zones goes from few up to tens of meters and they typically occur in the connection between two phases of a substation or in the connection between two substations. These zones serve to electrically separate power sources, which results in areas where the train has no traction force since there is no power to do it [51]. As expected, the presence of neutral zone will influence the determination of the optimal speed profile.

The train operator knows the railway structure very well and, for this reason, he is acquainted

with all neutral zones in the line. The operator knows gradient and velocity limits, the kilometers points where a neutral zone starts and ends. So, that information must be available to be used in the algorithm as an input.

The received information given by train operator, related with neutral zones, was analyzed and posteriorly processed in order to define an input data form to be used on the TMS algorithm. The information given by the train operator only indicates the kilometers points where a neutral zone happens. Considering this, an input vector is generated, as presented in Table 3.3.

Table 3.3: Example of neutral zones location input

pk(m)	**Notch**
0	0
290	1
940	0

As usual, the first column shows kilometer points (in meters) and all values are given by considering the departure station as the reference. The second column correspond to the value of a flag created for neutral zones. The idea of this flag is to identify the zones without energy forcing the flag value to 1, while zones powered by substations are represented with a 0. To interpret the vector, the algorithm must search on kilometer points to decide if the train is inside or outside of a neutral zone. In the example presented, from the initial point to kilometer point 290 the train is in a non-neutral zone. From point 290 up to 940, the train is in a dead zone, and from this last point until the end of the journey, no more neutral points occur. Dead zones determination followed a line of implementation similar to that used in the previous cases. Two different searching algorithms were implemented with the purpose to search for neutral zones in the accelerating and cruising phase as well as in the braking and coasting ones. Once more, the implementation of two routines could be reduced to merely one, being only necessary at the beginning to define what kind of search in the vector would be done. In both cases, the routines implemented only returns the flag value corresponding to the train position.

Algorithm 5 shows the first routine implemented, used in acceleration and cruising phases. The structure of the searching routine is quite simple: it only looks at the train's previous position and returns the flag corresponding value, according to neutral zone positions.

Algorithm 5: Neutral zone determination

NZ_vector = $[pk, Flag\_NZ]_{K \times 2}$; ▷ Neutral Zones vector - dimension k variable

NotchRestriction(NZ_vector, x_{i-1}, x_F)

for *k=2:1:K* **do**

  if $(x_{i-1} < NZ\_vector[k,1])$ && $(x_{i-1} \geqslant NZ\_vector[k-1,1])$ **then**
    $Flag\_NZ_i = NZ\_vector[k-1,2]$;
  else
    $Flag\_NZ_i = NZ\_vector[k,2]$;
  end

end

returns $Flag\_NZ_i$

A similar approach to identify neutral zones in braking and coasting phase was adopted. Algorithm 6 shows routine *NotchRestriction_backwards*.

Algorithm 6: Neutral zones determination

NZ_vector = $[pk, Flag\_NZ]_{K \times 2}$; ▷ Neutral Zones vector - dimension k variable

NotchRestriction_backwards(NZ_vector, x_{i-1}, x_F)

for *k=K:-1:2* **do**

  if $(k == K$ && $x_{i-1} < x_F - NZ\_vector[k,1])$ **then**
    $Flag\_NZ_i = NZ\_vector[k,2]$;
  else if $k \geqslant 2$ && $x_{i-1} > x_F - NZ\_vector[k,1]$ && $x_{i-1} \leqslant x_F - NZ\_vector[k,1]$ **then**
    $Flag\_NZ_i = NZ\_vector[k-1,2]$;
  else if $k == 2$ && $x_{i-1} < x_F - NZ\_vector[k,1]$ **then**
    $Flag\_NZ_i = NZ\_vector[k-1,2]$;
  else
    $Flag\_NZ_i = 0$;
  end

end

returns $Flag\_NZ_i$

Routine *NotchRestriction_backwards* was implemented to search for neutral zones, concerning

the input vector, in driving phases where speed profiles are determined from the end to the beginning. The output is the corresponding flag value, to be used posteriorly on TMS algorithm.

3.5 Train Model State Machine

The introduction of line constraints on the TMS algorithm makes a change in its structure necessary. As the number of line constraints increases, it will be more difficult to determine new velocity profiles. In some cases, with the actual structure, the algorithm is not able to produce solutions, being stuck, without giving any feedback. The reason for this problem to happen has its origin in the algorithm formulation, and consequently it must change somehow. So far, velocity profiles are determined using the sequence of driving regimes proposed by the optimal control theory. This means an acceleration phase, in the beginning of the journey is always applied, followed by cruising, coasting and braking phases. When line constraints are included, into the speed profiles determination, depending on the case, strictly following this driving regimes sequence may not be the most appropriate strategy. So, there was a need to change the way that calculations of new profiles are carried out. Driven by the need to have more freedom of choices, at a driving regime sequence, the train dynamic model implementation was changed. As an alternative to the work already done, the implementation of a train model based on a state machine was proposed. The state machine has four states, one for each driving regime. State selection is dependent on line actual constraints, and in each one, forces applied to the train, as well as resultant acceleration, are calculated. The new train dynamic model was implemented as exposed in Algorithm 7.

Algorithm 7: Train dynamics algorithm

train_dynamics ($state$, x_{i-1}, v_{i-1}, θ)

$F_{g_i} = Mg\theta$

$F_{r_i} = A + Bv_{i-1} + Cv_{i-1}^2$

switch *state* **do**

case *1* **do**

▷ Acceleration Regime

Determine F_{t_i} max with $F_{t_i} \geq 0$;

Determine F_{Total_1} and a_i;

end

case *2* **do**

▷ Cruising Regime

$F_{t_i} = F_{g_i} + F_{r_i}$

Determine F_{Total_1} and a_i;

end

case *3* **do**

▷ Coasting Regime

$F_{t_i} = 0$

Determine F_{Total_1} and a_i;

end

case *4* **do**

▷ Braking Regime

Determine $F_{t_i} = 0$ max with $F_{t_i} \leq 0$

Determine F_{Total_1} and a_i;

end

end

returns F_{t_i}, F_{r_i}, F_{g_i}, F_{Total_i}, a_i

Basically, each state defines a driving regime and, dependent on it, the train traction force and all acting forces are determined. With this implementation, the speed profiles calculation has more freedom of choice in what concerns the order of driving regimes. As an example, considering a line with velocity restrictions, more precisely, when the maximum allowed velocity decreases, the algorithm is able to switch between an acceleration to a braking phase in order to accomplish the

imposed line restrictions.

State transitions are defined by several parameters / variables. The train's actual velocity, actual and next velocity limits, as well as initial and final velocity, are cases of variables considered for state machine selection of the actual state. The state machine implementation will be better explained in the algorithm's final version.

3.6 Train Motion Simulator with Line Constraints

The implementation of train dynamics as a state machine leads to the inclusion of a new algorithm. This new approach considers non-constant accelerations together with associated line constraints. The use of the state machine allows for greater flexibility in the driving regimes choice, which can be applied along the journey. The implemented algorithm follows the flowchart presented in Fig. 3.10.

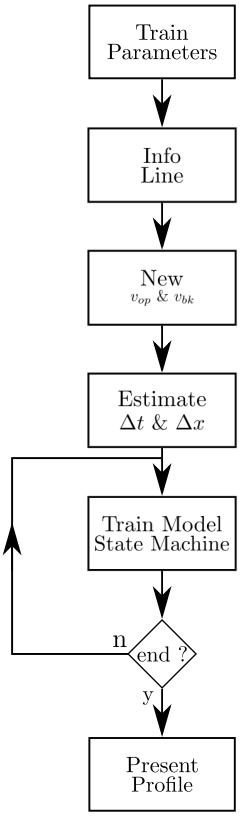


Figure 3.10: TMS algorithm using the train model state machine.

So, the algorithm starts by reading all inputs available and needed for the speed profiles determination. The first part is related to train parameters. The second one is linked to line constraints, as

well as journey information. The information about train parameters is required to set up the right train dynamic model for the vehicle in use. The line information contains information about line constraints, needed to determine speed profiles with all restrictions and journey information that is used to the initialization of the variables. The latter, more specifically vectors to store all train dynamics' information, happens after knowing the total available travelling time. This is so because the developed train model considers time as the independent variable and new values are determined at each new time instant. To start the speed profiles generation, the algorithm waits for values which define cruising and braking velocities, v_{op} and v_{bk} respectively. The next step, after reading and initializing all algorithm variables, is an estimation of time and space needed for each phase. This estimation is made by applying the equations presented in Section 3.3.4. Thus, given an initial and final value for velocities for each speed profile phase, the respective time and space associated to the current phase are determined. Time and space needed for the acceleration phase are the first ones to be estimated, equations (3.44) and (3.45). Braking, (3.46) and (3.47), and coasting, (3.48) and (3.49), come in second place. Calculation of time and space for cruising uses results from the already made calculations, acceleration, braking and coasting, as well as total journey time and space (3.50).

Once the estimation of each speed profile phase is made, the algorithm jumps to train dynamics. Train dynamics was implemented as a state machine which runs inside a while statement. The implemented state machine represents line train dynamics and is used to determine train forces as well as acceleration, velocity and position in each time instant. The while statement uses the total time available for actual journey as control variable. Thus, it runs considering train initial velocity and position equal to zero, coincident with the departure state. It ends when train reaches zero velocity, at a position near the arrival station. During the while statement execution, state machine transitions are based on train actual velocity, actual and next velocity limits and space needed in each phase. The state machine starts with the acceleration phase, defined as its initial state. The state machine's natural flow, which can be applied to a line without any velocity limits changes, is acceleration followed by cruising, coasting and the last braking. The natural state machine's flow happens activating the transitions between states, based on the distance travelled. On the other hand, in railway lines or journeys with velocity limits changes, the algorithm switches between the four speed profile phases, depending on line conditions. Independently of the speed profiles phases sequence, they all always start with an acceleration and end with a braking phase.

After running train dynamics, the algorithm ends returning to the determined speed profile. The

speed profile is graphically represented. Also, the difference between available and travelled time is calculated, as well as travelled distance. These last two are called time and distance errors. Both errors, considering future developments, were introduced in order to be used in cost functions associated to the energy minimization.

Unfortunately, and after several tests, this TMS algorithm implementation had to be abandoned due to multiple reasons for that. In fact, and especially after the introduction of the velocity limits several drawbacks emerged. The first one was the processing time needed to find a set of optimal speed profiles solutions to be analyzed by the energy consumption optimization algorithm. It proved to be quite high, making its use into a real time application impossible. Besides processing time, the introduction of speed limits, together with in-line slopes, led to the conclusion that the algorithm was somehow inefficient in speed calculation, according to travel specifications. The problem was associated to the implementation of the estimation of each driving regime. Basically, space/ time estimation needed for each speed profile phase was implemented considering an ideal line. In other words, the estimation only considered train traction force characteristics as well as resistive force, as presented in (3.51) and (3.52), thus ignoring velocity limits.

$$\Delta t_{coa} = \int_{v_i}^{v_f} \frac{1}{a_{coa}} \, \partial v = \int_{v_i}^{v_f} \frac{1}{\frac{F_t(v)-F_r(v)}{M(1+\sigma)+M_p}} \, \partial v \tag{3.51}$$

$$\Delta x_{coa} = \int_{v_i}^{v_f} \frac{v}{a_{coa}} \, \partial v = \int_{v_i}^{v_f} \frac{v}{\frac{F_t(v)-F_r(v)}{M(1+\sigma)+M_p}} \, \partial v \tag{3.52}$$

The estimation of each speed profile phase was posteriorly used, in train dynamics, as a velocity reference to follow. With the introduction of velocity limits, the final velocity profile calculated with train dynamics exhibited a distance error not considered by the estimation. This distance error increased with maximum velocity limits changes as well as with the line gradients. When first exposed, this problem led to a tentative solution, regarding the introduction of a cost function, posteriorly implemented into the optimization algorithm. On the other hand, this solution solved somehow the problem; on the other hand, the number of bad solutions associated increased. Therefore, considering all these problems, the algorithm implementation was abandoned and has undergone several changes in order to be operational. These changes are presented in the next section.

3.7 Speed Profiles Generator

The TMS algorithm's final structure resulted from the analysis of the previous versions and the correction of all revealed limitations. The path from the first to the final algorithm version was a complex work, which required considerable dedication and study time. This development implied new research work necessary to achieve a final solution able to determine speed profiles regardless of line conditions, and usable in real time. Comparing the last version of the algorithm flowchart with earlier versions, it can be seen that important changes occurred. In fact, this last version keeps the implementation of train dynamics as a state machine. The use of the train dynamics state machine is advantageous since it gives great algorithm flexibility associated to the driving regimes selection. Concerning the changes, the first to be performed was the removal of the initial estimation of the time and space needed to complete each one of the speed profile phases. Removing this first estimation, the algorithm starts with train movement determination as soon as it receives new values for cruising and braking velocities. The train movement determination uses train model equations, train forces, acceleration, velocity and the corresponding position to calculate space for each time instant. To determine train movement as well as velocity profiles, the algorithm was divided and organized by phases. The first decision to be made was the definition of the best sequence of speed profile driving phases. After looking at all possible sequences, it was chosen to follow the one presented in Fig. 3.11.

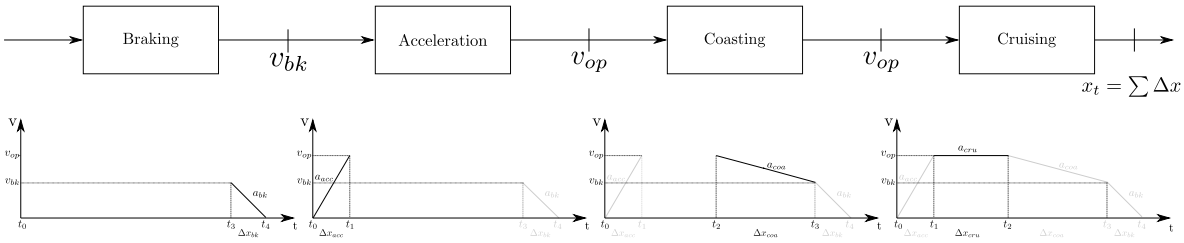


Figure 3.11: TMS algorithm - driving regimes sequence.

As can be seen, the algorithm structure for speed profiles determination is divided into four stages, corresponding to the four driving regimes, defined by control theory. These stages are acceleration, cruising, coasting and braking. Each stage is separately calculated. Inside of each speed profile phase, one or more driving regimes can be applied. As an example, the acceleration phase is defined as the stage applied at the beginning of the journey. Its purpose is train acceleration from the initial velocity to the cruising one. As expected, to accomplish its main purpose, the acceleration driving regime is continuously applied. However, when line characteristics do not allow to do that, the acceleration

regime must be interrupted either by braking or by applying another regime. Hereupon, it is up to the algorithm to choose the correct sequence of driving regimes that best fits the current journey. The use of train dynamics implemented as a state machine allows for an easy switch between driving arrangements within each phase, in order to meet the purpose of the journey as well as the limits imposed by the line.

Using the sequences of speed profiles phases presented in Fig. 3.11, the algorithm used to select the appropriate driving regimes sequence was implemented following the flowchart presented in Fig. 3.12.

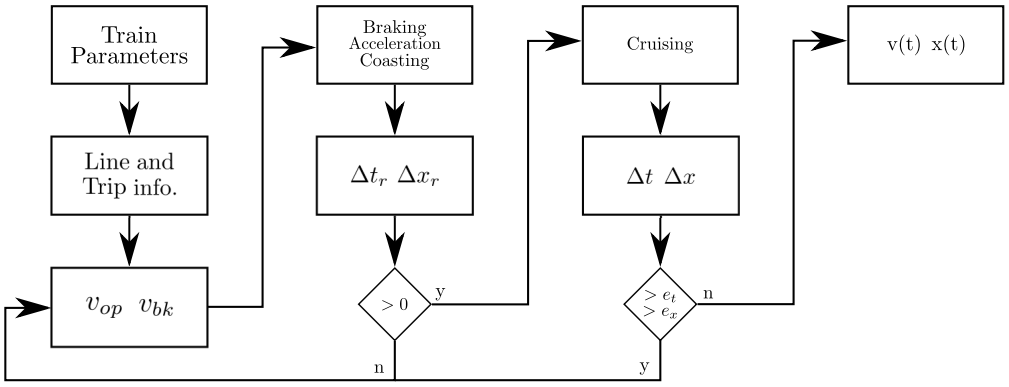


Figure 3.12: TMS algorithm flowchart.

The algorithm flowchart uses the following steps. The first task performed is reading the train parameters, usually given by the train operator. Train parameters are read from input files and assigned to the corresponding algorithm variables in order to introduce the correct vehicle characteristics into the dynamic model. Having all parameters of the train model, the algorithm continues to read information about the actual journey. At this point, information about velocity limits, gradients values and neutral zones position is received. Besides that, information related with the actual journey, as distance to be travelled and corresponding available time, are also at disposal. Having uploaded all available information about the train model, line and journey, the algorithm makes the variables and vectors initialization to store the final results as well as some intermediate operations.

The determination of velocity profiles starts only after the algorithm receives new values for cruising and braking velocity. The mechanism implemented to generate new velocity values will be explained in the next chapter. Once having a model which uses time as independent variable, the algorithm determines all forces, train acceleration, velocity and position for each time interval defined. The first speed profile phase to be considered is braking, which is followed by acceleration and then

coasting. The braking phase was selected to be the first one in order to assure the correct train stop at the arrival station. The second speed profile phase to be determined is acceleration, used to figure out how much time and space the train needs to achieve cruising velocity. Having both phases determined, the algorithm determines the coasting phase to connect cruising to braking velocity. Each driving phase was implemented as a new function, and in each function a time series with a length equal to total traveling time is created together with some temporary variables to store intermediate values. The definition of a time series equal to total travelling time is due to the fact that at the instant when each phase is determined, the needed time and distance are unknown. In each function, as soon as it is called, the algorithm enters into a cycle of operations that ends as soon as the phase boundary conditions are fulfilled. If they are not met, the function returns to the main algorithm with a result which indicates the impossibility of calculating a profile under the actual journey conditions. Boundary conditions are identified during each phase calculation.

Returning to the algorithm's flowchart description, once the first three phases are calculated, a first solution verification is carried out. This first verification consists in verifying if it is possible to determine all three phases, using the given cruising and braking velocity values, within the time and distance set for actual trip. This verification just subtracts the distance and time travelled to the available time and distance for the actual journey. This means that it is verified if there is any remaining time and distance. If remaining time or distance are negative, the algorithm drops the actual solution and waits for new velocity values. If the remaining time and distance are positive, the algorithm starts to calculate the last phase, cruising. This phase is used to travel the remaining distance, maintaining a constant velocity. The implementation is similar to the first three phases, where a new time series is defined and the algorithm runs train dynamic model to determine the time and space travelled. When the end condition of the cruising phase is reached, the algorithm re-checks the time and distance travelled. Once velocity values are given, the total traveling distance and time needed for train travel between the initial and the final station are compared with time and space defined for the trip. If traveling values are within an acceptable range, the current solution is accepted as a possible candidate. If it falls out of the speeds acceptable range, the algorithm returns to the starting position and receives new cruising and braking velocity values, and the whole process starts over. When the solution is accepted, outputs of all phases are reorganized in one single result, and the resulting speed profile is saved to posterior spent energy evaluation. The algorithm structure was developed considering the previous train model's implementation as well as the future integration with

an optimization algorithm.

3.7.1 Speed Profile Phases

3.7.1.1 Braking

The algorithm starts the speed profile determination with the braking phase. As mentioned before, this is the first phase on speed profiles determination to ensure a safe train arrival at destination with zero speed. On a speed profile, the braking phase can be defined as the one where the train decelerates, following train braking characteristics, from braking velocity to the final one (commonly zero velocity). This phase happens at the end of the journey, matching the final speed with the position of the arrival station.

To completely define the braking phase, it is necessary to determine how much time and space are needed to stop the train. Having information about the initial and final the velocity as well as final position for braking phase, the determination of this phase is quite challenging since the position where it must start is unknown. So, for calculations associated to the braking phase, it is necessary to apply a strategy that allows to know where it should start. Analyzing all the information, the idea of making braking phase calculations backwards, this is, starting from the end station, comes to mind. This means that the braking phase will be analyzed as an acceleration one. Fig. 3.13 shows the assumed strategy, demonstrating that the braking phase, instead of being determined to starting at v_{bk}, at unknown initial position, and decelerating the train to final velocity at arrival station, x_F, it is determined as an acceleration phase considering zero velocity as initial and ending at v_{bk}. Determining the braking phase as an acceleration allows to get around the problem of not knowing the starting braking position.

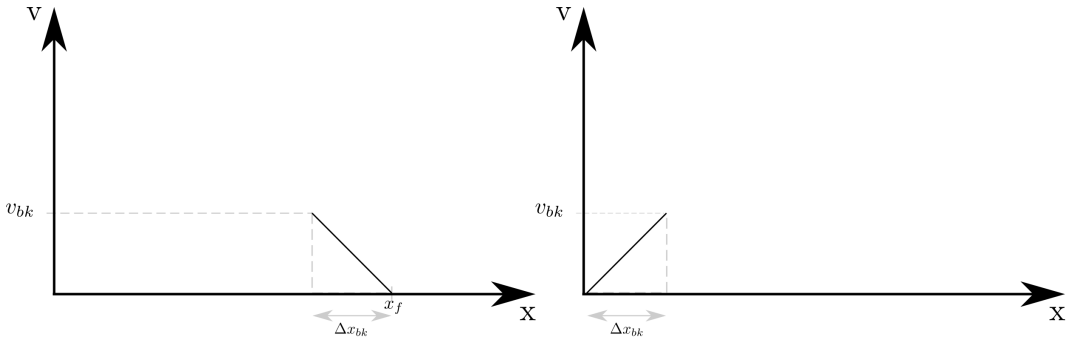


Figure 3.13: Braking phase - graphical representation of the phase determination.

The braking phase backwards requires some changes concerning the train dynamic model. The algorithm implements the train dynamic model as a state machine (which was already seen in function train_dynamics , presented in Algorithm 7). So, all driving regimes were kept unchanged except for the braking one. Whit this purpose in mind, some changes were introduced, more precisely in the way the acceleration value is returned. All forces acting on train as well as traction force needs are determined in the same manner as presented, and only the returned acceleration value is changed. This means that after determining all forces and train acceleration, the algorithm receives a positive acceleration value instead of a negative one. In each phase of speed profile, any of the four driving regimes can be chosen, depending on line constraints. The driving regime selection considers the train's actual and final velocity as well as current and next velocity limits. Depending on current and following velocity limits, the algorithm determines which driving regime is the most appropriate to accomplish it. The determination of current and following speed limits determination is carried out using the function previously presented in Section 3.4. In each time instant train forces, accelerations and respective velocity and position are calculated. The train's current position is used to locate where train is at the actual journey and analyze the line constraints. The determination of line restrictions, at braking phase uses a backward search on its input vectors. This is so due to the fact that the braking phase also uses a backward calculation.

As already seen, the driving regimes selection is dependent on the line's current and following velocity conditions. Before defining the driving regime to be applied at the following time instant, the final velocity for braking (v_{bk}) is compared with current and following velocity limits, and a decision is made. Considering these three variables as decision variables, a set of cases can be anticipated to be posteriorly implemented. Figure 3.14 shows all anticipated cases.

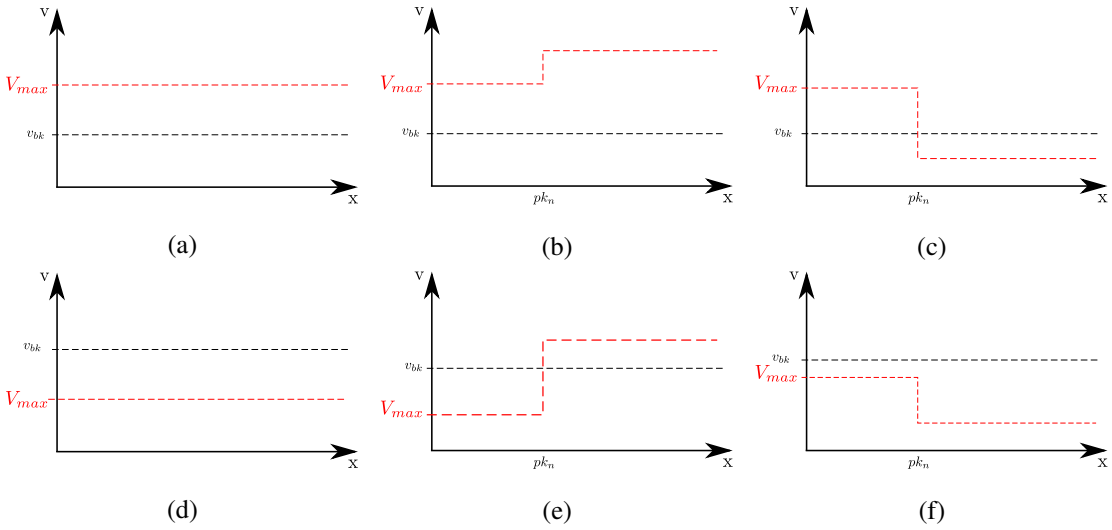


Figure 3.14: Braking phase: comparison between v_{bk}, current and following velocity limit, V_{MAX}.

All presented cases in Fig. 3.14 were considered during algorithm implementation as possible situations that may occur in real railway lines. In a first analysis, some cases do not represent a real problem to speed profile determination while some of them requires a special attention. The first two cases, Fig. 3.14a and Fig. 3.14b, present the simplest situations. In both cases, velocity limits are always above braking velocity, which results on a braking phase without any exchange concerning the driving regime. Case c, Fig. 3.14a, is more complex since it represents a situation where, at a certain position, the velocity limit changes, the actual limit being higher than the following one. The velocity limit's drop can occur to values above or below the braking speed. When braking velocity is still lower than following velocity limit, the case does not represent a real line constraint since a braking phase can be determined without any exchange of driving regime. So, this first scenario is the same than the one in the first two cases. On the other hand, when the following velocity limit falls below braking velocity, some additional operations must be carried out in order to calculate the speed profile within line constraints. A more detailed image representing what happens when the following limit drops below braking velocity is presented in Fig. 3.15.

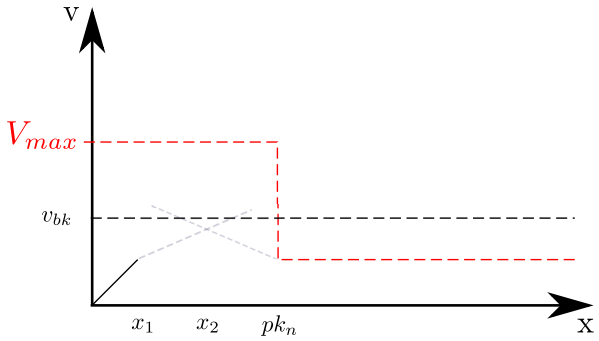


Figure 3.15: Braking phase: acceleration regime estimation.

The determination of a braking phase when the following velocity limit falls below braking velocity is now divided into two stages. The first stage goes from the initial position, with the initial velocity, to the moment when the train reaches the following velocity limit value, at position x_1. At this position, the algorithm starts the second stage of the braking phase determination. Once train velocity exceeds the following limit, an acceleration must be estimated between the train's current velocity and the following limit value. It is necessary to determine if there is enough space to go from the following velocity limit to the train's current velocity. On the braking phase, the use of an acceleration regime can be quite controversial as this phase is used to stop the train. Nonetheless, after analyzing all journeys, in some of them the travelling time available was short or appeared to be tight to complete the journey successfully. Thus, the braking phase was implemented considering the hypothesis of integrating an acceleration between the following limit value and the train's final velocity, at position x_2. So, after reaching the following limit value, position x_1, the algorithm starts to estimate an acceleration between the following limit value and the train's current velocity. In each time step, the algorithm determines the current velocity on left side, it means decelerating the train, and at same time, after updating the velocity on deceleration, the acceleration estimation is updated. Furthermore, it is verified if there is enough space to complete both deceleration and acceleration. This process occurs iteratively until v_{bk} is reached, or at a lower speed value after acceleration and deceleration intersect at a single point, at x_2.

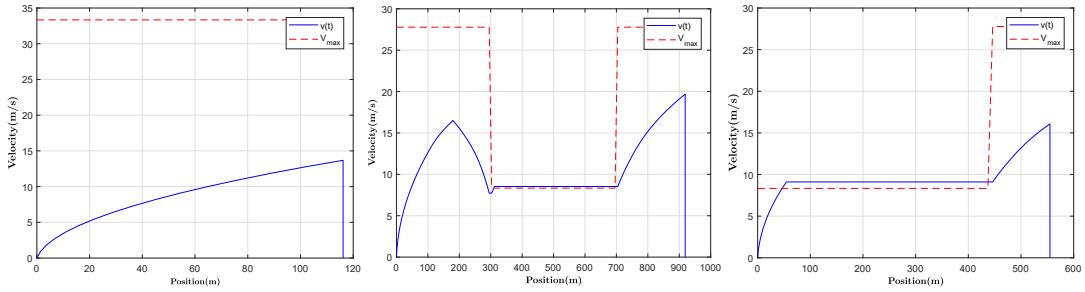
After having completed the analysis of the first row of Fig. 3.14 it is now examined the second one. This row represents all cases where the current velocity limit is lower than the final velocity for braking phase, v_{bk}. All cases now represented require, at the outset, some attention as there is a high probability that it will not be possible to calculate a velocity profile from the initial to the final velocity without any driving regime change. For the cases presented in Fig. 3.14d and Fig. 3.14e, the

algorithm is arranged to determine train velocity from the initial to the limit value, and while the train is within the limit zone, its velocity is maintained. As the speed limit increases, train velocity also increases to the lowest value between the following velocity limit or the final value for braking phase. The last case, Fig. 3.14f, is the most critical case, since it represents the case where current limit is lower than braking velocity as well as the following limit, after the occurrence of a limit drop. For this purpose, the algorithm makes an approach similar to the one developed for case 3.14c, the acceleration estimation being made between velocity limits. In all cases, when braking velocity is higher than any velocity limit imposed on the line, the braking phase is not determined and the algorithm waits for a new braking velocity value input.

In addition to speed limits, the braking phase also considers neutral zones location, which may appear on the line, as a decision variable to determine which driving regimes must be used. Since these zones are characterized by locations where there is no catenary, as soon as the algorithm detects that train is inside it, the current driving regime is automatically switched to the coasting regime. As soon as the train passes the dead zone, the coasting regime switches back to braking, to safely complete the braking phase.

The implementation of this braking phase, was, at the beginning, quite problematic, having become easier when the way it is calculated was changed. After being defined, the speed profile representation was some kind messy since braking was represented as accelerations and vice-versa. At the end, its calculation, as an acceleration phase shows that it could solve the problem at hand.

Before going into another driving regime phase implementation, some tests were carried out so as to verify the algorithm response, the results being presented in Fig. 3.16. Algorithm tests consists in using the data related to a line and it is expected that they will be able to calculate a braking profile. In order to test the maximum number of anticipated cases, the speed limits on the line were forced because at this time there is no actual travel record that can be used to do this kind of tests.



(a) $v_{bk} = 13.1\ m/s$ (b) $v_{bk} = 19.4\ m/s$ (c) $v_{bk} = 15.8\ m/s$

Figure 3.16: Results for braking phase determination.

The first result, Fig. 3.16a, represents an example of a braking phase with velocity limits above v_{bk}. When this happens, a braking phase can be determined without any driving regime exchange, resulting in a continuous braking from v_{op} to final speed. Recalling Fig. 3.14, this first image is an example of the first two anticipated cases presented in Fig. 3.14a and Fig. 3.14b, respectively.

Fig. 3.16b represents an example of a braking phase on a line with more than one velocity limit. A tight velocity limit near to arrival station was introduced to force an exchange of driving regimes inside the braking phase. The journey has a velocity limit of 28 m/s and in the last 700 journey meters a limit to 8 m/s is imposed. The reduced speed zone lasts 400 m, and in the lasts 300 m the maximum speed goes back to 28 m/s. The speed profile calculation, for this journey, combines two of the anticipated cases presented in Fig. 3.14. For its analysis the figure is divided in two parts. The first part involves the first 500 m and the second part the remaining 500 m. Starting with the first five hundred meters, the speed profile determination follows the methodology presented in Fig. 3.15. As already explained, the calculation is made in two stages. The first one begins at 0 m, with initial velocity, and ends when the train reaches the following velocity limit value, approximately at 180 m. During this first stage, the braking phase is calculated without any problem, a braking regime is applied. After this calculation for the velocity profile, from the initial to the following limit value, the algorithm starts the second stage. During this stage, an acceleration between the train's current velocity and the following limit value is estimated and the maximum velocity reached, before the reduced speed zone is also determined. Once the maximum before the start of acceleration regime velocity is calculated an iterative process is defined. During this stage, the algorithm, at each time step, increases braking velocity and, at same time, starts to determine the space needed to accelerate between the following limit and the train's current velocity. This process ends when both profiles,

acceleration and braking, intersect and this point corresponds to the maximum velocity value. Having calculated the maximum velocity value, the sequence of driving regimes applied to determine braking phase is the following: braking, from zero to maximum; acceleration, between maximum velocity and limit value; and cruising, throughout limit length. The last 500 meters represents a second case, similar to the one shown in the following figure, Fig. 3.23c. Both will be examined in the next lines.

The last result studied, Fig. 3.23c, represents a new case, already anticipated, implemented in the algorithm, Fig. 3.14e. This characterizes a case where current velocity limit is lower than braking velocity value, and at a certain kilometer point, a limit increase happens. The resultant speed profile is at first a braking from initial velocity to velocity limit, being selected cruising regime to maintain a velocity equal to the limit during its length. In the kilometer point where limit changes, the algorithm switches again the braking regime until train reaches braking velocity.

In conclusion, analyzing all the results presented in the figures, it can be realized that the algorithm presents a good performance in all tested situations. The calculated speed profiles are as expected, once they follow the rules imposed during implementation. The only detail missing is that, in some of the calculated speed profiles, the recommended speed value for train slightly exceeds the limits. This little detail is visible in one of the presented results. Even so, this result is accepted as the train cannot be driven with this level of accuracy. After analyzing the results, this appears to be caused by approximations made during calculations.

3.7.1.2 Acceleration

The second speed profile phase to be determined is acceleration. The determination of this phase starts after braking calculations. This phase of speed profile is used to accelerate the train from an initial velocity to a cruising one, defined as v_{op}. Train acceleration happens in the beginning of the journey, starting with zero velocity at departure stations and ending at the moment when the train reaches v_{op}. During this phase, train traction's characteristics are considered, avoiding the determination of speed profiles impossible to be achieved. Since cruising velocity is given to the TMS algorithm as input, the determination of the acceleration phase includes the time and space needed as well as the entire velocity profile representation.

