

Evolving Dynamic Traffic Assignments

Andrés Arribas

Thesis submitted to

Obtain the degree of

Master of Science

In Artificial Intelligence

**Promotor:**

Prof. dr. Tom Holvoet.

**Advisors:**

Rutger. Claes

Rinde. Van Lon

**Academic Year 2012 - 2013**

© Copyright by K.U.Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promoter(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot de K.U.Leuven, Faculteit Ingenieurswetenschappen - Kasteelpark Arenberg 1, B-3001 Heverlee (België). Telefoon +32-16-32 13 50 & Fax. +32-16-32 19 88.

Voorafgaande schriftelijke toestemming van de promoter(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma’s voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

© Copyright by K.U.Leuven

Without written permission of the supervisor(s) and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to K.U.Leuven, Faculty of Engineering - Kasteelpark Arenberg 1, B-3001 Heverlee (Belgium). Telephone +32-16-32 13 50 & Fax. +32-16-32 19 88.

A written permission of the supervisor(s) is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.

Foreword

May this thesis be the high note with which this challenging but rewarding year at KU Leuven comes to a close.

I would like to express my gratitude to my daily advisors Rutger Claes and Rinde van Lon, for their advice and clarity of ideas. My appreciation too, to Willem Himpe who was generous with his time and explained to me the most important traffic concepts and the details of the original simulator.

Finally, I would like to dedicate this work to my family in Spain, always supportive despite the distance, and to Judith, my partner in crime, the one with whom I ride the risks of life.

Andrés Arribas

Table of Contents

Foreword iii

Table of Contents v

Abstract viii

List of figures and tables ix

List of figures ix

List of tables x

List of abbreviations and symbols xi

Chapter 1: Introduction 1

Chapter 2: Traffic Theory 3

2.1 Traffic Systems Modelling 3

2.2 Demand Model 4

2.3 Supply Model 5

2.3.1 Network Structure 5

2.3.2 Flows 6

2.3.3 Performance Variables and Transportation Costs 7

2.4 Traffic Flow Model 7

2.4.1 Dynamic Network Loading 8

2.4.1.1 Link Transmission Model 8

2.4.1.1.1 Cumulative Vehicle Numbers 9

2.4.1.1.2 Traffic Propagation 10

2.4.2 Dynamic Traffic Assignment 11

2.4.2.1 DUE and DTA Basics 12

2.4.2.2 Route Choice Algorithms 14

2.4.2.3 MSA 15

2.4.2.4 DEC 15

2.5 Conclusion 17

Chapter3: Genetic Programming 18

3.1 Genetic Algorithms 18

3.2 Overview of Genetic Programming 22

3.3 Details of Genetic Programming 23

3.4 Conclusion 28

Chapter4: Experiments 29

4.1 Objectives and Overview 29

4.2 Tools 29

4.3 Control over the Evolved Heuristic 30

4.4 Additional Control and Speed-Ups 32

4.5 Fitness Evaluation 32

4.6 Genetic Programming Parameters 34

4.7 Experiment 1 34

4.7.1 Training Set 34

4.7.2 Final Fitness 37

4.7.3 DTA Control Parameters 37

4.7.4 Test Set 38

4.7.5 Test Cases / Comparisons 39

4.7.6 Results 43

4.7.6.1 Fitness Progression 43

4.7.6.2 Training Set 44

4.7.6.3 Test Cases 47

4.7.6.4 Conclusions 56

4.7.6.5 Why DEC does not perform up to the expectations 57

4.2 Experiment 2 58

4.8.1 Training Set 58

4.8.2 Final Fitness 59

4.8.3 DTA Control Parameters 59

4.8.4 Test Set 59

4.8.5 Test Cases / Comparisons 59

4.8.6 Results 60

4.8.6.1 Fitness Progression 60

4.8.6.2 Training Set 61

4.8.6.3 Test Cases 65

Chapter 5: Conclusion 73

Appendices 74

Appendix A: Source Code 75

Appendix B: Stagnation in Early Experiment 76

Appendix C: Best Individual in Experiment 1 77

Appendix D: Best Individual in Experiment 2 78

Bibliography 79

Master’s Thesis file 80

Abstract

Maximum 1 page

List of figures and tables

List of figures

Figure 1 Example of OD Matrix (from Cascetta [2]) 4

Figure 2 Example of Network, Paths and OD Pairs (from Cascetta [2]) 6

Figure 3 Cumulative Vehicle Numbers (from Corthout [7]) 10

Figure 4 Triangular Fundamental Diagram (from Corthout [7]) 11

Figure 5 DTA, the solution algorithm to the DUE (from Rahamin et al [4]) 15

Figure 6 Flowchart of the conventional genetic algorithm (from Koza [3]) 21

Figure 7 Example of tree for S-expression 24

Figure 8 Two parental computer programs (from Koza [3]) 26

Figure 9 Crossover fragments resulting from selection of point 2 of the first parent and point 6 of the second parent (from Koza [3]) 26

Figure 10 Resulting offspring (from Koza [3)) 26

Figure 11 Evaluation of Individual 33

Figure 12 Training Networks 36

Figure 13 Network 6 (First appeared in Rahamin et al. [7]) 38

Figure 14 Map of Rotterdam (Network 7) 39

Figure 15 Highlighted map of Rotterdam 42

Figure 16 Statistics Experiment 1 43

Figure 17 MSA vs DEC vs GP1 for Network1 44

Figure 18 MSA vs DEC vs GP1 for Network 2 45

Figure 19 MSA vs DEC vs GP1 for Network 3 45

Figure 20 MSA vs DEC vs GP1 for Network 4 46

Figure 21 MSA vs DEC vs GP1 for Network 5 46

Figure 22 Statistics Experiment 2 49

Figure 23 MSA vs DEC vs GP2 for Network1 50

Figure 24 MSA vs DEC vs GP2 for Network 2 51

Figure 25 MSA vs DEC vs GP2 for Network 3 51

Figure 26 MSA vs DEC vs GP2 for Network 4 52

Figure 28 MSA vs DEC vs GP2 for Network 5 52

Figure 29 MSA vs DEC vs GP2 for Network 6 53

List of tables

Table 1 Demand flows in Case 4 37

List of abbreviations and symbols

π the number pi

∞ infinity …

Chapter 1: Introduction

Simulation of traffic propagation and its impact over a traffic network is a powerful and common tool to study default user behaviour and impact of events and changes to the network.

A simulation of dynamic traffic, i.e., in which a user departing at a given time may influence users departing at different times, is typically considered realistic if it has reached Dynamic User Equilibrium (DUE). DUE states that no vehicle may be able to reach its destination faster by choosing an alternative route. The DUE is based on the assumption that users are rational and take their decisions based on experience.

The Dynamic Traffic Assignment (DTA) is the process at the hearth of such simulations. More specifically, the DTA is an iterative process that incorporates a route assignment step and a step computing traffic propagation and network performance (network loading). In essence, the route assignment splits the units of travel demand (vehicles) according to their departure times, and assigns them to pre-computed routes connecting their origin and destination.

In complex traffic networks the DTA rarely attains equilibrium in the first iteration. In practice, given the lack of an exact formal solution, a heuristic, which is a computationally efficient approximation algorithm, that does, however, not guarantee optimality, is used to perform the route assignment towards equilibrium. The speed of convergence of the DTA towards DUE depends, first and foremost, on this heuristic.

Despite a number of attempts at designing such assignment heuristics, two of which are considered in this work, it has proven difficult to produce by hand efficient heuristics due to the complexity of the problem.

The objectives in this thesis are, first, to consider the adequacy of using a particular automatic computer-based method to generate such heuristics: genetic programming, and, secondly, by using genetic programming, attempt to ameliorate existing manual heuristics.

Genetic programming is a technique that mimics natural selection to evolve computer programs. Given that many problems can be reformulated as requiring the search for a computer program, genetic programming has been applied to many problems, often with success. Nevertheless, genetic programming has never been tried on this specific problem in traffic.

The rest of this thesis is organised as follows: chapter 2 covers the essentials of traffic theory including demand and supply modelling, Dynamic Network Loading, the DTA and Route Assignment; chapter 3 covers genetic programming by first introducing genetic algorithms, and then focusing on genetic programming specifics; chapter 4 is a detailed account of all performed experiments and their results; finally, chapter 6 concludes and proposes further research steps.

Chapter 2: Traffic Theory[[1]](#footnote-1)

In this chapter basics of traffic theory are explained. The objective is to serve as an introduction to the most important concepts directly used in this thesis. This chapter is intended at readers with little or no background in Traffic theory. Out of scope remains to give a detailed picture of the field.

2.1 Traffic Systems Modelling

Transportation systems can be studied from a variety of angles e.g. economic, social, purely mathematical, etc., and at different scales e.g. regional, local or state wise.

In this thesis we are only concerned with a simplified view of what a transportation system is: we will focus on simulating simple small traffic networks and measuring their efficiency in terms of performance (e.g. travel times) given predefined fictional travel demands.

In order to simulate a traffic system, the first step is to model it.

A traffic model is a mathematical representation of the physical and organisational elements comprising a traffic network, as well as the travel demand and the emerging interactions.

Even the simplest traffic models have to take into consideration the inherent complexity of transportation systems, which arises from the multiplicity of non-linear interactions and feedback cycles.

In order to do so, a traffic model is most commonly defined by means of a demand model plus a combination of a traffic flow model and network model. The former is used to analyse and simulate the performances of the main supply elements, the latter to represent the topological and functional structure of a system.

The subset of traffic flow theory used in this work is a very specific one that requires careful consideration. But first, let us briefly summarize the demand representation to be used and the fundamentals of network/supply models.

2.2 Demand Model

The number of users consuming particular services offered by a transportation system in a given time period is the travel-demand flow. Travel demand flows, result from the aggregation of individual trips made in the area of study during the period of reference. A trip is defined as the act of moving from an origin to a destination.

The spatial characterization of trips is made by grouping them by place of origin and destination; accordingly demand flows can be arranged in tables commonly referred to as O-D matrices, whose rows correspond to the different origin and destination zones, respectively.

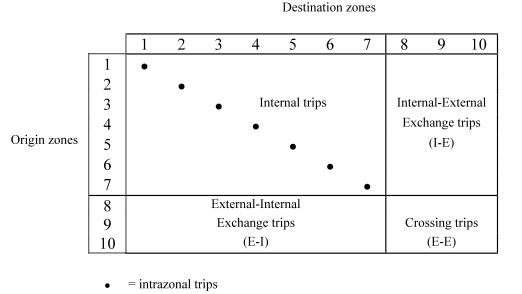
**

Figure 1 Example of OD Matrix (from Cascetta [2])

A cell in the matrix gives the number of trips in the reference period from origin o to destination d.

The accumulated number of trips with origin o is therefore:

And the accumulated number of trips attracted by destination d is:

2.3 Supply Model

In this section we look at the foundations of congested network models.

2.3.1 Network Structure

A network structure is represented by a directed graph. A directed graph consists of a set of nodes, N, and a set of connections between pairs of nodes, L, called links, such that L ≥ NxN. In a directed graph links are oriented.

A link does not necessarily correspond to a construction in the physical world. Links rather represent phases and/or activities of possible trips between different traffic zones. The core idea behind a link is that its physical and functional characteristics can be assumed to be homogeneous for the whole link. In this sense, links can be seen as the partition of trips into segments, each of which has certain characteristics.

Nodes correspond to significant events delimiting trip phases. A node can represent the same event occurring at different time instants (between two trip phases). For example, the different entry or exit times in a road segment.

A trip is a sequence of several phases/links called a path. A path is defined from an initial node, the origin, to a final node, the destination. Each path is unambiguously associated with one, and only one O-D pair, whereas several paths can connect the same O-D pair.

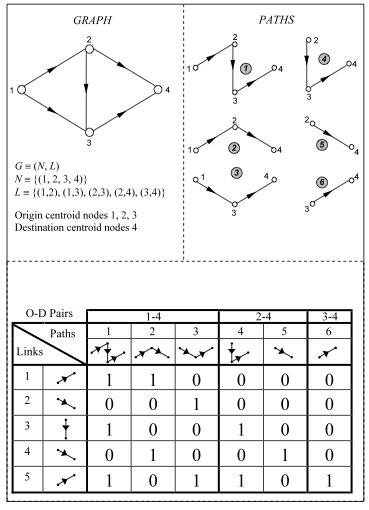


Figure 2 Example of Network, Paths and OD Pairs (from Cascetta [2])

2.3.2 Flows

An instantaneous link flow is the instantaneous number of units using a link (i.e. units in that phase of the trip at that time instant).

Typically, when computing the link flow, the difference between types of units has to be taken into account, however, in this work, we consider all units to be of the same homogenous type.

A path flow is obtained as the sum of the link flows for all links in the path, considering in the flow calculations only those units matching the origin and destination of the path.

2.3.3 Performance Variables and Transportation Costs

Some variables perceived by the users can be associated to individual trip phases (hence links). Examples of such variables are travel times, monetary cost, and discomfort. These variables are referred as level-of-service or performance attributes. In general, performance variables correspond to disutilities or costs for the users.

Given our limited scope and simplified approach we focus on travel times as the only performance variable. Accordingly, from now onwards, “travel time”, “cost” and “performance variable/indicator” may be used indistinctly.

Remark that a cost will reflect not only the assigned physical characteristics of the link at hand (e.g. length, free speed), but also the interaction of the units in the link. In particular, the phenomenon of congestion, which we define precisely in the next section, will have a significant impact on the cost.

Similarly to the flows, a path cost can be computed as the sum of the cost of all links in the path.

2.4 Traffic Flow Model

Usually traffic flow models can be split in stationary models, where there is no variance in time of variables, and dynamic models, where there is variance in time.

