Route Planning Algorithms for Car Navigation

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Flinsenberg, Ingrid C.M.

Route planning algorithms for car navigation /
by Ingrid C.M. Flinsenberg. -
Eindhoven : Technische Universiteit Eindhoven, 2004.
Proefschrift. - ISBN 90-386-0902-7
NUR 919
Subject headings : routing / algorithms / electronic navigation / stochastic graphs / combinatorial optimisation / graphs / automobilism
2000 Mathematics Subject Classification : 05C38, 65K10, 05C85

The work described in this thesis has been carried out at the Development Center of Siemens VDO Trading B.V. (subsidiary of Siemens VDO Automotive) in Eindhoven, the Netherlands. It may contain Intellectual Property Rights (IPRs) of Siemens VDO Automotive AG. All rights reserved.

Route Planning Algorithms for Car Navigation

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. R.A. van Santen, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op donderdag 30 september 2004 om 16.00 uur

door

Ingrid Christina Maria Flinsenberg

geboren te Helmond

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts
en
prof.dr. J. van Leeuwen

Copromotor:
dr. J.H. Verriet

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
IPA Dissertation Series 2004-12

Preface

When I attended the defense of my aunt's thesis approximately nine years ago at Tilburg University, where I just started to study econometrics, the idea that I would be defending my own thesis one day never occurred to me. However, four years ago, when professor Emile Aarts offered me a position as a PhD-student at Eindhoven University of Technology on the topic of advanced route planning strategies for car navigation in cooperation with Siemens VDO Automotive, the topic immediately appealed to me. Nevertheless I gave quite some thought to the question whether I wanted to spend the next four years doing research on this particular topic. I have never regretted my decision to do it, and thanks to the support of many people, you are now holding the result.

I would especially like to thank my two supervisors prof. dr. Emile Aarts and dr. Jacques Verriet. Jacques Verriet has been my daily supervisor for the last four years. I would like to thank Jacques for the discussions that usually lead to new ideas and for his help on implementation issues. I would also like to thank him for his perseverance in reading all my manuscripts. His never ending comments and indications to improve my manuscripts really made a difference. Furthermore, I would like to thank Emile Aarts for our fruitful discussions and for his confidence in me, which was very stimulating.

Furthermore, I would like to thank Edgar den Boef, Martijn van der Horst and Grace Zhu for their valuable contributions and Teun Hendriks and Karin Lim for making sure my research continued to be useful to Siemens VDO Automotive. I consider myself fortunate in having the members of the route planner group from Siemens VDO Automotive as my colleagues. They helped me get valuable insight in the peculiarities of route planning in car navigation systems and were always very helpful in solving problems with car navigation software. Furthermore, I would like to thank my colleagues from the (Eindhoven) Embedded Systems Institute, for providing a pleasant working environment at the university as well. I would also like to thank my friends from Chikara, whose friendship indirectly had quite an impact. Furthermore, I am grateful for the interest and support of my family.

Finally, I would like to thank my parents for always supporting me. Without their continuing support this thesis would not have been possible.

Contents

1

Introduction

Mobility is very important in our society. People live in one city and work in another. They go to visit friends and family living in different parts of the country. Even leisure time is not always spent in their residence. Despite the Dutch government's efforts to increase the use of public transportation, the car is still the most widely used means of transportation. Every time someone travels from one location to another (by car), he or she first determines the best route to reach the destination. Depending on the time of day and the day of the week, this route may be different. For example, the driver may know that a certain road is always congested at a particular time and day. As a consequence, he chooses to use a different route to avoid this congestion.

Car navigation systems are being offered as a special feature of new cars of an increasing number of car-brands. These car navigation systems are capable of taking over some of the tasks that are performed by the driver such as reading the map and determining the best route to the destination. They should also take daily congestion patterns into account. Because a car navigation system uses a built-in computer to determine a route, it can compare many different routes and the user expects the system to determine the best possible, or optimum route fast. This thesis is concerned with route planning algorithms that enable a car navigation system to plan optimum routes on very large real-world road networks in very little time, taking daily congestion patterns into account.

To that end, Section 1.1 briefly explains the concept of a car navigation system as well as the main functionality of such a system. The route planning problem that we

will try to solve is presented in general terms in Section 1.2. Section 1.3 presents an overview of publications on related topics. An outline of the remainder of this thesis is presented in Section 1.4. But first, we give a non-comprehensive overview of a car navigation system.

1.1 Car Navigation Systems

Currently, an increasing number of consumers has gained interest in car navigation equipment. Car navigation is no longer a luxury that is only for the rich. A car navigation system is offered as one of the many extras or as an advertising stunt of more and more "middle-class" cars. In the future, car navigation equipment may become as normal as air-conditioning. Of course, also other vehicles can use car navigation, such as trucks, busses and motorcycles. A navigation system offers the driver the possibility to be guided to his destination, by means of spoken and/or visual advices. In order to achieve this, the driver first has to enter his destination into the system. Such a destination may be a city center, an entire street, or an address including a house number. Information on every possible destination is contained in a database that is stored on a CD or DVD that has to be inserted into the car navigation system. Also data from external providers about interesting places such as hotels and restaurants for example is stored on the CD or DVD. The driver can ask for a list of all hotels in a certain area for example.

Figure 1.1. *A car navigation system.*

Figure 1.1 gives an overview of the main elements of a car navigation system. It shows on the left the sensors (gyro and tacho) and a GPS satellite used for determining the position of the car. The computer, which contains the CD or DVD player, and the screen for displaying the map and visual advice are shown in the middle. On the right, a GSM connection and TMC-RDS broadcasts used for receiving traffic information are shown.

The key components of a car navigation system are positioning, i.e. determining the current position of the car on the road network, route planning, i.e. planning

a route from the position of the car to the destination, and guidance, i.e. giving instructions to the driver. Schlott [1997] and Zhao [1997] give an overview of these key components. Figure 1.2 gives an overview of the most important components of a car navigation system.

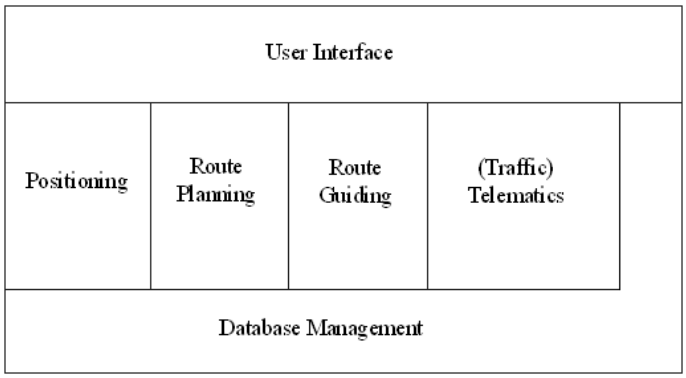


Figure 1.2. *The key components of a car navigation system.*

First, we explain positioning. In order to determine the current location of the car on the road network, a car navigation system tries to determine the geographical position of the car. This is done by "dead reckoning" [Zhao, 1997], which is the calculation of the geographical position based on calibrated sensor information. In particular the driven distance from the tacho and the angular rate from the gyro are used to determine the position of the car. This position can be improved by using the GPS position and speed. Subsequently, this geographical position is matched to the available road network. This is done by computing the likelihood that the car is currently at a particular position on a certain road segment. The route presented to the driver is also used in this map matching process. A position on the current route is given a higher preference than other positions. If no position on the road network has a sufficiently high likelihood, the car is not positioned on any of the road segments of the road network, i.e. the car is "off map".

Once the position of the car has been determined, a route can be planned from that car position to the destination. This route is planned using the available road network that is stored on a CD or DVD. The algorithm used for planning a route is (generally) based on Dijkstra's single source shortest path algorithm [Dijkstra, 1959]. To be able to determine a route fast, several approximations are used. For example, if the car position and destination are located sufficiently far apart, only important roads are used to determine the major part of the route. Furthermore, estimations of the remaining distance to the destination are used to speed up the planning process.

After a route has been found, the system provides the driver with spoken and/or visual advice to guide him to his destination. The position of the car is used in combination with the current route to determine the necessary advice and the timing

of giving advice. Usually, the driver first receives a warning that he should prepare to make a certain turn. Of course, this advice has to be given in time for the driver to actually make the necessary preparations, such as changing lanes and slowing down. Then the actual advice is given. Also this advice has to be timed carefully. While the driver is progressing towards his destination, the car navigation system monitors the progress of the car. That is done by comparing the current position of the car and the presented route. Of course, not all drivers (correctly) follow all instructions, sometimes a driver deviates from his route. If the car is not positioned on the current route for a certain amount of time, then the system concludes the driver has deviated from his route, and a new route needs to be planned from the position of the car to the destination.

This thesis focuses on the route planning functionality of a car navigation system. Car navigation systems usually provide the option to choose among several different optimization criteria, or cost functions. In general, the driver can choose between planning a fastest route, a shortest route, a fastest route giving preference to motorways, and a fastest route giving a penalty to motorways. Also an option to avoid toll roads or ferries may be available. In the future, we expect to see an increase in the personalization of planned routes, by an increasing adaptation of the used cost functions to the preferences of the driver.

Another aspect of real-life that influences the quality of a planned route, is the presence of traffic jams on the presented route. Traffic information, i.e. information on traffic jams, road works and road conditions for example, are received via RDS-TMC (Radio Data System - Traffic Message Channel), or by a connected GSM phone. One of the challenges of a navigation system is to let route planning take this traffic information into account, and guide the driver around a traffic problem. Because of the increasing number of traffic jams, and the frustration of the driver when he gets stuck in one of them, this would be valuable to the driver.

If for some reason the driver is not satisfied with his route, he is usually given the option to ask for an alternative route. The reasons for asking for such an alternative route may vary from a reported traffic jam on the current route, to preferences that do not correspond with the used cost function. Alternative routes are meant to differ significantly from the presented route. However, if one particular street in the presented route is under construction, the driver may not want a significantly different route, he just wants to be guided around the construction works, and back to his old route again. In most navigation system the driver can achieve this by asking for a local detour. The system should then determine a route that locally avoids the problem area.