The acceleration phase is determined from the initial point, coincident with the departure station, to the moment when the train reaches cruising velocity, previously unknown. Once this phase is used in train acceleration, the default driving regime applied is acceleration. Since there are some con-

straints on the line, the driving regime can be changed during each acceleration phase. Following the same methodology used at the braking phase, the algorithm implementation included some anticipated cases which can occur in real railway lines. Those cases are presented in Fig. 3.17.

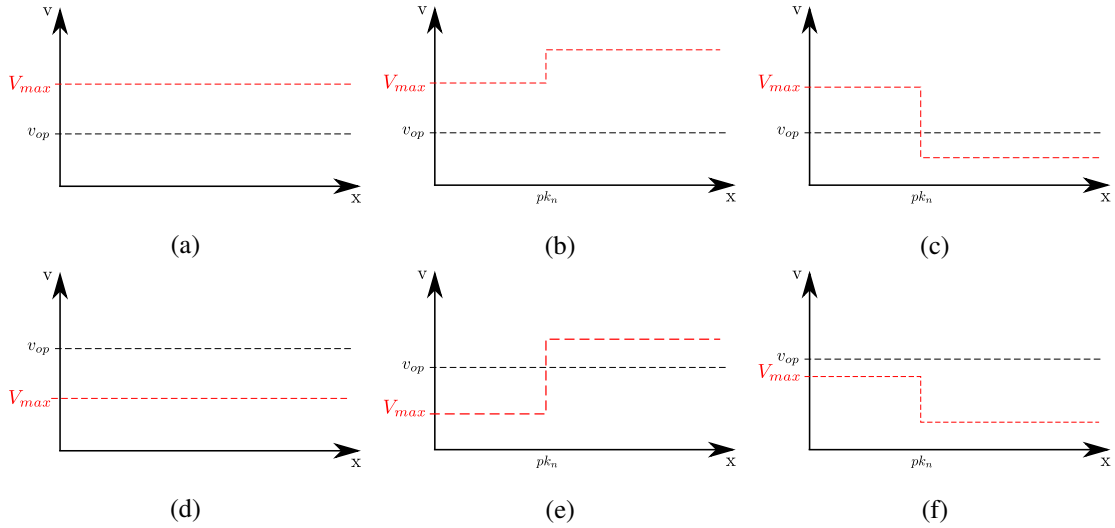


Figure 3.17: Acceleration phase: comparison between v_{op}, current and following velocity limit.

The first two cases, Fig. 3.17a and Fig. 3.17b, are the simplest ones since they cannot be seen as real constraints for speed profile determination. Velocity limits are always higher than the final velocity for the acceleration phase (v_{op}). So, the train driver will be advised to accelerate the vehicle throughout the phase without any concern. On the other hand, the third case, Fig. 3.17c, represents all lines where the current velocity limit is higher than v_{op}, and at a determined kilometer point, the limit falls down to a value lower than cruising velocity. The determination of an acceleration phase under these conditions is carried out in two stages, as presented in Fig. 3.18, and is next explained.

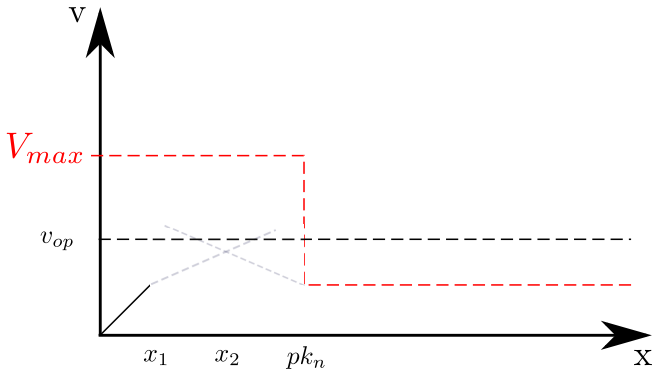


Figure 3.18: Acceleration phase: braking estimation.

Figure 3.18 shows a graphic representation about the acceleration phase determination on lines where the following velocity limit drops below cruising velocity. As stated before, the determination is carried out in two stages. The first one is the velocity calculation from the initial point to the moment when the train reaches the following velocity limit value, identified with x_1. Then the algorithm launches the second stage. Knowing that the following limit will require a reduction in the train speed to be met, the train driver should be warned in advance, to start the braking process. This means that the second stage consists of an estimation of the braking regime to meet the following limit value. Knowing the train's current position and velocity as well as the following limit value and position in each time step, train actual velocity in the acceleration phase is determined and, at same time, the space needed to brake the train between the current velocity and the following limit is estimated. When space needed to brake between both velocities is higher than the distance between train's current position and the one at which the velocity limit changes, the train driver is advised to start a braking regime. The velocity reached by the train may be the cruising speed or any speed which may be safely reduced within limits. In fact, in this case, if cruising velocity is reached, acceleration phase is considered done. In the event that, at position x_2, train velocity is lower than v_{op}, the acceleration phase should include, in addition to the braking phase, a speed maintenance along the limit. As soon as the limit increases, the train will accelerate again until cruising velocity is reached.

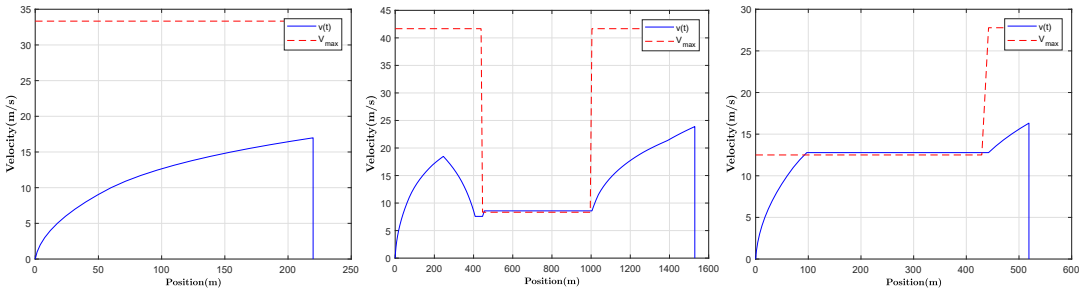
In the second line of Fig. 3.18 first row, the remaining represented cases are lines where the current limit is lower than cruising velocity. These will now be examined. Case (d), Fig. 3.17d, is the one where the current and the following limits are always lower than cruising velocity. This results in a small acceleration from the initial point to the limit value. Once the train has reached the line limit, a cruising regime is used to maintain a constant velocity. In the presence of a similar case, the algorithm will maintain the train velocity equal to the limit value, and the train driver will be advised to accelerate when an increase in the limit happens. Case (e), Fig. 3.17e, is an extension of case d), showing the point where velocity limit increases and exceeds cruising velocity. For this case, the algorithm is ready to determine a speed profile. The train driver is advised to start with an acceleration from the initial point to the instant at which the train speed matches the velocity limit. As soon as the train speed equals the limit within the reduced speed zone, the algorithm shifts from the driving regime to cruising one. After crossing the slow speed zone, the driving regime is changed back to acceleration so as to complete the acceleration phase. The last case, Fig. 3.17f is similar to the one presented in Fig. 3.17c. The algorithm is ready to estimate braking between velocity limits values. In

this last case, the train driver is advised to press the brake between velocity limits at an appropriate position, and is advised to accelerate only when a velocity limit increase happens.

As happens with braking, neutral zones location is also revelant at this phase. This line characteristic is considered since it is intended to use the algorithm in real railway lines, where these zones are quite probable to appear. The algorithm uses function NotchRestriction, presented on Section 3.4, to search in the input files where neutral zones appear, and as soon as one is detected, the driving regime is automatically switched to coasting. This driving regime is the most appropriate since it causes a train deceleration affected by train mass because there is no applied traction force. As soon as the train completes the dead zone, the algorithm switches back to acceleration, or to any other driving regime if there are other line constraints requiring it.

The implementation of the acceleration regime was easier than braking since the driving regimes selection had a better agreement with real train profiles. The implementation of braking regime calculations required some attention related with how braking is determined and how the information should be stored in order to reduce the processing time. In the end, the implementation of the acceleration phase exhibited appropriate results in accordance with line constraints and algorithm objectives.

After implementing the acceleration phase, the algorithm was subjected to some tests, with the purpose of understanding if it met the purposed requirements. These tests were done considering a railway line with some imposed limits in order to have a good number of diverse situations. The results are presented in Fig. 3.19, and since velocity limits where manually introduced, there is no real data to be compared with the algorithm outputs.



(a) $v_{op} = 16.6\ m/s$ (b) $v_{op} = 23.8\ m/s$ (c) $v_{op} = 15.8\ m/s$

Figure 3.19: Acceleration phase: examples of speed profiles.

The first result presented, Fig. 3.19a, shows an acceleration phase determination without any constraints since velocity limits are, in the whole acceleration phase length, always above train velocity.

As can be seen, the speed profile resultant is a continuous acceleration from the initial velocity to the cruising one, v_{op}.

Figure 3.19b shows the second result obtained. It exemplifies an acceleration phase with more than one driving regime, caused by changes on speed limits. Following the explanation given before, this result can be divided in two parts. Defining the speed profile division at kilometer point 800 m, it can be seen that this first part represents an example of acceleration phase determination considering case (c), shown in Fig. 3.17c. As it has been explained, in this situation, the speed profile is determined in two stages. The first stage is used to determine first acceleration, from initial velocity to the following velocity limit value. When train achieves a velocity equal to the following limit value, the algorithm starts the second stage, where braking between the train's current velocity and the following limit is estimated. The maximum velocity value is also determined, through an iterative process. Train velocity for acceleration regime (left side) is updated in each time step. At same time, the braking regime (right side) between train actual velocity and next limit value is updated. When both profiles intersect, the maximum velocity is found, which can be any value between the following speed limit and the cruising speed. Since the maximum velocity reached is inferior to the cruising speed, the algorithm continues to calculate the acceleration phase after the speed limit. As a result, in this first part, the resultant speed profile is defined as an initial acceleration, followed by a braking one, with a final cruising regime to maintain the train's velocity at the reduced velocity limit zone. The second part of the figure represents a similar case to the result presented in Fig. 3.19c, and both will be discussed together in the next paragraphs.

The third result is an acceleration phase on a railway line with velocity limits, as presented in Fig. 3.17e. In this example, a speed limit was introduced in the beginning of the journey, forcing the algorithm to calculate an acceleration stage that includes a first acceleration phase, finished when v_{op} is reached. This is followed by a velocity maintenance and finally the train accelerates as soon as the speed limit finishes.

In the end, it can be concluded that the acceleration phase determination matches what would be expected. As happened at acceleration phase, in some cases there are some approximation errors, when velocity limits are imposed, but these are somewhat neglected as the train driver will not have the required level of precision in speed control.

3.7.1.3 Coasting

Reminding the algorithm flowchart, Fig. 3.12, and having concluded the analyses of braking and acceleration phases, the next phase to be determined is coasting. This phase, as explained before, is used to reduce energy consumption since no traction force is applied. This phase of the speed profile starts at the end of the cruising and ends in the beginning of the braking one. In terms of velocities, the coasting phase starts with an initial velocity equal to cruising, v_{op}, and ends with braking velocity, v_{bk}. Following the flowchart, at the moment the coasting phase is calculated, it is only known the point where it should end as well as its initial and final velocities. Since the initial point to start coasting is unknown, this phase is also determined from the end to the beginning, similarly to braking. This means that the coasting phase is determined as another acceleration. Fig. 3.20 shows graphically how the coasting phase is calculated. The final position and the velocity of braking phase are considered as initial points for coasting. This way, following coasting phase properties, the corresponding speed profile is determined as an acceleration until cruising velocity is reached.

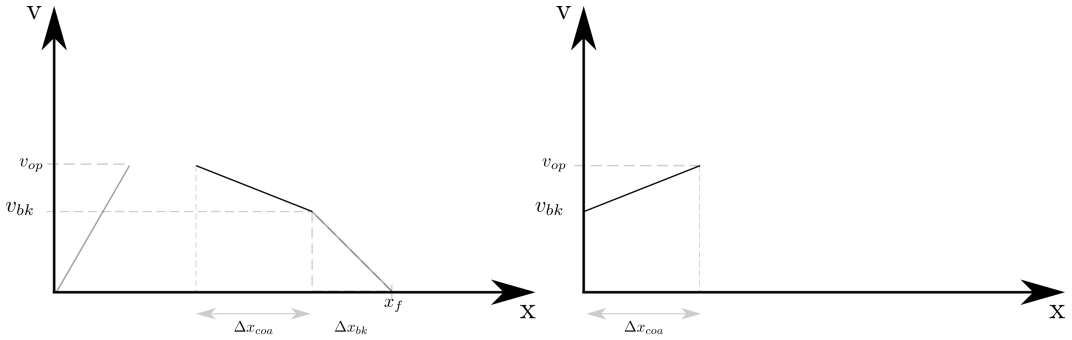


Figure 3.20: Coasting phase - graphical representation of the phase determination.

The change from a deceleration phase to an acceleration happens in the same way as it was done for the braking case. All train acting forces are determined considering the coasting phase as deceleration but at the moment when *train_dynamics* function returns train acceleration, it gives an opposite value. To search for line constraints, functions which implement a backward search are used, considering the correct initial position. In addition to these details, coasting phase is calculated in a similar manner to the other two phases.

Once more, speed profiles for coasting phase are limited to line constraints. The driving regime selection is based in the comparison of velocity limits imposed by the railway line together with the train's current and cruising velocity. One of the advantages of braking determination and coasting

phases as accelerations is the possibility to adapt all cases considered for the acceleration phase. Fig. 3.21 presents all cases contemplated at the coasting phase, and posteriorly implemented.

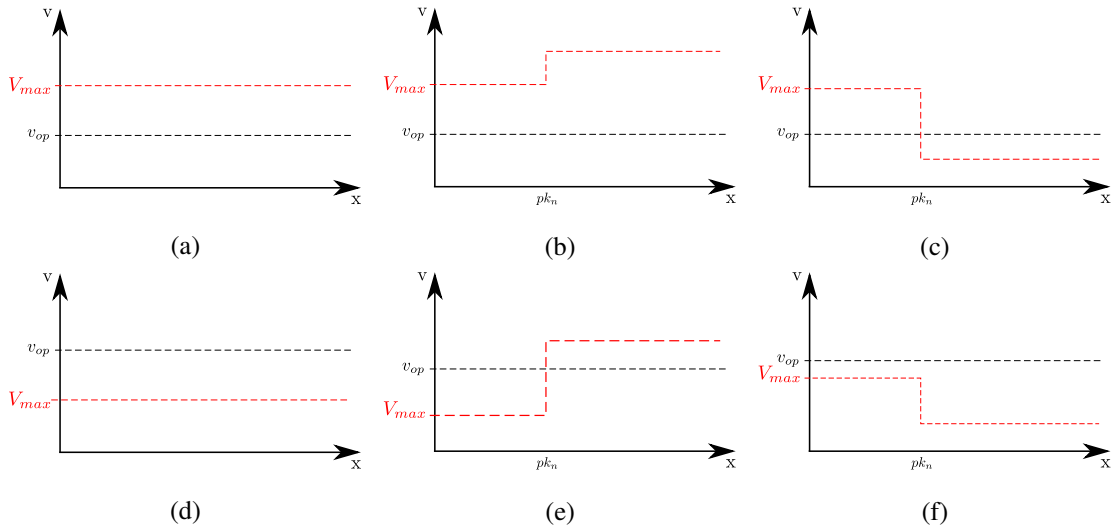
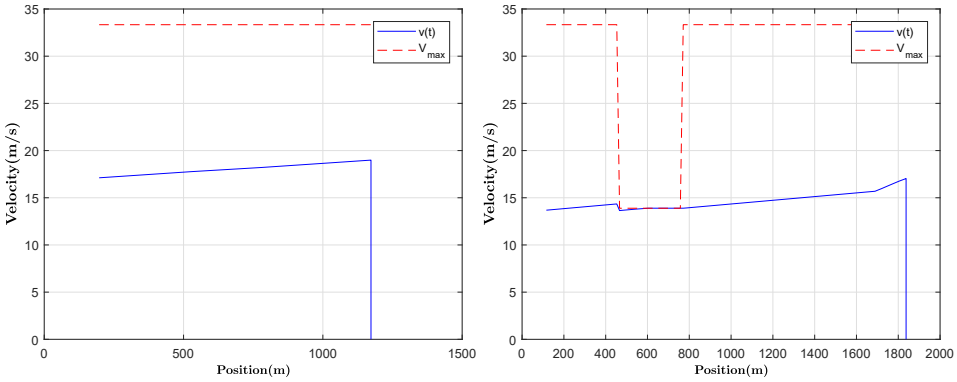


Figure 3.21: Coasting phase: comparison between v_{op} and velocity limits.

Looking at Fig. 3.21, it can be seen that all cases considered for coasting are the same as those considered at acceleration and braking phases. Since an explanation of each of those was previously presented, they will not be explained again. Comparing with acceleration phase, the difference consists in the considered driving regime. Instead of using acceleration as the default driving regime, coasting is the selected one. Another change is that braking estimations are changed to acceleration estimations, Fig. 3.21c, as at the braking phase. Although the initial speed of this phase is always set to a non-zero value, it does not fit into these comparisons since coasting always starts with cruising velocity and ends with the braking one. That is to say, that somehow this speed is guaranteed by the braking phase.

In terms of neutral zone locations, on coasting phase, when they are detected, the algorithm forces changes the current driving regime to coasting. In most of the cases, this action is not needed, but the algorithm is prepared so that, whenever necessary, the coasting regime is forced.

Figure 3.22 represents two examples of cruising phase determination. Both results were taken from a railway line where some velocity limits were imposed.



(a) $v_{op} = 18.5\ m/s$ (b) $v_{op} = 16.8\ m/s$

Figure 3.22: Coasting phase: examples of speed profiles.

The examples here presented show two different cases of coasting phase determination. The first one considers a line without any speed restriction while the second one a line with a speed limit zone. Starting with first result, Fig. 3.22a presents the case without any restriction. As explained, the algorithm starts coasting phase determination from the end, considering as initial conditions the ones defined at the braking phase. In the first result, as can be seen, the algorithm determines a coasting phase where the train is coasting from the beginning until the end. This happens since velocity limits on the line are always above train velocity.

The second figure, Fig. 3.22b, shows an example of a coasting phase determination with a speed limit. To force this result, a speed limit drop, from 33 m/s to 14 m/s was introduced in the position where the coasting phase was likely to occur. The result obtained, looking at the algorithm implementation's, goes as expected. This phase, as stated, is determined considering the final position as the initial point, which results in a slow speed increase, until the train reaches cruising velocity. As soon as speed limit is reached, a small acceleration phase is estimated in order to force a train velocity within the limit. The estimation of an acceleration is made once travelling time must be met. It should be noted that the coasting phase is usually used to reduce energy consumption, but it increases travelling time, which makes the introduction of this phase a trade-off between energy consumption and travelling time. Considering both variables, the algorithm imposes a slight acceleration before entering a speed limit zone. During the limit zone, train velocity is maintained at the maximum permitted and then the coasting regime is again selected.

Both results presented show that the algorithm is calculating coasting phases as expected. Although there is no associated image, another test was carried out, where a low velocity limit (under

8 m/s) was introduced into the line. This test was done several times, but in all runs the algorithm returned a speed profile where the coasting phase was out of the limit zone. The results obtained are quite expected, since the coasting phase is only used when it is possible. The introduction of a low speed limit condition into the speed profile results in the limit, which is covered by a braking or cruising phase. In conclusion, it can be stated that the coasting phase determination correctly answers line restrictions.

3.7.1.4 Cruising

The speed profile determination ends with cruising phase. This last phase is used to complete the remaining distance between the departure and the arrival station, which is still not covered by the other three phases. All other phases are determined considering final velocity, calculating the corresponding time and space needed. Before starting the determination of the cruising phase, the remaining space is determined and the algorithm imposes that cruising phase must cover all of it.

Cruising starts at the point where the acceleration phase ends, with an initial velocity equal to the cruising speed, v_{op}. It ends with the same velocity at the point where the train driver is advised to start coasting. During this phase, advices are given to the train driver to maintain a constant speed. This is valid in all cases where there are no line constraints. In the presence of line constraints, the algorithm must be able to determine a speed profile with the purpose to advise the train driver on how to drive without violating any restrictions as well as compromising passenger's safety. During the cruising phase, is expected the use of the corresponding driving regime: cruising. When a velocity limit change happens, this driving regime may not be the most appropriate, being necessary to switch to another one. As it happens in all other driving phases, the algorithm was developed anticipating possible cases, as presented in Fig. 3.23.

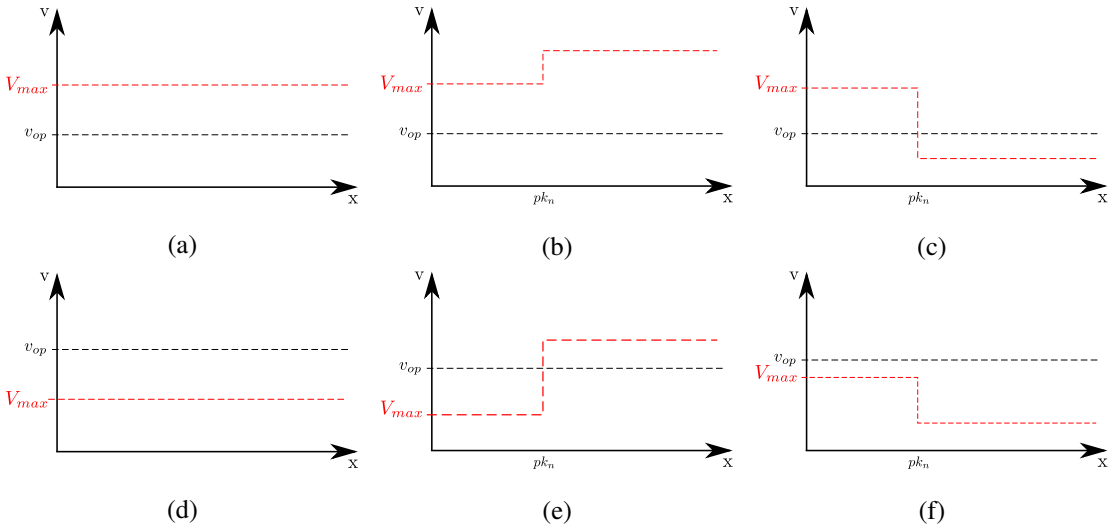


Figure 3.23: Cruising phase: comparison between v_{op} and velocity limits.

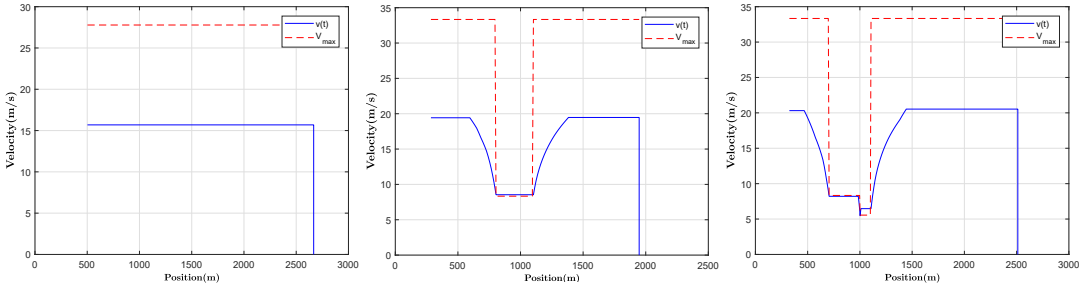
Considering cruising velocity and speed limits in the line, some cases may happen, being necessary to determine the speed profiles determination in these situations. Although this phase is already started with cruising speed, its implementation also considers cases where the current limit is lower than the cruising speed.

The handling of all cases considered is very similar to what was carried out at all other phases. Starting with case (a) and (b), presented in Fig. 3.23a and Fig. 3.23b, the speed profile can be determined without any exchange considering the riving regime, since limits are above train velocity. Fig. 3.23c represents a more delicate case, the change into a braking to make the train driver accomplish velocity limits being necessary. The process is similar to the one presented in acceleration, only being necessary to determine the point where braking must start, once maximum velocity is already known. Second line represent cases where actual velocity limit is lower than cruising. In all cases, the algorithm is ready to use the cruising regime to maintain the minimum velocity possible (limit or cruising). Once speed limit zone is crossed and train velocity is below cruising, the acceleration regime is selected in order to achieve maintain it.

Related with neutral zones, the algorithm is prepared to switch from the cruising regime to coasting, when they are detected. Once the train finishes the neutral zone, the algorithm returns to the cruising regime, or probably to an acceleration phase, to resume cruising speed.

After implementation, the algorithm with the purpose of analysing its performance. The algorithm tests were done forcing some particular situation, as tight velocity limits. Some algorithm results are

presented in Fig. 3.24.



(a) $v_{op} = 15.5\ m/s$ (b) $v_{op} = 23.8\ m/s$ (c) $v_{op} = 20.3\ m/s$

Figure 3.24: Cruising phase: examples of speed profiles.

An analysis of Fig. 3.24 examples for cruising phase are presented. The first example, Fig. 3.24a, shows a cruising phase without any driving regime change. It represents all cruising phases where speed limits are always above cruising velocity, resulting in a cruising phase without any restriction. As it can be seen, the cruising phase starts in the end of the acceleration phase with a velocity equal to v_{op}, and for that reason, speed profiles displayed do not start at the axis origin.

The second example, Fig. 3.24b, shows a railway line with a velocity limit coincident with the cruising phase, which represents case (c) and (e) from Fig. 3.23. The resulting speed profile includes driving regimes changes necessary to advise the train driver within line constraints. The cruising phase starts with v_{op} and the cruising regime was maintained during the first meters. As soon as the algorithm detects a drop in velocity limits, an estimation of the space required to decelerate the train starts. Having the point where deceleration must start, the algorithm changes from the cruising to the braking regime, and after the point where the train achieves a velocity equal to the speed limit, the cruising regime is again selected. Cruising regime is maintained until there is a further change in the speed limit, and since it increases, the acceleration regime is applied in order for the train to reach the cruising speed once more. Cruising velocity is then maintained until train travels the whole distance defined for this phase.

The last example, is similar to the previous one, the only difference being the introduction of a second velocity limit, lower than the former. In this case, the speed profile determination for the cruising phase requires the estimation of a second braking stage. The final velocity profile will include a sequence of cruising and braking regimes, being applied in the end an acceleration to finish the speed profile phase at cruising velocity.

As a conclusion, the algorithm seems to perform well for cruising phase determination. In all results, the algorithm shows velocity profiles which maintain cruising velocity during the whole phase, and accomplishes velocity limits imposed by the railway line. As happened in other phases, there are some errors on velocities, which are neglected by the reasons already presented.

3.7.2 Solution Construction

The previous section, Section 3.7.1, presented how each driving phase is determined for a given trip. Before presenting the algorithm's final solution, one last operation must be done. For a better explanation, an example of a speed profile determination will be presented.

The algorithm implemented has the purpose to make speed profiles to be used on a DAS. Given information about the actual journey, the algorithm receives possible values for cruising and braking velocities, which are posteriorly used on speed profile determination. As explained before, speed profile phases are calculated separately, and as consequence, after being all successfully determined, the result obtained consists of a set of vectors corresponding to each of the phases. Vector sets includes information about train position and velocity as well as acceleration and forces acting on the train in all journey, discretized in time. In addition, time spent as well as distance required in each phase are also provided at each phase result. All this information is useful to build the complete velocity profile in order to provide the train driver with it. To concatenate all information spread over several variables and vectors, the algorithm uses the time and space needed in each phase. Based on time spent for each speed profile phase, a new vector is filled in order to have only one result, which will be posteriorly presented. To accomplish this task, a function following the pseudo code presented in Algorithm 8 was implemented.

Algorithm 8: Speed Profile Construction

solution_construction (Δt_{acc}, Δt_{cru}, Δt_{coa}, Δt_{bk})

first_step=Δt_{acc};

second_step=$\Delta t_{acc} + \Delta t_{cru}$;

third_step=$\Delta t_{acc} + \Delta t_{cru} + \Delta t_{coa}$;

fourth_step=$\Delta t_{acc} + \Delta t_{cru} + \Delta t_{coa} + \Delta t_{bk}$:

for *i=1:1:first_step* **do**

$F_t(i) = F_{t_{acc}}(i)$;

$a(i) = a_{acc}(i)$;

$v(i) = v_{acc}(i)$;

$x(i) = x_{acc}(i)$;

end

for *i=first_step+1:1:second_step* **do**

$F_t(i) = F_{t_{cru}}(i - first\_step)$;

$a(i) = a_{cru}(i - first\_step)$;

$v(i) = v_{cru}(i - first\_step)$;

$x(i) = x_{cru}(i - first\_step)$;

end

for *i=second_step+1:1:third_step* **do**

$F_t(i) = F_{t_{coa}}(third\_step + 1 - i)$;

$a(i) = -1 \times a_{coa}(third\_step + 1 - i)$;

$v(i) = v_{coa}(third\_step + 1 - i)$;

$x(i) = x_F - x_{coa}(third\_step + 1 - i)$;

end

for *i=third_step+1:1:fourth_step* **do**

$F_t(i) = F_{t_{bk}}(fourth\_step + 1 - i)$;

$a(i) = -1 \times a_{bk}(fourth\_step + 1 - i)$;

$v(i) = v_{bk}(fourth\_step + 1 - i)$;

$x(i) = x_F - x_{bk}(fourth\_step + 1 - i)$;

end

returns $F_t(t)$, $a(t)$, $v(t)$ and $x(t)$;

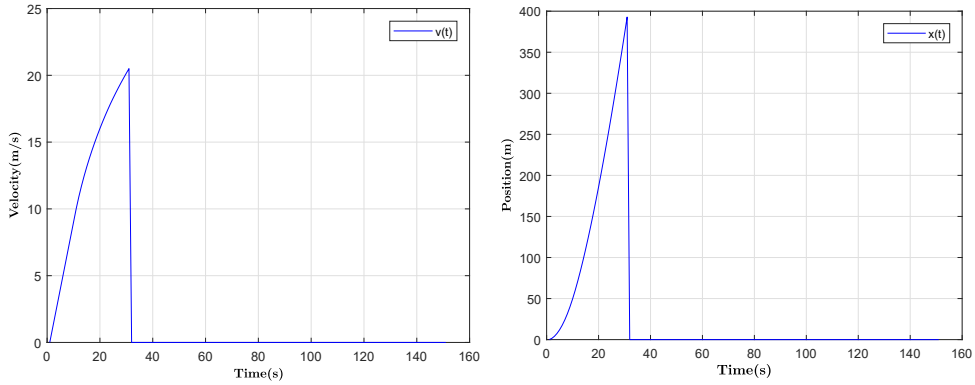
The information, as shown in the algorithm, is concatenated based on time spent in each phase. The reason to use time spent in each phase to fill new vectors is associated to the fact that the implemented train model uses time as an independent variable, which makes the algorithm determine, in each time interval, a new value for all train dynamics variables. Since in each time instant, a new value is determined, and knowing that all information stored have a time stamp associated with each value, a new routine was easy to implement with the purpose of introducing all determined values is a new vector, depending on time spent in each phase. The implemented function starts by defining new vectors to store the result for traction force, train acceleration, velocity and position. The vectors size is defined by the time spent to end the actual journey, that is to say the result is the sum of each individual time spent in each phase, affected by the time step size. After being initialized all vectors, the total travelling time is divided in all three phases, by determining the variables which control where the routine must copy individual results. Those variables used to control copying routines are defined as *first_step*, *second_step*, *third_step* and *fourth_step*, corresponding to the final position for acceleration, cruising, coasting and braking, respectively. Filling in the new vectors begins with copying information related with acceleration and cruising phases, being the easiest ones, since there are no additional operations. The last two phases to be copied are coasting and braking. For these, a more complex operation is used. Since both phases are determined in a reverse way, the introduction of each value into final vectors must also be carried out from the last to the initial position. Besides that, the train position in both phases is determined considering the arrival station as the initial position. To introduce the correct value of the train's position in the final result, the position determined in these two phases must be subtracted to the arrival station location (x_F).

As an example of how this function works, a simple journey with a travelling distance of 1710 m to be accomplish in 120 s was considered. The algorithm received as input cruising and braking velocity values, 20.3 m/s and $11,8$ m/s, respectively. The results obtained for each driving phase are presented in Table 3.4.

Table 3.4: Results for speed profile phases

	Δt (s)	Δx (m)
Acceleration	30	392.88
Cruising	16	317
Coasting	58	911.09
Braking	13	88.79

The obtained values are the result for each of the driving phases. The first phase presented in this case is acceleration, Fig. 3.25. The result obtained here is in accordance with journey needs and time and space shown.

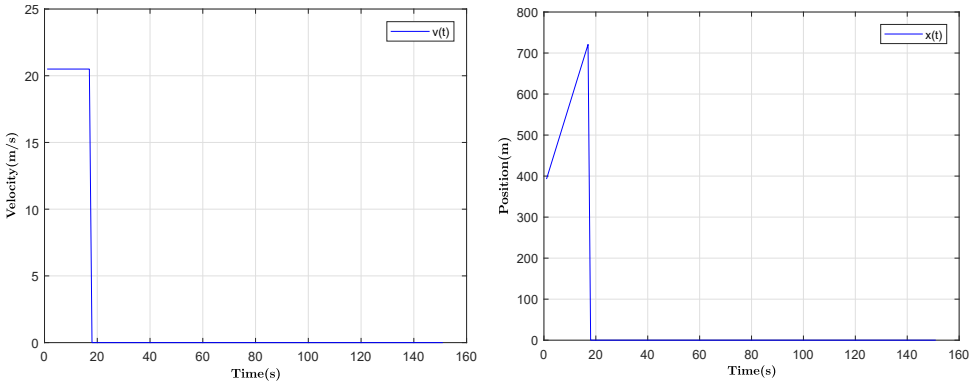


(a) Train velocity

(b) Train position

Figure 3.25: Acceleration phase: train velocity and position

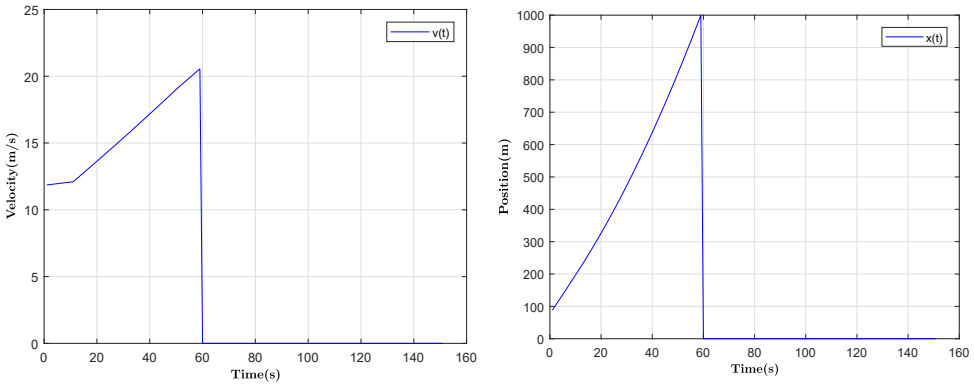
Figure 3.26 presents train velocity and position during the cruising phase, for the considered journey.