Stationarity could only be observed in the real work if demand, path choices and supply system remained constant for a very long period of time. In less than a day of time, this is not the case and the behaviour of the traffic system does not only depend on its characteristics but also on its history.

Hence, we are interested in dynamic models, that, contrary to stationary models, allow the effective, realistic, simulation of important phenomena, especially the creation, propagation and dissipation of queues.

There are several approaches to dynamic traffic modelling (also called “within-day dynamics) depending on the scale at which units are considered e.g. continuous fluid analogy, packet based, or individual units. We are concerned with so called “Macroscopic” models.

Macroscopic models are based on the analogy with fluid dynamics. In such a models the individual vehicles are treated as a continuous[[2]](#footnote-2) (one-dimensional) fluid for which variables such as flow, density and velocity are defined at each point in space and time. Macroscopic models comprise equations for the conservation of mass (vehicles) and the relation between flow and density.

More specifically, we are interested in “Space-discrete” Macroscopic models for which it is assumed that basic variables affecting link performance, such as density or speed, do not vary along the link.

A fundamental aspect to take into account given dynamicity is that the number of users within a link depends now on the travel times required to reach that link, which, in turn, depends on the number of users encountered on the network links in the previous instants. This circular dependency between flow and link performance, does not allow the resolution of the two models (flow propagation an link performance) in a sequential fashion. This problem is known as Dynamic Network Loading.

2.4.1 Dynamic Network Loading

The DNL model is responsible for propagating path demands through the network. The traffic flows are calculated in time and space, congestion may form, delays may be encountered and the travel times on the network are determined.

The DNL we use is the Link Transmission Model (LTM)

Whereas a mastery of the LTM or even in depth knowledge of it are not a requirement for this thesis it is desirable to look at a brief introduction to the LTM which will also serve as a means of covering some key aspects of traffic flow theory worth keeping in mind when assessing the upcoming sections and the final results.

2.4.1.1 Link Transmission Model[[3]](#footnote-3)

The Link Transmission Model is a DNL model proposed by Yperman at KU Leuven in 2007 [BIB].

Amongst its most significant characteristics we may count a high realism of representation of congestion formation (whenever the traffic in a link surpasses its free flow capacity) and spillback (backwards propagation of the effect of congestion) as well as computational efficiency..

We will now discuss firstly, the concept of Cumulative Vehicle Numbers, which forms the basis of the calculations in the LTM, and secondly, the involved basics of traffic propagation.

Many details of traffic theory of no interest to this work have been omitted.

2.4.1.1.1 Cumulative Vehicle Numbers

The aggregation of vehicles having passed a location x by time t is the Cumulative Vehicle Number, or CVN, denoted by N(x,t).

In the LTM units are assumed to follow a predefined route between origin and destination creating different streams of traffic given different origins and destinations.

For a particular link, CVN in the LTM are calculated at all upstream and downstream link boundaries. Through time the CVN, given that the locations of measurement are fixed, will draw a shape through time. Additionally, CVN curves are also computed for complete streams at origin and destination.

Once all the curves have being obtained different variables can be derived.

Link travel times can be derived as the horizontal distance between the curve at the upstream boundary N(,t), and the curve at the downstream boundary N(,t).

Analogously, total travel time for a path is the horizontal difference between CVN at origin and CVN ay destination.

The total number of vehicles in a link is the vertical distance between upstream and downstream CVN curves.

The flow q (veh/h) is simply the slope of the CVN curve.

The density k (veh/km) on a link can easily be calculated by dividing the number of vehicles ΔN by the length L.

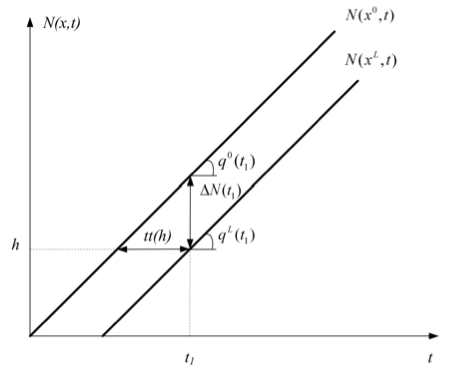


Figure 3 Cumulative Vehicle Numbers (from Corthout [7])

2.4.1.1.2 Traffic Propagation

In the LTM, the “fundamental triangular diagram” is considered an accurate representation of the propagation of traffic in unidirectional links. The diagram itself is just a plot of the flow in relation to the density. What is fundamental is the set of assumptions and definitions behind the diagram.

Below the critical density kc, which corresponds to the link’s capacity C, vehicles are assumed to travel with a fixed free speed vf (km/h) characteristic of the link.

Above the critical density kc , hence for congested traffic, the speed in the link v is given by q/k.

The maximum density is defined as the jam density, kjam, and corresponds to vehicles standing still.

Different traffic states are separated by shock waves that may propagate up- or downstream with a speed Ws, that can be derived as the slope of the connecting line between the adjoining traffic states.

The maximum negative shockwave speed w is the fastest possible speed with which congestion may spill back in the upstream direction.

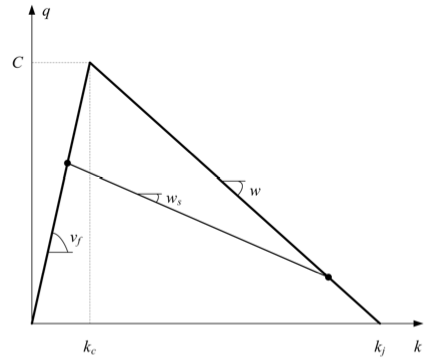


Figure 4 Triangular Fundamental Diagram (from Corthout [7])

In this thesis it suffices to know that all the above can be computed for a particular time step from CVN curves (including curves produced in previous time steps).

Finally, note that the above is a description of the propagation of traffic in a link.

The LTM may use different models to describe intersections i.e. propagation of incoming flows to outgoing links. These are out of scope; suffice to say, nevertheless, that once the flows are computed the CVN per link can be easily updated.

2.4.2 Dynamic Traffic Assignment

But, does DNL account for a realistic simulation of traffic? No, it does not on its own. DNL serves as a means of computing the model response for an a-priori assignment of traffic. That is, given that the path each vehicle will follow is known, we may use DNL to compute flow propagation and link performance. From the DNL point of view, it is not relevant whether the assignment reflects or does not reflect appropriate route choices.

It is the task of so called Dynamic Traffic Assignment (DTA) Models to, in addition, consider assignment accuracy and henceforth reproduce more adequately system dynamics within the period of simulation.

2.4.2.1 DUE and DTA Basics

Dynamic Traffic Assignments are used to study user route choice and propagation of traffic in time, including the reaction to changes and the impact on other users of congestion and driver’s decisions.

DTA models are mainly made of four sub-models, the demand model, the supply model, the DNL and the route choice/assignment model. We have already seen the first three; we will focus on the route choice.

Route choice, also called traffic assignment and traffic swapping, determines the division of traffic to alternative routes for every origin-destination pair, by entry time to the network. The final aim of the route choice is to attain user equilibrium, commonly called Dynamic User Equilibrium (DUE), stating that the cost of any route being used is not higher than the travel cost of any alternative route. In the dynamic case this condition must hold for all times. Analogously, no user can improve his situation (in terms of cost) by transferring to an alternate route, under the assumption that the users are rational and will chose the shortest route to their destination.

The general formulation of the DUE requires consideration of continuous time t. However, we are interested in practical applications, which require discretization of time into intervals of the form , that may be denoted .

The cost or travel time is designated by . The instantaneous route demand (veh/h) at a time t at the entry of route r is denoted by The total instantaneous demand assigned to any alternative route from origin p to destination q at time t is denoted by hence:

where is the set of alternative routes between origin p and destination q. The general DUE condition in continuous time is:

where indicates the minimal travel cost from origin p to destination q a time t. That is to say, at every point in time the travel cost for each route in use , is equal to the minimal cost of the origin-destination pair, and not higher than the travel cost of any alternative route.

In the discrete version we assume that the demands are constant within each interval, so for all and similarly for all . Hence,

Given that travel times are not really constant a key dilemma in the formulation of the DUE is which travel time should be associated with an interval of departure times. The most computationally stable option has been proven to associate with each interval the travel time at the end of the interval, i.e., .

In addition, route choice is often modelled as being the product of user experience. User experience is limited by the users perception of the traffic situation. In order to model the sensitivity of the user to traffic changes, the route choice algorithm may consider the users impervious to changes to the traffic cost for a period of time , also called Route-Time-Interval or RTI This leads us to an alternative formulation of the DUE condition:

The excess cost of a single route, denoted by , is defined as the difference between the travel cost on route r and the minimal travel cost from origin p to destination q. So that,

Total excess cost of traffic travelling on the route r denoted by is

In perfect equilibrium the TEC of all routes should be zero. Convergence is measured by the Global Excess Cost (GEC), which is the summation of the total excess travel costs for all departure times, in all the alternative routes and for all origin- destination pairs.

2.4.2.2 Route Choice Algorithms

The route choice algorithms we are concerned with follow an iterative process in which the assigned demand is updated between iterations according to a correction factor. . One can describe this as follows:

**,**

where F is a correction function suggesting a change to the demand flow matrix containing the demands for all routes and all departure intervals. The suggested change is dependent on the demand flow matrix of the previous iteration. The solution for iteration j is the weighted average of the proposed solution **,** in iteration j and of the solution attained in iteration j-1 , where represents the step size in iteration j.

Analogously, one can directly consider the correction function that expresses the proposed change matrix in iteration j. This correction matches the difference between the proposed solution in iteration j and the actual solution in iteration j-1.

=

In traffic dynamics, applying a change to a demand assignment for a route may affect, not only traffic departing at later times on the same route, but also already departed traffic on different routes. As a consequence, attaining the DUE requires the iterative process to compute a full DNL solution per iteration.

Notice that as a requisite to the usage of the DNL lays the need to specify the paths in a network. For small networks all possible paths may be enumerated a-priori, but with bigger networks and more realistic scenarios this becomes impractical very fast. Henceforth a pathfinder component is required.

Path finding is very costly computationally and as such is often applied incrementally. When computing inputs to the DNL for the first time, potential shortest routes are computed by the pathfinder based on costs a priori (assuming no congestion). Once a DNL solution has been obtained the new shortest routes according to the new costs are computed (and the GEC is calculated), if these differ from the existing ones they are added to the pool of potential paths.

The route choice algorithm uses the complete pool of paths in order to assign the demand to different routes with the DUE as an objective.

Given comparable DNL models and pathfinders, the route choice algorithm is the component that influences the most the speed of convergence of the DTA. Finding an efficient route choice algorithm is therefore a key challenge. We will now look at two of the existing simple methods.

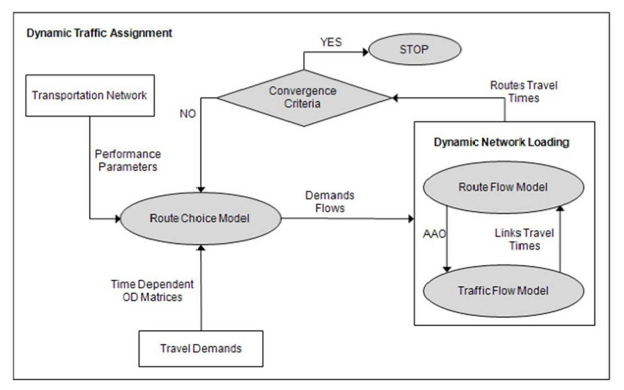


Figure 5 DTA, the solution algorithm to the DUE (from Rahamin et al [4])

2.4.2.3 MSA

MSA, or method of successive approximations, is a very popular, numerical and iterative, optimisation method. Applied to the problem of the DUE it consists on considering per DTA iteration an All-Or-Nothing assignment to minimum cost route as the correction function such that:

= **,**

where is the chosen minimum cost route.

Typically the step decreases such that **.** Under this condition MSA is guaranteed to converge.

2.4.2.4 DEC

DEC for Dynamic Excess Cost was first proposed by Rahamin et al. [4]. DEC is a heuristic that aims at stabilizing the system by taking advantage of the fact that allocation of traffic flows are dependent on change carried out in earlier times. This is achieved in practice, within the scope of a single DTA iteration, by limiting the amplitude of changes done to later departure times to, the excess cost, minus the cumulative of changes from earlier times:

The proposed change is initialized with =0.

In addition, DEC prevents feasibility violations in which the correction to a traffic flow could produce the appearance of negative traffic flows; and the appearance of opposite flow shifts (from lower cost route to higher cost route).

The update rule used in practice for DEC is:

Note that DEC, however, does not impose a particular constant. Typically the constant is fixed. Unfortunately very little explanations on how to chose were given in the original article cf. [4].

In this thesis the best for DEC was empirically found. A DTA for network 6 (see section 4.7.4) based on [4] was run with DEC as a heuristic for : 0.001, 0.1, 1, 10 and 100 (when working with relative travel demands, hence 1 means, the correction is of the order of the travel demand itself).

The fastest convergence was attained for of 1. We do not focus further on this aspect, unfortunately there is not enough theoretical ground on the matter and it is not the primary focus of this thesis. It is in fact a weak point of DEC, one that could require an entire separate piece of research to analyse.

2.5 Conclusion

Amongst the possible angles transportation systems can be studied from, we are only concerned with the simulation of simple traffic networks. A simulation is considered to reflect a realistic traffic situation if the Dynamic User Equilibrium (DUE) has been attained. In practice we may attain the DUE through a Dynamic Traffic Assignment (DTA) process. A demand model, a supply model, a Dynamic Network Loading (DNL) model, and a route choice/assignment model are all components of the DTA. The DTA is an iterative process that successively run the DNL for a pre-defined demand and supply, then computes the distance to the DUE and adjusts the route assignment to, once again, run the DNL. Route assignment heuristics are central to this thesis. We have seen two: MSA and DEC. In the next chapter we will introduce genetic programming, which may prove useful into searching automatically for new heuristics.