1.2 Problem Formulation

A car navigation system uses maps that contain a road network. CDs or DVDs containing the road network of entire Europe are now becoming available. This means that a car navigation system has to be able to plan routes on road networks that contain millions of road segments. On the other hand, a driver does not like to wait while his route is being calculated. Therefore, the planning of routes has to be done very fast, within a few seconds typically.

After the driver has entered his destination, and the system has determined the position of the car, a route has to be planned from the car's position to the destination. When planning this route, data has to be retrieved from the database repeatedly, because of the limited amount of internal memory that is available to the system. Reading information from this database is a time-consuming task, because the access to the secondary storage is slow. Also the various processes involved in planning a route, determining the position of the car and displaying the map are competing for the limited amount of computing power. Compared to a personal computer, an onboard car navigation system has considerably less computing power, due to hardware restrictions implied by the systems required tolerance for cold and heat, as well as its ability to withstand shocks.

To measure the quality of a planned route, a cost function is used to make "objective" comparisons between routes possible. Of course, different drivers have different preferences. Determining the cost function that correctly resembles the preferences of a particular, or of the "average" driver, is not an easy task. Determining such a cost function is beyond the scope of this research, and we assume that a cost function is available that correctly describes the preferences of the driver. A route that has lowest cost according to the used cost function, is called an optimum route. If the driver asks for the best route, then he expects that the system provides him the route with the absolutely lowest cost. This means that a car navigation system does not only have to plan a route, but has to plan the optimum route according to a particular cost function.

Planning optimum routes fast on very large road networks still poses a significant challenge to companies developing car navigation systems. At the same time, taking information on daily congestion patterns into account is getting more important. Drivers become increasingly demanding, and expect that traffic information is not only provided, but also used for determining the best route.

We aim at finding algorithms and approaches that enable a car navigation system to plan optimum routes on very large road networks in very little time, taking traffic information into account. At the same time, it should remain possible to maintain the full functionality of a car navigation system as described in Section 1.1. Also the introduction of all kinds of new functionality has to be possible.

Current car navigation systems use an approximation algorithm to plan a route

fast on large road networks. Ideally, we would like to be able to plan optimum routes faster than current systems plan a non-optimum route. The road networks on which routes have to be planned generally consist of millions of nodes and edges. Because route planning in the car has to be fast, it is necessary, with current hardware, to use some kind of pre-processing. The road network is too large to use as a whole to search for an optimum route. Therefore, the road network has to be divided into some kind of sub-networks that are smaller and can be searched more effectively.

We first have to find a pre-processing algorithm that enables the planning of optimum routes, or in other words, routes with minimum cost. Using the result of this pre-processing algorithm, the planning of optimum routes in very large road networks containing millions of nodes should be fast. So, we need to find a route planning algorithm using the result of the pre-processing step that plans optimum routes fast.

A challenge of a navigation system is to let the route planning take traffic information into account, and possibly guide the driver around a traffic problem. Current car navigation systems take traffic information into account by increasing the cost of roads that are congested when planning a route. However, traffic jams are usually only reported if they are longer than a few (typically two) kilometers, or if they cause a significant delay for drivers. However, less severe traffic jams or mild reductions in driving speed caused by everyday peak traffic are not always reported. These traffic circumstances do influence the best route however. Therefore, we would also like to take these daily recurring patterns into account when planning a route. Drivers familiar to a particular environment also take these circumstances into account when they are determining their route, and so should a car navigation system. Therefore, we want to take daily congestion patterns into account when planning a route. This should not only remain possible by using the pre-processing algorithm, but the planning of routes that take these congestion patterns into account should also be fast. Furthermore, these routes should be the best routes possible, which leads to a need for a definition of an optimum route in the presence of daily congestion. Finally, we need to develop a route planning algorithm for planning these optimum routes while taking congestion patterns into account.

Naturally, daily congestion is not exactly the same every day, so we have to deal with uncertainty. This not only requires the formulation of a model for handling uncertainty, but also the determination of the quality of a route with uncertain cost, and a route planning algorithm that determines a best possible route fast. Of course, this has to be combined with the use of the pre-processing algorithm.

Finally, the developed algorithms must be compatible with other features of a car navigation system. For example, a new route has to be planned if the driver deviates from his route, or if he requests an alternative route. Furthermore, the functionality of a car navigation system should be (easily) extendable. The designed algorithms should not restrict the introduction of new features in the future.

In determining an approach to achieve these goals, we make one important assumption. We assume that the planned routes do not influence the congestion patterns. If many drivers own a car navigation system using the same algorithm to determine an optimum route, then all these drivers receive the same advice and take the same route. This could cause congestion on this route which, in turn, may lead to another route becoming the optimum route. We assume that this is not the case, i.e., a car navigation system does not have to take the effect of its own advice into account. This is not an unreasonable assumption because we believe personalization will become increasingly important in car navigation systems. As a result, the used cost functions for planning optimum routes will differ between drivers. This most likely leads to different optimum routes for different drivers. Note that Jahn, Möhring & Schulz [2000] study the problem of planning good routes such that the usage of the road networks capacity is optimized.

1.3 Related Work

Finding a route in a road network with minimum cost is referred to as the shortest path problem. This problem has been studied extensively in the past decades. Dijkstra's algorithm [Dijkstra, 1959] is the most well-known algorithm for determining the shortest path from one location to all other locations in a road network. If only a shortest path between two locations has to be determined, Dijkstra's algorithm can be speeded up by taking an estimation of the cost from a location to the destination into account. This algorithm is called the $A^\star$-algorithm [Hart, Nilsson & Raphael, 1968]. If the used estimation satisfies a few conditions, then the $A^\star$-algorithm can be used to plan the shortest path between two nodes in a road network. Also many other shortest path algorithms exist, such as the Bellman-Ford algorithm [Bellman, 1958; Ford Jr. & Fulkerson, 1962], the D'Esopo-Pape algorithm [Pape, 1974] and the Floyd-Warshall algorithm [Floyd, 1962]. An overview is given by Bertsekas [1998]. More recently, Thorup [1997] has presented a deterministic linear time and space shortest path algorithm for undirected graphs with positive integer edge weights. However, the $A^\star$-algorithm (originally in [Hart, Nilsson & Raphael, 1968], see also [Gelperin, 1977] and [Pearl, 1984]) is the most commonly used shortest path algorithm in geographical networks.

New algorithms for planning the shortest path from one location to all other locations have been presented in [Meyer, 2001; Meyer, 2002; Meyer & Sanders, 2003; Pettie, 2002]. Meyer [2001] presents algorithms for planning the shortest path from one location to all other locations in arbitrary directed graphs with real edge weights uniformly distributed in $[0,1]$ in linear average-case time. The average-time complexity of computing these paths in parallel is studied by Meyer [2002]. Meyer & Sanders [2003] propose and analyze sequential and parallel versions of a label-correcting algorithm for the single source shortest path problem. Pettie [2002] presents an algo-

rithm for determining the shortest path between all pairs of nodes in arbitrary real-weighted directed graphs. Wang & Kaempke [2004] present an polynomial time algorithm for computing the shortest path in distributed systems. Huang, Jing & Rundensteiner [1996] present an algorithm for finding the shortest path between all pairs of nodes in a graph which takes the limited amount of main memory in an intelligent transportation system into account. Chen, Daescu, Hu & Xu [2003] present an algorithmic paradigm for finding the length of an optimal path without growing a single source shortest path tree. Finding a multi-criteria shortest path is discussed by Granat & Guerriero [2003]. Mandow & Pérez de la Cruz [2003] discuss multi-criteria heuristic search in more general terms.

Very specific for road networks is the presence of turn restrictions. Turn restrictions are traffic rules that forbid the driver to make a certain maneuver, for example, it may be forbidden to make a left turn at a particular intersection. These traffic rules or turn restrictions form a very specific aspect of real-life road networks, that is usually ignored when discussing route planning on real-world road networks. Turn restrictions can be modeled by costs on adjacent road segments, so that certain turns are given an additional cost. Turn restrictions have been studied recently by Schmid [2000], Winter [2002] and Szeider [2003]. Schmid [2000] discusses forbidden turns or equivalently, turn restrictions with infinite costs. He presents several algorithms and graph reformulations for planning optimum routes if forbidden turns exist in a road network. Winter [2002] considers turn costs on pairs of adjacent edges. He constructs the line graph of the road network, and proves that optimum routes in the original road network can be planned by applying a standard shortest path algorithm to the line graph. Szeider [2003] considers the problem of determining whether a simple path exists between two nodes in a finite undirected simple graph G with turn restrictions (i.e. in a graph with at most one edge between any two nodes and without loops), where a simple path is a route in which each node appears at most once. He constructs for each node u in graph G a transition graph that has a node for each edge adjacent to node u, and an edge between two nodes if the turn between the two corresponding edges is allowed. He proves that the problem of determining a simple path is *NP*-complete if the collection of these transition graphs contains at least one out of four sets of graphs.