(a) Train velocity (b) Train position

Figure 3.26: Cruising phase: train velocity and position

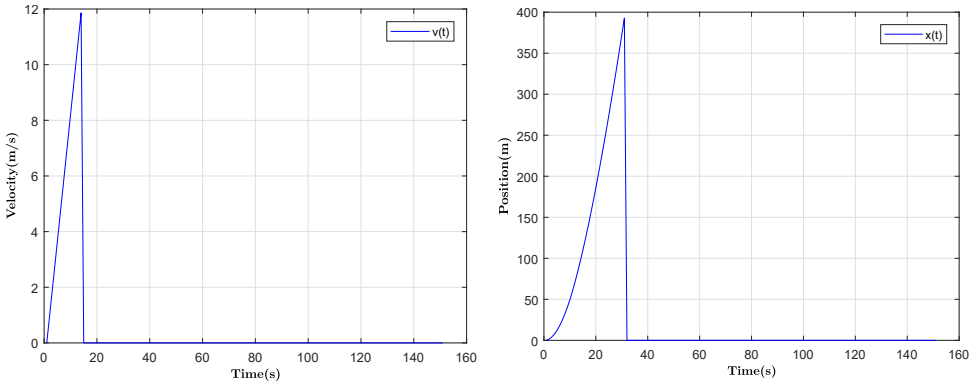
Following the sequence of speed profile phases used by optimal control, after cruising there will be a coasting phase. Train velocity and position in each time instant of this phase are presented in Fig. 3.27.



(a) Train velocity (b) Train position

Figure 3.27: Coasting phase: train velocity and position

The last result is braking phase, Fig. 3.28. As can be seen, braking phase starts the determination of train velocity from the end. In the same way, the position is calculated from an initial point, defined as 0. The algorithm that concatenates all this information into one makes the proper change of variable.



(a) Train velocity

(b) Train position

Figure 3.28: Braking phase: train velocity and position

Combining the information given in the table with the journey's total time spent, the algorithm determines final velocity profile, as presented in Fig. 3.29. Acceleration and cruising phase are the easiest phases, since it is only needed to know the initial and the final time for each phase, and the information contained in each phase output vectors is copied to final result vectors. On the other hand, for coasting and braking phases, in addition to determining the beginning and end of each phase, it is also necessary to copy the results in reverse. As can be seen, the routine implemented correctly accomplishes what is expected.

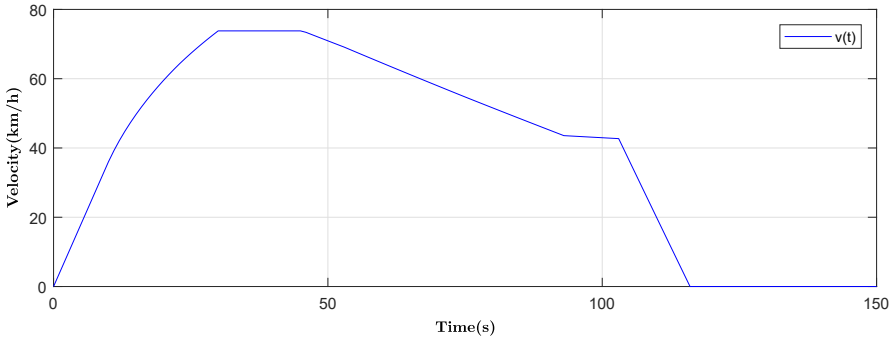


Figure 3.29: Train velocity vs. time graphic.

Regarding train position, presented in Fig. 3.30, it is also visible the good performance of the implemented function. Acceleration and cruising are once again easy to integrate in the final result; however, coasting and braking phases, as presented in the algorithm pseudo-code, are slightly more complex. The final position of the journey is used to determine the train's real position on the line, and, as it can be seen, it is combined without errors.

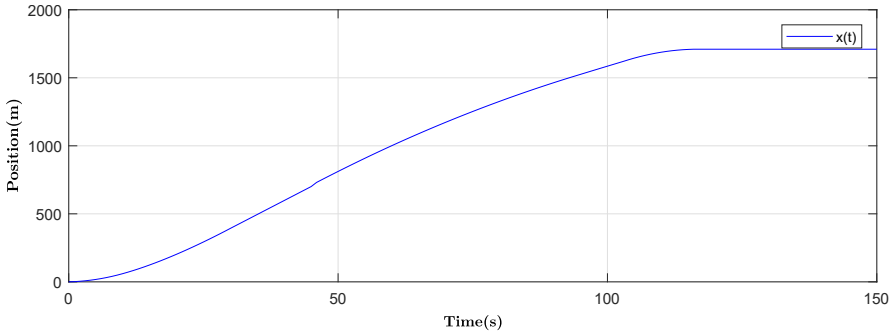


Figure 3.30: Train position vs. time graphic.

In the end, having train velocity in each time instant of the journey and the respective train position, also related with time instant, it is easy to plot the final result of the algorithm, which is a velocity versus position profile. The presentation of the final velocity profile in a graphic velocity versus time is in accordance to DAS, where the train driver must be advised in the right position about what action must be performed.

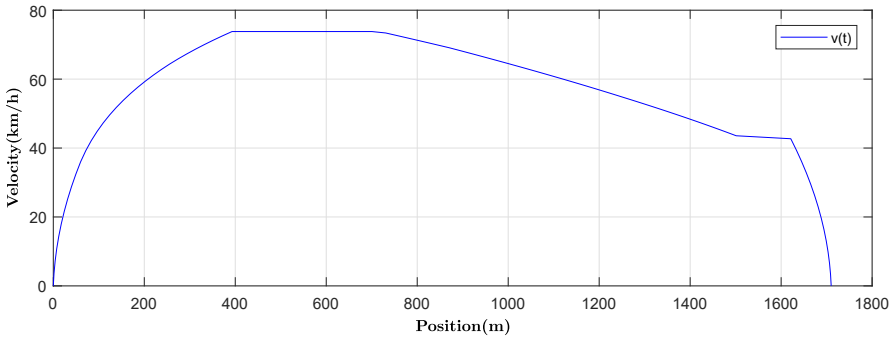


Figure 3.31: Train velocity vs. distance graphic.

CHAPTER 4

Optimization Algorithm

4.1 Introduction

This thesis, as presented before, envisages the development of an algorithm to work on a DAS. The algorithm must be able to determine velocity profiles following a defined objective function. The determination of a speed profile must be constraint to the train dynamic model, to line characteristics, and to time and distance requirements.

The optimization problem proposed during this thesis, is a large combinatorial one, associated to decisions about time and velocities, for each of the four considered traction regimes, constrained to line slopes and velocity limits. To solve this problem, a meta-heuristic approach called SA was adopted.

This chapter will start by presenting the SA algorithm. A short introduction about the algorithm and how it works will be given. After that, algorithm implementation for DAS will be presented and discussed. In the end of each section, the obtained results will be shown followed by a short analysis and discussion.

4.2 Simulated Annealing

The SA algorithm was created by Kirkpatrick in 1983, inspired by a metallurgical process used to organize material structures. The process name is annealing and it has the purpose to increase material ductility and to reduce its hardness. It starts by heating a material up to a certain temperature that allows free atoms movement inside of its structure. The atom movement is due to the amount of energy contained inside the material, caused by temperature increase. After being heated up, the material starts to cool slowly and, as a consequence, the atom movement starts to slow down. The atoms movement naturally will tend to become stable at a minimum energy structure. In the end, after

following a slowly cool down, the formed material will present a structure where atoms will be as aligned as possible. The material final structure, as expected, will tend to have a minimum energy possible, considering material initial conditions. From the physical process, some concepts must be taken, since they are highly important to the SA algorithm implementation. These concepts are:

- At higher temperature, atoms have higher movement;
- A slowly cool down scheme is highly important for a strong structure;
- Material final structure is dependent on initial temperature and a slow-down scheme.

Bearing this in mind, it can be concluded that the final structure of the material is dependent on its initial temperature, as well as how temperature decreases during all the process. In order to be able to solve combinatorial optimization problems, the SA algorithm emulates the annealing process, finding an optimal solution. The optimal solution presents the lowest value of the cost function. So, the algorithm implementation considers some operations which may be clearly defined and tuned:

- The generation mechanism;
- The temperature scheme;
- The cost function;
- The solution acceptance probability;
- The stop criteria.

Each one of these operations requires some knowledge about the system on which the SA will be implemented. The definitions of all the functions needs parameter tuning, constants determination and implementation rules. The process of tuning all functions is commonly an iterative process, being the best option defined after multiple iterations, after analyzing the algorithm's performance and the results' quality. Each one of those functions will be explored throughout this section.

The SA algorithm is used to find the optimal solution for a given problem. The algorithm's implementation lies on annealing process, used to find the minimum energy structure of materials. The algorithm will be used later for minimization. Because of that reason, it will be presented considering a minimization of the cost function. With the purpose of understand the similarity of annealing with SA and to make explanations easier, it is common to establish some parallelism between annealing processes and algorithm's operations and definitions:

- Each solution can be called a system state;
- The cost value is the corresponding energy of a state;
- The algorithm uses a control variable which represents temperature.

Based on the parallelism presented, algorithm implementation becomes easier. The algorithm here described is used to find an optimal solution for a given problem. In other words, optimal solution can be defined as the one with the lowest cost function value, which means the one with minimum value of energy. The SA starts by generating an initial guess, being posteriorly evaluated by the cost function. The generation of new values occurs through the generation mechanism. Once a solution is defined and there is enough material for objective function determination, the cost value associated is calculated. Depending on the cost value, and comparing with previous states, one of two possible hypotheses can happen, defining the algorithm's flow. From the comparison between actual energy and the best obtained solution, the algorithm can decide if the current solution must be accepted as a better solution or, on the contrary, rejected. If accepted, the algorithm follows for the following iteration, saving the current result as the best one, and generating a new solution in its neighborhood. The second option is to reject the current solution. The algorithm continues the search of solutions in the same manner as for the current solution. In fact, the SA algorithm is a little more complex. In the case of a solution being rejected, there is a probability that this worst solution will be accepted. This chance decreases when the number of iterations increases. This is so because using only a direct comparison between solution and following the first two hypotheses here described, the algorithm only accepts solutions which successively bring improvements to the cost function. Regardless of the gain between iterations, the algorithm only verifies if it progresses and thus accepts all the solutions fulfilling it. It turns out that following this, the algorithm could be unaware of the convergence to a global optimum, being stuck in a local one. This way, in order to circumvent this problem, the SA may also accept solutions that do not bring improvements to the cost function. It turns out that when a solution is not accepted by direct comparison, a probability of acceptance value is determined and compared to a random one. Then, depending on this comparison, the solution can still be accepted, even degrading the cost function [46, 52–56].

Figure 4.1 presents the generic flowchart of SA algorithm. Being a general SA flowchart, it presents an extra loop that corresponds to the so-called re-annealing process. In this thesis, the implementation of the re-annealing process was not considered.

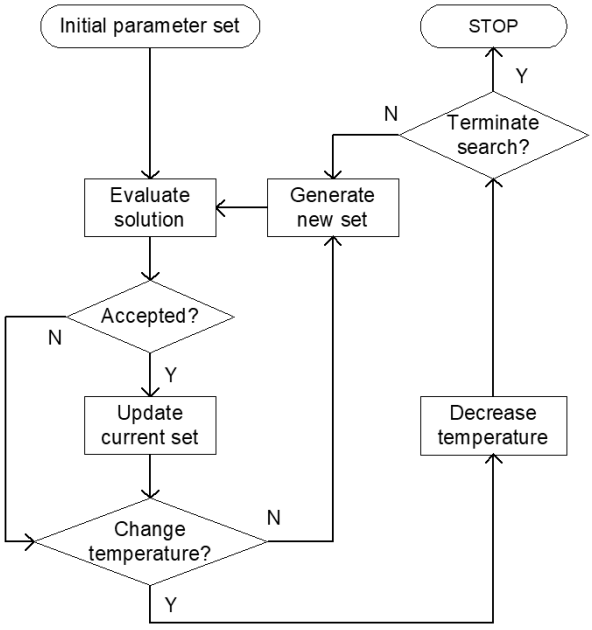


Figure 4.1: Simulated Annealing flowchart [46].

Throughout next sections, a description of all SA operation will be given. It will start with the generation mechanism on Section 4.2.1, followed by temperature scheme and cost function implementation in Section 4.2.2 and Section 4.2.3 respectively. The last part, concerning how acceptance criteria works, is discussed in Section 4.2.4 and the last operation is stop criteria, in Section 4.2.5.

4.2.1 Generation Mechanism

Generation mechanism on SA is responsible for determining new solutions. It specifies how a new solution must be generated. This mechanism is used in each algorithm iteration, to generate a new solution, considering the previous result as a reference point. The algorithm does a random search over the whole solution space. The most common practice, and also used here, is the generation of a new value, using a random Gaussian number with zero mean and considering a controlled deviation. To generate new values, in the first place the solutions space is defined. These solutions' space is limited by values which conduct to feasible solutions. After well defining the solutions space, the function that generates random values is used to perturb actual solution inside of all possible values. To perturb a solution, expression (4.1) is used. Basically, a new value generated for next algorithm iteration, x_{i+1}, depends on the actual value used, x_i. At a previous value, the algorithm uses a function that generates randomly values between 0 to 1 to select how much will be added, considering an

admissible value range, ΔX.

$$x_{i+1} = x_i + \Delta X \tag{4.1}$$

In this thesis, the generation mechanism considers two different rules. These two rules will be later explained, as well as and how they are used. Basically, those rules were implemented to define two different search types, local and global. Local search is implemented by applying (4.1), which represents a small disturbance of an actual solution. The second rule implements a global search within the whole solution space. This second rule is used when a big random jump is desired with the purpose to explore other parts of the solutions space.

4.2.2 Temperature Scheme

Temperature is a critical parameter on SA algorithm, used to control one of the main algorithm features. This parameter is essential for algorithm mechanism to escape to local minimums, since it directly affects the acceptance probability of a bad solution. Some authors denominate this variable as control loop, but sin it commonly represents temperature effect on material, temperature is the most appropriate name.

Since this is a critical parameter, it must be exposed properly. To determine temperature, two different stages must be defined in a first phase. These different stages can be a part of temperature scheme on the SA algorithm:

- Initial temperature value;
- Cool-down rate.

Starting with the initial value, temperature must start as high as possible since it happens with the annealing process, which starts with the heating of the material up to a high value of temperature in order to produce a free movement of its atoms' structure. In the SA algorithm, a similar behavior is desired. The initial temperature must also be high enough to allow bigger jumps in the solution space. This happens because temperature affects the acceptance probability of worst solutions, this being directly proportional to it. This means that the higher the temperature, the more chances a bad solution has to be accepted. So, the algorithm needs to have a good initial temperature estimation. Temperature which is too high can lead to instability, causing too many jumps inside the solution space, while a too small initial value can result in a local minimum convergence.

There are many approaches for initial temperature determination. One option, and the simplest, is setting a constant value as initial temperature. The exact value to be defined can result from several algorithm runs, in order to understand which one produces better results, [57,58]. Another alternative is to determine initial value by a relationship between the maximum amplitude value admitted for a bad cost function move with acceptance probability, as presented in [59]. Equation (4.2) shows how initial temperature can be determined.

$$T_0 = \frac{\Delta C_i}{ln(p_0)} \tag{4.2}$$

In (4.2), ΔC_i represents the maximum amplitude allowed for a bad move in the algorithm and p_0 is the corresponding probability of being accepted. ΔC_i can be determined after several algorithm runs, applying the average value of all bad moves, or can be simplified by using a unique example.

The second part of temperature scheme is the cooling rate definition. Regardless to the scheme choice, it is important to ensure a slow gradient of the temperature parameter. Looking once more at the annealing process, a fast cooling down results in a poor organization of material structure while a very slow one will consume more time for almost the same result. In the SA algorithm the right cooling rate also needs to achieve a good convergence, [60]. Some examples of cooling rates are [61,62]:

- Linear function as $T_i = sT_{i-1}$;
- Dynamic cooling rate.

Using cooling rate as a linear function, the first option is one simple way to implement it and it is the most applied. Cooling rate can be adjusted by parameter s, which is typically selected from a range between 0.8 and 0.99. The second option is more complex to be defined and implemented. Typically, it is used as a function dependent on initial and final values (estimated) for temperature, and, in some cases, it uses a number of iterations. An example of this type of cooling rate is the logarithmic cooling [63].

During the developments of this thesis, linear cooling rate was adopted as the most appropriate for decreasing temperature. Parameter s was tuned iteratively, several possibilities being defined. It was tested for several cases, and in the next sections, in the results demonstration, the applied s value will also be presented.

4.2.3 Cost Function

Cost function, often called objective function, is used by the SA to qualify solutions. The construction of cost function highly depends on the algorithm's objective, as well as the methodology adopted. The first thing to define when cost function is determined, is if a minimization or a maximization problem ins intended. After that, the variables which better represent the problem, or the most important for the main objective, must be selected. The cost function type, after these two steps, must be adjusted to the problem itself.

The way each solution will be qualified must also be defined. The cost function value can be, for instance, the result of an error or root square error. As an example, if the algorithm is used to determine parameters for a model that emulates a real physical behavior, the cost function can be the minimization of the difference between real measurements and the model output.

Later, in this chapter, the implemented cost functions will be presented.

4.2.4 Acceptance Probability

As mentioned several times before, one of SA main advantages is its capability of escaping from local minimums. This advantage is possible once algorithm can accept solutions that are worst then the selected last better one. The acceptance of bad solutions is based on a probabilistic method, determined by (4.3).

$$p = e^{\frac{-\Delta C}{T_i}} \tag{4.3}$$

So, the algorithm uses a Maxwell-Boltzmann distribution to determine if a bad solution must be accepted. This happens once the SA algorithm is based on Metropolis procedure. In Metropolis procedure, Maxwell-Boltzmann distribution is also used to determine if a small random perturbation on a multibody system is accepted, when system energy increases, [64]. In the SA algorithm, the acceptance probability is a function of cost function difference, ΔC, and the actual system temperature T_i. ΔC represents the difference between the current cost function value and the previous better solution, $\Delta C = C_i - C_{i-1}$. In accordance to the acceptance probability function, two different cases can happen, based on ΔC result. These cases are:

- $\Delta C \leq 0$, corresponds to solutions in accordance with cost function, which means the actual solution has a smaller cost function than the previous one;

- $\Delta C > 0$ happens when the actual solution is worse than the previous one.

The first situation relates to cases where the actual solution has improvements on cost function when compared with the previous one. This case is in accordance with cost function, which means that actual solution must be accepted. For all these cases, the acceptance probability is always higher than 1. The second case, corresponds to all scenarios where the current solution is worse than the previous one. These are all cases where the acceptance probability must be determined to define if it must be accepted or not.

When a worse solution appears, the acceptance probability is determined. After being determined, a function that generates random values between 0 and 1 is used to decide if the current solution must be accepted as better or, on the contrary, rejected. So, to decide that, the acceptance probability is compared with a random generated value, and in the event of being higher, the actual solution is accepted as better. If not, the SA considers that the actual solution must be rejected.

Another particularity of using (4.3) as acceptance probability is the fact that it is directly proportional to temperature. The acceptance probability decreases when temperature decreases, which means that a bad solution has higher changes to be accepted in the algorithm's first iterations than in the last ones. As ΔC increases, the acceptance probability decreases, which means the probability of a worst solution being accepted decreases with the distance from the solution to the main objective.

The acceptance probability was implemented in the developed algorithm as described in this section.

4.2.5 Stop Criteria

The SA algorithm searches for the solutions space for possible answers that introduce some gain on cost function. This search process is carried out iteratively, so as there is not any information about the time when it must stop, it never ends. Thus, before running the algorithm, a stop criterion must be set with the purpose of defining when the search must finish. Only after this stop criterion is satisfied, the algorithm is ready to present the best solution found. As it happens in all algorithm operations, there are also some possible options. The criterion choice depends on the algorithm's developer, which may define one without compromising algorithm convergence and implement the option which better meets the proposed objective. The stop criteria options can be:

- Maximum number of iterations;
- Cost function accepted value;
- Minimum temperature value;
- Maximum cost function gain between two consecutive iterations.

In this thesis, and in all SA algorithm implementations, the maximum number of iterations was used as stop criteria. Most of the applications carried out had the ultimate purpose of being executable in real time, hence the number of iterations was limited, to guarantee a minimum execution time.

4.3 Driving Assistant Algorithm

One of the main objectives of this thesis is the implementation of a real time algorithm suitable for a DAS. The development of this algorithm must consider the possibility of its use in a portable platform running side by side with a user interface. Typically, the portable platform goes near the train driver with the purpose of advising him about how to do the actual journey, in scheduled time and with minimum energy costs. During the course of algorithm development, new ideas were emerging, resulting in two possible implementations for it. Both implementations follow a similar methodology focused in solving most of actual railway system concerns, presenting solutions to reduce operations costs and maximize line capacity. The first methodology implemented follows the main focus of the thesis, which is the reduction of energy needs for train operation. The second approach is concerned with system punctuality, presenting solutions for reducing delays. Both approaches will be presented and subject to analysis throughout this section.

4.3.1 Energy Consumption

The first algorithm developed is aligned with the main objective of the thesis, which is the reduction of energy consumption during train operation. Most of the actual railway systems are near their full capacity, which requires careful management of the resources currently available. This resource management can be carried out in several perspectives, one possibility being the use of the algorithm developed within this thesis.

The developed algorithm aims to help train the driver to make decisions about the better way to run the train in order to comply with the schedule time, passengers' comfort, line constraints and minimum

energy consumption. The algorithm is not intended to make automatic driving of the vehicle, nor to replace actual drivers. Its main function is to advise them, being the driver's function to comply with the given advice. Obviously, the success of the algorithm will largely depend on the driver's compliance with it. So, the algorithm's main idea is, whenever a particular trip is intended to be made, that a solution must be completed during the time stopped at the departure station, after collecting the most accurate available information concerning trip constraints. Furthermore, it is clearly intended that the obtained solution has minimum energy consumption for current conditions.

4.3.1.1 Problem Formulation

The optimization algorithm will be applied to minimize energy consumption during train operation, as stated in (4.4). Since it is intend to test the algorithm on real railway lines, there are some limitations that must be imposed on speed profile determination. Also, there are some physical limitations, as explained previously, in Chapter 3, which should be accomplished to avoid any accident. These limitations should be expressed as algorithm constraints. The first constraint, (4.5), is used to guarantee that maximum velocity is never exceeded at any point of the journey. The second constraint limits total travelling time, (4.6), to the maximum allowed one. It is expected that all determined speed profiles will respect the schedule, defined by the train operator. This is important, since the amount of energy needed for a trip is highly dependent on travelling time. Considering a simple journey with fixed distance, consumed energy will decrease as the journey time increases. The last constraint, (4.7), was defined thinking of passenger security and comfort. It is important to have a controlled acceleration within security limits. This last constraint requires that for each value of the proposed train acceleration, it must be analyze and consequently verify if limit values are violated. If some of them surpass the maximum limit, or the maximum figure permitted by traction characteristics, it is automatically reduced to a minimum of two.

Minimize

$$T = \sum_{i=1}^{T} e_{c_i} + \mid e_t \mid \tag{4.4}$$

Constrained to

$$v_t \leqslant V_M \quad \text{for} \quad 1 \leq t \leq J \tag{4.5}$$

$$t4 \leqslant T \quad \text{for} \quad 1 \leq t \leq J \tag{4.6}$$

$$a_t \leqslant min\left[a_{max}, \frac{F_t - F_r - F_g}{M_{Total}}\right] \quad \text{for} \quad 1 \leq t \leq J \tag{4.7}$$

4.3.1.2 Algorithm Implementation

The implementation of the new algorithm started by deciding how TMS results can be used together with an optimization algorithm, with the purpose of helping the train driver with the task of reducing energy consumption during train operation. The selected optimization algorithm to accomplish this task was the SA. TheTMS algorithm's main role is the representation of the train dynamic model, determining speed profiles considering initial values given, as well as line conditions and journey requirements. The SA must evaluate each solution proposed and find the one that better fits all objectives. A flowchart representing all steps and operations carried out in the algorithm is presented in Fig. 4.2.

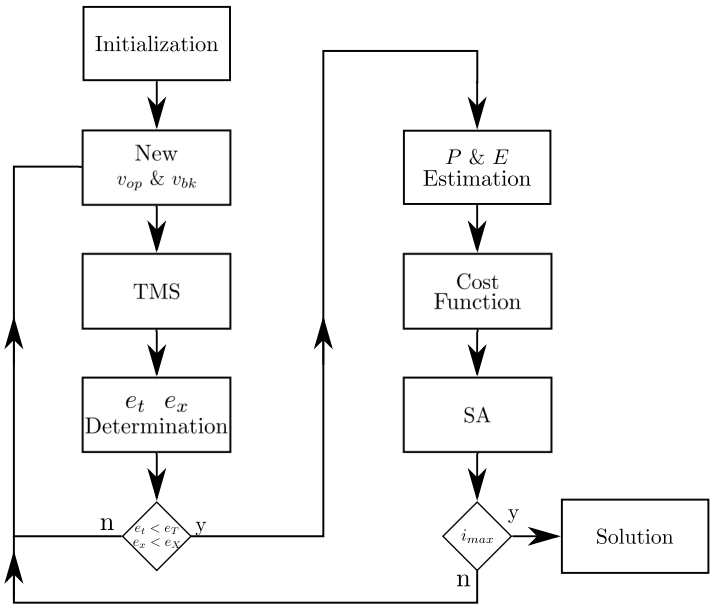


Figure 4.2: Algorithm Flowchart.

Before presenting an extensive explanation about how the algorithm works, the main operations are listed in order to make the following description easier:

- Generating random values for cruising, v_{op}, and braking, v_{bk}, velocities;
- Reading input data: distance, available time and line constraints;
- Using the TMS algorithm to determine the corresponding velocity profile;
- Verifying if the velocity profile respects initial requirements;
- Determining energy needs and the respective cost function value;
- Identifying whether the solution should be maintained or excluded (SA algorithm).

Following the order imposed by the algorithm, the first step is initialization. At initialization step, all inputs as train and line characteristics, and journey requirements are read. In addition to variables' reading, some variables are also initialized. SA needs to be initialized. In addition, the cooling rate and the maximum number of iterations must be defined. For the whole algorithm, in general, vectors and variables to store all intermediate and final operations are also defined and initialized. To record all done calculations and trying to avoid problems, it was decided to define the vectors' length based on the journey's available time. This is so because the TMS algorithm was implemented using time as an independent variable. This decision was based on the fact that the available trip time received as an input is known by the railway operator as being enough to travel from the departure to the destination station. Whenever the TMS runs, all variables related to train movement are updated at each considered time instant. Since this time was set to 1 s, as earlier explained, the vector dimension is based on the available total time divided by the step time size.

After finishing the initialization step, the algorithm jumps to the generation mechanism, which is responsible for generating new values for cruising and braking velocities, v_{op} and v_{bk} respectively. The generation of new values, as stated by the SA, is a random search inside the solution space. This means that, initially, a range of feasible values is defined by setting boundaries in order to create the solutions space. Once the latter is defined, a function that generates random values is used in order to search for a new set of solutions to be tested by the SA.

Inside the generation mechanism, a range of possible values was first defined, limiting the total number of feasible options for cruising and braking velocities, v_{op} and v_{bk}. The defined limits were the following:

1. **Cruising velocity:** From zero km/h up to maximum value allowed by train;

2. **Braking velocity:** From zero km/h and can goes up to cruising velocity.

More than limiting the solutions space, the generation mechanism also defines how new values must be determined. Those rules are highly important for the optimization algorithm, since they interfere in how new solutions perform and, consequently, in algorithm convergence. The process of tuning and rule adjustment was iterative, resulting from the performance of the algorithm, together with the sensitivity and perception of the problem. The result was the implementation of two ways, both based on random generation. Those ways can be regarded as rules, and the selection process of each one is dependent on the SA's previous result. Those generation rules are:

1. **Local search:** generating a new random value from the previous result neighborhood.

2. **Global search:** generating a new random value within the whole solution area.

Rule number 1 is used to introduce small disturbances to actual solution, thus generating a new one in a close neighborhood. This rule is used to verify if the actual solution can be improved in the near neighbor, before a big jump is taken in the solutions space. The purpose of this idea is to be used in the two following situations:

1. When the actual solution is accepted;

2. When a pre-defined number of solutions was tried without any of them being accepted.

Rule number 2 makes the use of a function that randomly generates numbers, following a normal distribution. The function in use generates random numbers on the range $[0, 1]$ and can be easily escalated. The result is the generation of a new value inside the problem's boundaries, corresponding to big and random jumps in the solutions space. This second rule is used in two different situations: generating the SA's initial solution and after that declining a fixed number of tried solutions. These two algorithm steps, the generation mechanism and the TMS algorithm, together form the algorithm's first loop, which is controlled by time and distance errors produced by the TMS algorithm, and it can be known as a control loop. As previously explained, the TMS algorithm simulates the train's dynamic behavior, and after defining cruising and braking velocities, a speed profile is determined and presented as a result. Once the TMS is obtained, it is possible to know how much time was needed

to go from the initial to the final velocity as well as the distance travelled. This means that the TMS algorithm somehow does not guarantee that the train travels a minimum distance in a defined time period. In fact, it only determines speed profiles from an initial to final velocity, accomplishing two intermediate velocities: cruising and braking. So, in order to reduce processing time and unnecessary operations, the following operation in the main algorithm, immediately after the TMS is the determination of time and distance errors. Errors are used to control if the algorithm must jump to the next operation, which is to determine energy needs. In fact, small errors for time and distance are allowed in order to ignore some approximations done during some numerical operations. In case of a distance or time error being higher than expected, the algorithm jumps back to the generation mechanism, with the purpose of finding another pair of velocities leading to a possible solution. When a new pair of velocities is needed, the algorithm applies the same rule used to generate the current solution, in order not to change SA algorithm's result.

Once a speed profile is determined, and according to initial time and distance requirements, the following step is the estimation of the spent train energy. Together with the estimation of energy consumption, a cost value is also determined, following a defined cost function. This step was the one that suffered more changes during the algorithm's developments, since the TMS algorithm also changed. All changes made were based on which variables should be considered as well as the change of some used penalty factors. The cost function tune was also an iterative process, requiring a lot of time spent on the running optimization algorithm and analyzing the results. Having a cost value associated to a speed profile determined by the TMS, the algorithm launches SA to analyze if actual solution brought some improvement to the current one. If it does it, the algorithm saves the current solution as the best obtained until that moment; if not, the acceptance mechanism described in the beginning of this chapter is used.

The algorithm here described runs until a maximum number of iterations is reached. After that, the solution representing the lowest cost function value is presented as the optimal solution for the problem, considering the actual conditions. After being identified as the best option, the algorithm presents the speed profile on a time vs velocity as well as a distance vs velocity plot.

The SA algorithm also required some attention. Some variables needed to be defined and some parameters had to be properly determined. First of all, the maximum number of iterations was set to 250. This number was defined based on a tradeoff between algorithm's convergence and the needed processing time. After a maximum number of iterations, all attention was directed to temperature

structure. Initial temperature as well as cooling scheme may be defined. Starting with the cooling pattern, a linear scheme was defined as being the most appropriate since the beginning. This linear scheme shows a good compromise between the algorithm's processing time and the difficulty of implementation. For this arrangement, is required to define the cooling rate factor. Bearing this in mind, and analyzing the literature review, it was defined as 0.8, thereby providing a slow cooling down. Also related with temperature structure, initial temperature should also be determined, and to reach this goal, the algorithm follows equation (4.2). Recalling the expression for the calculation of the initial temperature, it is necessary to define the maximum allowed amplitude for a bad move (cost function related) and respective probability. The maximum amplitude allowed for a bad move is not trivial to define, as it requires high knowledge about the system. A solution found to this problem was to make use of an initial estimation to determine this maximum amplitude. So, the process implemented to determine initial temperature is:

1. Defining random values for v_{op} and v_{bk} and run the algorithm's first iteration;
2. Determining the speed profile, energy needs and respective cost function;
3. Defining the first solution as the best cost achieved and generating another set of random values for v_{op} and v_{bk}.
4. Determining speed profile, energy needs and respective cost function for the second iteration;
5. Deciding the maximum amplitude allowed defined by the difference between the first and second iteration cost;
6. Determining the initial temperature, using (4.2) and attributing an acceptance probability of 0.8;
7. Continuing the algorithm, running naturally.

The results obtained with the optimization algorithm will be presented in the next section.

4.3.1.3 Results

The implemented algorithm was subjected to several tests with the purpose of verifying if is capable to meet all proposed requirements. Several results were obtained with the various versions of the TMS algorithm, but in this section, only the results found in the last version are presented. The implemented tests consisted in the simulation of a train on a railway line in the north of Portugal, used

mainly for commercial services of passengers. The algorithm results were compared with the acquired data, at the same line, in terms of speed values as well as time spent and energy consumption. In order to draw some conclusions about the algorithm's performance, based on the results' comparison with real measurements, train model parameters were defined according to the vehicle used in the trips.

Before running the algorithm, in order to look for new results, some parameters and configurations were set and maintained in all journeys. Table 4.1 presents the used SA parametrization.

Table 4.1: Algorithm configurations

Parameter	**Value**
Initial Temperature	Eq. (4.2)
s	0.8
Max iterations	250
Acceptance Probability	Maxwell-Boltzmann distribution

As already known, the most relevant algorithm configurations for the SA are: maximum number of iterations, initial temperature, cooling scheduling scheme, cooling factor and acceptance probability. With the purpose of improving the algorithm's search for new solutions, an acceptable error was defined for travelling time and distance, at the time when a solution is accepted. The introduction of error ranges in the acceptance of new solutions was made in order to help the algorithm search over the solution space. The idea is to avoid the non-acceptance of new solutions which may be classified as bad, but can be in an area of possible ones. On the other hand, the introduction of errors was made thinking of numerical approximations, which can happen during calculations. So, acceptable errors are 5 s for time and 750 m for distance, and both are constant for all journeys.