Chapter3: Genetic Programming

A central idea of genetic algorithms is that they are not based on the traditional correct, consistent, justifiable, certain, orderly, parsimonious, and decisive principles of the vast majority of techniques in machine learning. Rather than seeking an exact mathematical reasoning that leads us to the solution, genetic algorithms are based on entirely new principles, namely on applying the same kind of uncertain and non-deterministic evolutionary process observed in nature, where diversity is often wanted and therefore inconsistency and contradiction are considered of significant value.

Genetic programming (GP) is in fact a type of genetic algorithm (GA), which focuses on the evolution of computer programs. In the subsequent sections we will have a look at the basic of what a genetic algorithm is, then, we will re-interpret the genetic algorithm paradigm under the light of genetic programming, and finally we will consider some of the details and practicalities of genetic programming itself.

3.1 Genetic Algorithms

In nature, given a population of self-reproducing entities, diversity amongst them, and a connection between the entities diversity and their survival ability, we observe a tendency for the fittest individuals to reproduce more frequently, as well as a higher rate of survival of their genetic information. Through time this process reshapes the structure of the population towards a population dominated by fitter individuals. We say that the population has then evolved.

John Holland proposed in his book “Adaptation in Natural and Artificial Systems” to extend to artificial systems this sort of evolutionary process ruled by natural selection. The kind of algorithms that solve adaptation problems formulated in genetic terms is what we call a “genetic algorithm”.

A genetic algorithm is typically a highly parallel algorithm, which will transform a population of mathematical objects that we call individuals, each with an associated fitness value, into a new population using operations based on the principles of survival of the fitness and sexual reproduction.

Briefly put and in plain terms a typical genetic algorithm proceeds as follows:

It begins with an effort to learn something about the environment by testing a number of M randomly chosen points in the search space, equivalent to generating M individuals randomly. This is generation 0.

Per generation the fitness of each individual is assessed against the environment. Depending on the fitness of individuals and a probabilistic parameter fixed a-priori pr, a pool of individuals is created by reproducing (copying) individuals based on a criterion that favours the fittest ones. Individuals in this pool are then recombined into new individuals (mixing their parts to produce 2 individuals), this operation, which also favours fittest individuals and uses a new parameter pc, is called crossover and it allows exploring new points in the search space by replacing parents with their fitter offspring. An additional operation of mutation may uniformly computed per individual with parameter pm.

The iterative repetition of the previous steps has the potential to improve the average fitness of the population. The best individual encountered, often labelled best-so-far is our solution. Once the optimal solution has been reached or a maximum number of generations has been reached we have attained the termination criterion and the evolutionary process can be stopped.

More formally:

1) The preparation of a genetic algorithm requires:

* The selection of a representation scheme
* A fitness measure
* The parameters and variables controlling the algorithm
* A way of designating the result
* A termination criterion

The representation scheme is a mapping that allows expressing each possible point in the search space. A representation that facilitates the search for a solution by means of the genetic algorithm requires significant insight into the problem domain, for example, in the travelling salesman problem, or TSP, a possible solution or path may be represented by enumeration of all nodes: 1-3-4-6-5.

The fitness measure must ensure the evaluation of all possible individuals. It is often inherent to the problem, for example, in the TSP the fitness may be the length of the path.

The primary parameters for controlling a genetic algorithm are the population size M and the maximum number of generations to be run, G. Secondary parameters are pr, pc, and pm. Other control variables are often required depending on the specifics of the GA.

The most common way of designating the result is obviously to take the best individual so far.

A run may be terminated according to different criteria. The most frequent criteria are encountering the best possible result and reaching the maximum number of generations.

2) Once the preparation is complete the genetic algorithm can be run as follows:

* Randomly crate an initial population of individuals.
* Iteratively perform the following substeps on the population until the termination criterion has been satisfied

1. Evaluate the fitness of each individual in the population
2. Create a new population by applying at least the first two of the following operations, on individuals chose with a probability based on the fitness:
3. Copy existing individuals to the new population
4. Create two new individuals by genetically recombining two existing individuals
5. Create a new individual by randomly mutating an existing individual

* The best-so-far individual is then designated as the result of the genetic algorithm for the run.

This process is illustrated in the figure below..

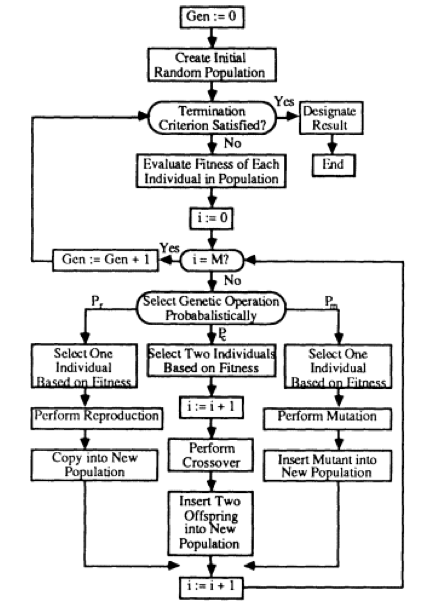
**

Figure 6 Flowchart of the conventional genetic algorithm (from Koza [3])

There are numerous variations on the basic genetic algorithm but this is the most frequent setting. It is important to understand that domain specific knowledge may help substantially into taking the best choices during the preparation step, which will in turn improve the performance of the genetic algorithm. However, the main point is that the genetic algorithm is a domain-independent approach to rapidly searching an unknown search space for good solutions.

For further explanations, particularly regarding why genetic algorithms work, the reader is referred to Koza’s book on Genetic Programming ([3]).

3.2 Overview of Genetic Programming

Genetic programming arises from the observation that numerous problems in machine learning, artificial intelligence and symbolic processing can be interpreted as requiring the discovery of a computer program that provides a certain output given a certain input.

The discovery of such a computer program can be reinterpreted as the search for the fittest computer program in a space of computer programs. As we have seen such a search can be achieved through the principles of survival of the fittest, reproduction and crossover.

The leap between basic genetic algorithms and genetic programming lies at the representation level. The type of structure undergoing adaptation is now a hierarchy of functions and terminals of varying size and shape. In addition, such computer programs are now “active” in the sense that they are capable of being executed.

Genetic programming is therefore a domain-independent method that provides a single unified approach to the problem of finding a computer program to solve a problem.

The execution of a genetic algorithm under the genetic programming paradigm becomes:

* Randomly crate an initial population of random compositions of the functions and terminals of the problem (computer programs)
* Iteratively perform the following substeps on the population until the termination criterion has been satisfied:

1. Execute each program in the population and assign it a fitness value according to how well it solves the problem.
2. Create a new population by applying at least the first two of the following operations, on computer programs in the population chosen with a probability based on their fitness:
3. Copy existing programs to the new population
4. Create new computer programs by genetically recombining two existing individuals
5. Create a new individual by randomly mutating an existing individual

* The best-so-far computer program is then designated as the result of the genetic algorithm for the run.

3.3 Details of Genetic Programming

As in Koza’s book we will describe in more detail genetic programming focusing on John Holland’s key features common to all adaptive systems.

These key features are:

* Structures that undergo adaptation
* Initial structures
* Fitness measure
* Operations that modify the structures
* State of the system at each stage
* Termination method
* Method to designate the result
* Parameters that control de process

The following explanations cover all that is needed in order to understand this thesis, not more.

**Structures that undergo adaptation**:

As already introduced the structures that undergo adaptation are computer programs. More precisely hierarchical structures made of functions and terminals.

The functions may be:

* Arithmetic operations, e.g. +,-,\*
* Mathematical functions, e.g. sin, exp
* Boolean operations, e.g. and, not
* Conditional operators, e.g. if, then
* Functions causing iteration, e.g. loop
* Functions causing recursion and
* Domain-specific functions that may be defined

The terminals are atomic entities, i.e., variables, constants or functions taking no arguments.

A common representation of these structures is the s-expression used in the LISP language, for example (OR (AND (NOT d0) (NOT d1) ) (AND d0 d1)), which has a corresponding tree representation.

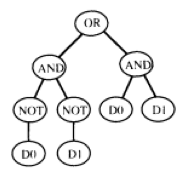
**

Figure 7 Example of tree for S-expression

**Initial structures:**

The initial structure in genetic programming consists of the individuals in the initial population of individual computer programs for the problem.

The generation of each initial computer program is done by randomly generating a tree structure made of functions and terminals.

In practice there are three dominant techniques to do this: full method, grow method and the ramped half-and-half.

The full method involves generating trees with a fixed length, in which all branches from root to the endpoint have same length.

On the other hand, the grow method produces trees that are variably shaped, but not deeper than a maximum length.

Ramped half-and-half is a method in which the population is split in two parts, often in half but not necessarily. The full method is then applied to one of the parts, whereas the grow method is applied to the other part.

**Fitness measure:**

The most common approach is to define an explicit measure of fitness for each individual. This is accomplished via an explicit well-defined evaluation procedure producing a scalar fitness value as output.

We are interested in 3 interpretations of the fitness value:

* Raw fitness, r: the fitness as normally measured in the problem domain.
* Standardized fitness, s: raw fitness manipulated so as to ensure that a lower fitness coincides with a better individual. In particular if the raw fitness increases for worse individuals s = r..
* Adjusted fitness, a: Defined as . The adjusted fitness lies between 0 and 1. Its is bigger for better individuals in the population.

**Operations that modify the structures:**

We already know that the primary operations in genetic programming are reproduction and crossover.

Reproduction is the basic engine of survival of the fittest. It consists of two steps. First and individual is selected. Second, the individual is copied, without alteration, from the current population into the new population (i.e., the new generation).

A popular method of selection is tournament selection, in which a specified group of individuals are chosen at random from the current population and the one with the better fitness is then selected. Note that tournament selection is a selection strategy with replacement; therefore the same individual may be selected to become part of the next generation several times. Tournament selection has the particularity of adding additional control to the selection pressure by increasing or decreasing the size of the group of individuals chosen at random.

The crossover starts from two parental structures and produces an offspring structure. Typically both parents are chosen with the same selection method used in the reproduction step. Subsequently, a point in the structure of the both parents is selected with a uniform probability. The crossover fragment for a particular parent is the rooted subtree which has as its root the crossover point for that parent and which consists of the entire subtree lying below the crossover point.

The first offspring is then produced by deleting the crossover fragment of the second parent at the crossover point of the first parent. The second offspring is produced in a symmetric manner. .

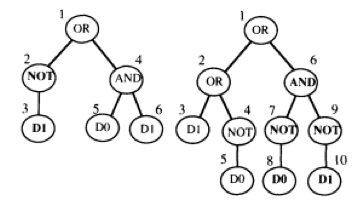
**

Figure 8 Two parental computer programs (from Koza [3])

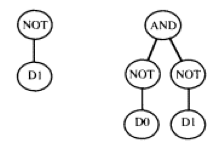
**

Figure 9 Crossover fragments resulting from selection of point 2 of the first parent and point 6 of the second parent (from Koza [3])

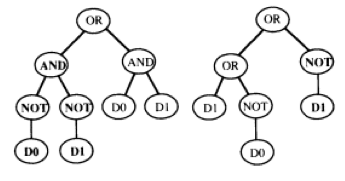
**

Figure 10 Resulting offspring (from Koza [3))

Mutation may be applied in addition to crossover and reproduction. However, it is considered a secondary operation because it does not substantially contribute to the evolutionary process. Its sole purpose is to potentially reintroduce diversity in a population that may be converging prematurely.

In mutation an individual is chosen randomly with the same or a similar selection method as in reproduction, then, a point in the structure of the individual is selected with random probability and the subtree rooted at that point is then replaced by a newly generated subtree. The depth of the inserted subtree is limited to a maximum parameter. This type of mutation is called one point mutation.

State of the system at each stage:

The current population gives the state of the system at any point in time. Control parameters (which may be dynamic in complex implementations) and best-so-far individual are commonly kept in memory too.

**Termination Method:**

As we have already discussed the most common termination criterion is to attain the maximum number of generations or to encounter a result that matches a given condition. Other criteria such as the loss of diversity (stagnation) may be part of the termination criteria.

**Result:**

Again, as already discussed the designated result is typically the best-so-far. An alternative is to focus on the best individual in the last generation. However, there is no guarantee that the best individual in the last generation is the overall best encountered result.

**Parameters that control the process:**

Values for the following control parameters must be defined in order to control the evolutionary process:

* Size of the population M.
* Maximum number of generations G
* Probability of crossover pc
* Probability of reproduction pr
* Probability of mutation pm
* Maximum depth for offspring after crossover
* Maximum depth for individuals in the initial population

In addition, in qualitative terms, a clear definition of the selection method must be provided, the type of fitness used has to be explicitly set, and the generative method for the initial population must also be clearly defined.

The optimal values for the control parameters are very hard to obtain. In fact, finding the optimal control parameters is in itself an optimisation problem. Hence, it is often empirically though experience that an approximation of the best values is inferred.

3.4 Conclusion

A genetic algorithm is a domain independent technique based on the principles of natural selection and sexual reproduction designed to find solutions to problems in artificial systems. Given the well-known execution flow and characteristics of a genetic algorithm, it is sufficient to give one conceptual step further to propose the search for computer programs as a powerful generic method to solve problems arriving to genetic programming. We have gone through the theoretical fundamentals of genetic programming; it is now the time to move from theory to practice and apply genetic programming to our problem of interest.