For a car navigation system, a standard Dijkstra-like algorithm [Dijkstra, 1959; Hart, Nilsson & Raphael, 1968] is not fast enough to plan optimum routes in large real-world road networks. Because of the high demands on planning speed, the route planning process has to be speeded up, which can be done by pre-processing the road network. Jung & Pramanik [2002] describe a graph partitioning approach to speed up the planning process. They divide the road network into a number of disjunct subgraphs that are connected by a boundary graph. The planning process is speeded up, by reducing the graph on which the optimum route is planned. Kim,

Yoo & Cha [1998] discuss handling real-time data in combination with graph partitions. They add all edges that may be subject to real-time data to the boundary graph. Henzinger, Klein, Rao & Subramanian [1997] use the planar separators of Lipton & Tarjan [1979] to achieve a faster route planning process. Their theorem gives efficient theoretic bounds on the running time of their shortest path algorithm, but we do not consider this approach to be practically feasible. Chen & Xu [2000] also use these planar separators to answer shortest path queries in undirected planar graphs with non-negative edge weights, but their appoach is based on the embedding of the graph in the plane. Graph separators are also studied by Alber, Fernau & Niedermeier [2003]. Chan & Zhang [2001] base their partitioning of a graph on rectangular grids and subgraphs may have nodes in common. Huang, Jing & Rundensteiner [1995] and Huang, Jing & Rundensteiner [1997] also use subgraphs that have nodes in common but store the minimum path cost between every pair of nodes in every subgraph and between a restricted number of node pairs in the boundary graph. These node pairs are selected according to the importance of the edges connected to these boundary nodes, thereby sacrificing optimality of the planned paths. In [Jing, Huang & Rundensteiner, 1996; Jing, Huang & Rundensteiner, 1998] they preserve optimality by storing all minimum path costs between boundary nodes. Shekhar, Fetterer & Goyal [1997] study the memory requirements of storing the minimum path cost between every pair of nodes for several node sets, and compare this with the decrease in the route planning time resulting from using these stored minimum path costs. Fernández-Madrigal & González [2002] introduce multiple hierarchies in a partition to plan a route but their approach sacrifices the optimality of the planned route. Car, Mehner & Taylor [1999] do not store shortest path costs but use a hierarchy based on average travel speeds to restrict the number of edges considered in finding a shortest path, thereby sacrificing optimality of the found path. Seong, Sung & Park [1998] use a similar approach. Ertl [1998] creates an implicit edge hierarchy by creating a "radius" for every edge. Only if the distance from the start node or destination node to the end node of an edge is smaller than its radius, the edge is evaluated when planning an optimum route between the two nodes. If the optimality of routes needs to be guaranteed, determining these radii is very time-consuming.

A road network can be pre-processed by graph partitioning, which is studied by several authors. Berry & Goldberg [1999] compare different algorithms for computing a graph partition. Falkner, Rendl & Wolkowicz [1994] study partitioning the nodes of a graph into k disjoint subsets of specified sizes so as to minimize the total weight of the edges connecting nodes in distinct subsets of the partition. They present a numerical study on the use of eigenvalue-based techniques to find upper and lower-bounds for this problem, based on graphs of several thousands of nodes. Huang, Jing & Rundensteiner [2000] compare alternative graph clustering solutions for storing data in square blocks that minimize the number of I/O operations. They compare

spatial partitioning clustering, that exploits spatial coordinates and high locality, 2-partitioning and approximate topological clustering. Their experiments on randomly generated graphs show that different algorithms should be used for different types of graphs. Krishnan, Ramanathan & Steenstrup [1999] study graph partitioning for Internet-like structures and require that the partitions are connected. Monien & Diekmann [1997] study bisection techniques for minimizing the number of connections between subgraphs. Pothen [1997] studies the same problem but compares several techniques. Baños, Gil, Ortega & Montoya [2004] minimize the number of cut edges allowing a pre-specified unbalance in the number of nodes in the created subgraphs. Kim & Moon [2004a] empirically study the local-optimum solution space of this problem for the Fiduccia-Mattheyses heuristic [Fiduccia & Matteyses, 1982]. Bollobás & Scott [2004] give optimal bounds for bipartitions of graphs with bounded maximum degree minimizing the maximum number of edges in each subgraph. Graph partitioning is also studied by Schloegel, Karypis & Kumar [2000], Karypis & Kumar [1998] and Kim & Moon [2004b]. Partitioning of flat terrain into polygons is studied by Rowe & Alexander [2000]. Cordone & Maffioli [2004] study the complexity of partitioning a graph into a given number of trees given various constraints on the locations and weights of the trees.

Another approach to speed up the route planning process is by taking the available computation time directly into account when designing the algorithm. Hiraishi, Ohwada & Mizoguchi [1999] and Shekhar & Hamidzadeh [1993] present an algorithm that finds a sub-optimal solution within a specified time. They also study the probability that a solution is found within the specified time.

Planning an optimum route that takes daily congestion patterns into account can be seen as planning a route in a graph that has costs that are not constant. Using time-dependent costs can be seen as one method to model daily congestion patterns. Planning fastest routes in time-dependent graphs has been studied by Orda & Rom [1990, 1991, 1996]. Wellman, Ford & Larson [1995] discuss the planning of fastest paths where the edge weights are given by time-dependent probability distributions. Subramanian [1997] and Chabini [2002] present routing algorithms for the k-shortest path problem for graphs with time-dependent edge costs. Horn [2000] presents and compares algorithms for planning a fastest time-dependent route and assumes the time-dependent travel speed changes continuously during the traversal of an edge. Da Cunha & Swait [2000] consider the problem of planning a minimum-cost path in a dense graph where each node can only be visited within a specific time-window. Glenn [2001] discusses a similar problem but also allows waiting at a pre-specified fixed cost. Fu [2001] presents an adaptive route planning algorithm for handling real-time information, and only determines the next road segment in the route. Gao [2002], Gao & Chabini [2002] and Chabini [2002] do the same for stochastic time-dependent travel times. Azaron & Kianfar [2003] apply stochastic dynamic program-

ming to find the dynamic shortest path in stochastic dynamic networks, in which the arc lengths are independent random variables with exponential distributions. Note that it is not realistic to assume that arc lengths are exponentially distributed because this results in arc length variations that are loo large. Golshani, Cortes-Rello & Howell [1996] use Dempster-Shafer theory to represent and reason with uncertain information on both costs and distance in an unknown environment for routing autonomous vehicles. Montemanni, Gambardella & Donati [2004] and Montemanni & Gambardella [2004] use an interval of possible values to model uncertainty and determine a path with minimum maximum deviation from the optimum path cost for all possible realizations of edge costs. An overview of research concerning real-time vehicle routing problems is presented by Ghiani, Guerriero, Laporte & Musmanno [2003].

Updating an optimum route in case the cost of one or more edges in the graph changes (due to congestion for example), or edges are added to or deleted from the graph has been studied by Frederickson [1985], Frigioni [1998], Klein & Subramanian [1998] and Narváez, Siu & Tzeng [2001] for example. They have developed algorithms for the dynamic maintenance of the minimum-cost path upon a change in edge costs or the insertion or deletion of edges. These algorithms assume the existence of a data structure, usually the minimum-cost path tree, that is updated in case of a change in the available data to avoid a full re-computation. Glenn [2001] discusses reoptimization in case of a changed departure time from the start node in time-dependent networks.

1.4 Thesis Outline

We introduce three road network models in Chapter 2. We start with a formal description of the road network and of the problem of planning an optimum route on this road network. We ignore the presence of daily congestion in this model. However, in order to be able to take this information into account when planning a route, we need to extend our model. Therefore, we subsequently introduce a time-dependent road network model. This model can be used to plan optimum routes in a road network taking daily congestion into consideration. However, in this model uncertainty in daily congestion is not taken into account. Therefore, we extend the time-dependent model further to a stochastic time-dependent road network model. This model is suited for taking uncertainty in daily congestion into consideration. Finally, we indicate how we are going to proceed, i.e., what strategy we use in order to end up with a route planning algorithm that enables us to plan optimum routes on a very large road network in very little time, taking uncertainty in daily congestion patterns into account when planning a route.

As said in Section 1.2, in order to achieve fast route planning, we need to develop a pre-processing algorithm. The result of this pre-processing algorithm should en-

able us to plan optimum routes on very large road networks in very little time. Chapter 3 describes a pre-processing approach as well as two pre-processing algorithms. We will use a partitioning approach to pre-process the road network. This leads to the definition of a partitioning problem. We show that this partitioning problem is strongly *NP*-hard. We then introduce and compare two approximation algorithms for creating a partition. The partitioning problem can be extended to create partitions of partitions, which we call a multi-level partition as opposed to a single-level partition. Finally, we present the conclusions on the creation of partitions.

After we have discussed the creation of partitions in Chapter 3, we discuss the planning of optimum routes in large real-world road networks using these partitions in Chapter 4. Daily congestion is still ignored in Chapter 4. We discuss planning an optimum route using single-level partitions and present some results. Similarly, we discuss planning with multi-level partitions and discuss the results. We will conclude that it is possible to plan optimum routes on very large road networks in very little time.

In Chapter 5 we turn to the problem of taking daily congestion patterns into account when planning a route. We first discuss the planning of optimum routes in such a road network. The planning algorithm, the data that we use to test and evaluate our algorithm, and the results are discussed. We also deal with the integration of the pre-processing step and the use of daily congestion. As we will see, we can effectively and practically combine the use of partitions and daily congestion patterns during route planning.

Chapter 6 discusses route planning with uncertainty in daily congestion. Also for route planning with uncertainty, we first look at the implications of taking uncertainty into account for planning optimum routes. The planning algorithm is then discussed. We have also gathered data on uncertainty in travel times. The results of the route planning algorithm are presented, and the combination of using partitions and planning with uncertainty is discussed. Again, we will see that using partitions can be combined with stochastic route planning as discussed in Chapter 6, both in theory and in practice.

To evaluate our route planning algorithms, we compare the results of our route planning algorithms with the results of the route planning algorithm used in commercially available car navigation systems in Chapter 7. We first briefly discuss route planning algorithms as they are used in car navigation systems that are currently on the market in order to facilitate a proper comparison. We compare the average performance of our route planning algorithm with the VDO Dayton route planning algorithm developed by Siemens VDO Automotive in Eindhoven. As we will see, our algorithm is on average able to plan optimum routes faster than the VDO Dayton route planning algorithm plans non-optimum routes. We also discuss some exceptional cases from which the benefits of our approach can be seen even more clearly.

Subsequently, we discuss the advantages and disadvantages of our route planning approach. Then we turn to the additional functionality of a route planning component in a car navigation system. We discuss what can be done if the costs of certain road segments in the road network change, if the driver deviates from his route, or if he enters a new destination. Requests for an alternative route to the current destination are also discussed. As we will see, all additional functionality can be combined with our approach.

Finally, in Chapter 8, we give an overview of the main conclusions and achievements.

2

Road Network Models

In this chapter we present three road network models. First, we discuss the model that we use to plan routes in a road network if all driving times are constant over time. This is the basic model, and it is discussed in Section 2.1. We then extend our model to incorporate time-dependent driving times as well as time-dependent costs. This model is used to take daily traffic congestion patterns into account when planning a route. The time-dependent model is discussed in Section 2.2. Finally, we extend the time-dependent model so that uncertainty in travel times and costs are also incorporated. This stochastic time-dependent model is discussed in Section 2.3. It can be used to take uncertainty in daily congestion into account when planning a route in a road network.