Each solution must be evaluated and posteriorly qualified in order to be defined as a good or bad one. This evaluation is made using the cost value, which is the result of applying the cost function, aiming to quantify the quality of a solution, based on some previously defined parameters. The cost function construction is directly related with algorithm's main objectives, and for the case here presented, the function presented in (4.8), is:

Minimize

$$E_c = \sum_{i=1}^{T} e_{c_i} + \mid e_t \mid \tag{4.8}$$

Looking at the cost function, the first term consists in energy consumption. Associated with each

velocity profile determined, an energy consumption estimation is done in order to be used in the cost function. In addition to the energy consumption estimation, the cost function contains a second term. This second term is related with the journey time needed, more precisely the difference between the total available time, set by scheduling, and the time needed by the train with the current solution. As explained previously, each determined solution for velocity profile is achieved within some tolerance errors. This way, only solutions inside an admissible range are accepted. The TMS algorithm itself, during speed profiles determination, has no concern with travelling time. Driving phases are determined following one sequence starting with braking, followed by acceleration, coasting and finishing with cruising. In each speed profile period, the necessary time and space are determined, and after the cruising phase is calculated, the remaining time and space for the actual journey are known. The calculation of the cruising phase considers the remaining distance as a control variable, only being completed when this distance is totally covered. The journey remaining time is not allocated to the cruising phase, only determining the time needed to travel the remaining distance at v_{cru} velocity. Once the cruising phase comes to an end, the time needed is determined and the algorithm confirms if the journey travelling time is in accordance with scheduling. Because of that, the difference between the time available for the actual journey and the time spent with the actual velocity profile is considered in cost function. After some algorithm tests, the inverse proportionality relation between travelling time and energy consumption was confirmed. So, considering a journey, the energy consumption estimation reduces with the increase of travelling time, having as result a positive time error when a journey is faster than expected but with higher energy consumption. To consider the travelling time error as a penalty factor its module is in use.

The first algorithm test considered a journey with 1710 m of distance between two consecutive stations with a travelling time of 120 s, defined on scheduling by the train operator. The profile for this first journey's, gradients and velocity limits values is presented in Fig. 4.3.

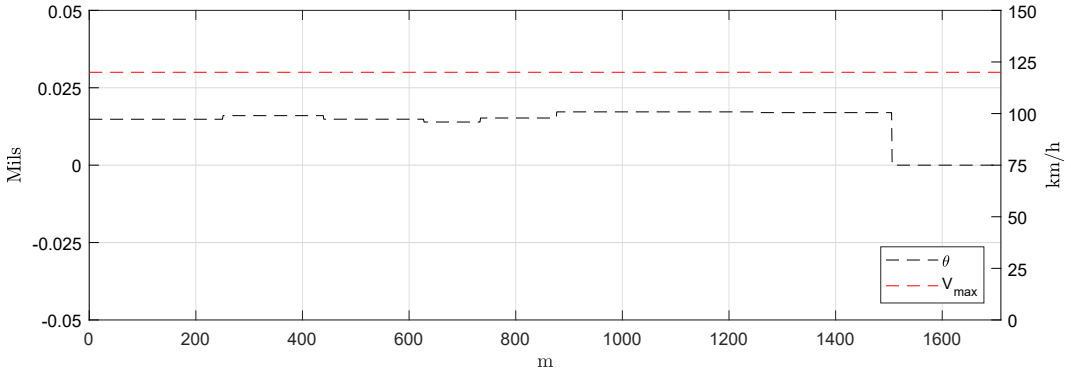
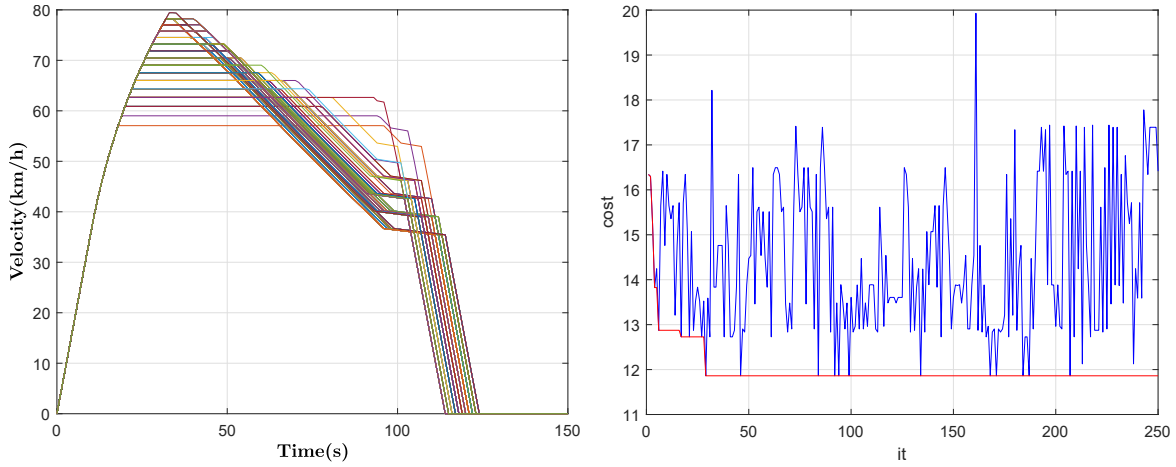


Figure 4.3: Line profile - gradients and velocity limits values.

Looking at line characteristics for the first journey, it can be seen that no restrictions are introduced in speed profile determinations. The velocity limit is defined as 120 km/h in all line extension, which is higher than the cruising velocity needed to accomplish the journey within scheduling. The line gradient is almost constant, being positive in most of the journey. This means that actual journey is situated on an uphill line. The optimization algorithm, as projected, uses an iterative process as methodology in order to find the best solution for each case which is the one presenting the minimum or maximum value for the cost function, depending on the algorithm's main objective, selected from a bunch of possible solutions. After running all algorithm iterations and before the best solution is identified, a set of possible candidate solutions that accomplish the journey's initial conditions are presented. Fig. 4.4a shows all velocity profiles determined during the algorithm's execution. Associated with each velocity profile, a cost value is determined, consisting of a measure of the solution's quality. The representation of the cost function's evolution associated to the algorithm's iterations is presented in Fig. 4.4b.

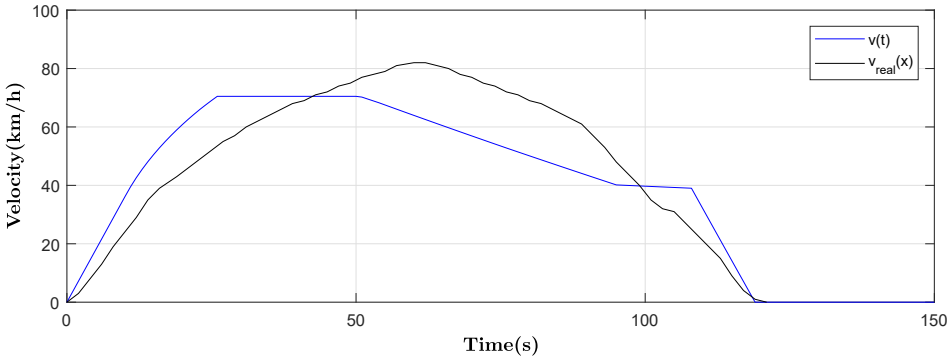


(a) (b)

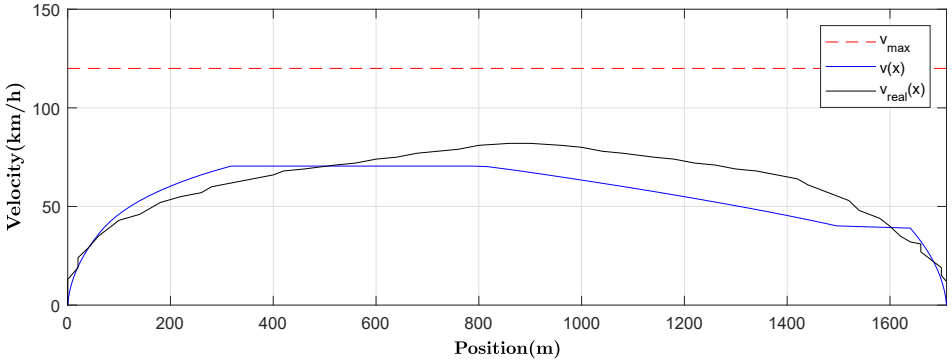
Figure 4.4: Algorithm solutions - Speed profiles and cost function values.

Figure 4.4a shows all velocity profiles calculated during the algorithm's execution. As already stated, line characteristics in these tests are almost constant during all journey extension, so they do not impose constraints on velocity profile determination. Therefore, it is expected that all determined velocity profiles follow the same sequence of driving regimes, as seen in the figure above. The big difference between the various profiles is the choice of cruising and braking speeds. Changing those velocity values implies that each driving regime needs a space and time change, resulting in different total travelling times and energy needs.

Figure 4.4b shows the evolution of cost function during algorithm's iterations. An important detail is that the minimum cost is reached during the first 100 iterations. This trend also occurred during several executions of this algorithm, using the same trip, which allows to conclude that the maximum number of iterations can be reduced, without bringing major changes to the final result. The advantage of reducing the maximum number of iterations is the decrease of the execution time, increasing the chances that the algorithm runs at real time. The algorithm's main objective is to determine the best solution, considering the actual trip as well as initial conditions. The determination of a best solution is made after several possible candidates being calculated, selecting the one with the lowest cost function value. From the algorithm's output, Fig. 4.4a, the lowest cost solution value was selected, and presented in Fig. 4.5.



(a)



(b)

Figure 4.5: Algorithm solutions – Calculated speed profiles and cost function value.

Figure 4.5 presents the algorithm output for this first journey. This figure is divided in two sub-figures, where calculated speed profiles are presented in a velocity versus time graphic, Fig. 4.5a, and a second sub-figure where train velocity is represented considering train position, Fig. 4.5b. In both figures, the algorithm result is compared with the current applied speed profile, acquired during a regular service and represented by a black line. The train was operating without any DAS, so the driver was only concerned with schedule accomplishment without violating velocity limits. In what concerns velocity limits, both velocity profiles are always below velocity limits, as can be seen in Fig. 4.5b. The acquired data shows, for this journey, a total traction energy of 14.78 kWh. To make a fair comparison, a train model in the algorithm was initialized with a train mass equal to real train measurements, together with speed velocity. The algorithm result proposes, as the best option, the speed profile represented in blue, estimating an energy consumption of 11.86 kWh. Comparing both speed profiles in terms of journey time, the algorithm results estimates a train arrival on time, while real measurements show a 1 s delay. The algorithm needed 2.1 seconds to output the solution, so it

enables its use in real-time applications, as intended. Table 4.2 compares the most important results of the algorithm with data received from measurements (first column is train weight).

Table 4.2: Results comparison for case 1

	$P(ton)$	$\Delta T(s)$	$E_c(kWh)$	$T_{exe}(s)$
Real	118	121	14.79	$--$
Algorithm	118	120	11.86	2.1

A second journey was used as another test for the algorithm. This second journey's line profile is shown in Fig. 4.6.

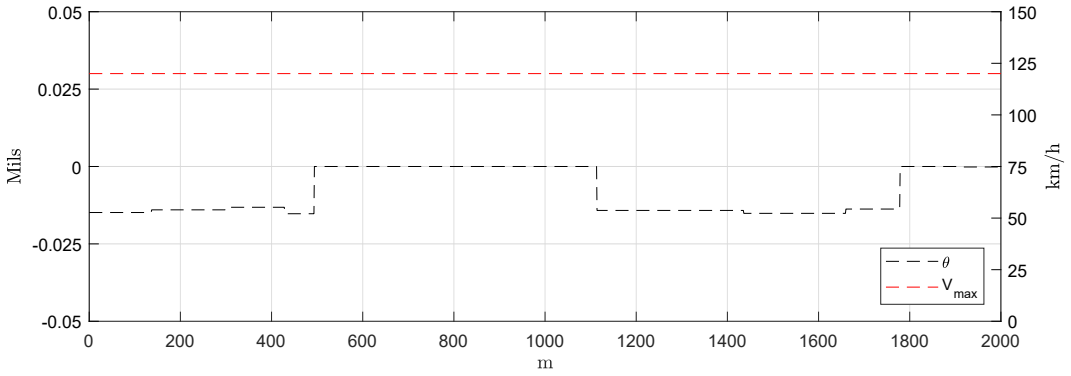
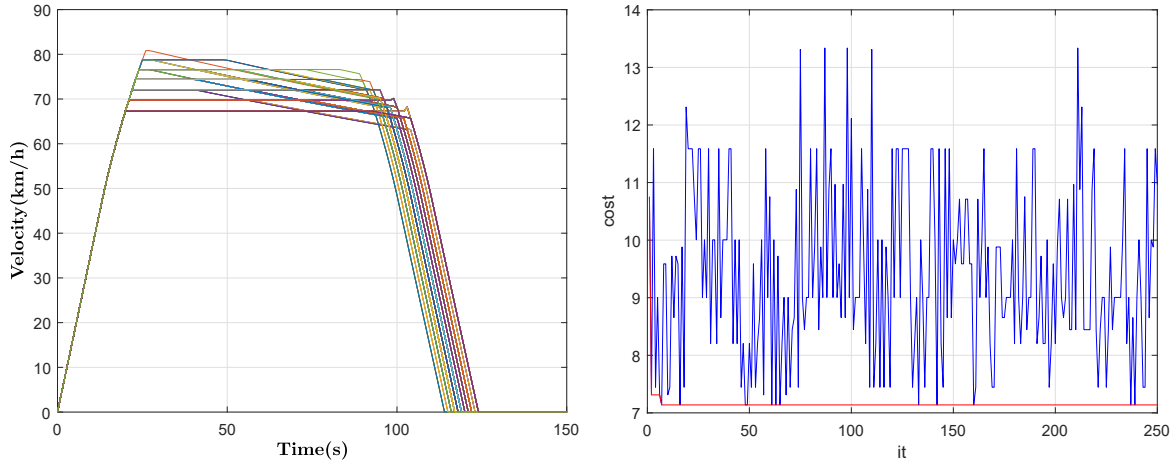


Figure 4.6: Line profile - gradients and velocity limits values.

As can be observed, this second journey has a distance to be travelled of approximately 2000 m. The train operator defined on scheduling an available time of 120 s. Throughout the journey, velocity limits are constant and equal to 120 km/h during all line extension. With velocity limits of this magnitude and without any changes, they can be seen as non-constraint to velocity profile determination. Related to line gradients, this second journey is on a line with some variations between zero and negative values. Although there are variations between downhill and flat zones, it can be said that all the length of the trip the profile is favorable to vehicle driving with minimum energy.

Once again, the algorithm was defined to run 250 iterations to look for a speed profile which can be able to minimize the energy consumption in the actual journey. After reaching the maximum number of iterations the algorithm presents a set of speed profiles as possible candidates to the best solution. Figure 4.7a presents all velocity profiles calculated for this journey. All determined speed profiles follow the same driving regimes sequence, the major differences concerning cruising and

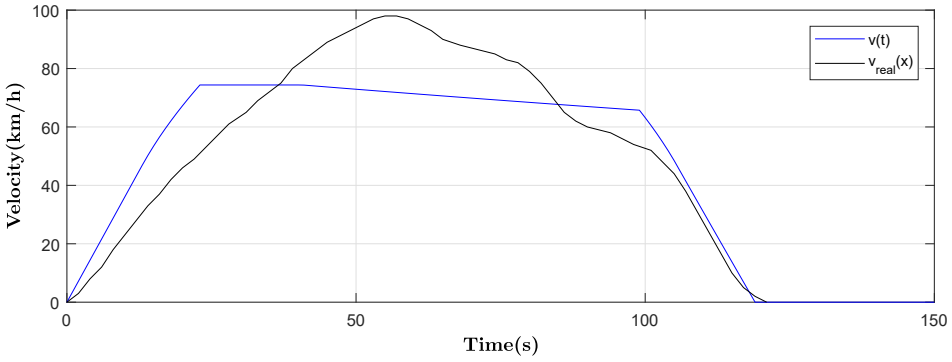
braking velocity values. The second sub-figure, Fig. 4.7b, presents cost function values during the algorithm's iterations. The lowest value for cost function, as happened in the previous journey, was determined during the first 100 iterations. Following iterations were used to explore other regions within the solution space, but without any improvements on cost function. The algorithm was tested more than once for this journey, and in all tests, cost function minimum value was always determined during the first 100 iterations. As a first conclusion, the maximum number of iterations can be reduced with the purpose of decreasing the algorithm time response without compromising its quality.



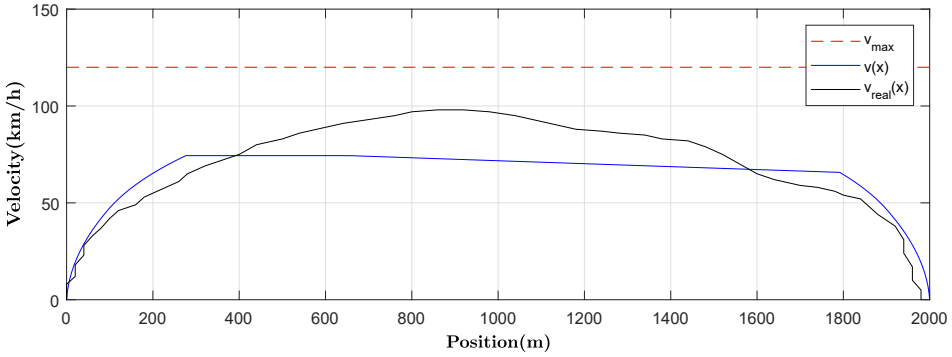
(a) (b)

Figure 4.7: Algorithm solutions - Speed profiles determined and cost function value

Considering all velocity profiles presented in Fig. 4.7, the algorithm identified the solution presented in Fig. 4.8 as the best. The best solution result is represented in a blue line.



(a)



(b)

Figure 4.8: Algorithm's solutions – Calculated speed profiles and cost function value.

Figure 4.8 presents the algorithm result, represented as the blue line, together with the current applied speed profile, corresponding to the black line. In Fig. 4.8a, both speed profiles are presented in a velocity versus time plot, while in Fig. 4.8b velocity profiles are presented in a velocity versus train position. The speed profile calculated by the algorithm advises the train driver to achieve cruising velocity in the shortest time. Reaching cruising velocity fast leads to a reduced speed to be achieved. A lower value of v_{op} enables a consumed energy decrease due to a reduction of motion resistance forces. Comparing both velocity profiles, the current measurements showed an energy consumption of 11.43 kWh, while the algorithm's output estimates a consumption of 7.14 kWh. Related with journey time, the algorithm purposes a speed profile within scheduling, while real measurements shown a 1 s delay. Since it is intended to run this algorithm on a real time application, the processing time was also registered to prove the feasibility of its implementation. The output was produced after 4.3 s. Table 4.3 presents a summary of the most important analysis criteria considered (first column is train weigh).

Table 4.3: Results comparison for case 2

	$P(ton)$	$\Delta T(s)$	$E_c(kWh)$	$T_{exe}(s)$
Real	121	121	11.43	$--$
Algorithm	121	120	07.14	4.3

The third test considers a railway line with characteristics presented in Fig. 4.9.

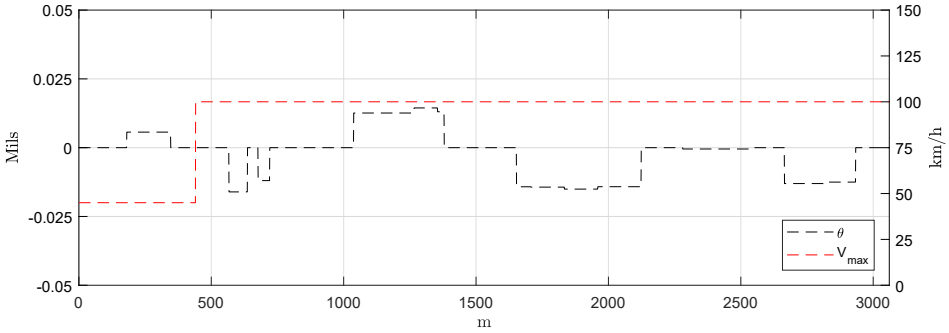
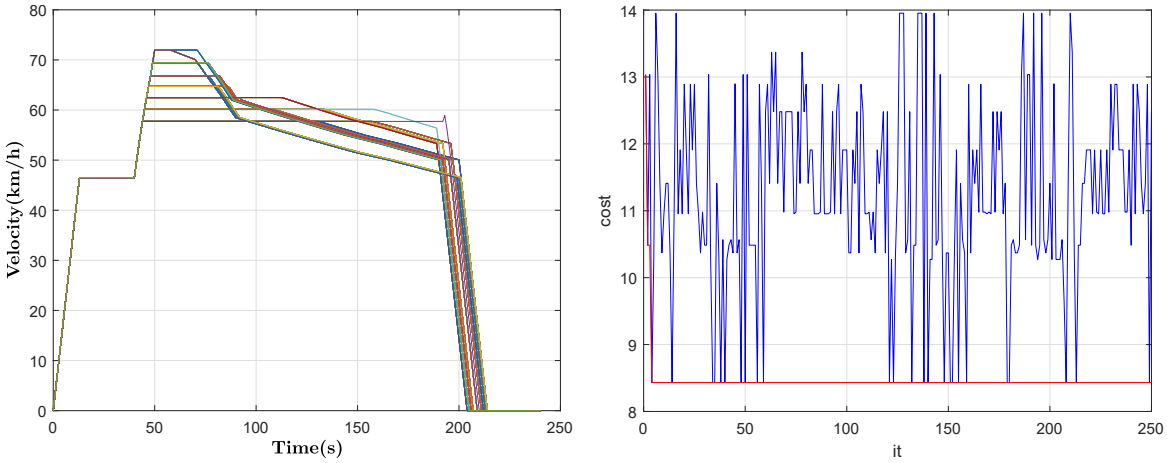


Figure 4.9: Line profile - gradients and velocity limits values.

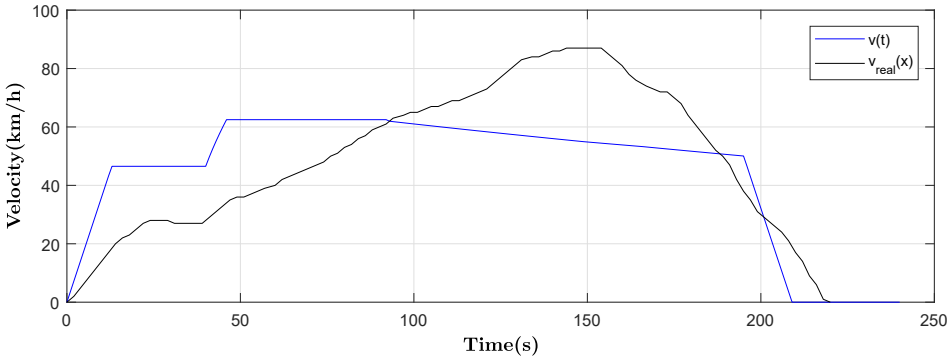
This journey has a 45 km/h velocity limit during approximately the first 500 m. The velocity limit location is coincident with the position of the departure station. With this velocity constraint at the departure station, it is expected that the train will be accelerate until the 45 km/h velocity limit is achieved. At kilometric point 500 m, velocity limit increases and is established at 100 km/h. Although this line section has a speed limit, the travelling time defined by scheduling for this journey is not tight, since most of the journey length is covered by the highest velocity limit. In fact, this velocity limit will only affect the acceleration phase, bringing some freedom to the remaining speed profile phases. Nevertheless, speed profile determination for this journey is more complex, when compared with previous cases. The algorithm must be able to identify the reduced velocity limit zone, and to determine a profile within constraints. Regarding the gradient profile, this trip is on a line section with a variable gradient. Along the trip, there are some uphill, downhill as well and flat zones. The line information transmitted to the TMS algorithm, and after defining a maximum number of iterations equal to 250, the latter determines several speed profiles within initial constraints and aligned with the algorithm's main objective. Figure 4.10 presents all speed profiles and cost function values calculated by the TMS algorithm.



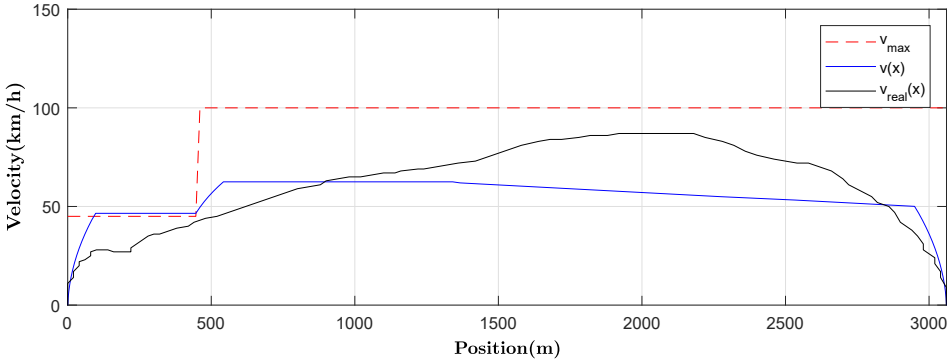
(a) (b)

Figure 4.10: Algorithm's solutions – Calculated speed profiles and cost function values.

All calculated speed profiles, as presented in Fig. 4.10a, are within the initial velocity limit. In the acceleration phase, the initial acceleration is interrupted by a cruising regime before the train reaches cruising velocity. At this point, the determined speed profile follows the same structure as the one presented for the optimal speed profile. Even though there is a speed limit at the beginning of the journey, the algorithm determines velocity profiles within the four phases, occupying the total available travelling time. This happens because the available time to travel between the two stations that are part of this third trip is more than enough to introduce a coasting phase. Figure 4.10b presents the cost function evolution, the minimum cost being determined during the initial 100 iterations. The solution with the best cost function is presented in Fig. 4.11.



(a)



(b)

Figure 4.11: Coasting phase: train velocity and position

The algorithm solution, when compared with the current applied speed profile, shows a faster acceleration at the beginning of the journey, with the purpose of reaching cruising velocity as fast as possible. The output is lower cruising velocity, resulting on less energy spent in motion resistance forces. The data acquired for this journey show a late arrival to the destination, 10 seconds after, with an energy consumption of 10.83 kWh. The speed profile determined by the algorithm estimates an arrival on-time, suggesting at the same time a sequence of driving regimes and velocities values leading to an energy consumption reduction of about 1.5 kWh, which results in a total consumption of 8.43 kWh. The algorithm calculates the solution in 10 s. Since line constraints for this journey are more complex, the increase of the processing time was already expected. Even so, the algorithm solution can be considered to have been determined in real time. Table 4.4 shows the values obtained by real measurements and algorithm outputs.

Table 4.4: Results comparison for case 3

	$P(ton)$	$\Delta T(s)$	$E_c(kWh)$	$T_{exe}(s)$
Real	117	220	10.83	$--$
Algorithm	117	210	08.43	10.0

The first three tests being completed, a new line has been introduced in the input data to perform a new test. The line profile is presented in Fig. 4.12.

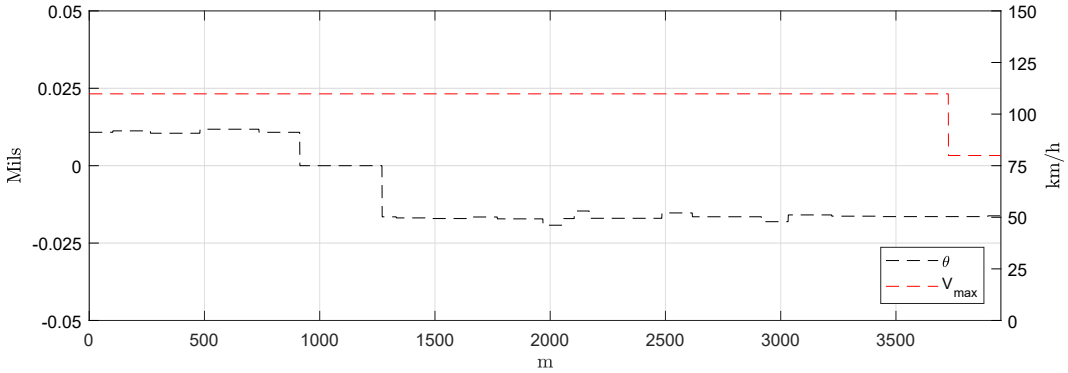
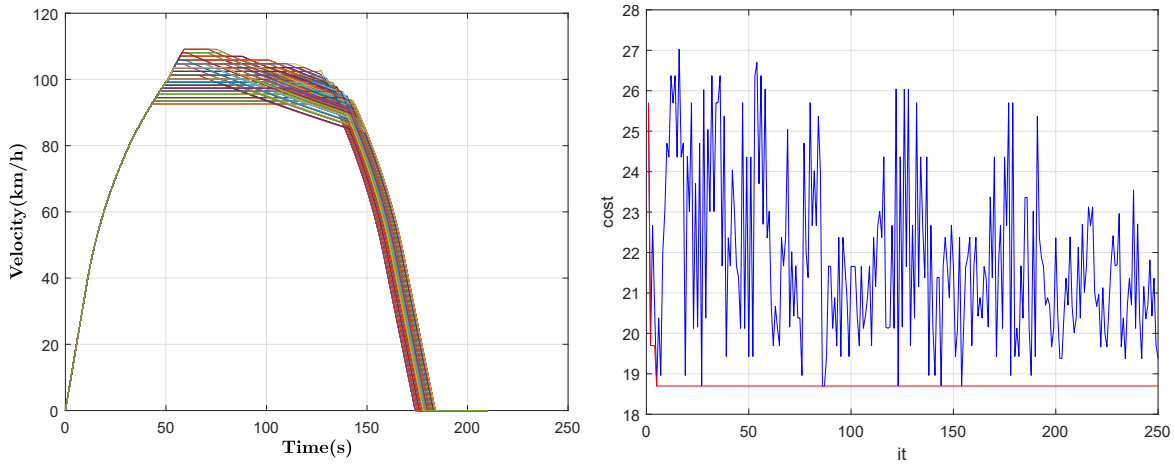


Figure 4.12: Line profile - gradients and velocity limits values.

This fourth test considers a journey with 4000 m of distance between stations, to be travelled in 180 s. This journey is located in a line section where speed limits are set at 110 km/h for the most of the journey length. In the end of the journey, velocity limits suffer a small drop, from 110 km/h to 80 km/h, near the arrival station. The limit in question will not cause major constraints in calculating speed profiles as it is close to the arrival station, where the train is likely to going through the process of braking or at least driven at a lower speed. Related with line gradients, the first kilometers occur in an uphill zone, while the last occurs downhill, as most of the journey length. Between both, a small flat zone appears. The line gradient is more or less constant inside each region, not being difficult for the train to overcome it.

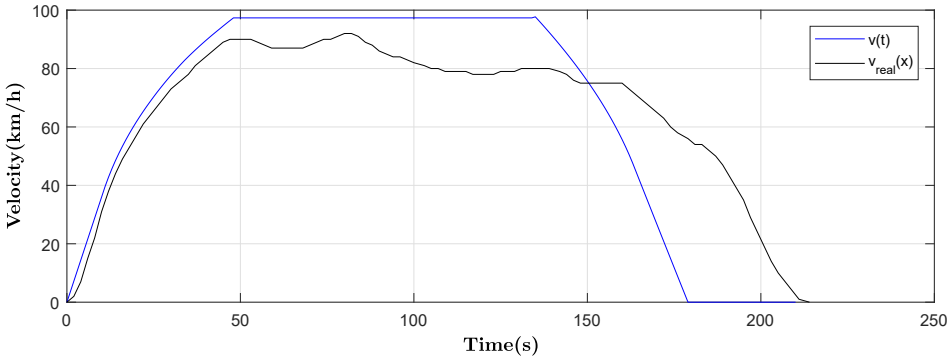
Following the initial configurations and given all information about the line and the actual journey, a set of possible solutions is determined, as presented in Fig. 4.13.



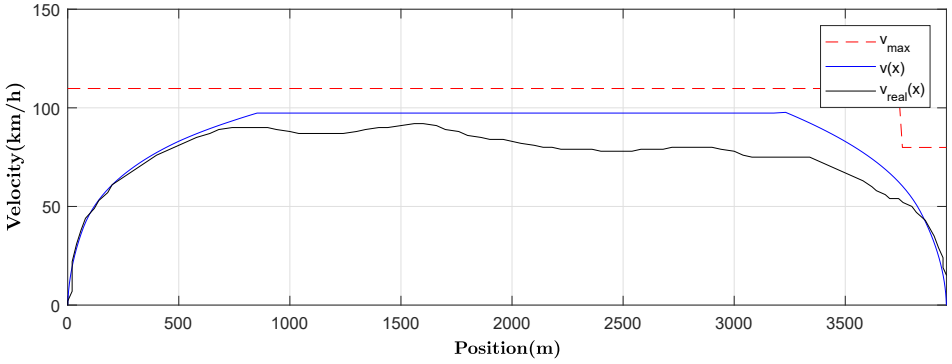
(a) (b)

Figure 4.13: Algorithm's solutions - Speed profiles determined and cost function values

This figure (4.13a) shows all calculated speed profiles after 250 iterations, beneath journey constraints. Looking at all the determined speed profiles, it can be verified that the velocity limit near the arrival station does not affect the driving regime sequence. All follows the same driving regime sequence, the major differences concerning cruising and braking velocity values. Even though it has no influence in the calculated profiles, the algorithm tries to make sure that this same limit is never exceeded. The second image presented, Fig.4.13b, shows the cost function value in each algorithm iteration. As happened in previous journeys, the algorithm found the minimum cost function value during the first 100 iterations, which is good in a way, since this maximum number of iterations can be reduced, without changing the algorithm's result, and thus allowing for the generation of solutions in a shorter time. After searching for new solutions, the algorithm selects the one with the lowest cost function value. The presented algorithm's speed profile is the one that has minimum energy consumption associated, presented in Fig. 4.14.