Chapter4: Experiments

In this chapter we will go through the experiments that are the heart of the research undertaken in this thesis.

4.1 Objectives and Overview

Our objectives are, to assess the adequacy of using genetic programming regarding route assignment heuristic generation, and to attempt to improve on existing manual heuristics.

The proposed method to achieve the target research objectives is to evolve a heuristic using genetic programming, then compare against MSA and DEC.

For this we devise two experiments: experiment one evolves a heuristic using a training set of traffic networks; the result of which is then tested against MSA and DEC using a test set of traffic networks under different assumptions. Experiment two is very similar. Again, a heuristic is evolved for a training set of traffic networks; this training set is in fact the original from experiment one, but modified taking into consideration the results of experiment one; the test set is also different but we measure once again against MSA and DEC under different assumptions.

In both cases, the main metric to consider when evaluating and comparing heuristics is the speed of convergence towards DUE.

4.2 Tools

The following software tools are at our disposal:

1. A port to Java of a traffic simulator provided by Willem Himpe from KU Leuven. The object oriented re-interpretation of the original code in Matlab and its porting to Java was a substantial time consuming task in this thesis. The software development aspect however, is not the focus of this work; the reader is referred to the source code cf. Appendix A. The simulator provides the capability to run a full DTA process, which embeds the LTM as DNL component, and Dynamic Dijkstra as pathfinder. The simulator has also de capability to receive and run a user defined route assignment.
2. ECJ by Sean Luc et al. from Goerge Mason University, and a wrapper provided by Rinde Van Lon from KU Leuven. ECJ is an evolutionary computation system. ECJ is a popular java framework for evolutionary computing and genetic programming in particular. Part of its popularity is due to its simplicity of use. ECJ, on the other hand, often requires the user to create prodigious amounts of separate files. The wrapper allows us to control directly with java code ECJ reducing the amount of extra files. In addition the wrapper allows the distribution of ECJ’s computations by integrating with the open-source java-based parallel computing framework JPPF.
3. Finally, a Java application integrating the previous tools into one; developed specifically for this thesis, this application contains all the logic and control aspects behind the definition and execution of the experiments.

In terms of hardware, all experiments are run on a computer with an AMD Athlon 64 Dual Core Processor 4600+ with 4GB of RAM.

ECJ was always run with 2 threads for breeding and evaluation.

4.3 Control over the Evolved Heuristic

Genetic Programming is a domain-independent technique. However, in practice, if one wants to obtain valid programs in a realistic time frame, one has to reduce the search space and direct the search in such a way that mostly valid programs are visited.

In our case we direct the search by aiming at route assignment heuristics that are similar to DEC. More precisely, and following the notation introduced at the end of chapter 2 we aim for heuristics with an update rule of the form:

Where is the -a-priori- proposed demand change to routes that are not optimal/shortest for and OD pair, and time interval. Only is in fact provided by the program that we evolve: As in DEC, we allow the –a-priori- proposed correction, in this case, to surpass the maximum and minimum demand for an OD pair on a route, but we explicitly limit the final change to the route to the maximum and minimum possible demand for that OD pair on the route.

This update rule is also convenient because the updates to the demand on the shortest routes in based on the updates to all non-shortest routes. This simple “normalization” allows the genetic programming to focus on changes to the non-optimal routes.

In addition we reduce the search space by constraining the heuristics to those formed of a reduced set functions and terminals.

**Set of Functions:**

* MUL, Multiplication: receives two scalars, returns their product.
* DIV, Division: receives two scalars, returns the first one divided by the second one.
* SUB, subtraction: receives two scalars, returns the first one minus the second one.
* ADD, addition: receives two scalars, returns their addition.
* POW, exponentiation: receives two scalars, returns the first one to the power of the second one.
* IF4: receives 4 scalars, if the first value is greater than the second the third value is returned, otherwise the second.

**Set of Terminals:**

Constants: 0 and 1.

Variables (all variables are computed per time instant , OD pair pq, and route r:

* **INV\_ITER**: the inverse of the current iteration number. We use the inverse because we work with normalised demands; we are therefore interested in operating with values between 0 and 1.
* **CDEMAND**: Normalised demand for that route, OD pair and time, as defined in the previous iteration.
* **ODEMAND**: Normalised demand for the optimal route, OD pair and time, as defined in the previous iteration.
* **NORMCOST**: Normalised cost difference between the current route and the optimal route. Following the notation in chapter 2:

Again we normalise the cost difference to keep values between 0 and 1. Manual heuristics that make use of normalised cost have been proposed in the past as seen in [5]

* **CUMUDELTA**: The cumulative of deltas computed for previous time instants.

4.4 Additional Control and Speed-Ups

First and foremost, we want to produce valid individuals as frequently as possible. In order to accomplish this, we add a double staged manipulation that increases the validity of the evolved programs; first, we compute all deltas proposed in the first iteration for all OD pairs, routes and time instants; Subsequently, we look at the prevailing sign for the deltas. The deltas are expected to be mostly positive. If not the case, all deltas are multiplied by -1 from the first iteration onwards. Secondly, if any of those initial deltas surpasses the maximum possible demand we divide all deltas by that value from the first iteration onwards.

Despite the previous, it is still very probable to generate invalid heuristics that we do not want to spend time evaluating. A set of additional control mechanisms has been devised to detect such undesired heuristics:

1. Zero Delta: If the deltas are all zero in the first iteration, the DTA is aborted.
2. Increasing GEC: if the GEC augments for more than 10 iterations the DTA is aborted.
3. GEC Stalled: if the GEC does not change for more than 10 iterations the DTA is aborted.
4. Average GEC Stalled: If the GEP oscillates, but results 2 times, in exactly the same GEC average computed over 5 iterations of the DTA, the DTA is aborted.

4.5 Fitness Evaluation

Fitness evaluation is a crucial aspect in the evolutionary process; it is an explicit formal way of measuring how good the heuristic is in accordance to pre-defined criteria. The way we evaluate the fitness, that is, the criteria, invariably biases the search towards a certain type of heuristic. Therefore, we must first carefully consider the positive aspects that make a good assignment heuristic.

A good assignment heuristic should proceed fast (in the least number of DTA iterations possible) towards the DUE, independently of the setting; this implies that for traffic networks with diverse complexity, diverse size and under the influence of travel demand of different scales, the assignment heuristic should converge quickly towards a GEC of 0.

The best possible experiment would be to measure the speed of a heuristic for multiple networks (e.g., 100+) and ever changing settings. In our case we are limited on one side by the availability of the networks, which we have to design manually due to the lack of an open pool of networks directly usable, and on the other side, by the computational resources at hand: it is utmost important that the genetic programming process can be executed for a considerable amount of individuals and generations in a reasonable amount of time. The speed of the simulator becomes then the limiting factor. Unfortunately the speed of the simulator degrades exponentially with the duration of and the number of nodes in the network.

The compromise we reach is to use a carefully crafted set of simple traffic networks and travel demands. These networks and demands conform the training set. Because they are different for each experiment the different training sets are detailed in sections [Ref to sections]. This is also the case for the parameters of the DTA process.

Finally, the fitness evaluation of an assignment heuristic, i.e., the individual, consists on running one simulation (DTA process) at a time, for a network, if the simulation is aborted the final fitness of the individual is automatically set to the maximum raw fitness value; otherwise a partial fitness is computed. Next, the process starts again with the next network. If all networks have been evaluated the final raw fitness for the individual is computed as a weighted average of all the partial fitness.

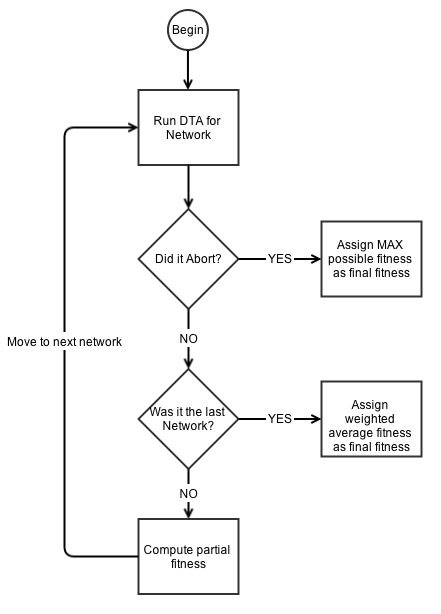


Figure 11 Evaluation of Individual

The partial fitness is a relative measure of the distance between the best possible progression towards equilibrium, in which GEC 0 is attained after 2 iterations, and the measured progression:

Where GECinit designates the initial GEC and GECfinal designates the final GEC after convergence or after the maximum number of iterations has been reached.

We normalise by the best possible progression so that the partial fitness is independent of the initial gap, GECinit, and the size and complexity of the network (often affecting the of number of iterations needed to achieve equilibrium).

4.6 Genetic Programming Parameters

In both experiments we use tournament selection with a tournament size of 7, and the following control parameters:

* Size of the population M = 500
* Maximum number of generations G = 100
* Probability of crossover pc = 0.90
* Probability of reproduction pr = 1 (once selected the individual becomes part of the reproduction pool)
* Probability of mutation pm = 0.10
* Maximum depth for offspring after crossover = 17
* Maximum depth for individuals in the initial population = 17

Originally an experiment with 50 individuals and 1000 generations was attempted with all other parameters the same. This experiment suffered from early convergence (no improvement over the best result after the second generations) and very low fitness. Upon studying the characteristics of the best individual (see Appendix B) it became clear that the initial generation did not have enough diversity. The solution was then to move to 500 individuals and 100 generations.

4.7 Experiment 1

4.7.1 Training Set

We use five different networks (figure 12), all networks are based on default link values:

* Length L=1 km
* Capacity C= 1000 veh/h
* Freespeed S= 120 km/h
* Kjam = 120

Whenever a parameter of a link is different from the default value, it has been indicated in the diagrams, except for Kjam. The Kjam has been adjusted so that whenever there is a non-default capacity the spillback speed W is 18 km/h.

The demand for an OD pair is fixed to 1000veh/h between time instant 0.005 and 0.70.

The exact demand flows per OD pair and network are listed below, using the notation origin\_node – destination\_node.

* Net1: 1-6
* Net2: 1-6
* Net3: 1-7
* Net4: 1-7 2-8
* Net5: 1-10 2-10

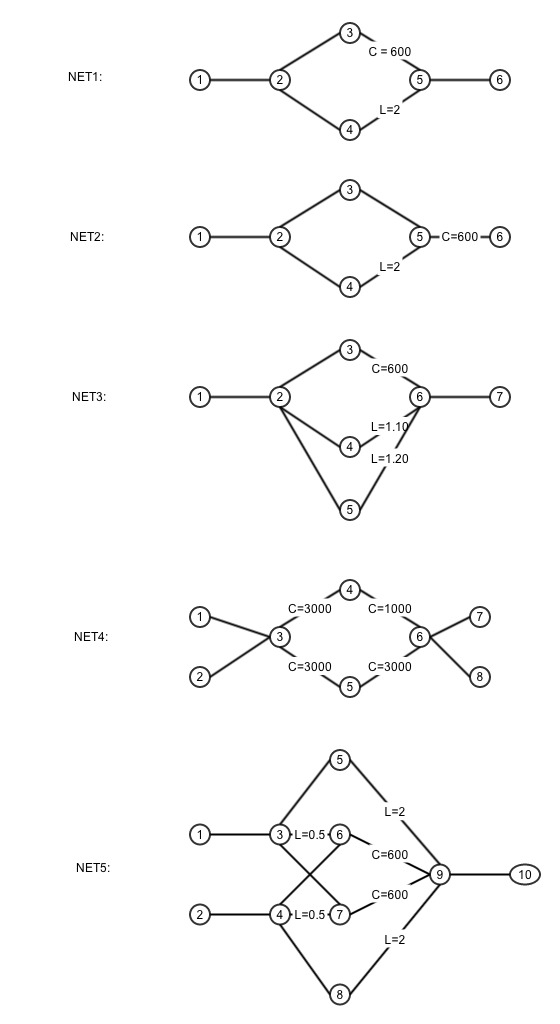


Figure 12 Training Networks

Network 1 is a simple network with 2 possible paths, where the shortest route in length 1-2-3-5-6 becomes congested. The heuristic is then forced to transfer part of the traffic flow to the secondary path 1-2-4-5-6 in order to reach the optimal assignment.

Network 2 follows the same structure of network 1, with the exception that the congestion is provoked at the end of both routes, in the link 5-6. This congestion creates queues and once again the heuristic is forced to transfer traffic from 1-2-3-5-6 to 1-2-4-5-6. Together with network 1, these networks conform a basic assignment scenario that a good heuristic shall be capable of solving easily.

Network 3 is a slightly more sophisticated network in which we extend the scenario of network 1 to 3 paths.

Network 4, introduces a new traffic flow. The segments 3-4-6 and 3-5-6 can now be used by both flows, and the assignment heuristic is forced to deal with situations in in which the traffic swapping has to take into account the interaction of flows.

Network 5, is a more sophisticated case in which there are multiple route choices for two traffic flows. Once again, the interactions between flows are a major factor, with the added complexity of having 3 possible paths per flow. The method used to produce congestion is to limit the capacity at the exit link 9-10, as in network 3.

4.7.2 Final Fitness

All networks are equally weighted therefore:

Where means, fitness measured for network X.

4.7.3 DTA Control Parameters

All networks are simulated for a DTA process with the following fixed set of parameters:

* Discretization Time Interval: 0.0025h
* Route Interval: 75 time intervals
* Maximum number of DTA iterations: 300 (this limit is essentially imposed to limit the simulation time to a maximum)
* Termination GEC: 0.002 of the initial gap.