2.1 Time-Independent Planning

A car navigation system uses a map containing a road network to plan routes. This road network consists of road segments connected at intersections. A lot of information is stored about both intersections as well as road segments, such as coordinates, length, speed, road type, number of lanes, street names, house numbers, importance, whether it is located in a urban or rural area, and so on. Furthermore, information on illegal maneuvers, such as a restriction on making a left turn at an intersection, is available. Not all available information is needed to plan, for example, the fastest or shortest route from one location to another. We now formulate a graph-theoretic

model that incorporates the necessary information to plan a route.

A road network can be represented by a *multi-graph*. A multi-graph consists of nodes and edges. The intersections in a road network form the *nodes* of the multi-graph. The road segments between intersections in a road network form the *edges* of the multi-graph. Because one-way roads exist, all edges are directed. There may exist multiple edges between two nodes. For example, there may exist parallel roads without side streets that diverge and join again. Therefore, the road network is represented as a multi-graph, which can have multiple directed edges between the same two nodes. Let N denote the finite, non-empty set of *nodes*, with $|N| = n$ the total number of nodes. Let E denote the finite, non-empty set of *directed edges* between pairs of nodes. For an edge e from node u to node v, we define $\delta_1(e) = u$ and $\delta_2(e) = v$. Our goal is to plan a route from one position to another. In a graph we choose to represent this by planning a route from a start node s to a destination node d. A route is represented by a sequence of connected edges.

Definition 2.1. A *route* p in multi-graph $G = (N, E)$ is a sequence of edges $p = \langle e_1, \ldots, e_\ell \rangle$, with $e_i \in E$, and $\delta_2(e_i) = \delta_1(e_{i+1})$, for $i = 1, \ldots, \ell - 1$. □

We say route p is a route *from* start node $s = \delta_1(e_1)$ *to* destination $d = \delta_2(e_\ell)$. Define $E(p)$ as the *edge set* of route p, $E(p) = \{e_1, \ldots, e_\ell\}$, so $E(p)$ contains the edges of route p. We use $N(p)$ to represent the *node set* of route p, so $N(p) = \{\delta_1(e_1), \ldots, \delta_1(e_\ell)\} \cup \{\delta_2(e_\ell)\}$. Similarly, $N(G)$ is used to denote the node set of graph G, so $N(G) = N$, and $E(G)$ for the edge set of graph G, so $E(G) = E$. The number of edges in a route is denoted by $L(p)$, so for $p = \langle e_1, \ldots, e_\ell \rangle$, we have $L(p) = \ell$. Furthermore, let $P(G)$ denote the collection of all possible routes in graph G, and let $P^{(\ell)}(G)$ denote all routes in graph G containing ℓ edges.

Definition 2.2. Let e_1 and e_2 be edges in graph G, and let u and v be nodes in G. We define edge e_2 to be *adjacent* to edge e_1 if and only if $\langle e_1, e_2 \rangle \in P^{(2)}(G)$. Edges e_1 and e_2 are adjacent if and only if either e_1 is adjacent to e_2 or e_2 is adjacent to e_1. Furthermore, node u is adjacent to edge e_1 and edge e_1 is adjacent to node u if and only if $\delta_1(e_1) = u$ or $\delta_2(e_1) = u$. Finally, node u is adjacent to node v if and only if there exists an edge e in G with $\delta_1(e) = u$ and $\delta_2(e) = v$ or $\delta_1(e) = v$ and $\delta_2(e) = u$. □

When planning a route in a road network, the costs of road segments and intersections have to be taken into account. These costs can represent for example, the length of the road segment, the time needed to drive over the road segment and the costs of making turns. Therefore, we need to define cost functions for edges.

Definition 2.3. The costs of the edges in the graph $G = (N, E)$ are represented by a function $w_e : E \longrightarrow \mathbb{R}_0^+$ where $w_e(e)$ is the non-negative *cost* of edge $e \in E$.[1] □

[1] $\mathbb{R}_0^+$ denotes the collection of non-negative real numbers.

Another aspect of a road network is the existence of traffic rules. At a particular intersection, the driver may, for example, not be allowed to make a right turn. Due to these traffic rules, not all possible routes are allowed in a road network. Naturally, these traffic rules have to be incorporated into our model (otherwise illegal routes may be presented to the driver). Traffic rules can be modeled by a cost function on pairs of adjacent edges. If a traffic rule exists that forbids the driver to go from one road segment to another, then an infinite cost is associated with that pair of adjacent edges.

Definition 2.4. Define $w_{\mathrm{r}} : P^{(2)}(G) \longrightarrow \mathbb{R}_0^+ \cup \{\infty\}$. If $w_{\mathrm{r}}(\langle e_1, e_2 \rangle) > 0$, then we say there exists a *rule* from edge e_1 to e_2 with *rule cost* $w_{\mathrm{r}}(\langle e_1, e_2 \rangle)$. □

In the remainder of this thesis we denote $w_{\mathrm{r}}(\langle e_1, e_2 \rangle)$ by $w_{\mathrm{r}}(e_1, e_2)$. A road network can be represented by a multi-graph that has all these properties. This leads to the following definition of a *roadgraph.*

Definition 2.5. A *roadgraph* is a tuple $G = (N, E, w_{\mathrm{e}}, w_{\mathrm{r}})$. □

Two different operations on (road)graphs are defined in Definition 2.6.

Definition 2.6. The *union* of two graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ is defined as $G_1 \cup G_2 = (N_1 \cup N_2, E_1 \cup E_2)$. We define the *difference* of graph G_1 with graph G_2 as $G_1 \setminus G_2 = (N(E_1 \setminus E_2) \cup (N_1 \setminus N_2), E_1 \setminus E_2)$. □

Now, we can define the cost of a route. The total cost of a route is equal to the sum of the costs of all edges in the route, plus the costs of all rules on the route.

Definition 2.7. The *route cost* of a route is represented by $w_{\mathrm{p}} : P(G) \longrightarrow \mathbb{R}_0^+ \cup \{\infty\}$. Let $p = \langle e_1, \ldots, e_\ell \rangle \in P(G)$. We define

$$w_{\mathrm{p}}(p) = \sum_{e \in E(p)} w_{\mathrm{e}}(e) + \sum_{i=1}^{\ell-1} w_{\mathrm{r}}(e_i, e_{i+1}).$$

□

We denote the cost of an optimum route from start node s to destination node d in roadgraph G by $w_{\mathrm{p}}^{\star}(G, s, d)$.

When one is planning a route in a roadgraph, it has to be checked whether the route does not contain any forbidden turns. Therefore we define a *feasible route.* A feasible route is a route that does not contain any forbidden turns. Note that one-way streets are not formulated as traffic rules and can only be contained in a route in the allowed direction, as follows from Definition 2.1. Therefore, infeasibility cannot be caused by one-way streets.

Definition 2.8. A feasible route p is a route that satisfies $w_{\mathrm{p}}(p) \neq \infty$. □

In a car navigation system, a route has to be planned from the car to the destination that is specified by the driver. We consider the following problem.

Problem 2.1. Plan a route p in roadgraph G, from a node s to a node d, such that $w_{\mathrm{p}}(p)$ is minimum. □

2.2 Time-Dependent Planning

In Section 2.1 we introduced the time-independent model. Costs in this model do not depend on the time of day. However, certain properties of a road network change over time. Roads may be closed during specific time periods. For example, a road can be closed for construction work during several hours or days. Furthermore, during rush hours the driving time of a route is typically longer. In order to take this into account, we extend our time-independent model, so that it can also handle time-dependent costs. In this section we introduce the time-dependent model.

We assume that the node and edge sets of a roadgraph are constant over time, and that only the costs change over time. This poses no restriction because if an edge is not available at certain time intervals, this can be modeled for example by setting the edge cost equal to infinity during those time intervals. So, we introduce time-dependent edge and rule costs. We assume that time $t \in \mathbb{R}_0^+$.

Definition 2.9. We define the *time-dependent edge cost* function as $w_{\mathrm{e}}^{\mathrm{t}} : E \times \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+ \cup \{\infty\}$. $w_{\mathrm{e}}^{\mathrm{t}}(e,t)$ is the cost of edge $e \in E$ if the driver departs from $\delta_1(e)$ at time $t \in \mathbb{R}_0^+$. The *time-dependent rule cost* function is defined as $w_{\mathrm{r}}^{\mathrm{t}} : P^{(2)}(G) \times \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+ \cup \{\infty\}$. $w_{\mathrm{r}}^{\mathrm{t}}(\langle e_1, e_2\rangle, t)$ is the cost of a rule on adjacent edge pair $\langle e_1, e_2\rangle \in P^{(2)}(G)$ for a driver departing from node $\delta_2(e_1) = \delta_1(e_2)$ at time $t \in \mathbb{R}_0^+$. □

In the remainder of this thesis we denote $w_{\mathrm{r}}^{\mathrm{t}}(\langle e_1, e_2\rangle, t)$ by $w_{\mathrm{r}}^{\mathrm{t}}(e_1, e_2, t)$. Having introduced time-dependent costs, we also need to determine the time of traversing an edge, or making a turn. The driving time of an edge is given by a driving time function $t_{\mathrm{e}}^{\mathrm{t}}$, and the turning time of a turn is given by function $t_{\mathrm{r}}^{\mathrm{t}}$.