(a)



(b)

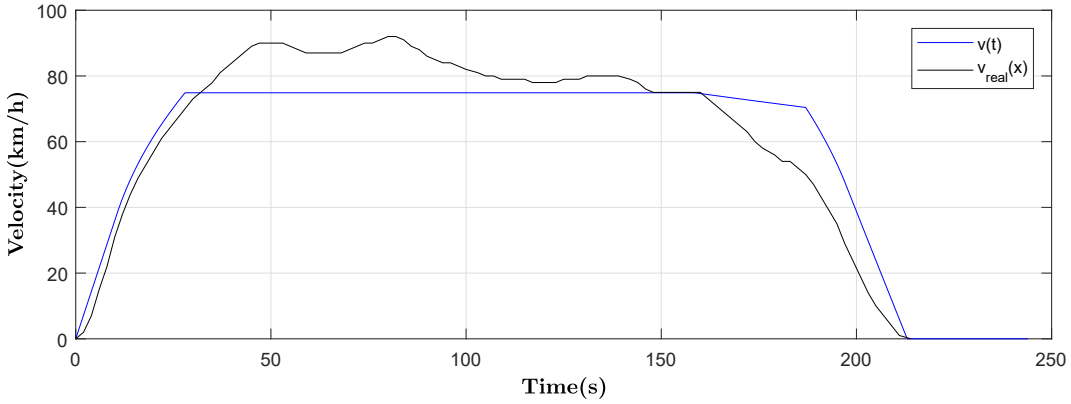
Figure 4.14: Coasting phase: train velocity and position

In figure 4.14, it can be seen the algorithm's output, together with the actual applied speed profile. The actual speed profile in use was acquired in a train without any DAS system. In Fig. 4.14a, both speed profiles are presented in a time versus velocity graphic, while in Fig. 4.14b speed profiles are represented in a train velocity versus position graphic, together with line velocity limits. Comparing the algorithm's output with the current applied speed profile, it can be seen that the algorithm output presents a profile where the journey's total time is shorter than real measurements. This happens due to the fact that the algorithm determined all speed profiles within scheduling, while real measurements were made on a journey where train arrived more than 30 s later than expected. Even though it is possible to calculate the velocity profile within the time defined by scheduling, it is clear that the time available for this journey is tight, since the algorithm did not introduce a coasting phase in the final solution.

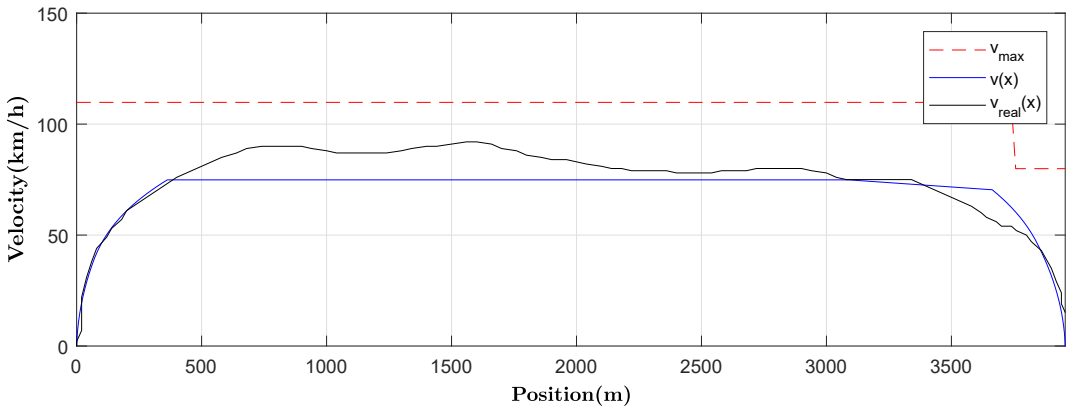
The comparison between both speed profiles in terms of energy consumption could not be considered fair. In fact, the algorithm's output has a higher energy consumption. An energy consumption

of 18.70 kWh is estimate, while real measurements showed a consumption of 15.10 kWh. Even with a higher consumption, the result of the algorithm remains the most correct, since it complies with the established time. In fact, as already known, the longer travelling time, the less energy consumption. This result was calculated in 6.1 s.

Since the travelling time difference between the algorithm's result and the train's measurements is high, about 30 s, the algorithm was again tested using this journey, considering the journey's available time equal to measurements, 214 s. The goal associated to doing this second test was to get a fairer comparison with this journey, and to reach some conclusions about algorithm's performance related with energy consumption reduction. The algorithm's results are presented in Fig. 4.15.



(a)



(b)

Figure 4.15: Coasting phase: train velocity and position

Comparing both results, the increase of the journey time allowed for the introduction of a coasting phase between cruising and braking, something that did not happen in previous results. As it can be seen, the algorithm responds again with a velocity profile capable of finishing the trip within

the time defined in the inputs. Comparing these results with the actual speed profile measured in terms of energy consumption, it can be seen that the algorithm estimates a reduction of approximately 2 kWh. The algorithm's output was calculated in 9.2 s after being initialized, which allows a real time implementation. Table 4.5 presents numerical results obtained for the fourth journey, comparing real measurements with both algorithm runs.

Table 4.5: Results comparison for case 4

	$P(ton)$	$\Delta T(s)$	$E_c(kWh)$	$T_{exe}(s)$
Real	119	214	15.10	$--$
Algorithm	119	180	18.70	6.1
Algorithm	119	214	12.97	9.2

To verify the algorithm behavior in different situations, one more journey was introduced into its tests, Fig .4.16.

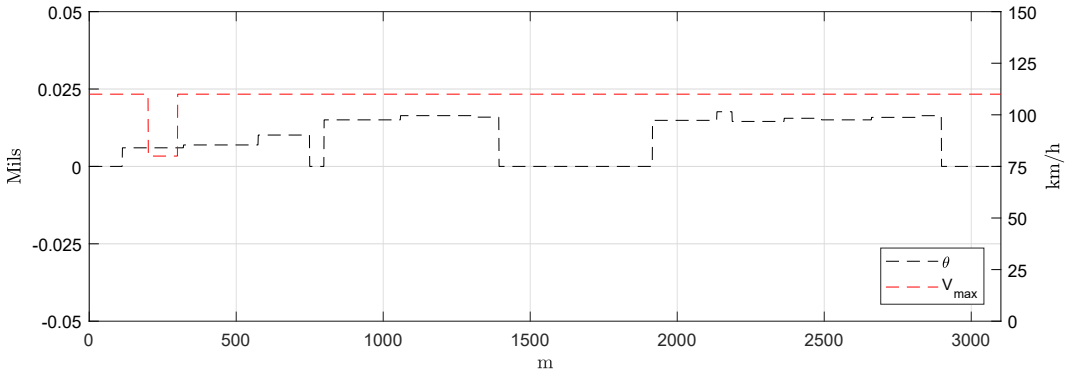
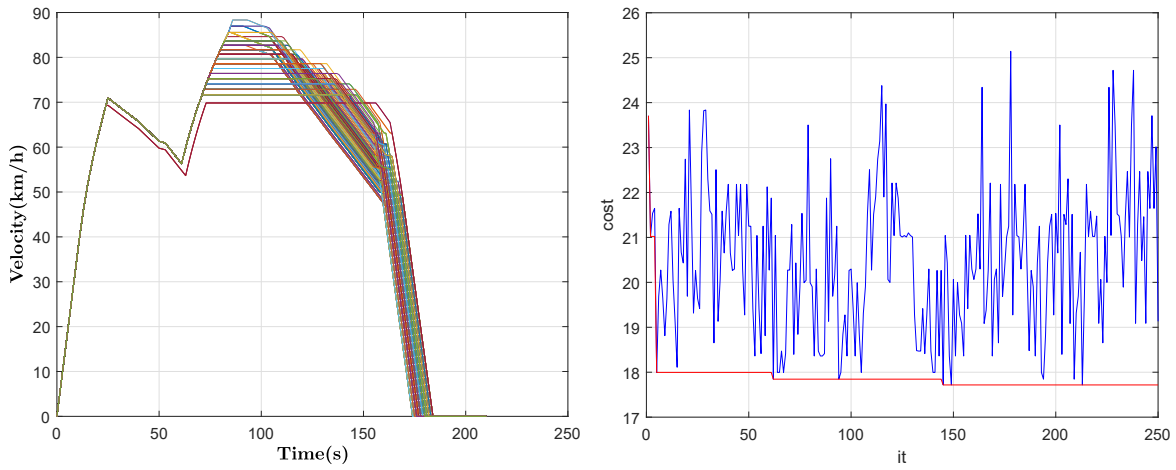


Figure 4.16: Line profile - gradients and velocity limits values.

This last journey is on a railway line with some gradient variation together with restricted velocity limit and a small neutral zone. The velocity limit in this journey appears some meters after the departure station, and it only lasts a few dozen meters. In spite of having a small extension, this speed limit zone cannot be ignored, leading to the calculation of speed profiles taking into account its location as well as the maximum allowed value. Line gradients in this journey alternate between flat zones and positive gradients, associated to uphill sections. The neutral zone appears at kilometric point 300 m. It has a length of 600 m and will be noticeable on speed profiles determination, since the algorithm will force a coasting driving regime when a dead zone is detected.

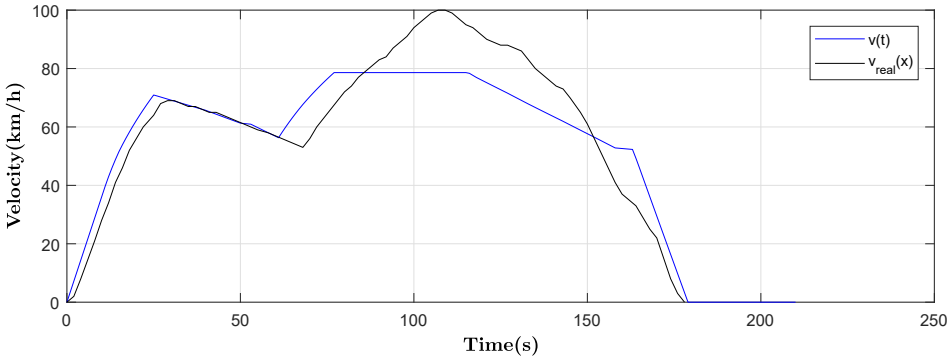
Following the same structure as presented in previous cases, the first result presents the speed profiles determined after the algorithm runs all iterations, Fig. 4.17.



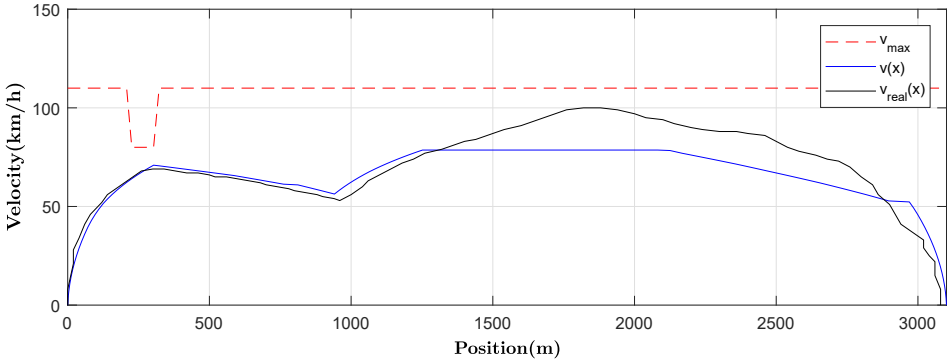
(a) (b)

Figure 4.17: Coasting phase: train velocity and position

The first figure, Fig. 4.17a, shows all calculated speed profiles. In all of them, the neutral zone is well identified since it affects all speed profile determination. After the train's departure, an acceleration phase is initiated, starting with an acceleration driving regime. As soon as the train reaches a neutral zone, the algorithm changes from acceleration to the coasting driving regime, which is changed back to acceleration when the neutral zone is over. Once this zone is crossed, the algorithm determines the remaining speed profile, following the sequence of optimal speed profile acceleration until the train reaches v_{op}, followed by cruising, coasting and in the end, near the arrival station, braking. The second figure, Fig. 4.17b, shows cost function values during the speed profile determination. In this journey, the algorithm took more than 100 iterations to find the lowest cost solution. The solution selected, the one with the lowest value for the cost function, is presented in Fig. 4.18.



(a)



(b)

Figure 4.18: Coasting phase: train velocity and position

Figure 4.18a presents the calculated speed profile, in blue, together with the train's measured profile, in black. The biggest difference between them refers to cruising velocity. The algorithm adopts the methodology of applying stronger acceleration and braking to minimize cruising speed, thereby reducing some of the energy that is wasted to overcome forces against movement, which increase with velocity. The real profile shows an early arrival and an energy consumption of 23.57 kWh while the algorithm proposes a profile which estimates an arrival on-time with a reduction in consumption of about 5 kWh. Table 4.6 summarizes the results obtained by the algorithm, comparing them with the actual data.

Table 4.6: Results comparison for case 5

	$P(ton)$	$\Delta T(s)$	$E_c(kWh)$	$T_{exe}(s)$
Real	128	178	23.57	$--$
Algorithm	128	180	17.72	3.6

Reviewing all the results presented during this section, some conclusions about the algorithm's performance can be drawn. This section presented the algorithm implemented with the purpose of determining speed profiles focused on accomplishing the scheduling imposed by the train operator as well as reducing energy consumption during normal operation. It is intended that the algorithm will be integrated on a DAS so as to advise the train driver on how he should drive the vehicle in each journey. In each departure station, the algorithm must determine a speed profile for the actual journey before the train driver starts acceleration. Looking at all the results, it can be seen a good algorithm performance within the initially proposed objectives. Algorithm results shows velocity profiles according to the time available by the operator as well as an estimation of energy consumption reduction in each trip, according to the journey's actual conditions. Another important aspect in the results is the time required to calculate each solution. Looking at the processing times obtained during the algorithm's tests, it can be concluded that time is considerably short and that it shows ability to be used into a real time application. In conclusion, it can be said that the algorithm acts according to the initially defined specifications.

4.3.2 Travelling Time

In addition to the development of the algorithm presented in previous section, a second one was also developed with the purpose of minimizing time delays on train operation. These delays, can happen naturally in non-regular operations, which can be divided in two different categories:

- **Late delays**
- **Line works**

Both cases consider unforeseen situations that can occur in every railway system. A very common problem is delayed departures, which may represent a major problem since they increase the chances of a delayed departure also delaying the arrival.

Starting with the first category, there are many factors that can cause system delays, such as entrance and exit of passengers, the vehicle or the driver not being ready to leave on time, or some any other factor that obstructs the passage on the line. The second category considers line works as a possible source of a non-regular train operation. Line works can happen any time at railway lines, caused by maintenance works or the removal of an external object that compromises the system's safety. Jobs on the line are usually carried out by workers. To ensure the latter's safety, the railway

operator defines tight speed limits for corresponding line sections. As a result, these line sections' average velocity must be reduced, thus requiring a longer time interval to travel the same distance.

Looking at both categories, it can easily be seen that they will introduce delays in every railway system. The solution to this problem is linked to train drivers, who, without compromising the passengers and the railway's system safety, seek to make the journey at the highest speed in order to reduce or even eliminate these delays. This means that any driver will not try to carry out energetically economic trips in order to reduce any delay. This implies running the train with the fastest velocity possible, to arrive as soon as possible at the next station.

Without compromising system security, it is necessary to take into account that this system arises from the necessity of transporting goods and people. Hence a great need to meet established schedules is required to maintain a high degree of satisfaction as well as confidence. On the other hand, if all train drivers meet the established schedules, it will be possible to maximize the use of railway lines, therefore also increasing the system's capacity.

4.3.2.1 Problem Formulation

Aiming at the development of a DAS capable of determining the OSP for real railway systems, it is necessary to offer options that cover all system needs. The train driver, may be able to choose between being advised about an economic driving or minimizing travelling time delays. In the development of this thesis, a second algorithm was developed in order to cover the new needs imposed by the system. The idea of this second algorithm is to be available at the DAS in order to be chosen, after the latter receives some feedback from the driver. This second algorithm is not supposed to replace the previous one. In fact, they are supposed to operate side by side.

Aligned with the objective of reducing time delays, problem formulation was revised before being implemented. Now, the problem to be solved, is how to determine an OSP that minimizes time delays or in other words, reduces the journey's total time, T, that is:

Minimize

$$T = \Delta x + \sum_{t=1}^{N} \Delta t_i \tag{4.9}$$

Constrained to

$$v_t \leqslant V_M \quad \text{for} \quad 1 \leq t \leq J \tag{4.10}$$

$$a_t \leqslant min\left[a_{max}, \frac{F_t - F_r - F_g}{M_{Total}}\right] \quad \text{for} \quad 1 \leq t \leq J \tag{4.11}$$

The algorithm will continue to be implemented in order to minimize the main subject. The energy consumption as cost function was abandoned and total travelling time took place, (4.9). Total travelling time will be determined as the result of all steps associated to time, used for the OSP determination, which will be better explained during this section. The distance error was added to total travelling time between the travelling distance and the distance between stations. This second term was added to the cost function in order to improve the algorithm's results based on how this is structured.

In terms of constraints, the problem was defined with the same restrictions, one being removed. The decision of maintaining some constraints was already expected, since the algorithm works over the same system. Anyway, all speed profiles determined must not exceed the maximum speed allowed at any point of the journey, (4.10). The second and the last restriction are used to set a limit to train acceleration, (4.11). Limits are related with train traction's characteristics and considering passengers comfort.

4.3.2.2 Algorithm Implementation

The implementation of an algorithm to determine speed profiles with the purpose of reducing time delays, requires an algorithm capable of estimating travelling time. So, algorithm developments started by integrating the TMS algorithm developed. TMS, as explained before, uses time as an independent variable on the train dynamic model, which means that for each variable present in the train dynamic model, it is necessary to determine the respective value for each time step defined. As it happened in previous developments, the time step was defined as 1 s, since train acceleration can be considered as constant during that period of time and due to this, TMS equations are still valid. To store all values, the previous DAS algorithm started by looking at the available time for the trip. Once the step time size was known, it was possible to define a fixed dimension for all vectors. A previous implementation was made considering the time available for the trip as enough to carry out the current journey. This step concerning variables definition was a drawback in adapting the algorithm to the case of non-regular operations. In some cases, more precisely, in the occurrence of works on the line, the algorithm does not have enough information to predict the delay that will be caused. To overcome this, the algorithm initializes all vectors with a pré-defined size, big enough to store all determined

values. After that, the algorithm receives one of two possible available times:

- In case of late delays, the algorithm receives the remaining time available;
- For line works, it receives the time available from scheduling.

After receiving a possible available time for actual trip, the algorithm starts to determine an OSP which intends to reduce time delays. Figure 4.19 presents the algorithm flowchart.

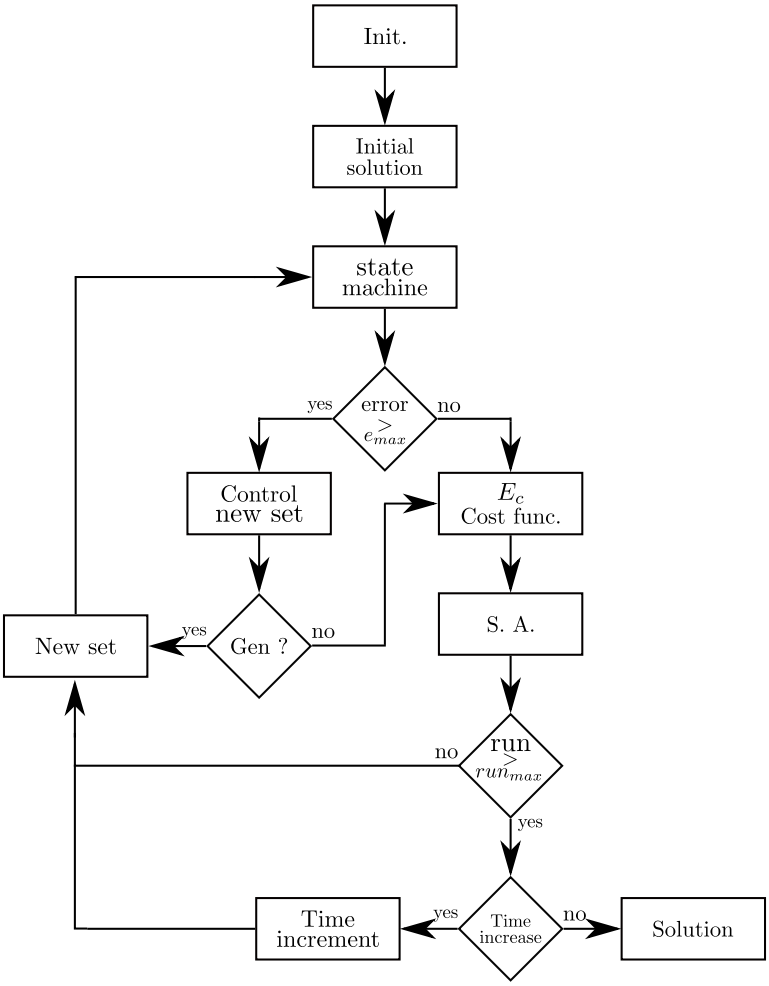


Figure 4.19: Algorithm Flowchart.

Summarizing all operations performed in the algorithm and following the sequence imposed, it can be described as:

- Generating values for cruising, v_{op}, and braking. v_{bk}, velocities;
- Reading input data: distance, time available and line constraints;

- Using TMS algorithm to determine associated velocity profile;
- Determining energy needs and respective cost function value;
- Identifying whether the solution should be maintained or excluded (SA algorithm);
- Before presenting the OSP, checking the feasibility of calculated speed profiles.

After presenting all operations, a more detailed algorithm description will be given. Looking at the algorithm's flowchart, two loops can be identified:

- **Control loop:** it contains generation mechanism, TMS algorithm and a mechanism to verify if it is possible to determine speed profiles, given the initial conditions. It is a part of the external loop. For each execution of the external loop it can be executed more than once. It appears after variable initialization and before cost function determination;
- **External loop:** this loop contains the remaining algorithm operations. Those operations are cost function determination and the SA algorithm. At the end of the execution, the external loop also verifies the feasibility of all calculated solutions.

The algorithm starts with the initialization step, used to define variables and vectors required to store all intermediate and final calculations. Besides variables to be used to store all calculations, SA parameters as well as other constants and parameters for the main algorithm are also initialized. In this algorithm step, the generation mechanism, which is responsible for generating new values for v_{op} and v_{bk}, to be used on TMS, can also be included.

Once initialization is carried out, the algorithm starts the control loop. With information about the actual trip and values for both velocities, the TMS algorithm runs and presents one velocity profile as a possible solution. The result is analyzed and it is verified if time and distance have been fulfilled with the initial requirements imposed by the algorithm's inputs. This analysis consists of time and distance's error determination, with the purpose of understanding if the train has enough time to travel between the departure and the arrival station with the calculated speed profile. The calculation of these errors allows the algorithm to move forward, since both variables manage control loop. At the end of control loop iteration, there are possible situations based on time and distance errors:

1. If both errors are within the acceptable range, control loop ends its execution;

2. If any of the errors exceeds the maximum allowed range, control loop runs one more iteration.

Situation 1 is, in fact, the most desired one. It means, only one try was necessary to determine a valid speed profile. At same time, once it happens, the algorithm automatically discovers that it is not necessary to increase the available time for the actual trip, since at least one speed profile was determined in accordance with all the algorithm's requirements and constraints.

Second situation, number 2, happens when a velocity profile within initial requirements was not determined, with values generated for cruising and braking velocities. When this happens, the algorithm jumps back to the initial point of control loop. Again, a new generation of values for both velocities is achieved, and TMS algorithm runs again. This situation is repeated until the control loop's maximum number of iterations is reached. To prevent the algorithm from being stuck when there is no solution for actual conditions, an extra control variable was defined to determine how many trials can occur. When the maximum number of attempts is reached, and if it has not yet been possible to determine a velocity profile, a dummy profile is generated associated with a high cost. The idea to associate a high cost is to guarantee that a bad solution will not be selected by optimization algorithm as a good solution.

Once control loop iterations end, the algorithm returns to the external one, where cost value is determined. The cost value is saved and the SA starts to verify if the actual solution must be saved or not. After a decision over the actual solution is made, the algorithm performs the last step within the external loop. This last step is the generation of a new set of velocities, v_{op} and v_{bk}, to be tested at the next algorithm iteration. The external loop runs until the maximum number of iterations defined before its starting.

The end of the algorithm is the OSP presentation, which starts by verifying if a feasible solution was found. As explained, considering initial conditions, if there is no solution, the algorithm determines hypothetical speed profiles. This happens when, after a maximum number of the control loop's maximum iterations is achieved without any solution. To confirm it, before presenting the algorithm's solution, it is necessary to check how many times the control loop reached the maximum number of iterations. Comparing it with the maximum number of iterations of the external loop, two situations can happen:

1. The control loop reaches the maximum number of iterations in every external loop iterations, meaning that a speed profile was not determined;

2. The control loop finds a solution, at least once, during external loop iterations. If the control loop fails at least once, the maximum number of iterations means that a speed profile was determined, regarding time and distance requirements.

In the first case, case number 1, the algorithm does not find a solution, which means there is no feasible speed profile. The reason is the fact that the time available for the trip is quite short, and for this reason the algorithm increases the time available in 5 s and restarts the iterations number. In case number 2 if in one iteration at least a speed profile is found, it means that it is a possible solution, presented as the best option.

As happened with the previous algorithm, for this second implementation the SA also required some attention on parameters' tuning.

4.3.2.3 Results

The algorithm implemented with the purpose of reducing travelling time was tested in order to analyze if its implementation was in accordance with the algorithm's main objective. A railway line in the north of Portugal, the same line used in previous tests, was used as a test line. From this line, distance between stations, gradients profile and velocity limits are used in the algorithm's tests. Contrarily to what was presented in the previous section, the results obtained with this second algorithm were not compared to real measurements. Since this section is dedicated to an algorithm implemented to reduce time travelling delays, some possible velocity limits were introduced in real line's information, simulating temporary velocity limits which commonly affect time scheduling. Looking at all data available to be used in this thesis, there is no information whether the journey was made with the purpose of reducing travelling time, or if the train was driven with a focus on reducing journey time delay. So, the analysis of these results is based on train accelerations, the resultant speed profile, travelling time and distance, being very difficult to make a direct comparison between the algorithm's results and real measurements.

Before presenting the algorithm's results, the needed initializations will be described in the first place. For this second implementation, beyond SA parameters, it is also necessary to initialize some control variables which define the algorithm's flow. Starting with SA configurations, initial tempera-

ture was defined as 100, s being the cooling factor, equal to 0.8. The acceptance probability follows Maxwell - Boltzmann distribution, again. During the algorithm's implementation, it was defined that it would run until a maximum number of iterations was reached. Looking at the algorithm's flowchart, it is clear that when a solution is not found, the algorithm reinitializes the maximum time available fort he actual trip and starts again from iteration one. Because of this, the maximum number of iterations was reduced to 50 in order to find an algorithm result in real time. In cases where no solution is found, a maximum number of 10 trials was defined near the actual solution in order to verify if determining a solution is really impossible. Bearing this in mind, in the worst case, when a solution is not found, the algorithm will run at least 50×10 iterations. Table 4.7 presents a summary of SA parameters initialization.

Table 4.7: Algorithm initializations.

Parameter	**Value**
Initial Temperature	100
s	0.8
Max iterations	50
Max iterations per round	10
Acceptance Probability	Maxwell-Boltzmann distribution

The algorithm's results, presented in this section, are organized by journey, and in each one, three different examples are presented. The main idea of this thesis section, is to present the algorithm's results when temporary velocity limits are considered on real railway lines and late departures are simulated. In all journeys, three possible scenarios were simulated as described:

1. A velocity limit was introduced in the line, without changing the journey's available time;
2. A tight velocity limit was introduced with the purpose of increasing the journey's time;
3. Train departure with delay occurred and, consequently, the time available was not be enough.

Considering those different scenarios in each journey, it is expected to have enough material to draw some conclusions about the algorithm's response.

Starting with the first journey, it is considered the same first journey previously taken into consideration. The distance between the departure and the arrival station is 1710 m and, according to the

scheduling, the time available for this journey is 120 s. Considering line conditions, the following scenarios were implemented:

1. Two velocity limits were introduced in the line: between 300 $m - 350$ m and 1200 $m - 1250$ m. Both limits were set at 50 km/h;
2. A 30 km/h velocity limit between kilometric points 900 $m - 1200$ m was added to the journey;
3. The time available for the journey was reduced from 120 s to 90 s.

The results obtained for this first journey with the new algorithm are presented in Fig. 4.20

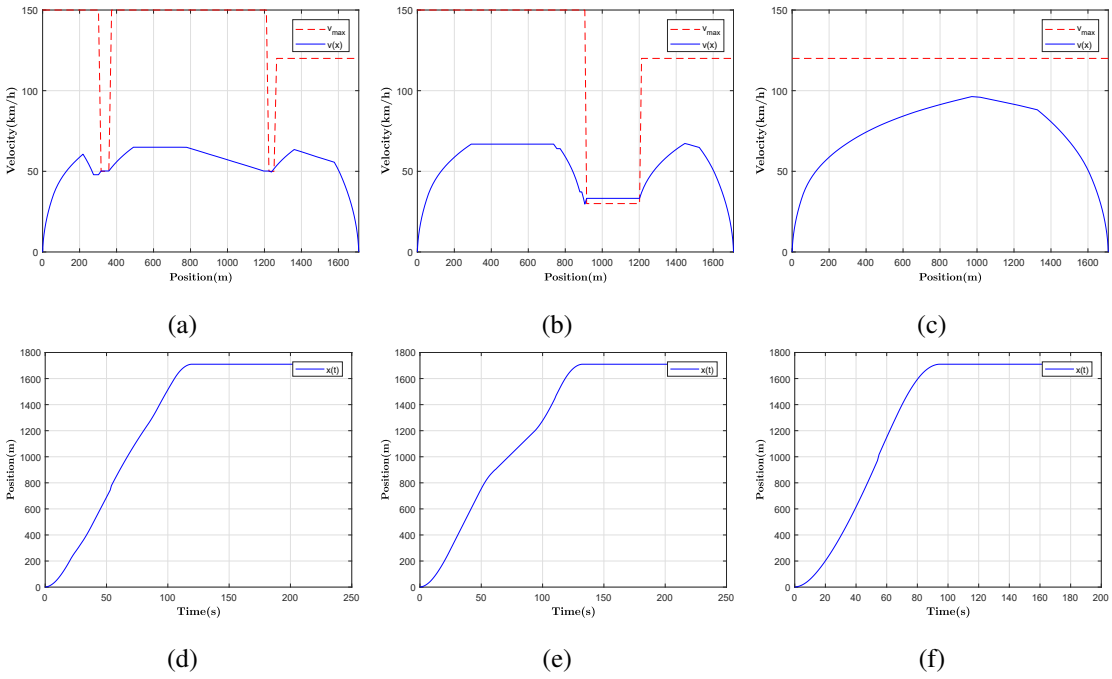


Figure 4.20: Algorithm results for all scenarios simulated in journey 1.

The results are divided in two figures. For each simulated scenario, a figure is shown, where the speed profile is represented according to the train's position; the second figure shows the train's position related to the journey's time. First of all, the idea of presenting both figures per scenario shows that the algorithm is capable of determining speed profiles following line and journey constraints and of showing that the defined time is accomplished.

The first calculated speed profile scenario is presented in Fig. 4.20a, while the train's position appears in Fig. 4.20d. With these results, it is possible to conclude that the algorithm is able to determine velocity profiles within velocity limits. This case simulates two temporary velocity limits

introduced in the line. Since there was no change concerning the journey's time, the algorithm determined a speed profile within scheduling. The second scenario, represented in Fig. 4.20b and Fig. 4.20e, simulates the introduction of a temporary velocity limit on the line, which causes an increase of the journey's time. The search for a solution starts considering the time available equal to 120 s, and after running all 100 iterations, discovering that it is impossible to find one. In face of this, the algorithm restarts the search and increases the total available time. This process is repeated again, being discovered that the journey's time increased almost 10 s. The third scenario is presented in Fig. 4.20c and Fig. 4.20f representing a late departure, the algorithm being run with the objective of determining a solution that reduces the actual delay. The algorithm was initialized with an available time of 90 s, 30 s less than scheduling, and was expected the algorithm expected to have the need to increase the available time in order to find a solution. Curiously, the time available was sufficient and the algorithm did not have to add extra time to the search. This is confirmed by analyzing Fig. 4.20f, where it can be seen the train arrival at station 90 s after initializing acceleration. Given this, and in order to reproduce the expected result, travelling time was reduced by an additional 5 s, resulting in a total of 85 s. With these conditions, the algorithm added twice the travelling time (in 5 s each), and the solution found goes in accordance with the previous one. For this second example of scenario 3, there are no figures representing the result once speed profiles were very similar. As a conclusion, for the first journey, all important numerical results are presented in Table 4.8, where time needed by the algorithm to determine a solution is presented in the last column. As can be seen, the algorithm presents processing times within initial requirements, showing real time capabilities. In the table it is presented the initial time defined as the total available for actual journey, $t_{f_{ini}}$, and the following column shows the final value for the total available time for the journey, $t_{f_{fin}}$, obtained in the end of the algorithm run. After these first two columns, two more values are presented, both determined in the algorithm. These are the error in time resulting from the profile chosen as optimal and the error in distance. The determination of time and distance errors occurs by following (4.12) and (4.13), respectively.

$$\Delta t = t_{f_{fin}} - t_F \tag{4.12}$$

$$\Delta x = x_{total} - x_F \tag{4.13}$$

Both errors can admit a positive or negative value in what concerns the total journey available

time. Considering $t_{f_{fin}}$ as the central value for all determined speed profiles, the algorithm's journey takes a time variation between -5 s up to 5 s. This happens since it is the admissible value for time errors, considered also in previous algorithm implementation. The last column on the table shows the algorithm's processing time.

Table 4.8: Results for journey 1

	$t_{f_{ini}}$	$t_{f_{fin}}$	Δt	Δx	$T_{exe}(s)$
Scenario 1	120	120	0	-1.59	5
Scenario 2	120	130	-3	-14.44	12
Scenario 3	90	90	-5	-7.92	7
Scenario 3	85	95	0	-7.17	9

In the second journey, similar scenarios were created with the purpose to test the algorithm once more. This journey consists of a travelling distance of 2000 m between stations, scheduling a time interval of 120 s being defined.

1. A low velocity limit zone was introduced between kilometric points 800 $m - 900$ m. Velocity limit during this 100 m was set as 50 km/h;

2. A temporary velocity limit of about 20 km/h was introduced between kilometric point 400 $m -$ 700 m;

3. The time available for the journey was reduced from 120 s to 80 s, simulating a late departure.

The results obtained for all three scenarios created on the second journey's line are presented in Fig. 4.21.