This combination of parameters is essentially an attempt at keeping the evolutionary process short while giving the time to the heuristic to proceed sufficiently towards the equilibrium as to provide a meaningful fitness.

4.7.4 Test Set

The test-set of networks is formed by two networks, network 6 and 7 that we use in different test cases.

Network 6 is exactly the network used in the original Rahamin et al. article where DEC was introduced [ref]. As such we reproduce hereby the original description and the original diagram.

“The squares A–F represent interchanges, which are expanded as illustrated for B and E. The net- work presents a loop road, typical to many cities. It consists of 30 road links and 18 potential routes. The values shown in the figure along the links represent their lengths in km. The network includes 9 merge nodes and 9 diverging nodes.”

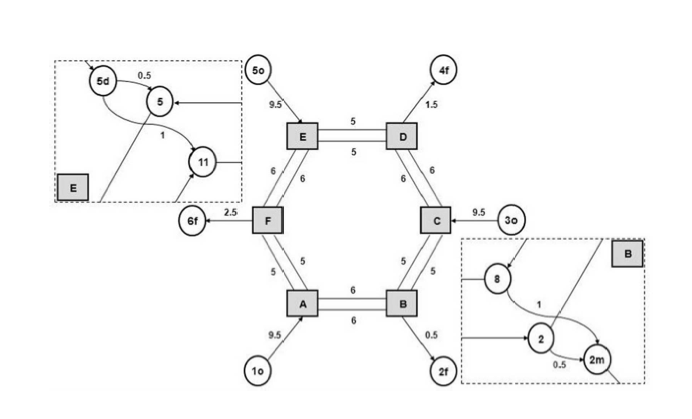
**

Figure 13 Network 6 (First appeared in Rahamin et al. [7])

No information regarding freespeed and kjam was provided in the original article. The defaults of 120km/h for the freespeed and 120 for the kjam are applicable.

Network 7 on the other hand is a model of the city of Rotterdam. Due to the amount of space it would take to reproduce all link attributes in the model of Rotterdam, (this network has 293 nodes and 476 links) such information is not included in this document. The model and the details of the model can be downloaded nevertheless from the Github link hosting the source code of the simulator [appendix ref].

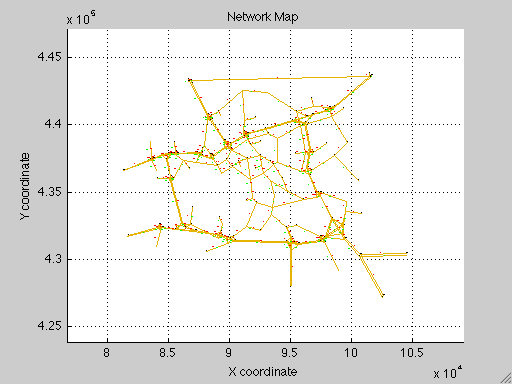
**

Figure 14 Map of Rotterdam (Network 7)

4.7.5 Test Cases / Comparisons

All cases are run for route traffic choice time intervals of 50, 75 and 100, i.e., 0.125h, 0.1875h and 0.250h respectively.

Cases 1 to 3 are based on network 7. The final set of demand flows in case 3 matches the original travel demand flows used in Rahamin et al. Case 4 is a case crafted exclusively for this thesis with the aim of testing the assignment heuristics against a more realistic network structure and different scales of travel demand flows.

Case 1:

We compare the evolved heuristic against DEC and MSA for network 6 with a travel demand of 1000veh/h, between nodes 1 and 4, and time instants 0.5 to 3. The time step remains 0.0025h

Case 2:

We add to case 1 a demand flow of 1000veh/h, between nodes 3 and 6, and time instants 0.5 to 3. The time step remains 0.0025h.

Case 3:

We add to case 2 a demand flow of 3000veh/h, between nodes 5 and 2, and time instants 0.5 to 3. The time step remains 0.0025h.

Case 4:

We compare the evolved heuristic against DEC and MSA for network 7. Due to the amount of possible paths and the fact that the simulator is designed to compute fast, not to save memory, we are constrained to short and simple simulations. Hence, we limit ourselves to the given travel demand flows between time instants 0.005 and 0.50. All flows are defined towards node 1 and 2.

|  |  |  |
| --- | --- | --- |
| Origin Node | Demand Flow From Origin  to node 1 | Demand Flow From Origin  to node 2 |
| 1 | 0 | 4,45120000000000 |
| 2 | 3,36240000000000 | 0 |
| 3 | 3,48000000000000 | 66,7360000000000 |
| 4 | 4,74879999999999 | 82,1672000000001 |
| 5 | 0 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 4,94399999999999 | 62,5808000000000 |
| 9 | 265,774960000000 | 77,2156000000000 |
| 10 | 0 | 0 |
| 11 | 110,868000000000 | 10,0464000000000 |
| 12 | 31,9992000000000 | 4,65360000000000 |
| 13 | 1,85120000000000 | 23,9928000000000 |
| 14 | 0 | 0 |
| 15 | 0 | 0 |
| 16 | 0 | 0 |
| 17 | 0 | 0 |
| 18 | 0 | 0 |
| 19 | 4,97120000000000 | 2,25440000000000 |
| 20 | 0 | 0 |
| 21 | 0 | 0 |
| 22 | 4,42640000000000 | 12,5744000000000 |
| 23 | 44,5056000000000 | 4,88639999999999 |
| 24 | 26,3056000000000 | 5,59839999999999 |
| 25 | 0 | 0 |
| 26 | 0 | 0 |
| 27 | 0 | 0 |
| 28 | 4,15840000000000 | 22,3304000000000 |
| 29 | 0 | 0 |
| 30 | 0 | 0 |
| 31 | 12,0632000000000 | 2,15440000000000 |
| 32 | 0 | 0 |
| 33 | 0 | 0 |
| 34 | 6,24960000000000 | 3,90159999999999 |
| 35 | 0 | 0 |
| 36 | 1,90640000000000 | 1,12000000000000 |
| 37 | 0 | 0 |
| 38 | 42,1480000000000 | 6,16800000000000 |
| 39 | 19,8720000000000 | 8,35199999999998 |
| 40 | 4,02720000000000 | 0,597600000000000 |
| 41 | 0 | 0 |
| 42 | 0 | 0 |
| 43 | 0 | 0 |
| 44 | 1,75760000000000 | 1,44400000000000 |
| 45 | 0 | 0 |
| 46 | 0 | 0 |
| 47 | 6,51040000000000 | 9,19360000000000 |
| 48 | 0 | 0 |
| 49 | 17,0392000000000 | 3,83039999999999 |
| 50 | 0 | 0 |
| 51 | 0 | 0 |
| 52 | 8,76319999999998 | 0,584000000000000 |
| 53 | 0 | 0 |
| 54 | 0 | 0 |
| 55 | 124,871120000000 | 24,4851200000000 |
| 56 | 0 | 0 |
| 57 | 0 | 0 |
| 58 | 67,8515200000000 | 29,5653600000000 |
| 59 | 0 | 0 |
| 60 | 308,437600000000 | 8,54159999999999 |
| 61 | 18,9448000000000 | 2,03360000000000 |
| 62 | 0 | 0 |
| 63 | 0 | 0 |
| 64 | 15,8496000000000 | 2,33120000000000 |
| 65 | 0 | 0 |
| 66 | 10,8328000000000 | 4,09919999999999 |
| 67 | 0 | 0 |
| 68 | 1,11280000000000 | 3,48959999999999 |
| 69 | 0 | 0 |
| 70 | 18,8808000000000 | 5,13200000000000 |
| 71 | 0 | 0 |
| 72 | 14,9968000000000 | 2,98799999999998 |
| 73 | 2,11040000000000 | 0,392000000000000 |
| 74 | 0 | 0 |
| 75 | 24,2464000000000 | 3,83120000000000 |
| 76 | 0 | 0 |
| 77 | 0 | 0 |
| 78 | 0 | 0 |
| 79 | 0 | 0 |
| 80 | 8,00239999999999 | 16,6744000000000 |
| 81 | 0 | 0 |
| 82 | 6,84239999999999 | 8,07759999999998 |
| 83 | 8,42960000000000 | 0,563200000000000 |
| 84 | 0 | 0 |
| 85 | 0 | 0 |
| 86 | 0 | 0 |
| 87 | 0 | 0 |
| 88 | 0 | 0 |
| 89 | 6,13360000000000 | 1,36800000000000 |
| 90 | 2,25680000000000 | 60,5272000000000 |

Table 1 Demand flows in Case 4

In the following map all origin nodes between node 3 and node 90 are highlighted in blue. Nodes 1 and 2 are highlighted in red.

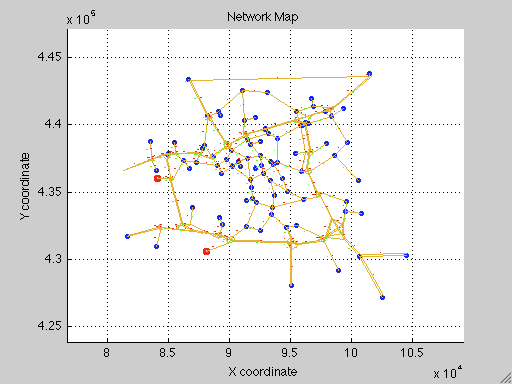
**

Figure 15 Highlighted map of Rotterdam

* + 1. Results

It took 3 days to run the evolution.

4.7.6.1 Fitness Progression

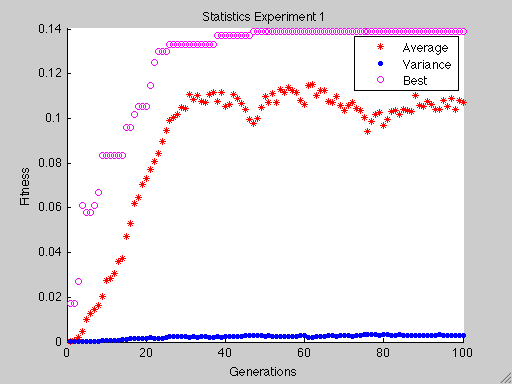
**

Figure 16 Statistics Experiment 1

Fitness in the graph is adjusted fitness.

The best adjusted fitness of experiment one is 0.13882221,i.e., 6.20345829389 raw. fitness.

Between generations 1 and 30 we observe a steady improvement of the fitness. Around generation 30 the average fitness nearly attains its best value. There are no significant improvements afterwards, but oscillations.

We observe too that the best-so-far fitness stalls at generation 49. The little progress achieved afterwards seems to indicate that stopping after 100 generations is an acceptable criterion.

The variance stays small (close to 0.001) but over 0 for the entire evolution. This indicates that certain diversity is maintained during the evolution as required.

4.7.6.2 Training Set

The following figures depict the gap reduction for MSA, DEC and the best individual from the genetic programming that we call GP1, for all training networks.

* All simulations were run for 300 iterations, but the graphs are limited to 100 iterations because all relevant information can be obtained from 100 iterations.
* The termination criterion, or *convergence*, was a reduction of 99.85% of the initial gap, i.e., .
* The vertical axis of the plots is limited to a maximum equal to GECinit and a minimum of 0.
* The plots are continuous to ease the interpretation but the data is not continuous.

**Unless otherwise stated the previous points remain true for all plots in this chapter.**

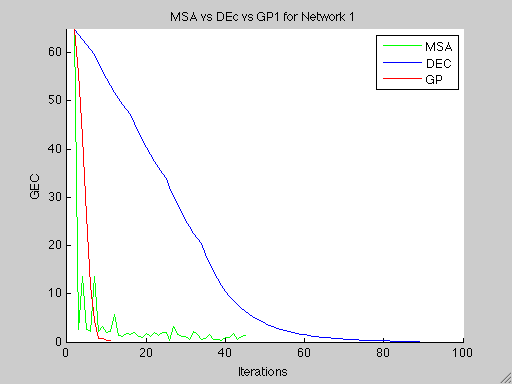


Figure 17 MSA vs DEC vs GP1 for Network 1

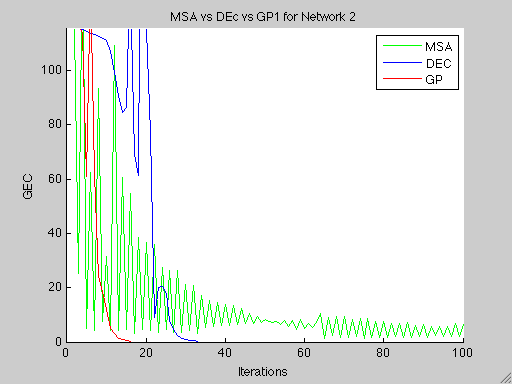


Figure 18 MSA vs DEC vs GP1 for Network 2

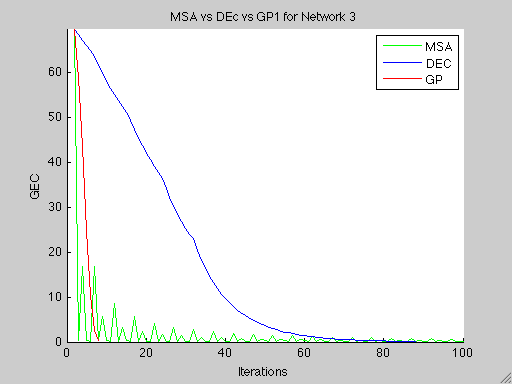


Figure 19 MSA vs DEC vs GP1 for Network 3

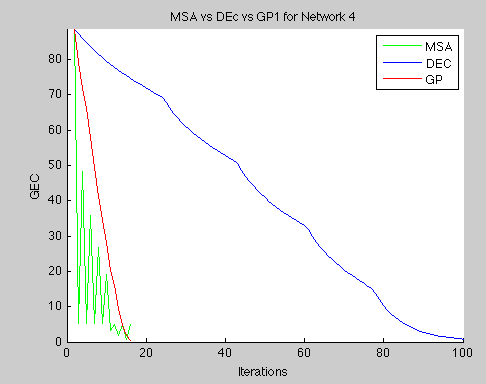


Figure 20 MSA vs DEC vs GP1 for Network 4

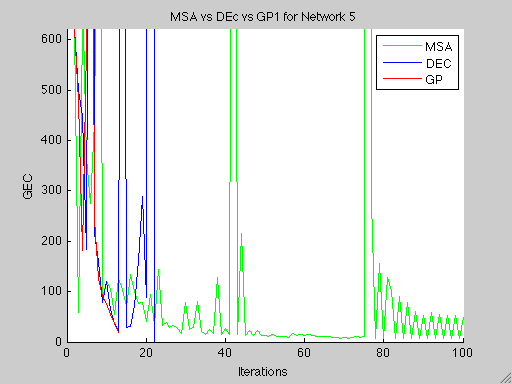


Figure 21 MSA vs DEC vs GP1 for Network 5

The table below summarizes key performance data. In case of convergence, the corresponding cell contains the iteration for which the convergence was achieved, otherwise the best\_gap\_reduction%/iteration.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MSA | DEC | GP1 |
| NET1 | 45 | 90 | 11 |
| NET2 | 99.7%/259 | 34 | 17 |
| NET3 | 149 | 89 | 9 |
| NET4 | 16 | 117 | 17 |
| NET5 | 99.36%/300 | 23 | 13 |