Definition 2.10. The *driving time* function is a function $t_{\mathrm{e}}^{\mathrm{t}} : E \times \mathbb{R}_0^+ \to \mathbb{R}_0^+ \cup \{\infty\}$, that gives for every edge the time it takes to traverse that edge, starting the traversal of the edge at time t. The *turning time* function is a function $t_{\mathrm{r}}^{\mathrm{t}} : P^{(2)}(G) \times \mathbb{R}_0^+ \to \mathbb{R}_0^+ \cup \{\infty\}$, that gives for every rule the time it takes to make the corresponding turn, starting the traversal of the second edge at time t. □

The driving time needed to traverse edge e starting at time t is denoted by $t_{\mathrm{e}}^{\mathrm{t}}(e,t)$. The turning time at time t of a rule on a pair of adjacent edges e_1, e_2 is given by $t_{\mathrm{r}}^{\mathrm{t}}(\langle e_1, e_2\rangle, t)$ which we denote by $t_{\mathrm{r}}^{\mathrm{t}}(e_1, e_2, t)$. By allowing a positive turning time, it is possible to take waiting times for traffic lights into account. If the timing of the traffic lights is known then we can take into account the time the driver has to wait for the next green light. Note that because the timing of traffic lights can be different for traffic that is going in different directions, we cannot spread this time over different edges. If a driver has to wait for a ferry for example, the additional costs can be incorporated into the edge functions of the graph.

This leads to the following definition of a time-dependent roadgraph.

Definition 2.11. A *time-dependent roadgraph* G^{t} is a tuple $G^{\mathrm{t}} = (N, E, w_{\mathrm{e}}^{\mathrm{t}}, w_{\mathrm{r}}^{\mathrm{t}}, t_{\mathrm{e}}^{\mathrm{t}}, t_{\mathrm{r}}^{\mathrm{t}})$. □

To make it possible to determine routes in a time-dependent roadgraph, it is also necessary to extend our definition of a route to incorporate time. Therefore, we formulate a *time-route*.

Definition 2.12. Given a *time-dependent roadgraph* $G^{\mathrm{t}} = (N, E, w_{\mathrm{e}}^{\mathrm{t}}, w_{\mathrm{r}}^{\mathrm{t}}, t_{\mathrm{e}}^{\mathrm{t}}, t_{\mathrm{r}}^{\mathrm{t}})$, a *time-route* is given by a tuple $\pi = (p, t)$. Here $p = \langle e_1, \ldots, e_\ell \rangle$ is a route in G^{t} and $t \in \mathbb{R}_0^+$ is the departure time from $\delta_1(e_1)$. □

Now we can determine the arrival time of the driver at the destination. Given time-route $\pi = (\langle e_1, \ldots, e_\ell \rangle, t_1)$, the *arrival time* at the destination is $t_{\ell+1}$, given that t_1 is the *departure time* of the driver from the start node, $t_2 = t_1 + t_{\mathrm{e}}^{\mathrm{t}}(e_1, t_1)$ and $t_{i+1} = t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i) + t_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i))$ for $i = 2, \ldots, \ell$. This leads to the following definition of the total route cost of a time-route.

Definition 2.13. Given a time-dependent roadgraph G^{t} and departure time t_1, let $\pi = (p, t_1)$ be a time-route in G^{t} with $p = \langle e_1, \ldots, e_\ell \rangle$ and departure time $t_1 \in \mathbb{R}_0^+$. The total route costs of time-route π are given by

$$w_{\mathrm{p}}(\pi) = w_{\mathrm{e}}^{\mathrm{t}}(e_1, t_1) + \sum_{i=2}^{\ell} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i)) + \sum_{i=2}^{\ell} w_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i),$$

with $t_2 = t_1 + t_{\mathrm{e}}^{\mathrm{t}}(e_1, t_1)$ and $t_{i+1} = t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i) + t_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, t_i))$ for $i = 2, \ldots, \ell$. □

We denote the cost of an optimum, i.e. minimum-cost route from start node s to destination node d in a time-dependent roadgraph G^{t} by $w_{\mathrm{p}}^{\star}(G^{\mathrm{t}}, s, d)$.

For an edge that cannot be traversed or a turn that cannot be made during a particular given time interval, we have to determine the edge and rule cost and duration. If waiting is not allowed for a particular edge or rule, then we set the edge or rule cost and duration during the restricted time interval equal to ∞. Consequently, all routes that start traversing that edge or making that turn during the restricted time interval have a route cost and time equal to ∞, and are infeasible. If waiting is allowed for a particular edge or rule, then the duration of traversing the edge or making the turn during a restricted time interval is set equal to the time from the moment of arrival to the moment the edge or rule is allowed again (i.e. the waiting time), plus the driving time of the edge or turning time of the rule at that time. The cost of the edge or rule (during the restricted time interval) is set equal to the cost of waiting from the moment of arrival to the moment the edge can be traversed or the turn can be made again, plus the cost of the edge or rule at that time. Note that if an edge can be traversed or a turn can be made at all times, then postponing the traversal of the edge or

making of the turn is not allowed (i.e. waiting is not allowed for these edges). Only for an edge that cannot be traversed or a turn that cannot be made during a particular given time interval, waiting may be allowed (depending on the selected edge and rule cost and time functions), but the time of departure and resulting cost are determined beforehand and incorporated in the edge and rule time and cost functions.

Now that we have defined a time-route in the time-dependent roadgraph, it is necessary to define when a time-route is feasible. Because the total route costs take the arrival time at a node into account, it is sufficient to check if the route costs are finite.

Definition 2.14. Let G^{t} be a time-dependent roadgraph, and π a time-route in G^{t}. Time-route π is feasible if and only if $w_{\mathrm{p}}(\pi) < \infty$. □

Our problem can now be formulated as follows.

Problem 2.2. Plan a time-route π in G^{t} from a start node s to a destination node d such that $w_{\mathrm{p}}(\pi)$ is minimum. □

2.3 Stochastic Time-Dependent Planning

The time-dependent costs from Section 2.2 can be used to model structural delays in travel time that are for example caused by heavy traffic during rush hours. In this way it becomes possible to take the effect of congestion into account when planning a route, even before a traffic jam report has been issued. Furthermore, small reductions in travel speed can be taken into account, while a traffic jam is only reported when the travel time has drastically increased. However, the used travel speeds cannot be assumed to be equal to the travel speed the driver really experiences at the moment he starts traversing a particular road segment. Therefore, in this section we introduce uncertainty into our model.

The function $t_{\mathrm{e}}^{\mathrm{t}}(e,t)$ denotes the time it takes for a driver to traverse edge e starting at time t. In case of a traffic jam, or a reduced travel speed, both $w_{\mathrm{e}}^{\mathrm{t}}(e,t)$ and $t_{\mathrm{e}}^{\mathrm{t}}(e,t)$ are stochastic, because the exact travel speed and therefore both the cost of the edge and the driving time of that edge are unknown. Usually, the cost of an edge depends on the driving time. Of course, the variance of the edge cost can always be set to 0, if no uncertainty exists. Also without traffic jams travel times are uncertain, for example due to traffic lights, which can be modeled by turning times and rule costs. However, we assume that neither the costs nor the duration of rules are subject to uncertainty.

We first define the *stochastic edge cost*, and the *stochastic driving time*.

Definition 2.15. The random (stochastic) variable $\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,t)$ on domain $\mathbb{R}_0^+ \cup \{\infty\}$ denotes the *stochastic (time-dependent) edge cost* of edge e for departure time $t \in \mathbb{R}_0^+$. The random (stochastic) variable $\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e,t)$ on domain $\mathbb{R}_0^+ \cup \{\infty\}$ denotes the *stochastic (time-dependent) driving time* of edge e for departure time $t \in \mathbb{R}_0^+$. □

This leads to the definition of a *stochastic roadgraph* $\tilde{G}^{\mathrm{t}}$.

Definition 2.16. We define a *stochastic (time-dependent) roadgraph* as a tuple $\tilde{G}^{\mathrm{t}} = (N, E, \tilde{w}_{\mathrm{e}}^{\mathrm{t}}, w_{\mathrm{r}}^{\mathrm{t}}, \tilde{t}_{\mathrm{e}}^{\mathrm{t}}, t_{\mathrm{r}}^{\mathrm{t}})$. □

The introduction of stochastic travel times causes the need for a new definition of a *stochastic time-route*, because the departure time for each edge in a route is no longer a given constant but a stochastic variable.

Definition 2.17. Given a stochastic time-dependent roadgraph $\tilde{G}^{\mathrm{t}}$, a *stochastic time-route* $\tilde{\pi}$ is defined as $\tilde{\pi} = (p,t) = (\langle e_1, \ldots, e_\ell \rangle, t)$, with p a route in graph (N,E) and $t \in \mathbb{R}_0^+$ the departure time from $\delta_1(e_1)$. □

In Definition 2.19, we define the cost of a stochastic route. But first we need to introduce some additional notation.

Definition 2.18. Let $\tilde{x}$ be a stochastic variable. The *mean* of $\tilde{x}$ is denoted by $\mu(\tilde{x})$ and the *variance* by $\sigma^2(\tilde{x})$. The *coefficient of variation* of x, $c(\tilde{x})$, is equal to $\frac{\sigma(\tilde{x})}{\mu(\tilde{x})}$, where $\sigma(\tilde{x})$ denotes the *standard deviation* of $\tilde{x}$. □

Definition 2.19. Given a roadgraph $\tilde{G}^{\mathrm{t}}$ and a stochastic time-route $\tilde{\pi} = (p, t_1)$ with $p = \langle e_1, \ldots, e_\ell \rangle$, the *stochastic cost* of $\tilde{\pi}$ is defined as

$$\tilde{w}_{\mathrm{p}}(\tilde{\pi}) = \tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_1, t_1) + \sum_{i=2}^{\ell} \tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_i, \tilde{t}_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, \tilde{t}_i)) + \sum_{i=2}^{\ell} w_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, \tilde{t}_i),$$

with $\tilde{t}_2 = t_1 + \tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_1, t_1)$, and $\tilde{t}_{i+1} = \tilde{t}_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, \tilde{t}_i) + \tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i, \tilde{t}_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1}, e_i, \tilde{t}_i))$ for $i = 2, \ldots, \ell$. □

The stochastic cost of stochastic route $\tilde{\pi}$ is very difficult to determine because the driving times are stochastic and the edge and rule costs depend on the stochastic departure time from the start node of an edge. Checking the feasibility of a route cannot be done so easily anymore. Specifically, the arrival time at the destination is uncertain, so it is possible that the route has an infinite cost with a certain positive probability strictly lower than 1. This means that it is possible that a route turns out to be feasible or infeasible, each with a positive probability. However, the condition that $w_{\mathrm{p}}(\pi) < \infty$ can be generalized in a relatively straightforward manner to $Pr(\tilde{w}_{\mathrm{p}}(\tilde{\pi}) < \infty) \geq \varepsilon$, which means that the probability that the route has a finite cost has to be greater than or equal to ε. If $\varepsilon = 1$ then it is known that the route always has a finite cost.