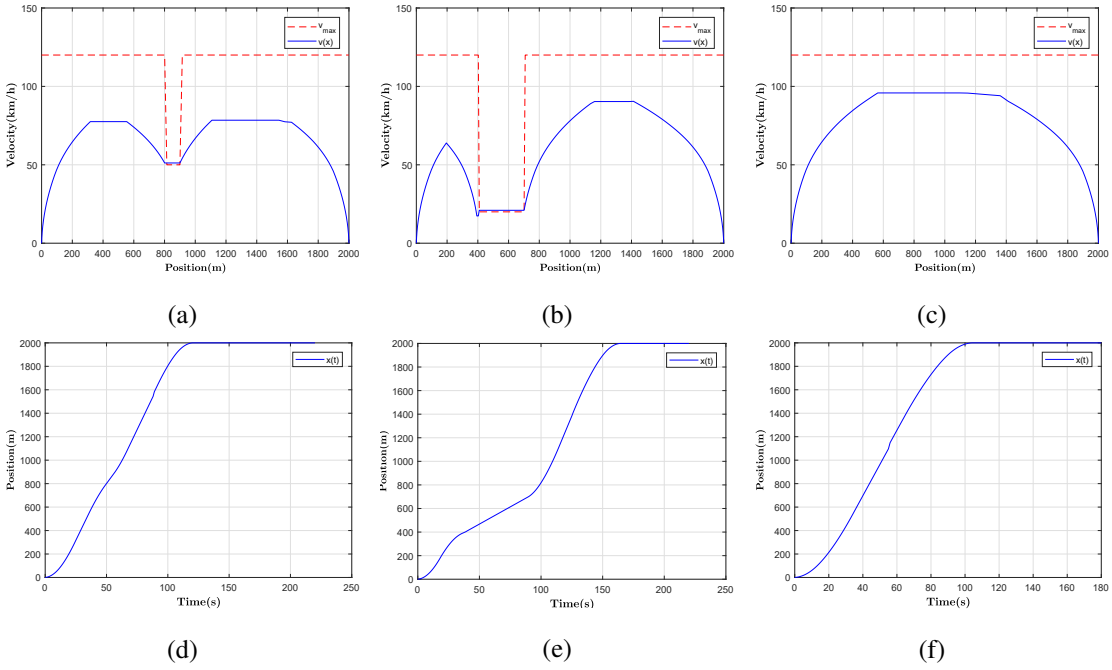


Figure 4.21: Algorithm results for all scenarios simulated on journey 2.

The first results' analysis, obtained with second journey's data shows everything as expected. The first scenario presented in Fig. 4.21a and Fig. 4.21d, which simulates a temporary velocity limit not leading to increased travelling time, shows that the algorithm follows all imposed constraints, and since there was not any change concerning the journey's time, the determined speed profile is within the available time. The second scenario, Fig. 4.21b and Fig. 4.21e, shows the introduction of a limit speed zone with a low limit value. The introduced speed limit, being particularly small, makes an increase in the average speed to travel the distance. Since, as expected, this time will not be sufficient, the algorithm should be able to look for new solutions by increasing the total travelling time. The algorithm was initialized with 120 *s*, as defined by scheduling, but this time was not enough. After being confirmed, the algorithm is internally prepared to increase travelling time in 5 *s* and reinitialize the solution determination. That said, after several attempts increases in travelling time, a solution was found near 160 *s*. Looking at the speed profile, Fig. 4.21b, it can be concluded that the algorithm follows the objective, since the velocity profile is calculated without overcoming the physical imposed limitations. From the analysis of the results, it is possible to see that the distance is fully met and that the algorithm always advises acceleration and braking within the defined limit. The last scenario presents a late delay, the algorithm being used to determine the fastest speed profile in order to try to

annul the delay. For this scenario, the time available for the actual journey was reduced from 120 s to 80 s, initializing the journey with less 40 s than usual. The algorithm started to determine a speed profile considering the remaining time, and after the first run it was confirmed that this amount of time was not enough. Increasing travelling time, the algorithm, after some runs, discovered a speed profile which can be used in 105 s, which is 25 s more than the available time and allows a delay reduction in 15 s. The result, presented in Fig. 4.21c and Fig. 4.21f, shows that all distance was covered by the speed profile, and the latter reduced the coasting phase with the purpose of decreasing travelling time. Table 4.9 shows the numerical results of the algorithm. For each scenario, time needed by the algorithm to return a solution was also accounted for. The average value is around 16 s, which allows, for its real time implementation within a DAS.

Table 4.9: Results for journey 2

	$t_{f_{ini}}$	$t_{f_{fin}}$	Δt	Δx	$T_{exe}(s)$
Scenario 1	120	120	-1	-4.33	10
Scenario 2	120	160	-5	-22.67	23
Scenario 3	80	100	-5	-0.49	16

After analyzing the results presented in this chapter, we can conclude that the algorithm meets the proposed objective. This second implementation, as explained before, is not intended to replace the previous algorithm presented, which was developed with the purpose of working side by side to offer the DAS the most different options concerning speed profiles determination. Since the previous algorithm was focused in determining speed profiles which lead to reduce energy consumption, this second implementation searches for a solution that only fulfills the trip within the time available or, if there is no solution, looks for a possible time, at least. This is quite noticeable from the results, as well as the concern with the train's physical limitations and limits accepted for passenger comfort.

Another indicator about the algorithm's response within the main objective is coasting phase times. In fact, in all speed profiles determined, constrained to tight journey times, coasting phases are shot, which leads to the conclusion that there was not enough time space to introduce one. In the beginning of this thesis, it was explained that the introduction of cruising phases is one of the tools applied to reduce the energy consumption in a journey, whenever there is room for it. Since time available was quite thigh, and the increase of coasting phases reduces energy but at the same time

increases journey time, the algorithm determined speed profiles with a shot coasting phase or even none.

Even with low processing times, during these tests, some points were exposed that may help to reduce the response time. One example of improvement is related with the initial generated value. The algorithm initializes the searching point by generating a random value for cruising velocity between 0 and maximum allowed by train traction characteristics. After running the algorithm, changing the initial cruising velocity equal to the maximum allowed train velocity, it was quite noticeable that the algorithm found the solution a little faster. This makes sense because calculating the velocity profile to meet the shortest possible time requires the use of a higher velocity to travel the same distance faster. Therefore, and initializing the algorithm with higher speeds, it would have been a little help in the first few attempts, since before randomly jumping to another zone of the solution space, iterations would walk around the initial generation. The second improvement identified is also related with new velocities values generation, used during the whole algorithm's execution. The second idea of improvement is related to the narrowing of the solutions space. When a new cruising velocity is generated, any value between 0 and the train's maximum speed value is considered, which opens a lot the number of possible combinations. After any algorithm runs, it is possible to conclude that only cruising velocities higher than the average are the ones that produce speed profiles, as expected. Based on this, the searching can be reduced, thus eliminating calculations about values that will possibly result in profiles that do not fall within specifications.

In conclusion, the algorithm here presented, with the objective of minimizing travelling time, proves to be able to produce results according to the spectra, and even though it is already in an advanced development phase, the addition of these small improvements will make it even more efficient, turning into a considerable contribution considering the optimization of the railway system operation.

CHAPTER 5

Parameters Estimation

5.1 Introduction

As already seen, OSP determination for the DAS use occurs using a TMS algorithm. The TMS algorithm, besides speed profile determination, is also used to estimate energy needs for a specific journey. Also, these operations are performed using a dynamic train model. The model is composed of a set of equations that mathematically emulate the train's dynamic behavior. Although the model is composed of a set of mathematical equations, the latter follow a certain physical meaning, as they are used to determine applied vehicle forces, and as a consequence, respective acceleration, velocity and position.

In an earlier chapter, namely Chapter 3 Section 3.1, it was presented a model to characterize train dynamic behavior. As it happens in any other model, its equations make use of particular parameters to emulate a specific behavior. Normally, these parameters have a physical meaning associated, as the case of Davis equation, (3.5). These parametric values must be in accordance with the train in use in order to find a model as accurate as possible. The TMS output accuracy is obviously dependent on train's model parameters. So, it is important to have a good parametric representation of the reality inside the algorithm.

The developed model can be used to characterize any train as well as different train topologies. To change train topology, or the particular train being used, it is only necessary to change parameters values or, in other cases, some forces representations. Looking at railway applications, a specific train representation is based on Davis equation constants, train mass and mass correction factor. Selecting a train topology to be represented in the algorithm, the needed information is sometimes available on manufacturers' official documents as white papers, technical documents and data sheets. In cases where this information is not accessible from manufacturers, or in the impossibility of obtaining it, it must be determined. The determination of this kind of information needs some field tests which

demand specific methodologies. Field tests also require a train, an available railway line, specialized technicians and some specific equipment. As a result, the development of new DAS, considering different running vehicles, is strongly conditioned by these field tests related with the acquisition of the needed parameters.

So, some trains have multiple sensors and are equipped with acquisition systems that allow data acquisition during regular operation. Dependent on the train's manufacturer, data collected includes several information from traction units, throttle position, energy meters and information about doors and auxiliary systems operational state. The data acquired is sent by a train communication system and is made available for line operator.

As an alternative to field tests, some research and developments are being carried out so as to decrease the operational complexity and costs associated with parametric determination. Those alternatives are based on parameter estimation. Having acquired data, related to train journeys, some algorithmic approach can be done allowing the determination of train parameters. These are used to run the model and obtained results are compared to the ones acquired with real measurements. So, in order to complete all the work already done, this chapter is dedicated to the development of a parameter estimation algorithm, to be effectively used in the established train model.

5.2 Methodology

With the aim of smoothing the DAS algorithm' development, as well as reducing the complexity of adapting a DAS to different train topologies, it is proposed an algorithm capable of a train model's parameters calculation. The proposed algorithm uses data collect on-board, during train operation. It is important to note that a real time solution for this train parameter calculation is not a requirement to the proposed algorithm. The use of train data requires, in the first place, data collection followed by the respective processing and analysis. Data collection is out of this thesis scope, having been provided by a train operator. The handling and analysis of the received data is still necessary since it is needed to understand and interpret all variables available in the dataset. This is a crucial step because it will condition the methodology to be adopted as well as the algorithm's results. An incorrect interpretation will conduct to bad results or even to none .

The accessed dataset contained information acquired in a group of trains, that operating in the north of Portugal and that carries out commercial passenger services. The data available was collected in two vehicles of the same model. The information stored in dataset is related to one month of

operation, and each variable was acquired with a sampling time of 2 *s*, approximately. Although both trains operate on the same railway line, saved data refers to different commercial services, which means a diverse sequence of stops as well as diverse journey times between consecutive stations. This data was acquired from the train's communications system, using all sensors and acquisition systems available from the train's manufacturer. The original dataset file has a huge amount of information, some variables, such as train total applied force, velocity, position, energy counters and journey times, being identified as the most important for the developments shown in this chapter. After selecting the variables for the train model parameter estimation, a data analysis was carried out in order to better understand the given information.

The new algorithm for parameter determination, was motivated by the already established TMS algorithm. Developments concerning the TMS algorithm were presented with detail in Chapter 3. Next, a short description of its operations is made. The developed TMS algorithm uses time as an independent variable concerning model equations, and starts the speed profiles determination with the train's gravitational force, based on the previous position. Then, resistance force is immediately calculated, the sequence of operations finishing with the determination of forces opposed to train movement. Once this task is accomplished, the next step associated to the TMS algorithm is the traction force calculation, which is dependent on the train's characteristics, the final velocity intended and the actual driving regime. Once all forces are determined, train acceleration, velocity and the resultant position are calculated. At implementation level, each train model variable is determined at each time instant. To make all calculations, TMS algorithm starts by reading the available time for the actual journey. Diving it by the sampling time, defined at 1 s, a time vector is built, as well as some other vectors with the same length so as to store the results. Figure 5.1 summarizes, in a diagram block, all operations performed in the TMS algorithm.

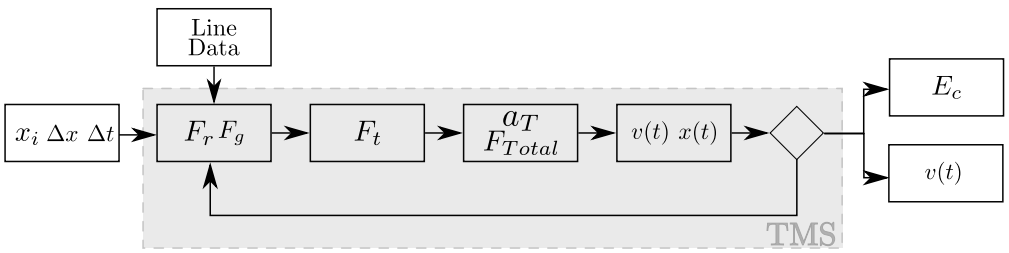


Figure 5.1: TMS basic operations.

Considering the TMS algorithm's operation as well as expressions, the proposed methodology

consists in the use of real available measurements in order to try to approximate the train model to acquired data. The chosen variables from the dataset, which are considered as the most important and to be posteriorly used in the algorithm, are:

- Traction force;
- Train velocity;
- Train position;
- Travelling time;
- Information on energy meters.

With the identification of all dataset variables that will be used, a methodology to develop the algorithm for the estimation of train model parameters started to be designed. Before starting the algorithm's implementation, it is necessary to define which train parameters will be estimated. Looking at dataset information and the TMS algorithm's operation, it was decided that the algorithm to be implemented will estimate:

- Davis equation constants: A, B and C;
- Mass correction factor, γ.

Taking the TMS algorithm as a starting point, the changes occurred are related to how traction force may be calculated. Instead of determining the train traction force dependent on the actual driving regime, this step was changed to a function that reads the input vector from the dataset. The use of the measured traction force at the train model will result in a velocity output that will be compared with the real velocity measurements. Based on this, the algorithm must be able to approximate the model to real measurements adjusting the parameters to be estimated. Figure 5.2 presents a block diagram showing the adopted methodology for the parameter estimation algorithm.

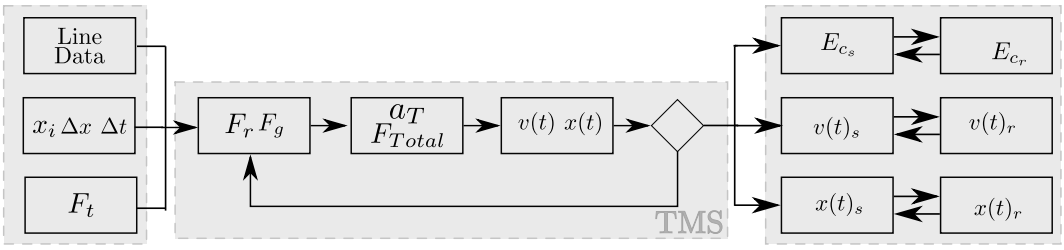


Figure 5.2: Parameters estimation methodology block diagram.

To achieve the algorithm main's objective, two different methods were studied and implemented. Both will be explained during this chapter, which also presents some conclusions drawn about this subject.

5.3 Least Square Methods Approach

The first applied method was the LSM. This is a well-known technique used to represent datasets through a function. This method can be used in applications like those presented in this thesis: datasets containing data from multiple experiments collected in the same phenomenon. One drawback of using the LSM is the need to define, a *priori*, the function to be used in order to characterize the dataset. In other words, the LSM is not used to determine the function itself but to estimate parameters that better fit some function to the dataset.

5.3.1 Theoretical demonstration

Before showing how the LSM was implemented for the determination of train model parameters, a brief theoretical review of the method will be given. The method here applied follows the one presented in [65], but a small description will be given for a better understanding about its implementation. LSM implementation expects a dataset that has good information about the system/process that is being analyzed. For this thesis, the plant and the system where data was acquired are known as well as the function that represents the system. Considering a dataset, defined by pairs (x_{r_i}, y_{r_i}) with a total of m samples, it can be characterized as a polynomial equation with degree n. So, (5.1) will be used to show how the LSM is implemented.

$$y_e(x) = p_n x^n + p_{n-1} x^{n-1} + ... + p_1 x + p_0 = \sum_{j=0}^{n} p_j x^j \tag{5.1}$$

Having access to input data and respective function, the LSM is responsible for determining the best function parameters. Best parameters are understood as the ones that better approximates estimated to real measurements. Following this definition of best parameters, a function to measure its quality is needed. Since it is expected to find parameters that better approximate the model to reality, the distance from an estimation to the respective real data can be used as that measure. This difference may be called as deviation, represented as d, and can be calculated as (5.2).

$$d_i = y_{r_i} - y_e(x_{r_i}) = y_{r_i} - \sum_{j=0}^{n} p_j x_{r_i}^j \tag{5.2}$$

The use of deviations as quality measurement may be useful but in some case, can hide a bad algorithm performance. Looking at expression (5.2), deviation result can be a positive or a negative value. This is only dependent on the position of the estimated point when compared to its real measurement. If all points are considered and by some reason they are equally spaced in magnitude, total deviation can be zero. In an ideal situation, a zero deviation should represent a well-defined dataset by the approximation point; however, in this case, it can signify a case where each individual error is nullified. A method to avoid this kind of situations is to use the quadratic value, (5.3), instead of the deviation itself.

$$e_e = \sum_{i=1}^{m} d_i^2 = \sum_{i=1}^{m} [y_{r_i} - y_e(x_{r_i})]^2 = \sum_{i=1}^{m} \left[y_{r_i} - \sum_{j=0}^{n} p_j x_{r_i}^j \right]^2 \tag{5.3}$$

Analyzing the quadratic value of deviations, it is possible to conclude that only ideal situations could produce a zero result. Another advantage of using the quadratic error is the fact that the more distant an estimation is from its corresponding measured point, the greater the penalty will be. Considering (5.3) as quality measurement, the LSM is implemented focused on minimizing this value. The minimization of deviation implies $\Delta e_e = 0$, or, considering all degrees on polynomial $y_e(x)$ from $k = 0, ..., n$:

$$\frac{\partial e_e}{\partial p_k} = 0 \tag{5.4}$$

Solving partial derivative, (5.4), in order to p_k:

$$\frac{\partial e_e}{\partial p_k} = -2 \sum_{i=1}^{m} \left[\left(y_{r_i} - \sum_{j=0}^{n} p_j x_{r_i}^j \right) \left(x_{r_i}^k \right) \right] = -2 \sum_{i=1}^{m} y_{r_i} x_{r_i}^k + 2 \sum_{j=0}^{n} p_j \sum_{i=1}^{m} x_{r_i}^{j+k} = 0 \tag{5.5}$$

Extending the partial derivatives to all p_k will result in a $n+1$ system of equations with $n+1$ unknown variables. These unknown variables are the parameters, p_j, and they can be calculated making each partial derivatives to 0 and solving the resultant system of equations. Taking the result of (5.5), it will be zero when:

$$\sum_{j=0}^{n} p_j \sum_{i=1}^{m} x_{r_i}^{j+k} = \sum_{i=1}^{m} y_{r_i} x_{r_i}^{k} \tag{5.6}$$

Expanding (5.6) results of the system of equations (5.7).

$$\begin{cases} p_0 \sum_{i=1}^{m} x_{r_i}^{0} & +p_1 \sum_{i=1}^{m} x_{r_i}^{1} & +\ldots & +p_n \sum_{i=1}^{m} x_{r_i}^{n} & = & \sum_{i=1}^{m} y_{r_i} x_{r_i}^{0} \\ p_0 \sum_{i=1}^{m} x_{r_i}^{1} & +p_1 \sum_{i=1}^{m} x_{r_i}^{2} & +\ldots & +p_n \sum_{i=1}^{m} x_{r_i}^{n+1} & = & \sum_{i=1}^{m} y_{r_i} x_{r_i}^{1} \\ & & \vdots & & \vdots & \vdots \\ p_0 \sum_{i=1}^{m} x_{r_i}^{n} & +p_1 \sum_{i=1}^{m} x_{r_i}^{n+1} & +\ldots & +p_n \sum_{i=1}^{m} x_{r_i}^{n+n} & = & \sum_{i=1}^{m} y_{r_i} x_{r_i}^{n} \end{cases} \tag{5.7}$$

To simplify method implementation, the set of equations (5.7) can be reduced to a matrix form, (5.8).

$$\left[A\right]\left[x\right] = \left[B\right] \equiv$$

$$\begin{bmatrix} \sum_{i=1}^{m} x_{r_i}^{0} x_{r_i}^{0} & \sum_{i=1}^{m} x_{r_i}^{0} x_{r_i}^{1} & \cdots & \sum_{i=1}^{m} x_{r_i}^{0} x_{r_i}^{n} \\ \sum_{i=1}^{m} x_{r_i}^{1} x_{r_i}^{0} & \sum_{i=1}^{m} x_{r_i}^{1} x_{r_i}^{1} & \cdots & \sum_{i=1}^{m} x_{r_i}^{1} x_{r_i}^{n} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} x_{r_i}^{n} x_{r_i}^{0} & \sum_{i=1}^{m} x_{r_i}^{n} x_{r_i}^{2} & \cdots & \sum_{i=1}^{m} x_{r_i}^{1} x_{r_i}^{n} \end{bmatrix} \times \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_{r_i} x_{r_i}^{0} \\ \sum_{i=1}^{m} y_{r_i} x_{r_i}^{1} \\ \vdots \\ \sum_{i=1}^{m} y_{r_i} x_{r_i}^{n} \end{bmatrix} \tag{5.8}$$

The first matrix, A, is a symmetric one, including all input sums, presented in (5.7). Matrix x, is a column matrix that contains all parameters to be determined. The last matrix contains the right part of (5.6), sums of products between system outputs and their corresponding inputs. To calculate the parameters, it is necessary to invert matrix A and multiply it by B, as shown in (5.9).

$$\left[x\right] = \left[A\right]^{-1}\left[B\right] \tag{5.9}$$

5.3.2 Method Application

Selecting a random journey, recorded in a given dataset, and plotting data relative to measured velocity on a velocity vs. time graphic, a plot as presented on Fig. 5.3 is obtained.

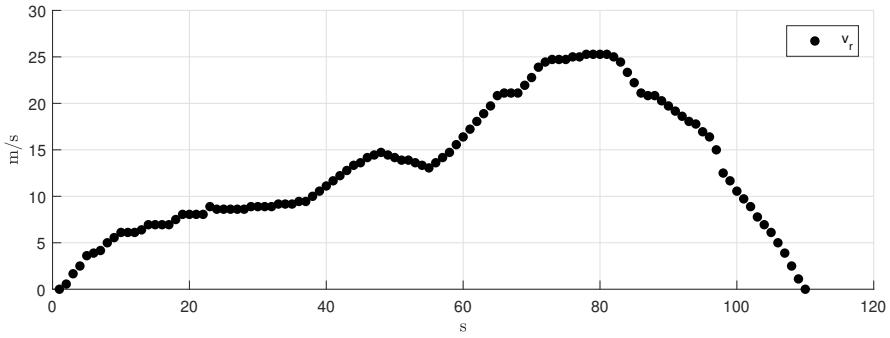


Figure 5.3: Input data - velocity vs. time.

As shown on Fig. 5.3, records on dataset consist of a group of measurements with an associated timestamp. The timestamp associated to each line of the dataset allows to trace and relate the various variables to each other. After some time spent on data analyses, a first problem was detected, related with the acquisition rate. In some journeys, without any explanation some consecutive samples appear with a sampling time higher than the expected one. This fault may be caused by failure in the communication line and it may cause some difficulties in the implementation of the LSM. Later on, it will be explained how this problem was avoided.

As mentioned before, the LSM implementation requires some knowledge about the system intended to be represented by a function. In this thesis, it is proposed to determine parameters for a train model, based in some data acquired in a real vehicle. Plotted data, as presented in Fig. 5.3, can help in function selection. In this case, and looking to Fig. 5.3, it is possible to conclude, that this set of points can be characterized by TMS equations. So, the parameter estimation considering the algorithm's measured input will use train traction force, and by applying TMS equations, a velocity profile will be estimated and posteriorly compared with corresponding measurements. Equations used in the algorithm were presented at Chapter 3 Section 3.1, but an explanation about how they were manipulated will still given, starting with Newton's second law, (5.10), which relates train movement with its mass and total force applied.

$$F_{Total} = Ma \tag{5.10}$$

Equation (5.10) can also be written as (5.11). The change of M to $M(1+\gamma)$ was presented on a previous chapter.

$$F_{Total} = M(1+\gamma)\frac{\partial v}{\partial t} \tag{5.11}$$

The total force, represented on (5.11) by F_{Total}, is the sum of all forces, for and against movement, applied to the train. Expanding total force results from the sum of traction, braking, gravitational and resistance force, (5.12).

$$F_{Total} = F_t - (F_r + F_g) = F_t - (A + Bv + Cv^2 + Msen(\theta)) \tag{5.12}$$

The replacement of F_{Total} expression, (5.12) in equation (5.11) results in (5.13), which combines all forces with train acceleration.

$$F_t = M(1+\gamma)\frac{\partial v}{\partial t} + A + Bv + Cv^2 + Msen(\theta) \tag{5.13}$$

(5.13) represents train motion in a continuous time scenario. It happens that the dataset describes train movement in a sequence of measurements periodically acquired, representing a discrete movement. So, discretizing (5.13) to adjust the function in use to input data results in equation (5.14).

$$F_{t_i} = M(1+\gamma)\frac{v_i - v_{i-1}}{T_s} + A + Bv_i + Cv_i^2 + Msen(\theta_i) \tag{5.14}$$

In equation (5.14), T_s is the time interval between two consecutive data records, better known as sampling time. M corresponds to train mass, acquired after train departure. These parameters are kept constant on the trip (between two consecutive stations), since there is no change concerning the number of passengers or loads. v_i is the velocity value recorded for instant i and v_{i-1} is the value acquired in the previous instant. θ_i is the gradient the train is subject to, at instant i. This last variable is not directly accessible, but it is determined considering data related with the train's position. (5.15) represents the function selected for LSM.

$$T_s(F_{t_i} - F_{g_i}) = M_i(1+\gamma)(v_i - v_{i-1}) + AT_s + BT_s v_i + CT_s v_i^2 \tag{5.15}$$

On the left part of this equation, can be find F_{t_i} which represents traction force, and it is a value given as input. Gravitational force, F_{g_i}, is subtracted from this traction force. It was defined this way since gravitational force can be interpreted as an input. The value for the gravitational force value

is not given nor directly accessible from the dataset, but it is previously determined and stored as an input vector. The function polynomials were divided into 1, v_i, v_i^2 and finally $M_i\,(v_i - v_{i-1})$. Table 5.1 shows how LSM polynomial and parameters were defined.

Table 5.1: LSM function organization

Parameters		**Functions**			
p_1	AT_s	g_1	1	y	$T_s\,(F_t - F_g)$
p_2	BT_s	g_2	v_i		
p_3	CT_s	g_3	v_i^2		
p_4	$(1+\gamma)$	g_4	$M_i\,(v_i - v_{i-1})$		

The application of LSM to (5.15) allows the determination of train model parameters. Davis parameters, A, B and C, are given by the first 3 results in the second column, as presented in Table 5.1, after their division by the sample time. The last column's result gives information about the mass correction factor, γ.

5.3.3 Method Implementation

Fig. 5.4 presents the algorithm structure in order to show how LSM was implemented.

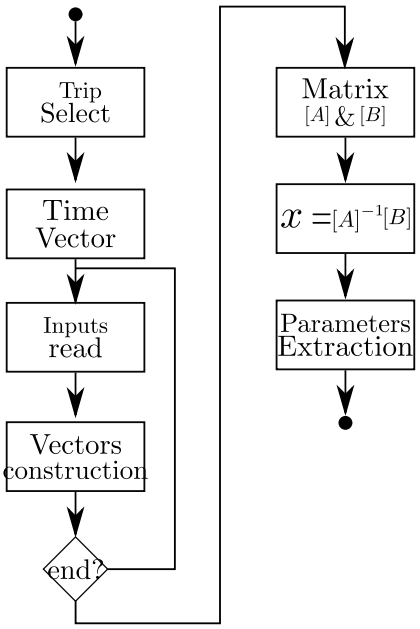


Figure 5.4: Algorithm flowchart for LSM.

Initial algorithm developments were made considering a single journey between two consecutive stations, which is enough to apply the LSM method. This approach was quickly abandoned, since the use of a single trip as input data would not be enough to extract parameters for a robust model. In fact, this would lead to a model that would representing the train in a specific condition, the trip for that gathered the input data. In other words, the use of a single journey would lead to a local optimization. Since a global optimization is expected, the algorithm's implementation was made with the possibility of considering one or more trips as input. For this reason, the first algorithm step is trip selection, where the definition of how many and which journeys are considered as inputs can be find.

The information related with each trip, before its availability to be used as input, should be, in the first place, analyzed and accordingly processed accordingly. The algorithm receives as input some vectors as input:

- Line characteristics: velocity limits and gradients;
- Total time available and corresponding distance to be travelled;
- Speed and positions logs;
- Train mass;
- Traction force applied by the train driver;
- Sampling time.

Input vectors are organized and saved as several files inside of different folders, divided by trips. This step required an extra work, since the information contained in the dataset was analyzed and divided into different files.

After defining the number of trips, the algorithm builds a time vector, which will be used as a base for all calculations. To perform this task, the algorithm reads all inputs related to time spent per trip. After reading all information about journey time, the size of the time vector is defined, being equal to the sum of the time interval of each individual trip. Certainly, beyond time of all trips, the size of the time vector is also constraint to defined time step. To reduce the algorithm's complexity, only integer values are accepted. For this reason, the minimum possible step time is 1 s, which does not present itself as a problem, since it will bring no gain to solve the motion equations at a higher frequency.

The next stage, after defining the time vector, is input reading. This way, the algorithm reads the information about each considered trip. That reading process is controlled by a statement defined by

the number of input trips considered and, in each run, train velocity, position, as well as, traction force and train mass are uploaded to be later used. The information of each individual trip is stored in an unique vector. The process to read and store input information can be described as:

1. Reading traction force, train mass, velocity and position;
2. Reading the sampling time and defining the time step;
3. Dividing the sampling time by the time step to determine how many positions must be filled;
4. Interpolating values between the initial and the final point (between the previous and the current sampling time);

After reading all inputs and properly storing all variables, the algorithm jumps to the LSM. Having all inputs available and each one of the variables on a vector, matrix A starts to be built, (5.8). All sums present on the matrix are determined and matrix A is fulfilled, element by element. Matrix B is also built, following the same order defined to matrix A. After that, the inverse of matrix A is determined so as to obtain the parameters matrix, x, (5.1).

In the end, the obtained result, by applying (5.9), is taken and Davis parameters A, B and C, as well as the mass correction factor are obtained by applying the relations presented on Table 5.2.

Table 5.2: Train model parameters

Model Parameters	
A	$\frac{x_{(1,1)}}{T_s}$
B	$\frac{x_{(2,1)}}{T_s}$
C	$\frac{x_{(2,1)}}{T_s}$
γ	$x_{(4,1)} - 1$

5.3.4 Results

Finally, the LSM algorithm was tested to verify the method performance. During this section, the algorithm's results will be presented as well as some conclusions. Before starting the algorithm's tests, data available was analyzed. As previously mentioned, data acquisition rate by communication system default is 2 s. It happens that sometimes this acquisition rate is not uniform in all data records. In some cases, perhaps due to communication failures, the acquisition rate changes. This detail can be considered as a problem concerning the LSM; to avoid that, the acquisition rate was changed to 1 s. Although an extra operation is required, this change in input data is advantageous for data estimation. The number of points per trip increases, which can be helpful for parameters estimation. Another change applied to input data was the synchronization between variables. In some trips, a desynchronization between train velocity and traction force was noticeable. To synchronize train velocity with traction force it was necessary to analyze each trip, one by one, and adjust both input vectors manually. This has demanded a huge effort as well as enormous time consumption.

The algorithm was tested in four different line sections. To increase the number of input cases, for each line section, four different trip records were considered. Figure 5.5 shows all velocity records used in algorithm tests.

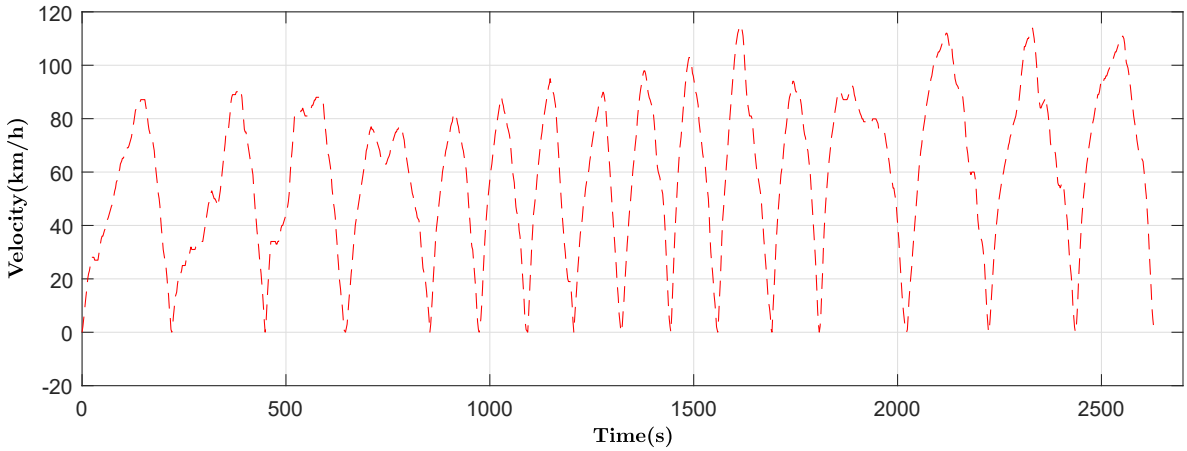


Figure 5.5: Velocity records for LSM algorithm tests

After running the algorithm, its output results can be seen on Table 5.3.

Table 5.3: LSM result

Parameters	**Result**
A	-7295.73
B	1117.30
C	-20.61
γ	-0.12

To verify the effectiveness of the estimation of the LSM's train model parameters, the TMS algorithm was configured with LSM results and all journeys have been simulated once again. The procedure was quite simple and alike the one used for parameter estimation. Basically, TMS algorithm was used to run once more, considering the measured traction force. The expected result would be to find velocity profiles most identical to the measurements performed. Figure 5.6 shows the measured speed profile in dashed black line while the determined profile is presented in blue. Figure 5.7 shows the train's position results compared with real measurements.