Table 2 Convergence Summary, Training Set, Experiment 1

Observations:

1. GP1 outperforms MSA and GEC for all training networks except network 4, for which it converges one step later than MSA.
2. GP1 displays, not only speed but also, except for a few oscillations in network 2 and 5, a nearly constant rate of descent, this is a characteristic that we were looking for.
3. Interestingly a problem that both MSA and DEC found hard (network 3) which required splitting traffic into 3 separate routes, was easy for GP1.
4. DEC does not consistently perform better than MSA. It does outperform MSA in 3 cases but it does reduce the gap at a smaller rate than MSA. DEC does however, display fewer oscillations.
5. The fact that DEC is not significantly better than MSA requires a separate analysis.

4.7.6.3 Test Cases

Case 1:

Case 1 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

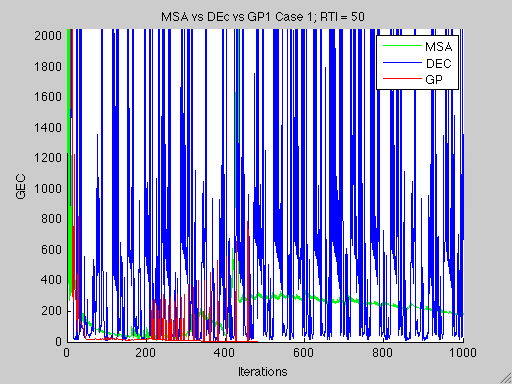


Figure 22 MSA vs DEC vs GP1 Case 1 RTI 50

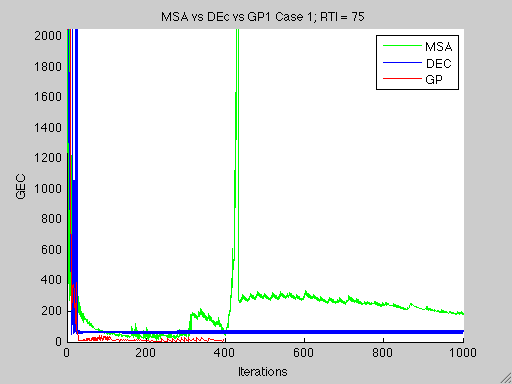


Figure 23 MSA vs DEC vs GP1 Case 1 RTI 75

Case 2:

Case 2 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

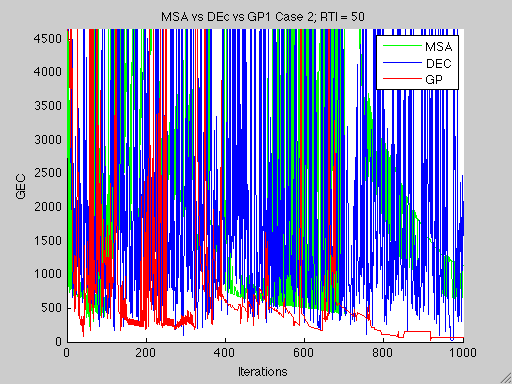


Figure 24 MSA vs DEC vs GP! Case 2 RTI 50

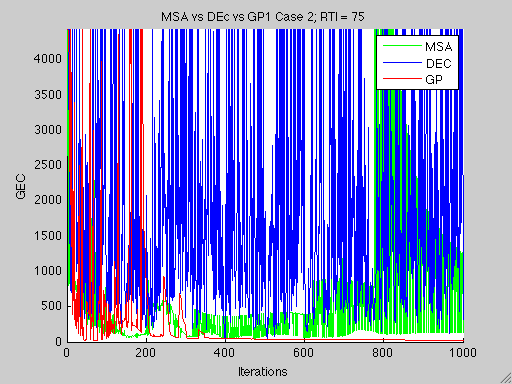


Figure 25 MSA vs DEC vs GP1 Case 2 RTI 75

Case 3:

Case 3 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

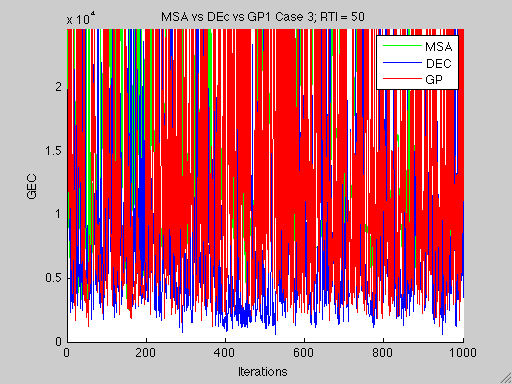


Figure 26 MSA vs DEC vs GP1 Case 3 RTI 50

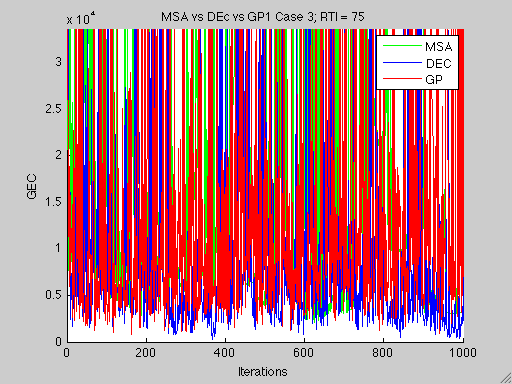


Figure 27 MSA vs DEC vs GP1 Case 3 RTI 75

Case 4:

Case 4 was run for 100 iterations, due to memory limitations, and a RTI of 50 and 75 time intervals.

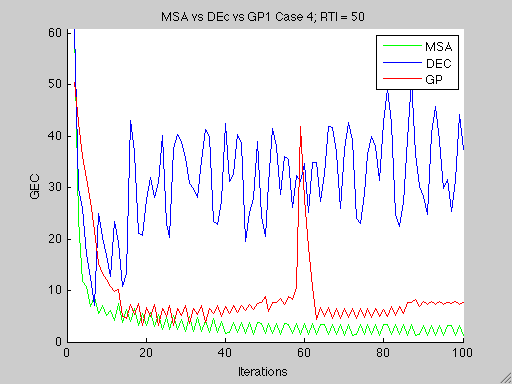


Figure 28 MSA vs DEC vs GP1 Case 4 RTI 50

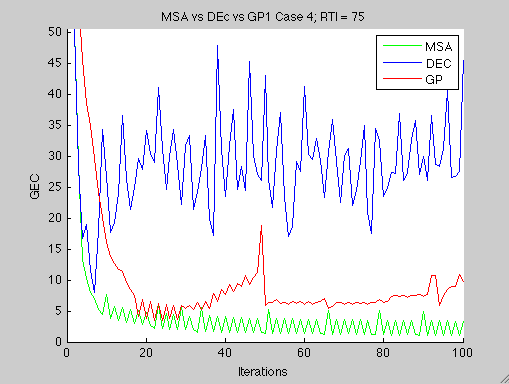


Figure 29 MSA vs DEC vs GP1 Case 4 RTI 75

|  |  |  |  |
| --- | --- | --- | --- |
|  | MSA | DEC | GP1 |
| CASE1, RTI 50 | 768 | 99.57%/95 | 481 |
| CASE1, RTI 75 | 99.39%/226 | 97.60%/10 | 393 |
| CASE2, RTI 50 | 95.07%/55 | 99.56%/971 | 99.38%/918 |
| CASE2, RTI 75 | 98.84%/418 | 99.33%/399 | 99.55%/67 |
| CASE3, RTI 50 | 89.53%/154 | 97.47%/525 | 95.18%/367 |
| CASE3, RTI 75 | 93.70%/623 | 99.33%/366 | 97.46%/163 |
| CASE4, RTI 50 | 98.07%/71 | 88.31%/6 | 94.29%/23 |
| CASE5, RTI 75 | 97.98%/96 | 84.31%/6 | 93.79%/22 |

Table 3 Convergence Summary, Test Cases Experiment 1

Observations:

1. DEC displays interesting behaviour in Case 1. For RTI 50, DEC oscillates during the whole simulation with amplitude of the order of GECinit, it does not manage to converge but reduces the gap a 99.57%. With RTI 75 the oscillations of amplitude GECinit seem to disappear after a few iterations, but instead DEC seems to oscillate round a GEC of 60 without further reducing the gap, which is certainly an undesired behaviour for a route assignment heuristic.
2. If we compare case 1, 2 and 3, we observe that the increasing complexity of the travel demand is reflected in the difficulty for the heuristics to reduce the gap in a smooth manner. This is most obvious in case 3: all heuristics oscillate repeatedly with amplitude GECinit or higher. Such abrupt change of GEC is the result of the 3 traffic demand flows interacting.
3. Focusing on case 3, we finally have a situation were DEC does the best job at reducing the gap. This matches our expectation, as case 3 (network and travel demand) is exactly the setting used in the original article that presented DEC for the first time ([4]).
4. DEC does a very poor job reducing the GAP in case 4 (9% less than MSA). DEC seems to be inadequate for this type of problem. On the other hand, we notice that MSA, despite not converging, reduces the gap the most. GP1 does not degrade like DEC.
5. We can summarize the behaviour of GP1 as being the best heuristic for cases 1 and 2.; the second best for case 3 and case 4. It does overall outperform, or perfom close to the best, for all cases.

4.7.6.4 Conclusions

Based on the observations made for the training set and the test cases we can conclude that:

1. Except for case 3, DEC does not perform up to the expectations set in Rahamin et al. [4]. We further explain this and analyse the possible causes in the next section.
2. Genetic programming as tackled in this thesis proved to be a viable way of obtaining a route assignment heuristic.
3. The GP heuristic was capable of outperforming MSA and DEC for the training set used in the fitness evaluation, and also for test cases 1 and 2, which resemble in terms of complexity of the travel demand flow, that same training set.
4. While outperformed by DEC in case 3, and by MSA in case 4, the GP heuristic is still capable of reducing the gap significantly and display desired characteristics, such as the tendency to keep a constant rate of descent and the capacity to recuperate fast from increments to the GEC.

Given the influence of the training set over the performance of the genetic heuristic we propose to slightly modify the training set in a second experiment using case 3 as an inspiration.

4.7.6.5 Why DEC does not perform up to the expectations

Recall that network 6, with travel demands defined as in case 3 was used in Rahamin et al., [4].

The one comparison provided in [4] presented DEC as performing far better than MSA (figure 30). We do not observe such an overwhelming difference between DEC and MSA in our experiment.

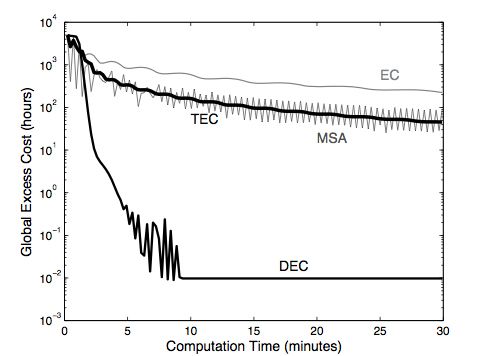


Figure 30 Original comparison MSA vs DEC vs TEC vs EC from Rahamin et al.

In addition we do notice that despite displaying minor oscillations in [4], neither MSA nor DEC seem to display the oscillations of amplitude of order GECinit as in our measurements of case 3.

So, why do we observe such a different behaviour between [4] and case 3?

The most important factor to take into account is that the DTA used in [4] is different than the DTA used in this thesis:

* In [4], all 18 paths are known by the DTA, whereas in our DTA we use a pathfinder that progressively updates the pool of possible paths. The fact that paths are discovered progressively often translates into sudden increments of GEC (for an iteration a new shortest route is discovered).
* In addition as stated in [4], “*Being iterative, an additional goal of this research was to find a way to integrate the DNL component with the route choice component in a way that minimizes the computational effort. This objective was fulfilled using a ”reuse” concept, in which the results obtained by the DNL component at the end of previous DUE iteration are those starting the DNL procedure at the next DUE iteration. The ”reuse” concept reduced the need for DNL iterations to a minimum of two inner iterations.”* It is unclear how this “reuse” was implemented [4] given that the propagation of flow depends on the route choice, but our approach does certainly not depend on any reuse, which may also explain the oscillations in our case and the lack of those in [4]. Notice that reusing the DNL solution may not be a good approach when confronted to many different travel demand flows, as more demand flows means more complex interactions and more ways of affecting the DNL solution.