Definition 2.20. Let $\tilde{G}^{\mathrm{t}}$ be a stochastic time-dependent roadgraph, and $\tilde{\pi} = (p,t)$ a stochastic time-route in $\tilde{G}^{\mathrm{t}}$. Time-route $\tilde{\pi}$ is ε-*feasible* if and only if $Pr(\tilde{w}_{\mathrm{p}}(\tilde{\pi}) < \infty) \geq \varepsilon$, with $\varepsilon \in [0,1]$. □

If a route is not 1-feasible there is a strictly positive probability that the route has infinite costs, so the expected route cost is infinite. Because of the complicated determination of the stochastic route cost, we also define the *estimated* cost of a stochastic route. For a route that is not 1-feasible the expected estimated route cost may still be finite. Note that we assume that the cost of an edge in a route only depends on the departure time, and not (directly) on the cost of the preceding edge in the route, so the costs and travel times of adjacent edges are assumed to be independent.

Definition 2.21. Given a roadgraph $\tilde{G}^{\mathrm{t}}$, a stochastic time-route $\tilde{\pi} = (p, t_1)$ with $p = \langle e_1, \ldots, e_\ell \rangle$, and $\beta, \gamma \in \mathbb{R}_0^+$, the *estimated cost* of stochastic route $\tilde{\pi}$ for given β, γ is defined as $w_{\mathrm{p}}(\tilde{\pi}) = \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) + \beta \cdot \sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}))$, with

$$\begin{aligned}
\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \mu(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \sum_{i=2}^{\ell} \mu(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))) + \sum_{i=2}^{\ell} w^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i), \\
\sigma^2(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \sigma^2(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \sum_{i=2}^{\ell} \sigma^2(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))),
\end{aligned}$$

with

$$\begin{aligned}
t_2 &= t_1 + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)), \\
t_{i+1} &= t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i) + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))) \\
&\quad + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))), \text{for } i = 2, \ldots, \ell.
\end{aligned}$$

□

The factor β is used to include the standard deviation of the route cost in the route cost. If β is large then the driver has a lot of aversion against uncertainty, and the optimum route cost will generally have a lower standard deviation, but a higher mean. The factor γ is used to increase the travel time per edge. If γ is large then the probability that the driver arrives earlier than estimated is high. We assume β and γ are fixed, but arbitrary.

In order to determine which route is the optimum stochastic route, we first need to be able to order stochastic routes according to their quality. Let $\tilde{x}$ and $\tilde{y}$ be two random variables. If $\tilde{x} \preceq \tilde{y}$ then we say $\tilde{x}$ is as least as good as $\tilde{y}$. We assume that relation "$\preceq$" is a *partial order* on a set S, see [Brightwell & West, 2000].

Definition 2.22. A relation "$\preceq$" is a *partial order* on a set S, $\preceq \subseteq S \times S$, if it has

Reflexivity:	$a \preceq a$ for all $a \in S$;
Antisymmetry:	$a \preceq b$ and $b \preceq a$ implies $a = b$;
Transitivity:	$a \preceq b$ and $b \preceq c$ implies $a \preceq c$.

□

If any two variables can be compared by relation "$\preceq$" then "$\preceq$" is a *total order* on a set S, see [Brightwell & West, 2000].

Definition 2.23. A relation "$\preceq$" is a *total order* on a set S, $\preceq \subseteq S \times S$, if it has

Reflexivity:	$a \preceq a$ for all $a \in S$;
Antisymmetry:	$a \preceq b$ and $b \preceq a$ implies $a = b$;
Transitivity:	$a \preceq b$ and $b \preceq c$ implies $a \preceq c$;
Comparability:	for any $a, b \in S$, either $a \preceq b$ or $b \preceq a$.

□

If $a \preceq b$ and $b \npreceq a$ then we denote this by $a \prec b$. For a given relation for a set of stochastic route costs, we define an optimum stochastic route as follows.

Definition 2.24. Let "$\preceq$" be a partial order on the stochastic route costs $\tilde{w}_{\mathrm{p}}(\tilde{\pi})$. A stochastic route $\tilde{\pi}_1 = (p_1, t)$ with stochastic cost $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)$ is an *optimum* stochastic route if and only if there does not exist a stochastic route $\tilde{\pi}_2 = (p_2, t)$ with stochastic cost $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)$ such that $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2) \prec \tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)$. □

If we decide to use the stochastic route cost to measure the quality of a route, then we need to base the relation of the stochastic routes on this stochastic route cost. Because the average route cost and its standard deviation are the most important factors in determining which route is preferred, we use the following relation for stochastic route costs. Note that this relation is a total ordering. Let $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)$ and $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)$ denote the stochastic cost or travel time of two routes for given $\beta, \gamma \in \mathbb{R}_0^+$.

If $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) < \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2))$ then $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1) \prec \tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)$.
If $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) \leq \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2))$ then $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1) \preceq \tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)$.
If $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1)) = \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}_2))$ then $\tilde{w}_{\mathrm{p}}(\tilde{\pi}_1) = \tilde{w}_{\mathrm{p}}(\tilde{\pi}_2)$.

Note that this corresponds to using the *estimated* cost of a stochastic route to measure the quality of stochastic routes. We denote the cost of the optimum route (according to the estimated route cost) from start node s to destination node d in a stochastic time-dependent roadgraph $\tilde{G}^{\mathrm{t}}$ by $w^{\star}_{\mathrm{p}}(\tilde{G}^{\mathrm{t}}, s, d)$.

We can choose to let the quality of a stochastic route be directly determined by the stochastic cost of the route. We can also choose to minimize the estimated cost of a route. This leads to the following two route planning problems.

Problem 2.3. Plan a stochastic time-route $\tilde{\pi}$ in $\tilde{G}^{\mathrm{t}}$, from a node s to a node d, such that either $\tilde{w}_{\mathrm{p}}(\tilde{\pi})$ is optimum according to an ordering "$\preceq$" or such that $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}))$ is minimum. □

Minimizing $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) + \beta\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}))$ can be used to let the driver indicate the importance of taking uncertainty into account when planning a route. For example for $\beta = 1$ a decrease of 10 minutes in the expected travel time and a decrease of 10 minutes in the standard deviation of the travel time are considered equally important. For higher values of β reducing uncertainty becomes more important than reducing the expected travel time. For lower values of β the situation is exactly the other way around.

2.4 Towards a Solution Strategy

A very important aspect of planning a route in a car navigation system is the time it takes from the moment the driver requests guidance until he receives his first advice. We call this the *time to first advice*. The time to first advice should be as small as possible, because a driver does not like to wait while the system is computing his route. Once he receives advice, the system can continue planning, but this time is in general not experienced as waiting time. Another important aspect of route planning in car navigation systems is the quality of the planned route. As said in Sections 2.1, 2.2 and 2.3, we aim at planning routes with minimum cost, or in other words, optimum routes. However, the number of edges in roadgraphs used by car navigation systems is very large. The standard $A^\star$-algorithm [Hart, Nilsson & Raphael, 1968] is an efficient algorithm for planning routes from a single start node to a single destination node. In fact, there does not exist an algorithm for planning single source, single destination routes in general graphs that is more efficient in worst-case, as was shown by Gelperin [1977]. However, using the standard $A^\star$-algorithm (or Thorup's algorithm [Thorup, 1997]) to plan an optimum route in a large real-world roadgraph, is not fast enough for route planning in car navigation systems.

Our first objective is to plan optimum routes in large roadgraphs considerably faster than the standard $A^\star$-algorithm. In order to increase the route planning speed and maintain the possibility to plan optimum routes, we choose to use pre-processing. By pre-processing the roadgraph, we perform some work on planning an optimum route beforehand. By using the results of the pre-processing step during the actual route planning, the speed of the route planning algorithm can be increased dramatically. In the pre-processing step, we partition the roadgraph into a number of disjoint subgraphs, called *cells*. The resulting division of the roadgraph is called a *partition*. The pre-processing step is described in detail in Chapter 3. In order to study the effect of the pre-processing step, we study the planning of optimum routes in a time-independent roadgraph G in Chapter 4. We particularly focus on the number of iterations needed by the route planning algorithm to plan optimum routes. So, Chapter 4 focusses on the route planning speed. Because pre-processing is used to increase the planning speed, we also study the quality of the created partitions.

After achieving our first objective in Chapter 4, we turn to planning optimum routes in time-dependent roadgraphs in Chapter 5. We first discuss planning optimum time-dependent routes in a time-dependent roadgraph. For some of these time-dependent roadgraphs, it is optimal to delay the arrival at a node to achieve an optimum time-dependent route. Naturally, this greatly complicates the search for an optimal route. Therefore we introduce the concept of consistency. If the roadgraph is consistent, then we can use a modified standard shortest path algorithm such as the $A^\star$-algorithm to plan optimum routes. If the roadgraph is not consistent, using such an algorithm may result in non-optimum routes. We then discuss the effects of

planning fastest routes in time-dependent roadgraphs by planning routes through the Netherlands using realistic driving speeds that lead to a consistent roadgraph. Secondly, we discuss the integration of planning time-dependent routes using a partition. We will see that route planning in time-dependent roadgraphs can be combined with using the partitions introduced in Chapter 3. As a result also time-dependent routes can be planned fast.

Finally, we study the planning of stochastic time-dependent routes in Chapter 6. Also for stochastic planning, consistency of the used roadgraphs is an important issue to assess the optimality of the planned routes. Because of the complexity of determining the stochastic cost of a route and the limited planning time available to a car navigation system, we plan stochastic routes with a minimum estimated stochastic cost. In particular, we plan fastest stochastic routes through the Netherlands to investigate the usefulness and behavior of stochastic time-dependent planning. The integration of using partitions with stochastic time-dependent planning is also discussed. We will see that also the planning of stochastic time-dependent routes can be combined with the use of partitions.