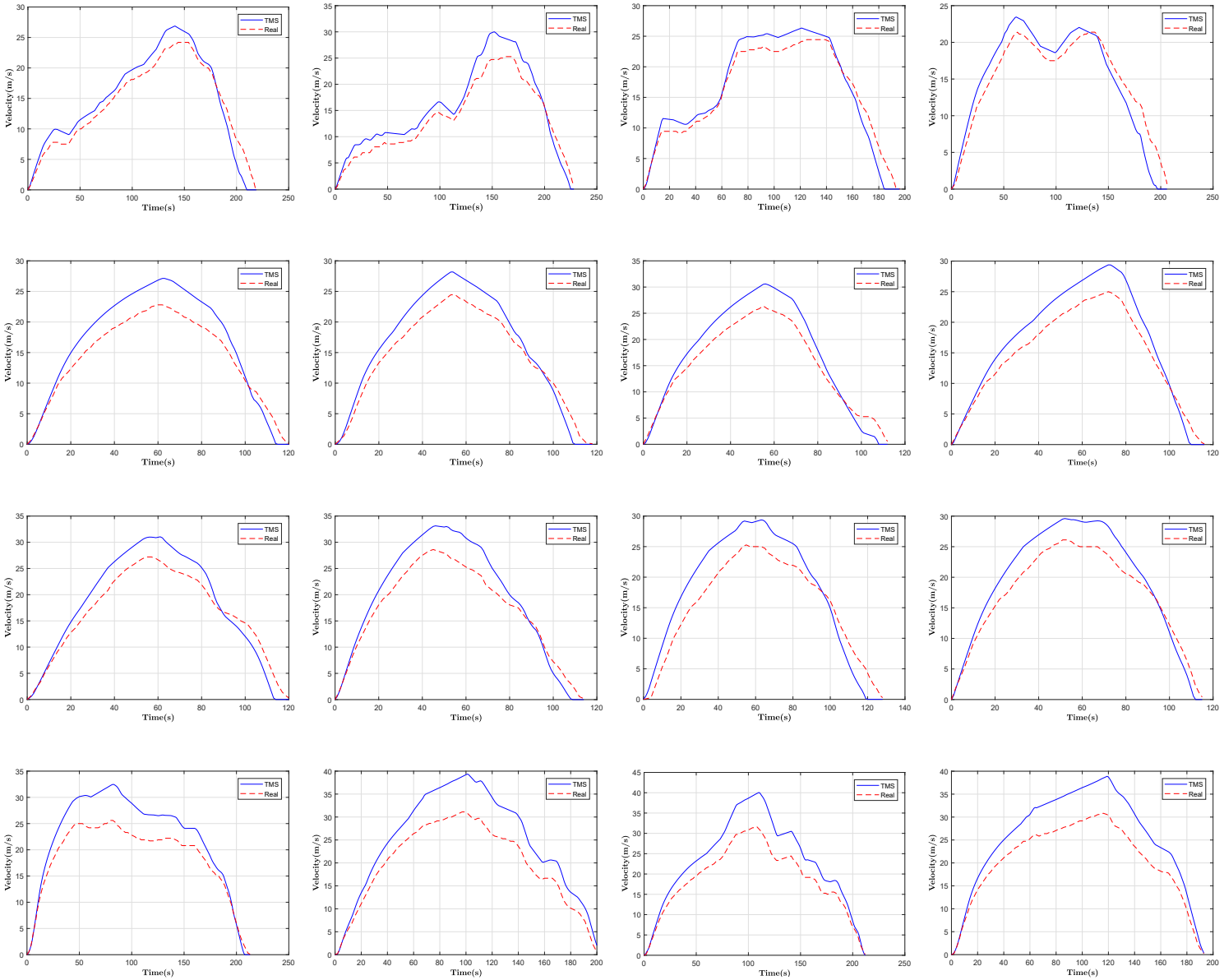


Figure 5.6: Velocity results using parameters from Table 5.3 - Measurements in red dashed line, TMS in blue line

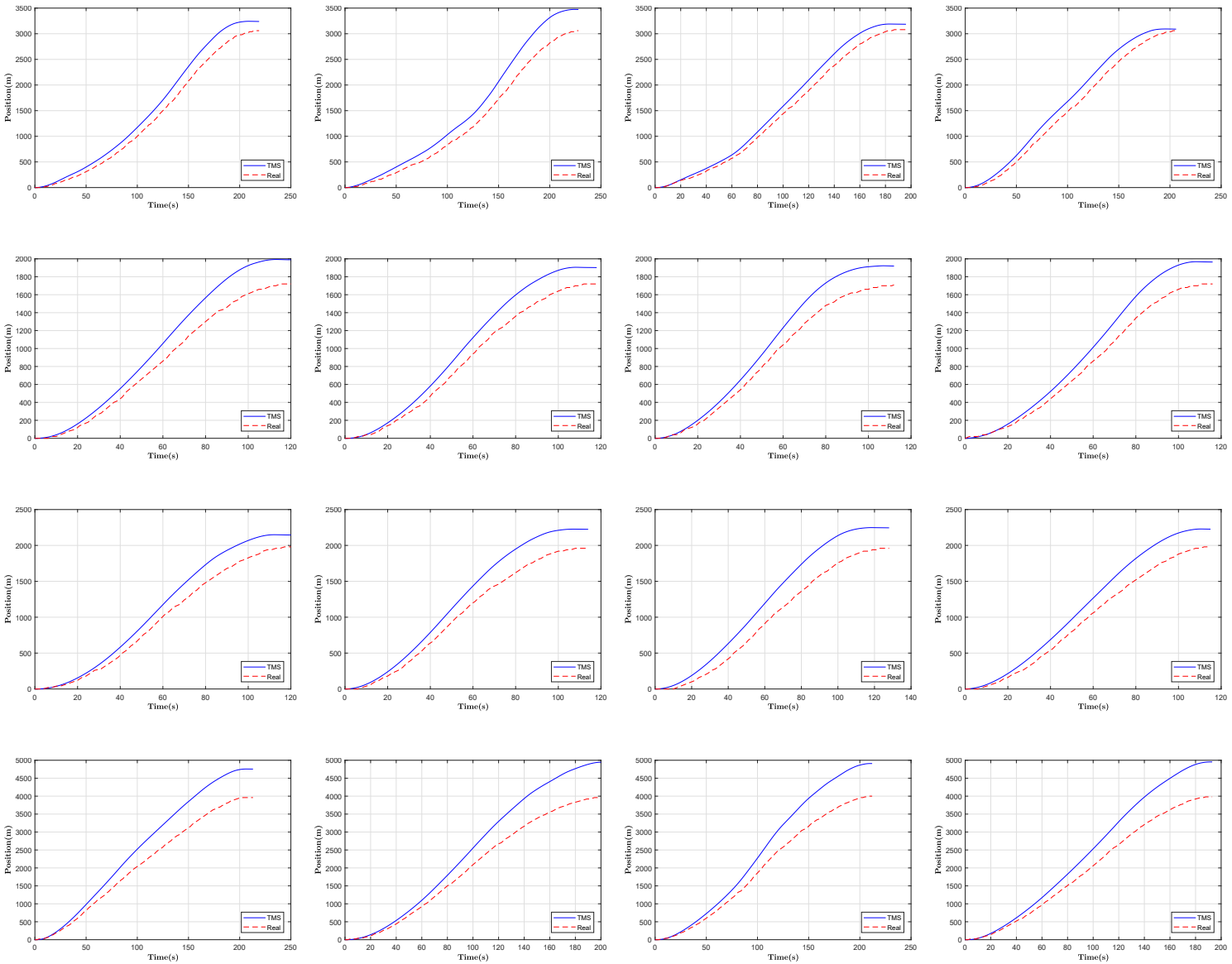


Figure 5.7: Position results using parameters from Table 5.3 - Measurements in red dashed line, TMS in blue line

This results were satisfactory but maybe another method would lead to even better ones. This new approach is described in the following section.

5.4 Simulated Annealing Approach

In order to try to improve LSM results, a second algorithm to estimate train model parameters was implemented. This second algorithm uses a heuristic optimization approach. The SA was again the selected one. The algorithm flowchart is shown in Fig. 5.8.

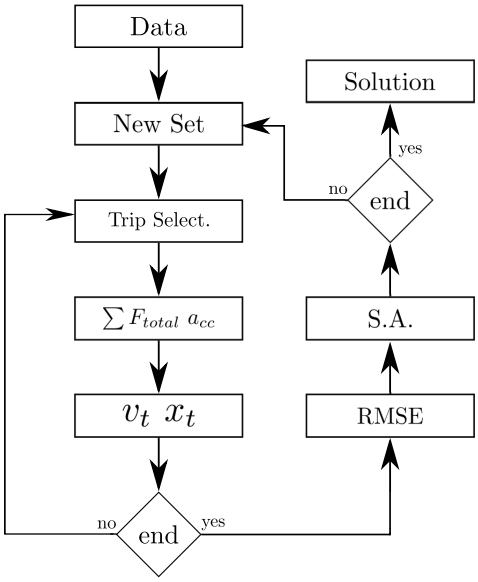


Figure 5.8: SA algorithm for parameters estimation.

Algorithm structure is based on two loops. For a better understanding, one loop was called the inner loop and the second one was denominated the external loop.

First, the inner loop, as the name suggests, runs inside the external one. This means that for each external loop run, an inner loop can run from one up to a maximum number of iterations which is defined by the number of input trips used to determine train model parameters. This internal loop is responsible for opening two main operations:

- Reading all information contained at input files;
- Using the TMS algorithm's operations to determine train dynamics.

The inner loop starts with the definition pf the time vector. As it happened with the LSM-based algorithm, the train model was implemented using time as an independent variable, so train dynamics

must be calculated at each defined time instant. In this algorithm, the definition of the time vector started by opening all input files to check for the longest journey. Once the most time-consuming trip is identified, the algorithm assigns this time as the maximum length of the time vector. After the time vector is defined, the inner loop starts to read and use all input available, and so train dynamics is determined in order to obtain a velocity profile as well as the resultant positions and energy needs. These calculations are divided by journey, and in the end, we have a group of tables presenting results concerning velocities, positions, forces and energy needs, divided by trip. The result of the inner loop is then used by the external one.

The main operations of the external loop are:

- Generating values for all parameters;
- Qualifying generated solutions;
- Running the optimization algorithm.

The SA algorithm's approach to determine train model parameters is essentially based on an algorithm that considers possible parameters values, runs train model dynamics and evaluates how far is the current result from the real solution. So, the initial stage of the external loop is the generation of a possible solution for all parameters. After defining a new generation set, the algorithm runs all inner loop iterations and generates a result to be used on the external loop. At this point, the second main operation of this loop is performed. Based on a well-defined function, the inner loop results are quantified as how good or bad, depending on the quality of the generated results. To determine this solution's quality, a cost function is used, constructed on the basis of what is intended to be analyzed. Considering the cost function value, the algorithm launches the SA algorithm to determine if the solution must be accepted. Then the algorithm launches a new set of parameters and the external loop runs once more, until reaching a maximum number of iterations, which is defined at beginning of the algorithm without following any criteria. In fact, this algorithm is to be run in an offline mode, thus there are no time restrictions, and a high number of iterations can be defined. When the maximum number of iterations is reached, the generated set with the best cost function value is presented as the best solution.

5.4.1 Generation mechanism

The generation mechanism is responsible for generating new solutions to be posteriorly tested. This is a crucial part of algorithms because, besides defining new solutions, it also controls the way how they are generated. A narrow search area may lead the algorithm to determine a solution that represents a local minimum, whereas a very large area will require a large number of iterations until the desired solution is found. The first defined point was to fix the boundaries for all variables. This restriction of variable boundary values keeps them within its physical meaning. In fact, without any boundaries, there will be no control over the searching area, which may lead to a value search that, in spite of fulfilling the searching criteria, does not have any practical meaning. Following this methodology, boundaries were defined, as presented in Table 5.4. Nevertheless, the algorithm is compatible with other value definition.

Table 5.4: Searching area for each parameter.

Variable	**Min**	**Max**
A_r	1300.00	1700.00
B_r	30.00	100.00
C_r	0.00	20.00
γ	0.05	0.10

The parameters that the SA algorithm's implementation searches for are the same as the ones estimated by the LSM. The first three are known as Davis equation parameters and the last one is the mass correction factor. Additionally, this algorithm allows for a search for a scalar factor, which can be applied to traction and braking forces. After being tested, this scalar factor was defined to be 1. Boundaries for A_r, B_r and C_r as well as γ where defined bellow the values found in literature.

The generation mechanism was built under two main rules. Both rules define how new values must be generated, rule selection being dependent on the SA algorithm's previous result. The defined rules are:

1. New values within boundaries;

2. New values in a last one generated neighborhood.

Rule 1 defines a new generation inside of the whole allowed range. The new value is generated

randomly, using a function that generates a random value between 0 and 1, which can be scalable to a corresponding value inside of the allowed range. The implementation of rule 2 forces the search at a restricted space (contained inside of maximum boundaries). This is used to perturb a sample, in order to verify if there are, in its neighborhood, new solution values that allow for the algorithm to obtain a better result. Considering searching boundaries for one variable, X_{max} and X_{min}, and defining the amplitude between them as ΔX, the perturbation step was defined as 2% of ΔX. A new value is determined as (5.16), where $rand()$ represents a function that generates a random value between 0 and 1.

$$x_{new} = x_{old} \pm 0.01X \tag{5.16}$$

In the generation mechanism, besides being necessary to define how new values are generated, it is also compulsory to know when each one of the rules is used. Throughout the parameters' search, the optimization algorithm can return one from two possible results. The returned result can be the acceptance or non-acceptance of actual solution. The following rules were created with the purpose of defining how new solutions must be generated.

- **Rule 1.:**
  - At the first algorithm's iteration, a random value is generated;
  - After ten consecutive iterations, without accepting any solution.
- **Rule 2.:**
  - Generating a new value after accepting a solution;
  - If the solution was not accepted (up to nine consecutive iterations).

In conclusion, rule 1 is used when there is no information to generate a new set of values, or to run away from a bad neighborhood. On the other hand, rule 2 keeps previous value in order to search for a better nearby solution or to test a limited number of new values before jumping to another region inside the solutions space.

5.4.2 Cost Function

The result of an optimization algorithm depends, as expected, on how a solution is evaluated. This evaluation is made through a cost function, which is based on some statements, quantified as a number: the quality of the solution.

The choice of an appropriate cost function is a task which requires some care once the algorithm's results depends on it. To select the most appropriated one, several cases were tested, being selected, at the end, the one with the best results. As a methodology to select the appropriate cost function, the algorithm's objective was defined and was analyzed which variables must be available.

As presented before, the main objective of implementing this algorithm is to have a tool to determine parameters for the train model. The results, after applying this tool, must be able to reproduce train motion as similar as it gets to a real one. Bearing this in mind, it is quite obvious that a comparison between real measurements and simulated values must be make so as to determine how alike both values are. The comparison between these values will be made by Root Mean Square Error (RMSE), calculated with (5.17). In this equation x represents one possible variable as, for instance, velocity or position.

$$e_{x_{RMS}} = \sqrt{\frac{\sum_{i=1}^{M} \left(x_{s_i} - x_{r_i}\right)^2}{M}} \tag{5.17}$$

RMSE gives the square root of mean square error. Mean square error is the average value of square errors between an estimated point, x_{s_i}, and its respective real value acquired x_{r_i}. The use of RMSE as a quality measurement enables for a higher penalization in solutions that are far from the objective than those that are closer. As a result, a faster algorithm convergence is expected.

After deciding how a solution will be evaluated, it is necessary to define how this evaluation will happen. Looking at the algorithm structure, Figure 5.8, the cost value is determined in each iteration, in the end of the inner loop. Depending on the number of considered trips, and analyzing the RMSE in each one, a vector with dimension equal to number of trips will come as a result. In order to have a single value per iteration, it was defined that the cost value will be the sum of all individual determined RMSE. Considering a total number of N trips, for each j iteration, the cost function will be determined as (5.18):

$$C_j = \sum_{n=1}^{N} e_{x_{RMS_n}} \tag{5.18}$$

Cost function tested:

- Minimizing the train's velocity:

$$\begin{aligned} C_1 &= min \sum_{n=1}^{N} e_{v_{RMS_n}} \\ &= min \sum_{n=1}^{N} \left[\sqrt{\frac{\sum_{i=1}^{M} \left(v_{s_{i,n}} - v_{r_{i,n}} \right)^2}{M}} \right] \end{aligned} \tag{5.19}$$

- Minimizing the sum of the train velocity's with position:

$$\begin{aligned} C_2 &= min \left[\sum_{n=1}^{N} e_{v_{RMS_n}} + \sum_{n=1}^{N} e_{s_{RMS_n}} \right] \\ &= min \sum_{n=1}^{N} \left[\sqrt{\frac{\sum_{i=1}^{M} \left(v_{s_{i,n}} - v_{r_{i,n}} \right)^2}{M}} + \sqrt{\frac{\sum_{i=1}^{M} \left(s_{s_{i,n}} - s_{r_{i,n}} \right)^2}{M}} \right] \end{aligned} \tag{5.20}$$

- Minimizing the sum of the train's velocity, position and energy consumption:

$$\begin{aligned} C_3 &= min \left[\sum_{n=1}^{N} e_{v_{RMS_n}} + \sum_{n=1}^{N} e_{s_{RMS_n}} + \sum_{n=1}^{N} e_{E_{c_n}} \right] \\ &= min \sum_{n=1}^{N} \left[\sqrt{\frac{\sum_{i=1}^{M} \left(v_{s_{i,n}} - v_{r_{i,n}} \right)^2}{M}} + \sqrt{\frac{\sum_{i=1}^{M} \left(s_{s_{i,n}} - s_{r_{i,n}} \right)^2}{M}} + \left(e_{E_{cs_n}} - e_{E_{cr_n}} \right)^2 \right] \end{aligned} \tag{5.21}$$

5.4.3 Results

This algorithm was tested in order to validate its performance in solving this type of problems. More than verifying the algorithm's convergence, it is also analyzed the quality of the TMS algorithm with the calculated parameters. The input data was the same as the one used before. Four journeys were used, and for each of them, 4 different records, taken in different days, were considered.

To run the SA algorithm, it is necessary to define some initial configurations. The maximum number of iterations is one of the parameters to be defined, and was chose 1000. The selection of a high number of iterations occurred since a real time solution is not a requirement. To assure the algorithm convergence, setting a high number of iterations increases the chances of more cases being tested, and by this, the probability of finding a global minimum is higher. Another definition for the SA algorithm is related to the way the algorithm decides if it must save or not a bad solution. This mechanism is known as acceptance probability and in this implementation Boltzmann distribution

was selected. The last parameter to be defined before running the SA was the cooling factor. As stated before, the cooling factor helps the algorithm search for a good solution. So, this parameter must be well defined. In this algorithm implementation, the cooling factor was set as 0.85, which corresponds to a slow temperature decreasing. Table 5.5 resumes all configurations defined, and posteriorly used.

Table 5.5: SA initial configurations.

Variable	**Min**
Max iteration	1000
Trips	16
Cooling factor	0.85
Acceptance Probability	$p = e^{(\frac{-1\times(\Delta E)}{T_k}}$

Cost functions presented on Section 5.4.2 were all tested to see which one got the best results. Following the order presented in the previous chapter, the first cost function tested was (5.19). The algorithm must start with an initial solution for all parameters to be searched. Instead of starting with a random initial guess, a set of initial points has been defined to start the algorithm search. Table 5.6 shows initial guesses for all parameters.

Table 5.6: Initial points for the SA algorithm using cost function (5.19)

Variable	**Initial Point**
$A_{r_{init}}$	2000.00
$B_{r_{init}}$	100.00
$C_{r_{init}}$	20.00
γ_{init}	0.10

After running the maximum number of iterations, the algorithm returned the following results, Table 5.7.

Table 5.7: SA results for cost function (5.19)

Variable	**Min**
A_{r_f}	1060.08
B_{r_f}	60.69
C_{r_f}	5.12
γ_f	0.05

Using the parameters given for the TMS algorithm, and considering the measured traction train force as input, the algorithm returns velocity profiles that can be compared to real velocity measurements. Fig. 5.9 shows TMS output and corresponding velocity, measured for all sixteen cases. The train's position, presented in Fig. 5.10, was also analyzed and compared with real measurements.

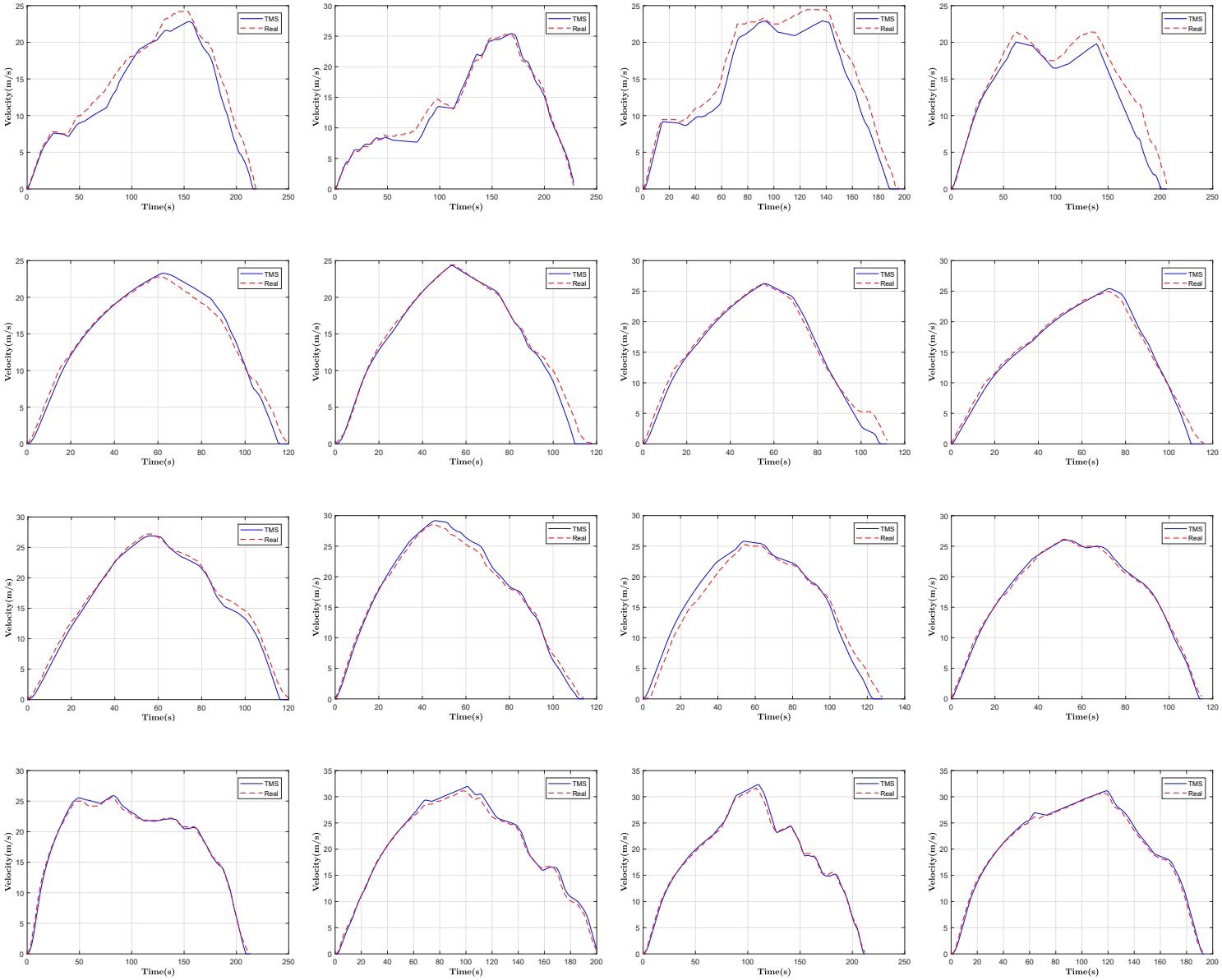


Figure 5.9: Velocity results using parameters from Table 5.7 - Measurements in red dashed line, TMS in blue line

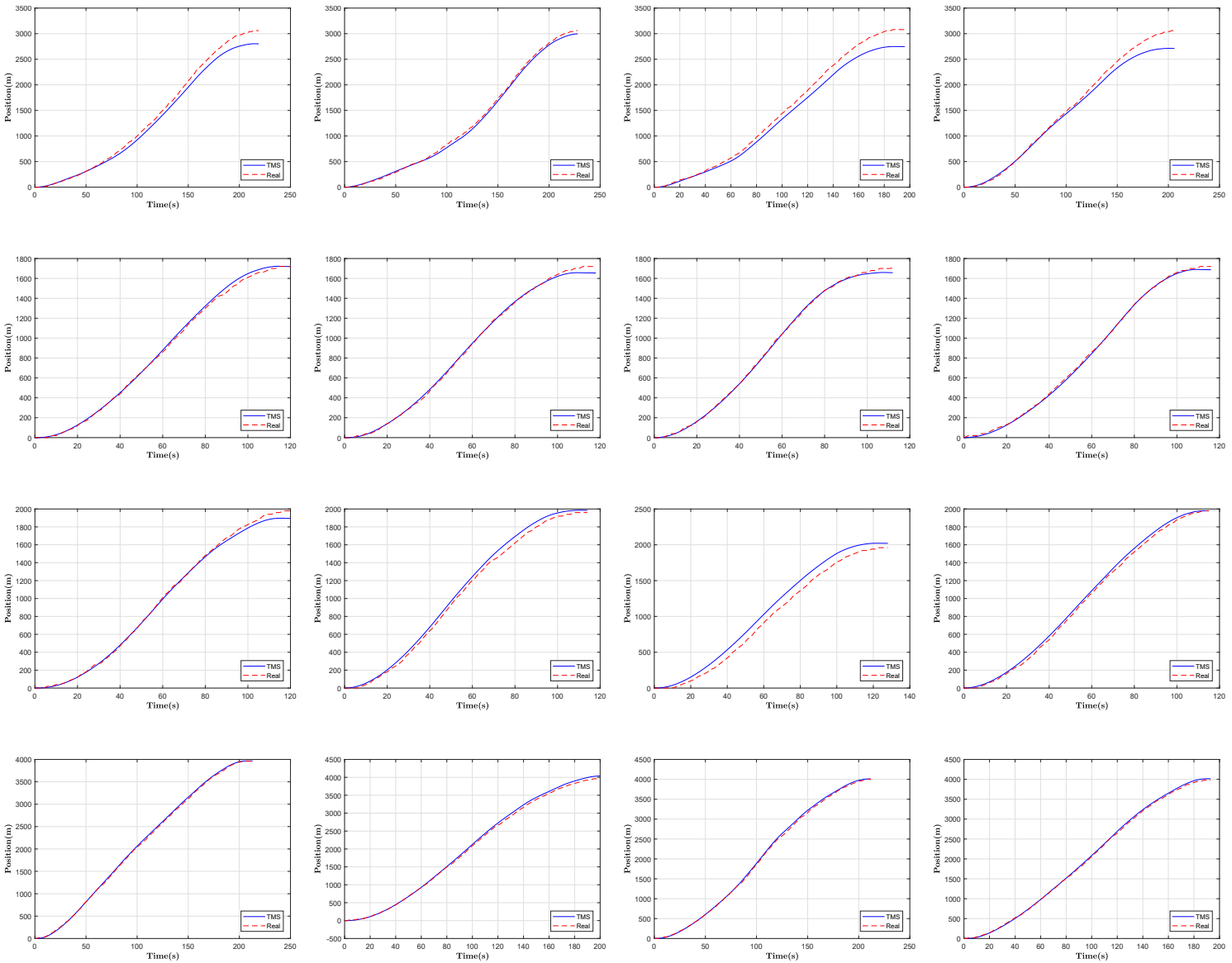


Figure 5.10: Position results using parameters from Table 5.7 - Measurements in red dashed line, TMS in blue line

The result presented last is the evolution of the cost function value over algorithm iterations, Fig .5.11. As it can be seen, the algorithm started with a bad initial solution, but it has evolved in order to improve the final result. It is also visible that some generated parameters produce worst solutions that allow to avoid the local minimum. The algorithm reached the minimum cost during the first half of iterations, and this behavior was common in other runs. Even so, the maximum number of iterations will be kept.

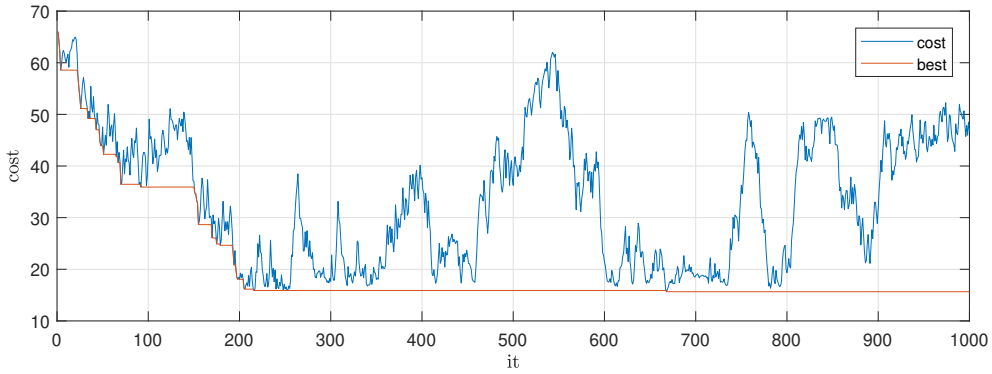


Figure 5.11: SA cost function (5.19).

The algorithm tests continued by using a second cost function, (5.20). This cost function considers a sum of velocity and position errors. The performance of the algorithm using this second cost function option was analyzed. The search for parameter values uses, as a starting point, the same values presented in Table 5.6. After the algorithm reaches its maximum number of iterations, the result with the lowest cost function value is the one presented in Table 5.8.

Table 5.8: SA results using cost function (5.20)

Variable	**Min**
A_{r_f}	1104.96
B_{r_f}	22.65
C_{r_f}	5.38
γ_f	0.07

Using the given parameters, Table 5.8, in the TMS algorithm, we reach the speed profiles shown in Fig. 5.12. in an overall analysis, it seems that this is a good approximation. Beyond speed, train position is presented on Fig. 5.13, and it is compared with real measurements.

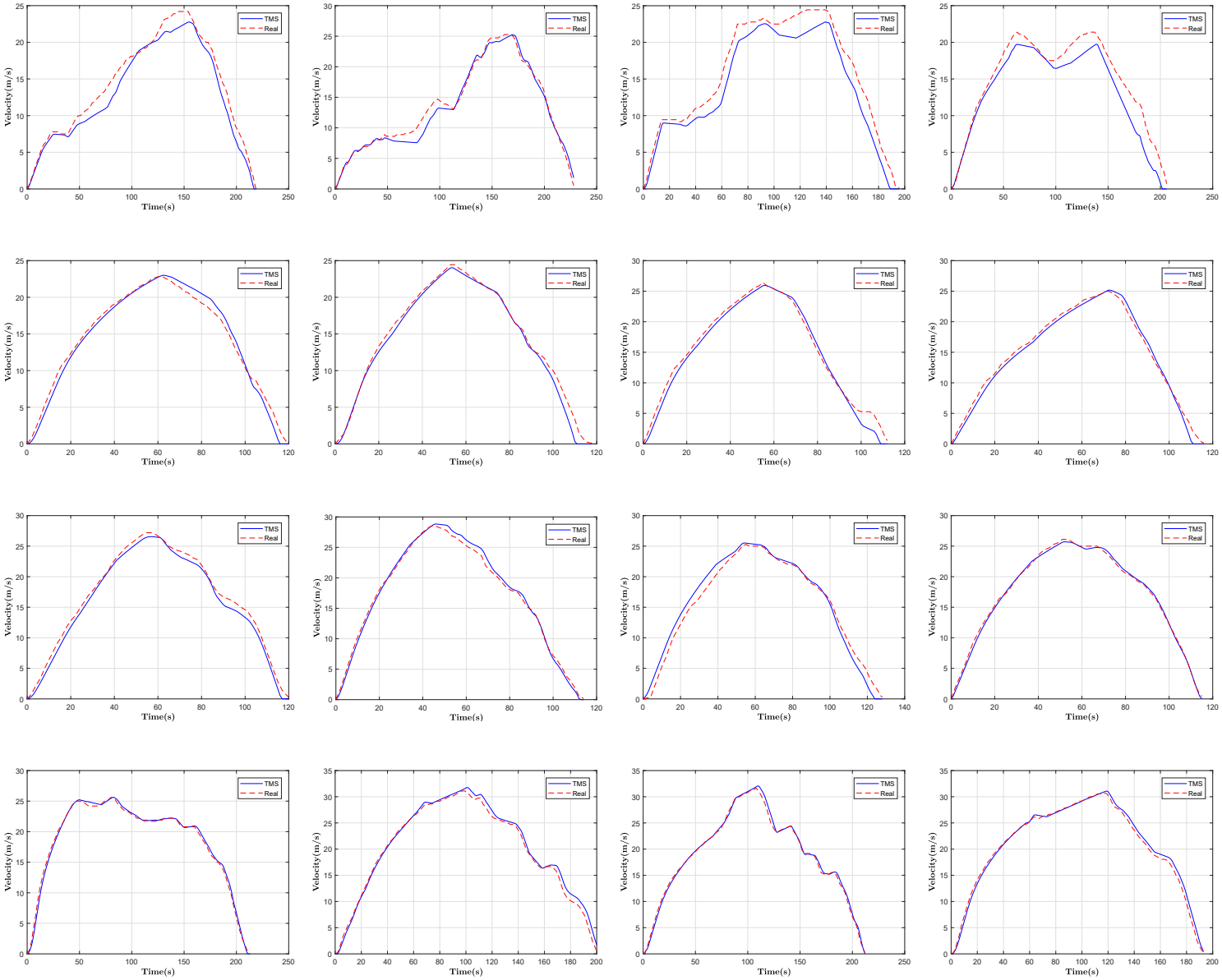


Figure 5.12: Velocity results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line

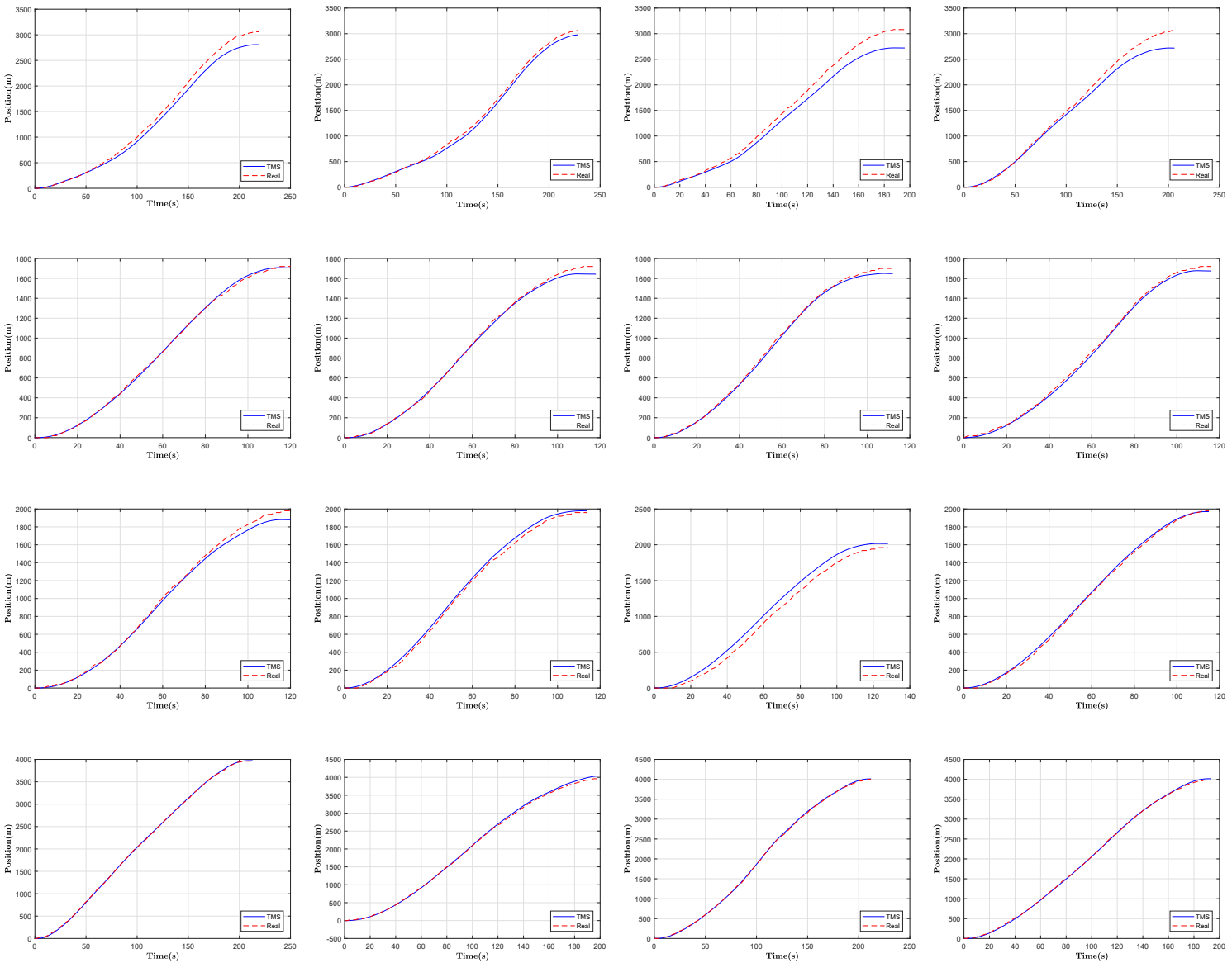


Figure 5.13: Position results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line

To understand the algorithm convergence, the cost function evolution must be analyze. Fig .5.14 shows the cost function value at each iteration, and even though there are some jumps to areas with worse solutions, the algorithm converged to a minimum cost, which was obtained after a few runs of the algorithm.