Overall, DEC seems to rely on a-priori knowledge of the network and reuse of the DNL solution. This could explain why DEC performs so poorly for case 4, which is a more realistic setting, with many traffic flows and many potential different routes.

Other aspects that may explain the difference between DEC in cases 1 to 4 and DEC in [4]:

* It is not explained in [4] how the chosen influences the convergence of DEC as only one result, presumably the best, is provided in [4]. The value of possibly depends on the characteristics of the network and the demand; given that is fixed during our experiments it seems sensible to observe disparity of behaviour.. The fact that DEC performed best for case 3 does, on the other hand, prove that our chosen is well suited for that problem, and henceforth close to the original value proposed in [4].
* Some details of the networks like freespeed and kjam are not provided in [4], consequently our network 6 is not the same as the one used in [4].

4.8 Experiment 2

4.8.1 Training Set

The training set used in the second experiment is composed of the same networks and travel demands as in experiment 1, with the addition of network 6, which was used as a test network in that same experiment. Demand flows for that network have been defined between nodes: 1-4, 3-6, and 5-2 as in case 3 but the flows have the default characteristics used in training (1000veh/h between time instants 0.005 and 0.70).

4.8.2 Final Fitness

Network 6 is considered much more complex and important now. We weight its partial fitness with a factor 3. Therefore the final fitness is:

4.8.3 DTA Control Parameters

All networks are simulated for a DTA process with the following set of parameters:

* Discretization Time Interval: 0.0025h
* Route Interval: 50 time intervals
* Maximum number of DTA iterations: 300
* Termination GEC: 0.15% of the initial gap

The route interval has been reduced which makes it harder to reduce the GEC; this, and the usage of a more complicated network makes experiment 2 more demanding than experiment 1.

4.8.4 Test Set

Test networks are exactly the same as in experiment 1.

4.8.5 Test Cases / Comparisons

Test cases are exactly the same as in experiment 1.

* + 1. Results

It took 2.5 days to run the evolution.

4.8.6.1 Fitness Progression

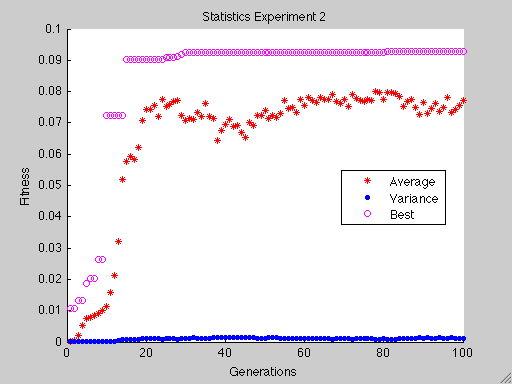
**

Figure 31 Statistics Experiment 2

Fitness in the graph is adjusted fitness.

The best adjusted fitness of experiment 2 is 0.09295833,i.e., 9.75750822976

raw. fitness. This raw fitness can nevertheless not be directly compared to the raw fitness of experiment 1 as it was calculated differently.

Between generations 1 and 20 we observe a steady improvement of the fitness (faster than experiment 1). At generation 27 the average fitness attains nearly its best value. There are no significant improvements afterwards, but oscillations.

In comparison to experiment 1, we observe that the best-so-far fitness is achieved a bit earlier at generation 35. Equally so, the little progress achieved afterwards seems to indicate that stopping after 100 generations is an acceptable criterion.

As in experiment 1 the variance remains over 0 for the entire evolution.

4.8.6.2 Training Set

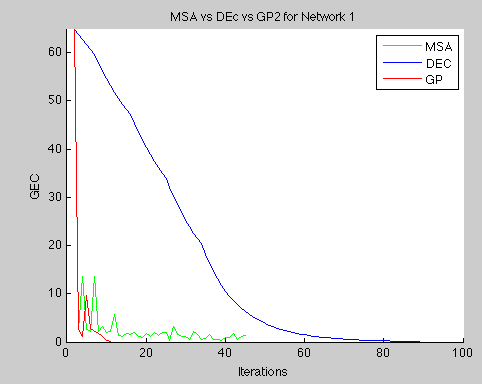


Figure 32 MSA vs DEC vs GP2 for Network 1

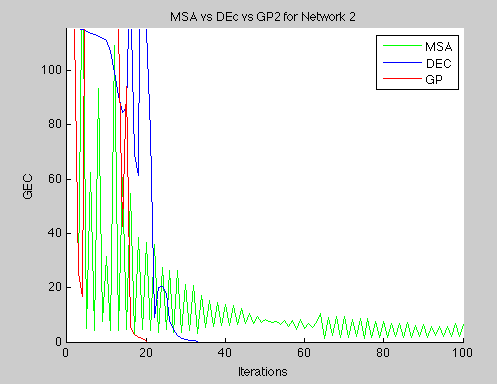


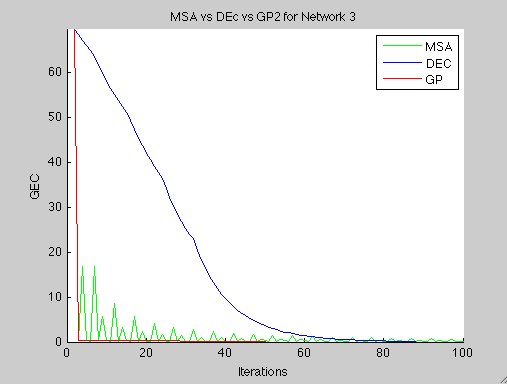
Figure 33 MSA vs DEC vs GP2 for Network 2

Figure 34 MSA vs DEC vs GP2 for Network 3

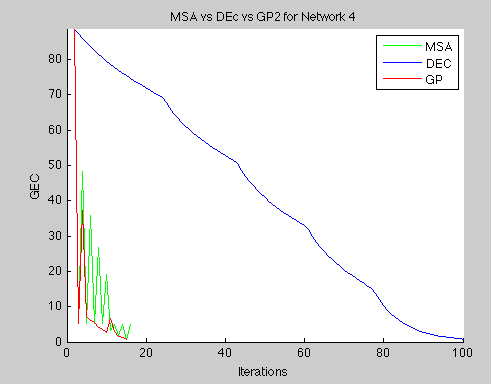


Figure 35 MSA vs DEC vs GP2 for Network 4

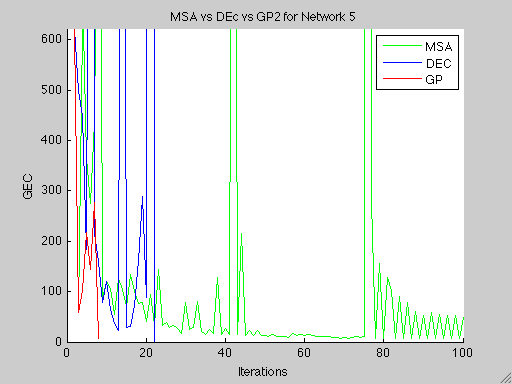


Figure 36 MSA vs DEC vs GP2 for Network 5

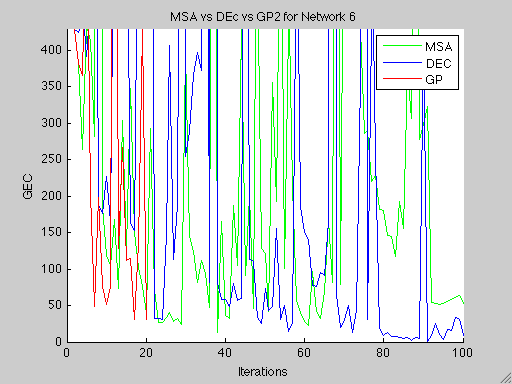


Figure 37 MSA vs DEC vs GP2 for Network 6

|  |  |  |  |
| --- | --- | --- | --- |
|  | MSA | DEC | GP2 |
| NET1 | 45 | 90 | 11 |
| NET2 | 99.7%/259 | 34 | 20 |
| NET3 | 149 | 89 | 51 |
| NET4 | 16 | 117 | 9 |
| NET5 | 99.36%/300 | 23 | 13 |
| NET6 | 96.84%/38 | 99.20%/94 | 21 |

Table 3 Performance Training Set Experiment 1

Observations:

1. The introduction of network 6, which, as predicted, is a much harder problem to solve, has had the effect of producing a different best-heuristic GP2.
2. GP2 outperforms MSA and GEC for all training networks but seems to be less smooth than GP1. In particular for network 2 despite converging in 20 iterations we observe a very significant oscillation around iteration 10. This is not desirable.
3. Network 3, which took 9 iterations to GP1, is now a hard problem for GP2.
4. Network 6, being significantly harder produces oscillations of the order of GECinit for all heuristics.
5. Observations 4 and 5 from experiment 1, section 4.7.7 also apply to experiment 2.

4.8.6.3 Test Cases

Case 1:

Case 1 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

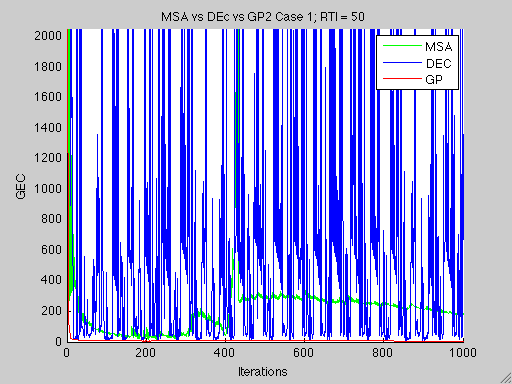


Figure 38 MSA vs DEC vs GP2 Case 1 RTI 50

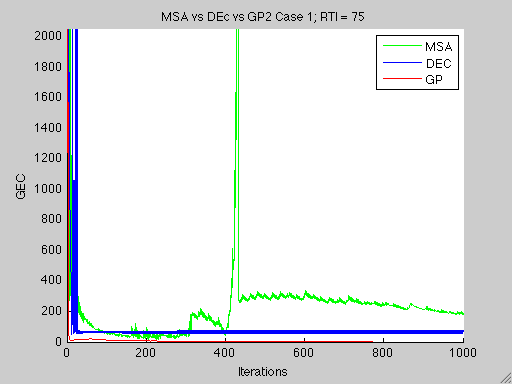


Figure 39 MSA vs DEC vs GP2 Case 1 RTI 75

Case 2:

Case 2 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

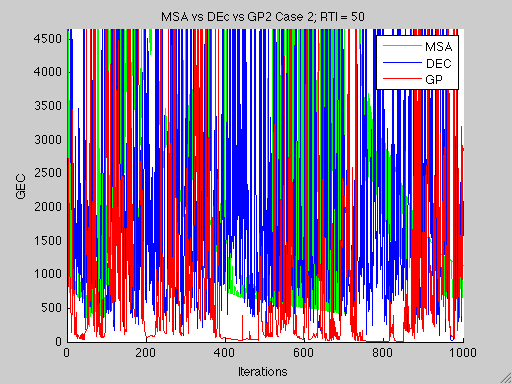


Figure 40 MSA vs DEC vs GP2 Case 2 RTI 50

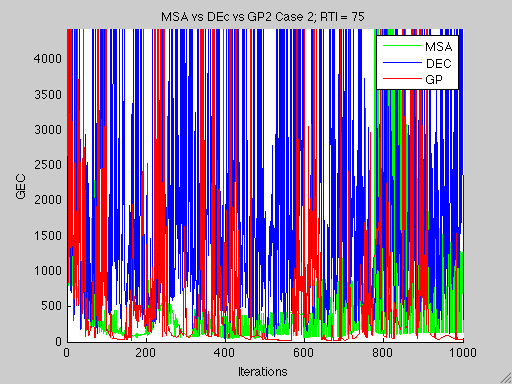


Figure 41 MSA vs DEC vs GP2 Case 2 RTI 75

Case 3:

Case 3 was run for 1000 iterations and a RTI of 50 and 75 time intervals.

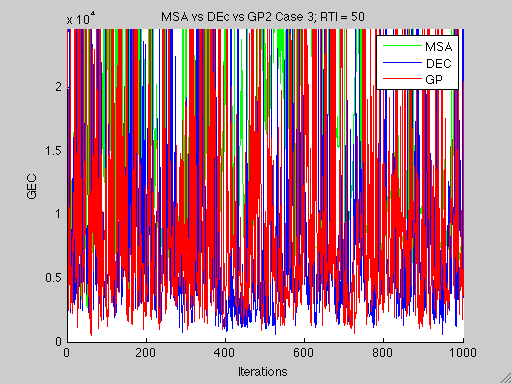


Figure 42 MSA vs DEC vs GP2 Case 3 RTI 50

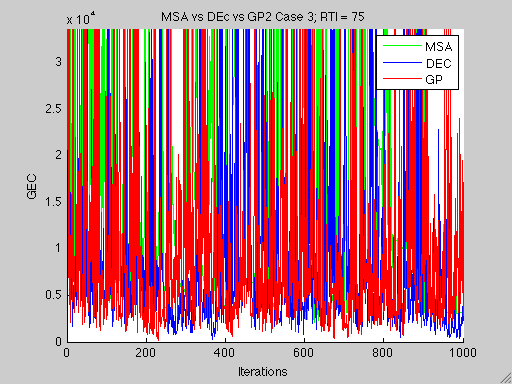


Figure 43 MSA vs DEC vs GP2 Case 3 RTI 75

Case 4:

Case 4 was run for 100 iterations and a RTI of 50 and 75 time intervals.