3

Partitioning

As was explained in Section 2.4, planning an optimum route on a real-world road network containing millions of nodes and edges, takes too long if a standard shortest path algorithm, such as Dijkstra's algorithm [Dijkstra, 1959] is used. The number of edges that have to be examined in order to plan an optimum route is just too large. In this chapter, we discuss an approach that enables us to reduce the size of the roadgraph used for planning routes, while it remains possible to plan an optimum route between any two nodes in the road network. This approach is based on the construction of a so-called *partition.* We define a partition in Section 3.1. Section 3.2 discusses the criteria used for determining the quality of a partition. The complexity of determining a partition with maximum quality is discussed in Section 3.3. We will see that this problem is strongly *NP*-hard. Since it is generally believed that it is not possible to determine an optimum solution efficiently for *NP*-hard problems, we discuss two approximation algorithms for creating a partition in Section 3.4. Section 3.5 discusses a possible extension to multi-level partitions. The conclusions are presented in Section 3.6.

3.1 Representing Partitions

The input of a route planning algorithm, before any pre-processing, consists of a roadgraph, for example the roadgraph in Figure 3.1. This roadgraph can be divided into several subgraphs, each consisting of a set of nodes and a set of edges. Such a

subgraph is called a *cell*. Every node in a roadgraph is contained in exactly one cell. All edges of the original graph that connect nodes of a single cell are also part of that cell and are called *internal edges*. Cells usually are formed by highly connected subgraphs of the original road network. For example, a city such as Eindhoven or Amsterdam may form a cell. For a road network of a city, cells may be formed by districts such as the Bijlmer in Amsterdam. Figure 3.2 gives the cells of the roadgraph in Figure 3.1 after the roadgraph has been divided into four cells, represented by rounded squares.

There are edges that connect nodes from different cells. Such an edge is called a *boundary edge*. The two end nodes of a boundary edge are called *boundary nodes*. All other nodes are called *internal nodes*. This leads to a collection of boundary nodes and boundary edges, the *boundary graph*, that is not necessarily connected. Figure 3.3 gives the resulting boundary graph for this partition.

Definition 3.1 gives the formal definition of a partition of a roadgraph G.

Definition 3.1. Let $G = (N, E, w_e, w_r)$ be a roadgraph, then a *partition* of G is a set of roadgraphs $\{C_1, \ldots, C_k\}$ where $C_i = (N_i, E_i, w_e, w_r)$ is a roadgraph induced by the node set N_i, such that $N_i \cap N_j = \emptyset$, for every $i \neq j$, and $\cup_{i=1}^{k} N_i = N$. Furthermore, for all rules in G, such that $w_r(e_1, e_2) > 0$, there has to exist a $C_i \in \{C_1, \ldots, C_k\}$ such that $e_1, e_2 \in C_i$. The edge and rule costs of *cell* C_i are equal to the edge and rule costs of the corresponding edges and rules in G. □

Definition 3.1 defines the cells of a partition, and its costs. These cells are usually connected by boundary edges. Definition 3.2 gives the formal definition of boundary edges.

Definition 3.2. Given a partition $\{C_1, \ldots, C_k\}$ of roadgraph $G = (N, E, w_e, w_r)$, the set of *boundary edges* is given by $E_B = E \setminus \{\cup_{i=1}^{k} E(C_i)\}$. □

The nodes that are connected by these boundary edges are called boundary nodes and are defined in Definition 3.3.

Definition 3.3. Given a partition $\{C_1, \ldots, C_k\}$ of roadgraph $G = (N, E, w_e, w_r)$, the set of *boundary nodes* of a cell C_i is given by $N_B(C_i) = \{u \in N(C_i) | \exists e \in E_B : u = \delta_j(e), j = 1, 2\}$. □

These boundary nodes and edges form the basis of the so-called *boundary graph* defined in Definition 3.4.

Definition 3.4. The *boundary graph* B of a partition $\{C_i, \ldots, C_k\}$ of roadgraph G is defined as $B = (N_B, E_B, w_e, w_r)$ with $N_B = \cup_{i=1}^{k} N_B(C_i)$ the collection of *boundary nodes* and E_B the collection of *boundary edges*. The costs of the edges of the boundary graph are equal to the costs of the corresponding edges in G. The costs of all possible rules in B are equal to 0. □

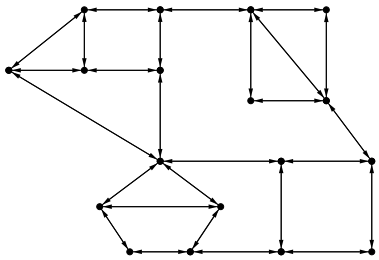


Figure 3.1. *A roadgraph before division into cells.*

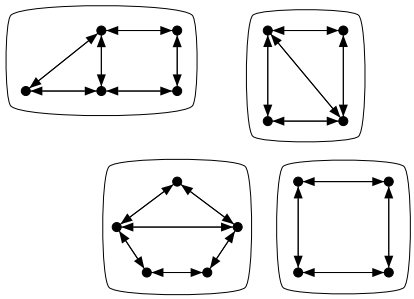


Figure 3.2. *Resulting cells after the roadgraph has been divided into four cells.*

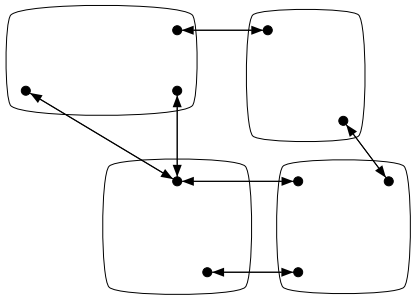


Figure 3.3. *Boundary edges after the roadgraph has been divided into four cells.*

Now that we have defined a partition and its boundary graph, we can give the formal definition of an internal node.

Definition 3.5. Given a partition $\{C_1, \ldots, C_k\}$ of roadgraph G with corresponding boundary graph B, the set of *internal nodes* of a cell C_i is given by $N_I(C_i) = N(C_i) \setminus N_B(C_i)$. □

Note that all edges of cell C_i are called internal edges, as is formally stated in Definition 3.6.

Definition 3.6. Given a cell $C_i = (N_i, E_i, w_e, w_r)$ the set of *internal edges* of cell C_i is given by $E_I(C_i) = E_i$. □

3.2 Partitioning Objectives

Now that we have defined a partition, we need to define its quality, so that we can determine the quality of a particular partition. The quality of a partition depends on the purpose of the partition, in our case planning optimum routes as fast as possible. Therefore, a partition has a high quality if it leads to a fast route planning process.

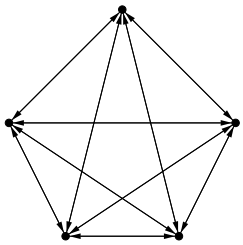
The quality of the partition depends on the size of the graph that is needed to plan optimum routes in a roadgraph, which we call the *searchgraph*. The partition defined in Section 3.1 forms the basis for this searchgraph. How this searchgraph can be created is discussed in the next subsection.

Creating a Searchgraph

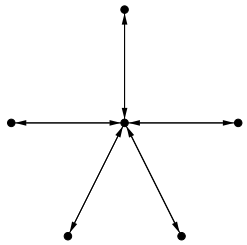
The general idea of creating a partition is that when a route needs to be planned, the areas of a road network, i.e. the cells of a partition, that the driver just passes through can be replaced by the optimum routes through those cells. These optimum routes can be stored on the CD or DVD containing the road network. For example, suppose a driver wants to travel from node s to node d in Figure 3.6. Then he just passes through the cells in the north-east and the south-west. So if we store an optimum route between the boundary nodes of these cells on the CD or DVD containing the road network, then we only need to determine these optimum routes once, instead of computing them anew each time the driver wants to travel from s to d. In order to determine an optimum route from s to d, we do not need to store the entire detailed optimum route between these boundary nodes however. Storing the cost of the optimum route between these boundary nodes is sufficient.

Before we can plan optimum routes with a partition, we need to store the cost of an optimum route between two boundary nodes of a single cell. The most logical way to do this, is by creating an additional edge, called a *route edge*, between every pair of boundary nodes of a single cell. The graph formed by these route edges is called a *route graph*. The number of route edges in the route graph of cell C_i is denoted by r_i. Figure 3.4(a) shows the route graph for a cell containing 5 boundary nodes

in case a route edge is created between every pair of boundary nodes to represent an optimum route between these two boundary nodes. This route graph contains $b_i(b_i-1)$ route edges where b_i denotes the number of boundary nodes of cell C_i, so we have $r_i = b_i(b_i-1)$. Two other options to store the costs of optimum routes between boundary nodes of a single cell can be found in Figures 3.4(b) and 3.4(c). The route graph in Figure 3.4(b) contains a dummy node and only $r_i = 2b_i$ route edges. The optimum route cost between two boundary nodes is not stored as an edge cost as is the case for the route graph in Figure 3.4(a), but as a rule between a pair of adjacent edges. Similarly, for Figure 3.4(c) the optimum route cost between two boundary nodes is stored as a rule on two edges entering and leaving a dummy node representing the cell. This route graph contains $r_i = 0$ route edges. In Chapter 4 we will discuss the representation of route graphs in more detail. We will also show that optimum routes can be planned with each of these three route graphs.



(a) *A route graph with* $r_i = b_i(b_i-1)$.



(b) *A route graph with* $r_i = 2b_i$.

(c) *A route graph with* $r_i = 0$.

Figure 3.4. *Three ways of storing route edges of a cell with 5 boundary nodes.*

For the three route graphs that we consider, the number of route edges in a cell depends solely on the number of boundary nodes of the cell. Define the route graph containing the route edges of a cell C_i by R_i, i.e. one possibility is to use $R_i = (N_B \cap C_i, \{e | \delta_1(e), \delta_2(e) \in N_B \cap C_i\})$. Figure 3.5 shows the route edges that have to be created for the partition in Figure 3.2 if we create an edge between every pair of boundary nodes in a cell.