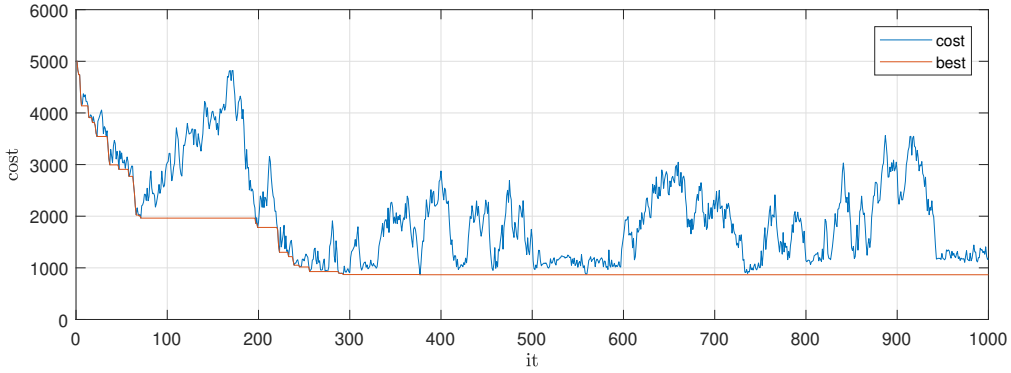


Figure 5.14: SA cost function (5.20).

The last cost function tested was (5.21). This cost function considers velocity and position errors as well as energy consumption. The inclusion of energy consumption in cost function is intends to help with the parameters' determination adjustment. Since energy consumption is influenced by opposing movement forces, a closer model to reality is expected.

Starting once more with the same initial points, Table 5.6, the algorithm returned, in the end, the parameters presented in Table 5.9

Table 5.9: SA results using cost function (5.21)

Variable	**Min**
A_{r_f}	1169.50
B_{r_f}	19.46
C_{r_f}	5.71
γ_f	0.07

Comparing this last result with the previous one, it seems that the inclusion of energy consumption in cost function did not bring major changes to the parameter search. When both solutions are compared, it is clear that the results are close.

Using this result, the TMS algorithm was used to see the how it performs in speed profiles deter-

mination. The outputs of the TMS algorithm were compared to real measurements, as presented in Fig. 5.15. The train's position was also determined and compared with the train's data, Fig. 5.16.

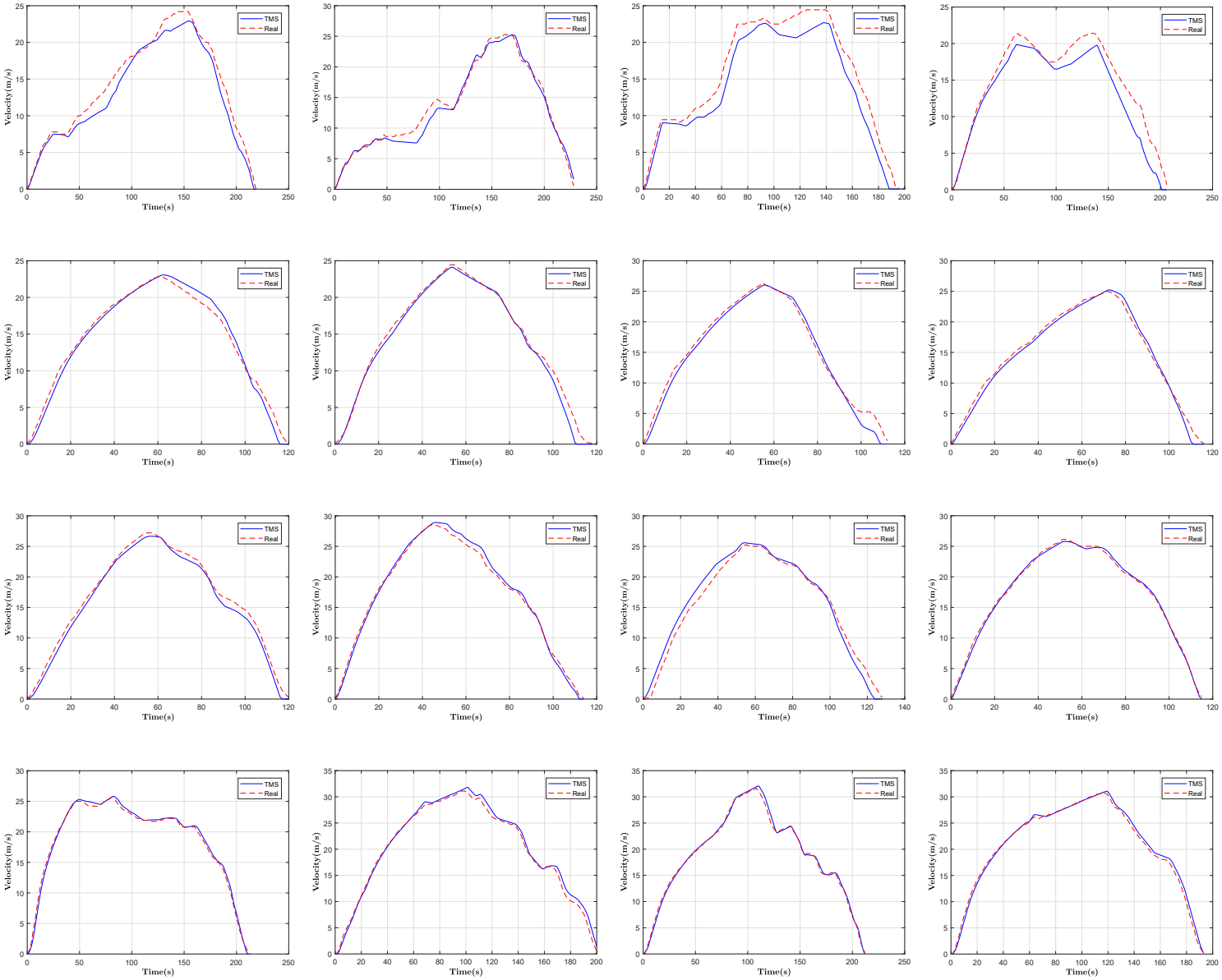


Figure 5.15: Velocity results using parameters from Table 5.8 - Measurements in red dashed line, TMS in blue line

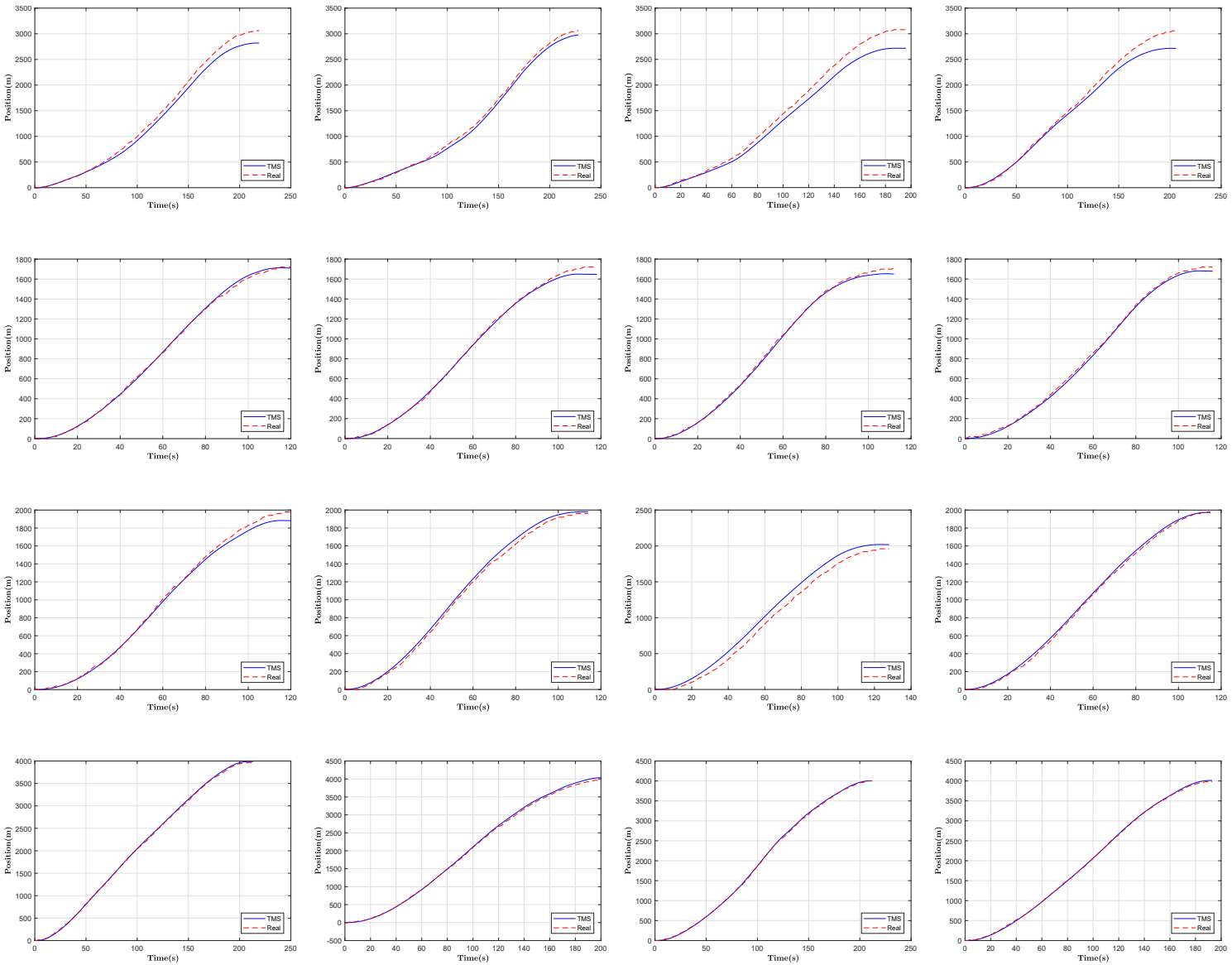


Figure 5.16: Positions results using parameters from Table 5.9 - Measurements in red dashed line, TMS in blue line

In spite of using a different cost function, the SA algorithm has a similar behavior, as can be seen by the analysis of the cost function evolution along iterations. Fig 5.17 shows cost function values.

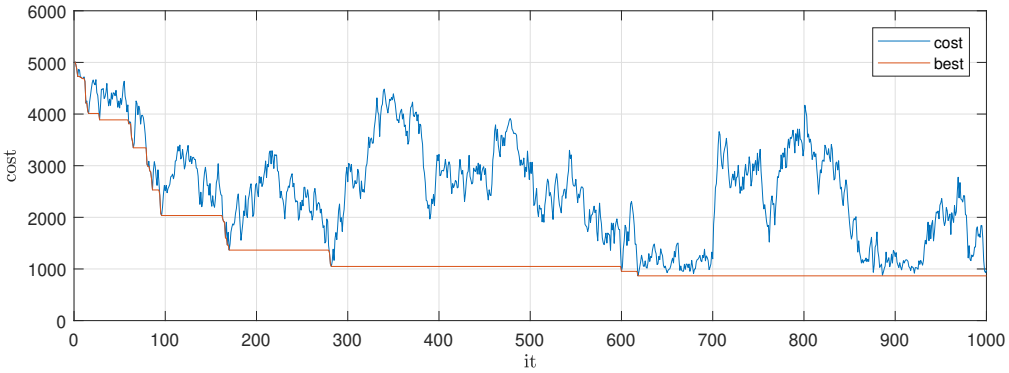


Figure 5.17: SA cost function (5.21).

5.5 Methodology Comparison

After performing all tests and analyzing all the results, a comparison between all approaches has been carried out. This comparison was made to understand which approach best meets the objective.

In order to use the same comparison for each of the solutions, the quadratic error was calculated for the determined speed, as well as for the position of the train and the consumed energy. Fig. 5.18 shows a bar plot where each quadratic error is represented.

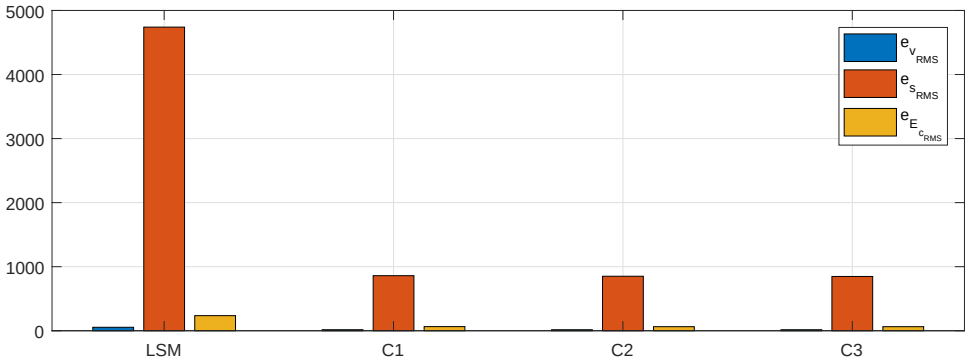


Figure 5.18: Results comparison.

The bar plot is composed of four groups and each one contains three bars. The first group represents LSM's results. The second group is related to the SA algorithm, using the first cost function.

The third and fourth groups are related to the second and third cost functions, respectively. In each group, the first bar corresponds to the quadratic error of velocity, the second to position and the last one to energy consumption. Looking at Fig. 5.18, the first thing that stands out is the low performance of LSM. Besides the first group, it is not possible, from the analyses of the figure, to decide which is the best cost function. So, to reach a better understanding of the results' analyses, a summary table was created, Table 5.10.

Table 5.10: Summary table of results

Method	$e_{v_{RMS}}$	$e_{s_{RMS}}$	$e_{E_{c_{RMS}}}$
LSM	54.11	4.74e3	235.58
C1	15.66	859.40	64.36
C2	15.50	850.90	63.28
C3	15.44	847.34	63.39

From that, it can be seen that there are no great differences between the various cost functions implemented. In fact, cost function three has lower velocity and position errors, which may lead to conclude that it would be the best option, but the difference is not so substantial.

CHAPTER 6

Conclusion

Before the end of this thesis, this last chapter shows the main conclusions. In addition, some suggestions for future work are presented.

6.1 Main Conclusions

During the development of this thesis, an algorithm was established, to be integrated into a DAS. The algorithm had several requirements, which orientated the developments of this thesis in order to accomplish all of them. The implemented algorithm searches for speed profiles, constraint to scheduling and line gradients, with the purpose of reducing energy consumption. So, the algorithm must receive line information about velocity limits, gradients values and dead zones locations, and before train departure, a new solution for the actual journey must be available for the train driver to be advised. In other words, it is expected that the algorithm is able to produce a real time solution in order to determine, at each train stop, a speed profile for the next journey.

The algorithm development started with the study of a dynamic model of the train, used to simulate the vehicle's behavior. The choice of the dynamic model to use was made taking into account some factors, such as complexity and vehicle simulation capability. Before starting the implementation of a dynamic model, a deep study on the state of the art was done, and it was verified that, unanimously, a large part of the studied works was using the same dynamic model. This model stands out for its low complexity in representing train movement, without losing much detail. Since this model has been widely used and train behavior is represented with considerable accuracy, it was selected for the implementation. Besides, since the algorithm is expected to run as a real time application, the low complexity was seen as an advantage of its use.

Following the thesis developments, after having a train model studied and implemented, the development of a TMS algorithm started. This was the task with the highest devoted time. In fact, putting into practice the TMS algorithm led to several implementation methodologies, starting by speed pro-

file determination for simple lines. Furthermore, after introducing restrictions to the algorithm, it finished on a stage that was able to determine a solution for any journey. The train model was defined as a state machine, each driving regime being defined as a state. With the used methodology, the algorithm, with its line constraints, follows a sequence of state transitions in order to define the best driving regime sequence. After looking at the algorithm's results, it can be realized that it follows the initial proposed requirements. The results are very good, which allowed the thesis progress towards the main objectives.

The TMS algorithm was combined with an OSP generator with the purpose of determining the speed profile which best fits the algorithm's objectives. The best solution determination is a task of the optimization algorithm, the SA algorithm being the one selected due to its advantages and potentialities, such as the ability to escape to local minima. The first implementation of the algorithm was in accordance with thesis main objective, which is the reduction of energy consumption. Throughout the thesis developments, a second trend of advances appeared. On actual railway lines, a very common problem is late train arrivals at destinations. Focused on a positive contribution to this second objective, a second algorithm version, aiming at reducing these delays, was implemented. Both optimization algorithms, after being tested, presented very good results, showing convergence to cost function minimum values. Since these algorithms are to be implemented in a real time application, the time needed to process each solution was also controlled, and both algorithms showed a high potential for this purpose. The implemented algorithms are ready to determine velocity profiles for real railway lines, with permanent and transitory constraints and are highly configurable.

The last development of this thesis was the implementation of an algorithm to determine train model parameters. This last development was carried out with the purpose of making the DAS algorithm more versatile with the possibility of being used in different types of trains. The algorithm's objectives were focused on its ability to correctly estimate values from train model parameters, without any restriction concerning the execution time. Two different approaches were applied in order to solve the problem. The first approach uses the nonlinear least square method, which showed a low performance, since the estimated parameters where quite far from the optimal values. The second approach uses an optimization algorithm to search for parameters values. The optimization algorithm used was similar to the one developed for the OSP algorithm, and this one showed a much better performance. Several cost functions were tested, showing no major differences, the cost function with less operations being selected. The results, as exposed, are very convincing, showing that the

algorithm using simulated annealing potentialities is able to solve the problem.

6.2 Future Work

In the end, after successfully achieving the thesis main objectives and looking at obtained results, there are some points which can be targeted for improvements. Some of the improvements here referred were introduced during the thesis development, but somehow were removed from the algorithm's last version. Starting with the OSP algorithm, some changes can be applied in order to improve the algorithm's performance, resulting in reducing the time needed to determine each solution. One improvement may be the determination of the average value of each journey before starting the speed profile determination. Knowing the velocity average value for the actual journey, and once in all journeys a cruising velocity higher than average is necessary, a new generation mechanism for cruising velocity can be obtained with limits between the maximum train velocity and its average value. With this modification, a faster algorithm convergence is expected, since it will reduce the number of impossible solutions tested in the actual algorithm's version. This algorithm change can be applied to both versions, energy and time minimization.

Still looking at the OSP algorithm, the second proposed point as future work, is related with the algorithm's implementation on a mobile platform. A first version of the algorithm was already converted from Matlab to other programming language. Since it is expected that it runs on a mobile platform, an android one was selected, and by this, the programming language to use on OSP was restricted to compatible ones. From some possible compatible languages, C++ was selected. At the moment, the algorithm was converted to be compatible with C++ language, the code translation being successfully obtained. Since some changes were applied to the algorithm after this translation, as future work, we also propose the final version conversion.

To conclude the algorithm analysis, as future work is also suggested the introduction of speed profiles on real railway lines. The algorithm's outputs were only analyzed by comparison with real data, showing great potential to help minimize energy consumption. In addition, the results were also analyzed by individuals linked to the development of railway systems, and it has been able to satisfy their demands. This way, the algorithm's first version is currently under tests at a Danish train track. As a final test, we proposed the use of this algorithm on other real train tracks, in its DAS, in order to have real data acquired on train, following the advices given by the algorithm. This is presented as future work because requesting the use of this algorithm on real trains is a lengthy process, as the

approval by multiple entities is required.

Finally, the algorithm implemented to determine train model parameters can also suffer some modifications. With satisfactory results, it would also be interesting to change the algorithm's implementation programming language. For the implementation of this algorithm on a portable interface to be possible, C++ programming language may be an option. In addition, the routines separately used to pre-process the available data could also be added to the algorithm, making the parameter estimation completely autonomous between the moment when raw data is received up to the moment when output is presented

References

[1] shift2rail.org, "Infrastructure innovation programme 3 shift2rail," June 2016. [Online]. Available: http://shift2rail.org/research-development/ip3/

[2] A. Fernández-Rodríguez, A. Fernández-Cardador, and A. P. Cucala, "Energy efficiency in high speed railway traffic operation: a real-time ecodriving algorithm," in *IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, June 2015, pp. 325–330.

[3] Y. Jiang, J. Liu, W. Tian, M. Shahidehpour, and M. Krishnamurthy, "Energy harvesting for the electrification of railway stations: Getting a charge from the regenerative braking of trains.a," *IEEE Electrification Magazine*, vol. 2, no. 3, pp. 39–48, Sept 2014.

[4] S. de la Torre, A. J. Sánchez-Racero, J. A. Aguado, M. Reyes, and O. Martínez, "Optimal sizing of energy storage for regenerative braking in electric railway systems," *IEEE Transactions on Power Systems*, vol. 30, no. 3, pp. 1492–1500, May 2015.

[5] A. Gonzalez-Gil, R. Palacin, and P. Batty, "Sustainable urban rail systems: Strategies and technologies for optimal management of regenerative braking energy," *Energy Conversion and Management*, vol. 75, pp. 374 – 388, 2013.

[6] M. Dominguez, A. Fernandez-Cardador, A. P. Cucala, and R. R. Pecharroman, "Energy savings in metropolitan railway substations through regenerative energy recovery and optimal design of ato speed profiles," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 3, pp. 496–504, July 2012.

[7] G. M. Scheepmaker, R. M. Goverde, and L. G. Kroon, "Review of energy-efficient train control and timetabling," *European Journal of Operational Research*, vol. 257, no. 2, pp. 355 – 376, 2017.

[8] P. Howlett, I. Milroy, and P. Pudney, "Energy-efficient train control," *Control Engineering Practice*, vol. 2, no. 2, pp. 193 – 200, 1994.

[9] J. Qu, X. Feng, and Q. Wang, "Real-time trajectory planning for rail transit train considering regenerative energy," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, Oct 2014, pp. 2738–2742.

[10] I. P. Milroy, "Aspects of automatic train control," Ph.D. dissertation, © Ian Peter Milroy, 1980.

[11] R. R. Liu and I. M. Golovitcher, "Energy-efficient operation of rail vehicles," *Transportation Research Part A: Policy and Practice*, vol. 37, no. 10, pp. 917 – 932, 2003.

[12] C. S. Chang and S. S. Sim, "Optimising train movements through coast control using genetic algorithms," *IEE Proceedings - Electric Power Applications*, vol. 144, no. 1, pp. 65–73, Jan 1997.

[13] K. K. Wong and T. K. Ho, "Dynamic coast control of train movement with genetic algorithm," *International Journal of Systems Science*, vol. 35, no. 13-14, pp. 835–846, 2004.

[14] S. Acikbas and M. T. Soylemez, "Coasting point optimisation for mass rail transit lines using artificial neural networks and genetic algorithms," *IET Electric Power Applications*, vol. 2, no. 3, pp. 172–182, May 2008.

[15] P. G. Howlett and P. J. Pudney, *Practical Strategy Optimisation*. Springer London, 1995, pp. 285–297.

[16] S. Su, T. Tang, L. Chen, and B. Liu, "Energy-efficient train control in urban rail transit systems," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 229, no. 4, pp. 446–454, 2015.

[17] Y. Huang, X. Ma, S. Su, and T. Tang, "Optimization of train operation in multiple interstations with multi-population genetic algorithm," *Energies*, vol. 8, no. 12, pp. 14 311–14 329, 2015.

[18] R. Chevrier, G. Marliere, B. Vulturescu, and J. Rodriguez, "Multi-objective evolutionary algorithm for speed tuning optimization with energy saving in railway: Application and case study," in *RailRome 2011*, Rome, 2011.

[19] R. Chevrier, P. Pellegrini, and J. Rodriguez, "Energy saving in railway timetabling: A bi-objective evolutionary approach for computing alternative running times," *Transportation Research Part C: Emerging Technologies*, vol. 37, pp. 20 – 41, 2013.

[20] S. Lu, S. Hillmansen, T. K. Ho, and C. Roberts, "Single-train trajectory optimization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 743–750, June 2013.

[21] Y. Huang, C. Yang, and S. Gong, "Energy optimization for train operation based on an improved ant colony optimization methodology," *Energies*, vol. 9, no. 8, 2016.

[22] M. Domínguez, A. Fernández-Cardador, A. P. Cucala, T. Gonsalves, and A. Fernández, "Multi objective particle swarm optimization algorithm for the design of efficient ATO speed profiles in metro lines," *Engineering Applications of Artificial Intelligence*, vol. 29, pp. 43 – 53, 2014.

[23] K. Kim and S. I.-J. Chien, "Optimal train operation for minimum energy consumption considering track alignment, speed limit, and schedule adherence," *Journal of Transportation Engineering*, vol. 137, no. 9, pp. 665–674, 2011.

[24] K. Keskin and A. Karamancioglu, "Energy-efficient train operation using nature-inspired algorithms," *Journal of Advanced Transportation*, vol. 2017, 2017.

[25] P. Lukaszewicz, "A simple method to determine train running resistance from full-scale measurements," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 221, no. 3, pp. 331–337, 2007.

[26] C. Somaschini, T. Argentini, D. Rocchi, P. Schito, and G. Tomasini, "A new methodology for the assessment of the running resistance of trains without knowing the characteristics of the track: Application to full-scale experimental data," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 232, no. 6, pp. 1814–1827, 2018.

[27] S. Aradi, T. Becsi, and P. Gaspar, "Estimation of running resistance of electric trains based on onboard telematics system," *International Journal of Heavy Vehicle Systems*, vol. 22, pp. 277–291, 01 2015.

[28] B. P. Rochard and F. Schmid, "A review of methods to measure and calculate train resistances," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 214, no. 4, pp. 185–199, 2000.

[29] C. Somaschini, D. Rocchi, G. Tomasini, and P. Schito, "Simplified estimation of train resistance parameters: Full scale experimental tests and analysis," 04 2016.

[30] R. BOSQUET, P. O. Vandanjon, A. Coiret, and T. Lorino, "Model of high-speed train energy consumption," in *International Conference on Railway Engineering and Management*, Denmark, Jun. 2013, pp. 5p, graphiques, tabl., bibliogr.

[31] P. O. VANDANJON, R. BOSQUET, A. Coiret, and M. Gautier, "Model of High-Speed train energy consumption," in *15th MINI CONFERENCE ON VEHICLE SYSTEM DYNAMICS, IDENTIFICATION AND ANOMALIES (VSDIA 2016)*, BUDAPEST, Hungary, Nov. 2016, pp. pp. 47–53.

[32] P. Howlett, P. Pudney, and X. Vu, "Estimating train parameters with an unscented kalman filter," *Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference*, 01 2004.

[33] X. Liu, B. Ning, J. Xun, C. Wang, X. Xiao, and T. Liu, "Parameter identification of train basic resistance using multi-innovation theory," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 637 – 642, 2018, 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018.

[34] N. Jaoua, P. Vanheeghe, N. Navarro, O. Langlois, and M. Iordache, "A bayesian approach for parameter estimation in railway systems," in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, March 2018, pp. 1–6.

[35] H. S. Hansen, M. U. Nawaz, and N. Olsson, "Using operational data to estimate the running resistance of trains. estimation of the resistance in a set of norwegian tunnels," *Journal of Rail Transport Planning & Management*, vol. 7, no. 1, pp. 62 – 76, 2017.

[36] S.-W. Kim, H.-B. Kwon, Y.-G. Kim, and T.-W. Park, "Calculation of resistance to motion of a high-speed train using acceleration measurements in irregular coasting conditions," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 220, no. 4, pp. 449–459, 2006.

[37] H. Kwon, "A study on the resistance force and the aerodynamic drag of korean high-speed trains," *Vehicle System Dynamics*, vol. 56, no. 8, pp. 1250–1268, 2018.

[38] J. Sun, J. M. Garibaldi, and C. Hodgman, "Parameter estimation using metaheuristics in systems biology: A comprehensive review," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 185–202, Jan 2012.

[39] D. Akman, O. Akman, and E. Schaefer, "Parameter estimation in ordinary differential equations modeling via particle swarm optimization," *Journal of Applied Mathematics*, vol. 2018, 2018.

[40] C. Zhan, W. Situ, L. F. Yeung, P. W. Tsang, and G. Yang, "A parameter estimation method for biological systems modelled by ode/dde models using spline approximation and differential evolution algorithm," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 11, no. 6, pp. 1066–1076, Nov 2014.

[41] O. Gonzalez, C. Kuper, K. Jung, P. Naval, and E. Mendoza, "Parameter estimation using simulated annealing for s-system models of biochemical networks," *Bioinformatics (Oxford, England)*, vol. 23, pp. 480–6, 03 2007.

[42] M. Ali, M. Pant, A. Abraham, and V. Snasel, "Modified differential evolution algorithm for parameter estimation in mathematical models," in *2010 IEEE International Conference on Systems, Man and Cybernetics*, Oct 2010, pp. 2767–2772.

[43] Z. Dai and L. Lai, "Differential simulated annealing: a robust and efficient global optimization algorithm for parameter estimation of biological networks," *Mol. BioSyst.*, vol. 10, pp. 1385–1392, 2014.

[44] R. K. Arora, *OPTIMIZATION - Algorithms and Applications*, 2015.

[45] B. Wang, Z. Yang, F. Lin, and W. Zhao, "An improved genetic algorithm for optimal stationary energy storage system locating and sizing," *Energies*, vol. 7, no. 10, p. 6434, 2014.

[46] R. Chibante, A. Araujo, and A. Carvalho, "Parameter identification of power semiconductor device models using metaheuristics," in *Simulated Annealing*, R. Chibante, Ed. Rijeka: InteChOpen, 2010, ch. 1.

[47] P. G. Howlett and P. J. Pudney, *Energy-Efficient Train Control*. London:Springer, 1995.

[48] H. Douglas, P. Weston, D. Kirkwood, S. Hillmansen, and C. Roberts, "Method for validating the train motion equations used for passenger rail vehicle simulation," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 231, no. 4, pp. 455–469, 2017.

[49] S. Yi, "Chapter 2 - traction calculation," in *Principles of Railway Location and Design*, S. Yi, Ed. Academic Press, 2018, pp. 73 – 157.

[50] M. U. Nawaz, "Estimation of running resistance in train tunnels," Master's thesis, NTNU, 2015.

[51] IP, "Léxico | infraestruturas de portugal," April 2019. [Online]. Available: https://www.infraestruturasdeportugal.pt/negocios-e-servicos/lexico/z

[52] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation. part i, graph partitioning," *Oper. Res.*, vol. 37, no. 6, pp. 865–892, Oct. 1989.

[53] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[54] B. Chopard and M. Tomassini, *Simulated Annealing*. Cham: Springer International Publishing, 2018, pp. 59–79.

[55] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines*. New York, NY; John Wiley and Sons Inc., 1988.

[56] Z. Michalewicz and D. B. Fogel, *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

[57] M. T. Outeiro, "A new high power efficient electronic converter for fuel cell applications," Ph.D. dissertation, FEUP, 2012.

[58] I. The MathWorks, "Simulated annealing options," Jul. 2019. [Online]. Available: https://www.mathworks.com/help/gads/simulated-annealing-options.html#bq26ky_-1

[59] W. Ben-Ameur, "Computing the initial temperature of simulated annealing," *Computational Optimization and Applications*, vol. 29, no. 3, pp. 369–385, Dec 2004.

[60] S. Ledesma, G. Aviña, and R. Sanchez, "Practical considerations for simulated annealing implementation," *Simulated annealing*, vol. 20, pp. 401–420, 2008.

[61] R. F. Chibante, "Desenvolvimento de um modelo para igbts optimizado por um metodo de base experimental," Ph.D. dissertation, FEUP, 2005.

[62] T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

[63] Y. Nourani and B. Andresen, “A comparison of simulated annealing cooling strategies,” *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, p. 8373, 1998.

[64] A. Y. Zomaya and R. Kazman, “Algorithms and theory of computation handbook,” M. J. Atallah and M. Blanton, Eds. Chapman & Hall/CRC, 2010, ch. Simulated Annealing Techniques, pp. 33–33.

[65] R. L. Burden and J. D. Faires, *Numerical analysis*, 7th ed. Australia Brooks/Cole, 2001.