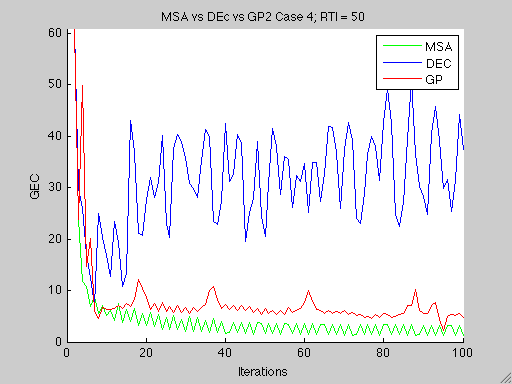


Figure 44 MSA vs DEC vs GP2 Case 4 RTI 50

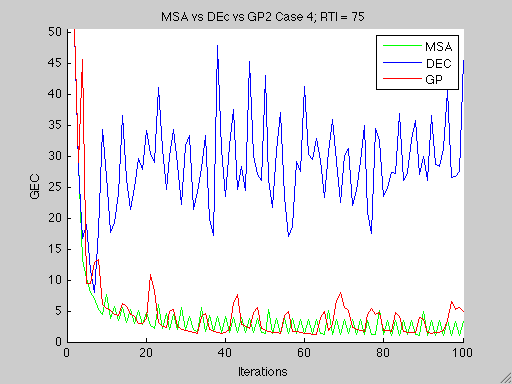


Figure 45 MSA vs DEC vs GP2 Case 4 RTI 75

|  |  |  |  |
| --- | --- | --- | --- |
|  | MSA | DEC | GP2 |
| CASE1, RTI 50 | 768 | 99.57%/95 | 99.77%/871 |
| CASE1, RTI 75 | 99.39%/226 | 97.60%/10 | 771 |
| CASE2, RTI 50 | 95.07%/55 | 99.56%/971 | 99.77%/807 |
| CASE2, RTI 75 | 98.84%/418 | 99.33%/399 | 99.63%/687 |
| CASE3, RTI 50 | 89.53%/154 | 97.47%/525 | 97.95%/62 |
| CASE3, RTI 75 | 93.70%/623 | 99.33%/366 | 99.37%/231 |
| CASE4, RTI 50 | 98.07%/71 | 88.31%/6 | 96.21%/94 |
| CASE5, RTI 75 | 97.98%/96 | 84.31%/6 | 97.64%/62 |

Table 4 Performance Test Cases Experiment 2

Observations:

1. Observations 1, 2, 3 and 4 from the test cases in experiment 1 (section 4.7.6.3) apply to experiment 2, except for case 1, RTI 50, given that MSA proves to be the best heuristic in that case (but GP2 is only 0.08% short of converging).
2. Despite performing slightly worse for case 1 than GP1, GP2 proves to perform better than GP1 for cases 2 and 3. In fact, in case 3, GP2 is now the best heuristic instead of DEC.
3. GP2 also improves the result obtained with GP1 for case 4. Not only GP2 reduces the GAP further but also displays a smoother progression.

4.8.6.4 Conclusions

Based on the observations made for the training set and the test cases we can add to the conclusions from experiment 1 that:

1. Increasing the complexity of the training set proves to be a viable way of obtaining a more performing GP heuristic. In particular, including a new complex network and new travel demand flows in the training has the effect of producing a GP heuristic capable of improving the gap reduction (not only for the expected test case (3) but also for a test case (4) with more, and more different demand flows).

4.2 Conclusion

If we consider together experiment 1 and experiment 2 we conclude that our objective of evaluating the adequacy of using genetic programming to evolve a route assignment heuristic has been fulfilled.

Our second objective of improving over DEC and MSA is fulfilled except for case 4, which unfortunately, due to memory limitations was only run for 100 iterations. It remains an open question whether the GP could have performed better given more iterations.

In addition, we explored the possibility of improving the heuristic by introducing changes to the training set used in the fitness evaluation. It seems possible to push the evolution towards a more performing heuristic with well-crafted networks and travel demands. This, together with the fact that the computational resources were relatively small for this kind of problem, opens the door to progressively obtaining better heuristics in practice.

However, all of the previous conclusions must be taken with caution. It is unclear how representative the networks and the demands used in this thesis are of common networks and demands. Despite our attempt to work in the direction of generality (note that network 6 is according to [4] a very typical case) our test cases are only but a few and therefore our results may not be statistical significant enough.

Chapter 5: Conclusion

The master’s thesis is brought to a close with a chapter summarising all the conclusions once again. This is also the place to include suggestions on further use of the results, in both industrial applications and further research.

Appendices

Appendix A: Source Code

The source code of the traffic simulator alone can be found at:

https://github.com/aarribas/DtaTrafficSimulator

The source code of the program combining the simulator and the genetic programming part, including the .mat files for all networks used in this work, can be found at:

<https://github.com/aarribas/GeneticProgammingForDTAHeuristics>

The ECJ wrapper from Rinde Van Lon can be found at:

https://github.com/rinde/RinECJ

The ECJ’s homepage is:

http://cs.gmu.edu/~eclab/projects/ecj/‎

Appendix B: Stagnation in Early Experiment

S-expression of the best final individual in an early experiment, with 50 individuals and 1000 iterations:

(if4 (if4 (if4 (if4 (if4 (if4 (if4 (if4 (if4 cumudelta cdemand inviteration (if4 (if4 inviteration cdemand inviteration odemand) odemand inviteration odemand)) (if4 inviteration cdemand (if4 inviteration cdemand inviteration odemand) odemand) inviteration (if4 inviteration (if4 (if4 (if4 (if4 (if4 inviteration cdemand inviteration odemand) cdemand inviteration odemand) cdemand inviteration odemand) (if4 cumudelta cdemand inviteration odemand) inviteration odemand) inviteration inviteration odemand) inviteration odemand)) cdemand inviteration

We observe that there is only one function used, if4. Given the low performance of the assignment heuristic and the stagnation of the fitness it was clear that there was a lack of diversity.

Appendix C: Best Individual in Experiment 1

S-expression of the best final individual experiment 1:

(mul (sub (div (add cdemand cumudelta) (div cdemand normcostdiff)) (mul (sub (div (add cdemand cumudelta) (div cdemand normcostdiff)) (mul (mul (mul (if4 (mul (mul (pow (if4 normcostdiff (mul (div cdemand 0.0) (mul normcostdiff (add inviteration odemand))) cdemand inviteration) cdemand) 1.0) (if4 1.0 odemand 1.0 inviteration)) cumudelta inviteration inviteration) (if4 odemand normcostdiff cumudelta cumudelta)) (sub (add (add cdemand cumudelta) odemand) (if4 odemand normcostdiff cumudelta cumudelta))) (mul inviteration inviteration))) (mul (pow (if4 normcostdiff (mul (div cdemand 0.0) (mul normcostdiff (sub (add (add cdemand cumudelta) odemand) (if4 odemand normcostdiff cumudelta cumudelta)))) (sub (sub (mul (mul (mul cumudelta cumudelta) (if4 odemand normcostdiff cumudelta cumudelta)) (mul inviteration inviteration)) (mul (if4 odemand normcostdiff (div (add cdemand cumudelta) (div cdemand normcostdiff)) cumudelta) (mul 0.0 (if4 normcostdiff (mul (mul cumudelta cumudelta) (mul normcostdiff (add inviteration odemand))) (sub (mul (mul (mul inviteration inviteration) (if4 odemand normcostdiff cumudelta cumudelta)) (div (add cdemand cumudelta) (div cdemand normcostdiff))) (mul (mul (if4 cumudelta cumudelta inviteration inviteration) (if4 odemand normcostdiff cumudelta cumudelta)) (mul normcostdiff (add inviteration odemand)))) inviteration)))) (mul (mul cumudelta cumudelta) (mul 0.0 (if4 odemand normcostdiff cumudelta cumudelta)))) inviteration) cdemand) 1.0))) (mul (pow (if4 normcostdiff (mul (div cdemand 0.0) (mul normcostdiff (add inviteration odemand))) cdemand inviteration) cdemand) 1.0))

Appendix D: Best Individual in Experiment 2

S-expression of the best final individual experiment 2:

(if4 (pow (sub (sub (div cdemand 1.0) (if4 odemand inviteration normcostdiff 1.0)) (pow 1.0 inviteration)) (div (add 0.0 1.0) (div cdemand 1.0))) (pow (mul 0.0 (add (if4 (if4 (pow (sub (sub (div cdemand 1.0) (if4 odemand inviteration normcostdiff 1.0)) (pow 1.0 inviteration)) (div (add 0.0 1.0) odemand)) (pow (mul 0.0 (add (pow (div cdemand (add (sub (mul (add 0.0 1.0) (div inviteration odemand)) cumudelta) (div inviteration cumudelta))) 1.0) inviteration)) (pow (div (add inviteration normcostdiff) (add (sub 0.0 cumudelta) (div (add 0.0 1.0) odemand))) 1.0)) (pow 0.0 normcostdiff) (pow (mul (mul 0.0 (add (pow (div cdemand (add (sub (mul (add 0.0 1.0) (div inviteration odemand)) cumudelta) (div inviteration cumudelta))) 1.0) inviteration)) normcostdiff) (pow (div cdemand 0.0) (if4 (pow 1.0 inviteration) (pow (if4 (div inviteration cumudelta) (mul 0.0 cdemand) (mul odemand cdemand) (pow (div (pow (div 1.0 0.0) (add inviteration 0.0)) (div cdemand 1.0)) (div inviteration odemand))) (pow (pow 0.0 normcostdiff) (div cumudelta 0.0))) (mul (add inviteration (add inviteration 0.0)) (mul 0.0 cdemand)) (mul 0.0 cdemand))))) (pow (div inviteration cumudelta) (div cdemand 0.0)) cumudelta (pow (mul (div (pow (div (add 0.0 1.0) (div cdemand 1.0)) (div inviteration odemand)) (add (sub 0.0 cumudelta) (div inviteration cumudelta))) (div inviteration odemand)) (pow (add 0.0 1.0) (pow 1.0 normcostdiff)))) inviteration)) (pow (div (div (sub (pow 1.0 inviteration) (mul odemand cdemand)) (pow cdemand inviteration)) (add (sub 0.0 cumudelta) (div inviteration cumudelta))) 1.0)) (mul (if4 (sub (add inviteration (pow (mul 0.0 (div inviteration cumudelta)) (pow (div (pow (div (add 0.0 1.0) (div cdemand 1.0)) (div inviteration odemand)) (add (sub 0.0 cumudelta) (div inviteration cumudelta))) 1.0))) cumudelta) (if4 (pow 1.0 inviteration) (pow (if4 (sub odemand cumudelta) (mul 0.0 cdemand) (mul odemand cdemand) (add inviteration 0.0)) (pow (pow 0.0 normcostdiff) (div cumudelta 0.0))) (pow (div (add 0.0 1.0) (div cdemand 1.0)) (div inviteration odemand)) (mul 0.0 cdemand)) (mul odemand cdemand) (add inviteration 0.0)) (pow (div (pow (div 1.0 0.0) (mul 0.0 (add (pow (div (div (sub (add inviteration (div cdemand (mul odemand cdemand))) (div cdemand 0.0)) (pow cdemand inviteration)) (div cdemand 1.0)) 1.0) inviteration))) (div cdemand 1.0)) (div inviteration odemand))) (pow (mul (add inviteration normcostdiff) normcostdiff) (pow (div (sub (add inviteration normcostdiff) (div odemand odemand)) (pow cdemand inviteration)) (pow (div (add 0.0 1.0) (div cdemand 1.0)) (div inviteration odemand)))))

Bibliography

[1] Poli, W. B. Langdon and N. F. McPhee, A Field Guide to Genetic Programming, Published via <http://llu.com> abd freely availabe at <http://www.gp-field-guide.org.uk>, 2008

[2] E.Cascetta, Transportation Systems Analysis: Models and Applications, Springer Optimization and Its Applications, September 2009

[3] Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection, [MIT Press](http://en.wikipedia.org/wiki/MIT_Press). [ISBN 0-262-11170-5](http://en.wikipedia.org/wiki/Special:BookSources/0262111705)

[4] Shmuel Rahamim, Michal Blumberg Nitzani and Hillel Bar-Gera. Continuous Correction Functions for Dynamic Route Choice Algorithms. Dynamic Traffic Assignment Journal, June 2008

[5] Chung-Cheng Lu, Hani S. Mahmassani, Xuesong Zhou, Equivalent gap function-based reformulation and solution algorithm for the dynamic user equilibrium problem, Journal of Transportation Research Part B 43, (2009) pages 345–364

[6] Isaak Yperman, The link transmission model for Dynamic Network Loading Ph.D Thesis at KU Leuven, (2007) , ISBN 978-90-5682-837-0

[7] Ruben Corthout, Intersection modelling and marginal simulation in macroscopic Dynamic Network Loading, Ph.D Thesis at KU Leuven, (2012) , ISBN 978-94-6018-506-9

K.U.Leuven Faculty of Engineering 20xx-20xx

Master’s Thesis file

Students: First name Surname

Title: Title of master’s thesis

English title: English translation of the title

UDC:

Content in brief: (maximum 500 words)

(Title of the article: Title of the article)

(English translation: English translation of the title)

Thesis submitted to obtain the degree of Master in Engineering: ….....................................

Supervisor(s):

Assessors:

Mentor(s):

1. This chapter is essentially based on reformulated excerpts from Transportation Systems Analysis by Ennio Cascetta [↑](#footnote-ref-1)
2. *Notice that despite the continuous fluid analogy the simulation and resolution of Macroscopic models requires the discretization of time.* [↑](#footnote-ref-2)
3. This section is essentially a reformulation of the relevant parts from Ruben Corthout’s PhD thesis ([7]) [↑](#footnote-ref-3)