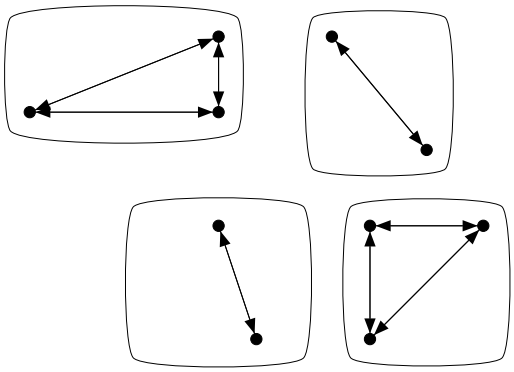


Figure 3.5. *Route edges of the partition in Figure 3.2.*

Now, when a route has to be planned between two nodes in a partition, we create a so-called *searchgraph*. A searchgraph consists of the boundary graph, the cells containing the start node and the destination node and all route edges of all other cells. So, we define the *searchgraph* $G_S(G, \{C_1, \ldots, C_k\}, s, d)$, which we denote by $G_S(s,d)$ or G_S for short, for planning an optimum route between start node s and destination node d as $G_S = C(s) \cup C(d) \cup B \cup \bigcup_{i=1}^{k} R_i \setminus \{R(s) \cup R(d)\}$, where B is the boundary graph, $C(u)$ is the cell containing node u, and $R(u)$ the graph formed by the set of route edges of cell $C(u)$. The searchgraph of the partition for start node s and destination node d is given in Figure 3.6, assuming we use a route graph as in Figure 3.4(a) to store the costs of optimum routes between boundary nodes of a cell.

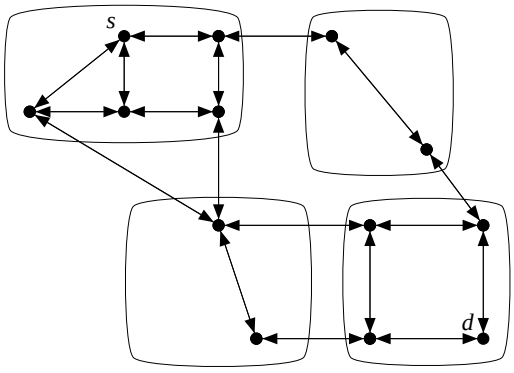


Figure 3.6. *Searchgraph for start node s and destination d.*

Note that route edges represent optimum routes between boundary nodes, so they do not correspond to actual road segments. After a route has been planned, we have to

make sure that every edge in the route is an edge from the original roadgraph. So if the route contains a route edge, this edge has to be replaced by the optimum route that it represents. The resulting route can then be presented to the driver. Notice that first advice can already be given before the route edges are replaced because the first part of the route consists of internal edges of the cell containing the start node.

In the next paragraph, we discuss the quality of a partition, which depends on the size of the searchgraph.

The Quality of a Partition

The most commonly used partitioning criterion in literature is minimizing the number of boundary edges, see for example [Pothen, 1997] and [Falkner, Rendl & Wolkowicz, 1994]. Creating a partition based on this criterion does not only not take the future planning process into account, it also cannot decide on the number of cells that have to be created. A partition containing more cells contains more boundary edges, and is therefore considered to be worse with respect to this partitioning criterion, i.e. the number of boundary edges. This means that the number of cells of the partition has to be given as input to a partitioning algorithm. However, we do not know in advance how many cells have to be created. Obviously, we can select an arbitrary number possibly based on intuition, or knowledge of the area that has to be partitioned. But, using a single criterion that also determines the number of cells in the partition is much more convenient in practice.

Because the running time of a route planning algorithm depends on the number of edges and nodes in the input graph, the running time of our route planning algorithm depends on the number of nodes and edges in the searchgraph G_S. Choosing to minimize the number of nodes in the searchgraph means that we completely ignore the number of route edges that we created, when determining the quality of the partition. However, the number of route edges may have a significant impact on the route planning speed, especially if the number of route edges is very large. Therefore, we minimize the number of *edges* in the searchgraph. Another reason for choosing to minimize the number of edges in the searchgraph is that Winter [2002] shows that for real-life road networks with turn restrictions, i.e. restrictions to make for example a right turn at an intersection, optimum routes can be planned by evaluating edges instead of nodes in a standard route planning algorithm. As a result the route planning speed no longer directly depends on the number of nodes in the searchgraph. So, in order to achieve a fast route planning process, we need to minimize the number of edges in the searchgraph G_S. Note that the searchgraph G_S is different for different start and end nodes, and so is the number of edges in the searchgraph.

We can choose to minimize the maximum number of edges, or the average number of edges in searchgraph G_S for all possible start and destination node pairs. We now explain why minimizing the average number of edges in the searchgraph is preferable to minimizing the maximum number of edges in G_S. First of all, a driver

is more interested in the average performance of a car navigation system, and thus of the route planning, than in its worst-case performance. Furthermore, consider a roadgraph G, with m edges, that is partitioned into two cells. If the start node and destination node are located in different cells, then the searchgraph consists of both cells and the boundary graph. Consequently, the searchgraph is the original roadgraph G. The maximum number of edges of the searchgraph for a partition consisting of two cells, is thus at least m, which is the number of edges of the roadgraph G. The number of edges in a partition with a single cell is equal to m. So, if we minimize the maximum number of edges in G_S, a partition containing two cells is never an improvement over a partition consisting of only one cell. However, in many situations we expect a partition containing two cells to be an improvement over a partition with only one cell, because if both the start and end node are located in the same cell, only one cell has to be searched in detail. Therefore, we choose to minimize the *average* number of edges in the searchgraph.

Note that the average number of edges in the searchgraph G_S is a criterion that can also be used to determine the number of cells in a partition. To mathematically formulate this criterion, we use the notation in Table 3.1.

k	The number of cells in the partition of G, $\{C_1,\ldots,C_k\}$.
n	The number of nodes of roadgraph G.
n_i	The number of nodes in cell C_i.
m_i	The number of edges in cell C_i.
r_i	The number of route edges of cell C_i.
m_B	The number of boundary edges in the partition.

Table 3.1. *Notation used for defining the quality of a partition.*

Definition 3.7. The *average number of edges* in searchgraph G_S for all possible start and destination node pairs is equal to the number of edges in the boundary graph, plus the average number of route edges in all cells except the cells containing the start and destination node for all possible start and destination node pairs, plus the average number of edges in the cells that contain the start and destination node for all possible start and destination node pairs. □

We assume that every node has an equal probability of being chosen as a start node or destination node. This means that the probability of choosing a node in a cell C_i is equal to $\frac{n_i}{n}$. Because a partition is not driver dependent (it has to be created during a pre-processing step and stored on a CD or DVD), this assumption is not unreasonable. Note that if the start and destination node are located in the same cell, the edges of only that cell contribute to the average number of edges in the searchgraph G_S. We denote the average number of edges in the searchgraph of partition $\{C_1,\ldots,C_k\}$ of G with boundary graph B as $AE(G,C_1,\ldots,C_k)$. The average number of edges in the

searchgraph, $AE(G,C_1,\ldots,C_k)$, is equal to (see Appendix A)

$$AE(G,C_1,\ldots,C_k) = \sum_{i=1}^{k} \left\{ \frac{n_i}{n}(2-\frac{n_i}{n})m_i + (1-\frac{n_i}{n})^2 r_i \right\} + m_B. \tag{3.1}$$

Note that our ultimate goal is to minimize the average number of evaluated edges by the route planning algorithm. The average number of edges in the searchgraph is an upperbound for this. The actual average number of evaluated edges is very hard to determine because it depends on various things, such as the planning algorithm, the partition, the planning criterion, the structure of the roadgraph, and the like. Therefore, we approximate the average number of evaluated edges.

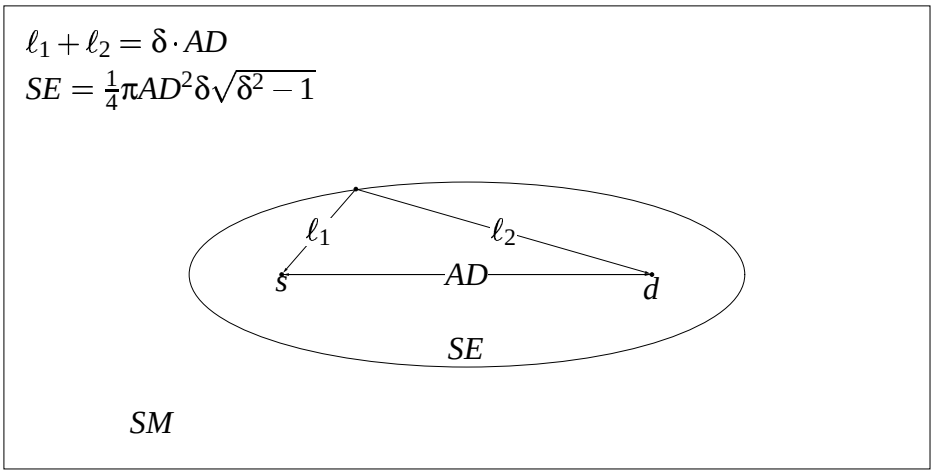


Figure 3.7. *Search area of the route planning algorithm $A^\star$.*

The standard $A^\star$-algorithm generally evaluates edges in an ellipse-shaped area, like in Figure 3.7. We assume the foci of the ellipse are the start node and the destination node. We denote the area of an ellipse by SE, the average Euclidean distance between the start and destination node by AD, and we introduce an average detour factor δ. We assume that the area searched by the route planning algorithm is an ellipse which has an average distance between start and end node equal to $AD > 0$, and a sum of distances from any point on the ellipse to both the start and destination node equal to $\delta \cdot AD$, $\delta > 1$. The area of an ellipse can then be expressed as $SE = \frac{1}{4}\pi AD^2\delta\sqrt{\delta^2-1}$. The area represented by the entire roadgraph is denoted by SM. The number of edges evaluated by the route planning algorithm can now be estimated by the average number of internal edges of the start and end cell plus $\alpha = \frac{SE}{SM}$ times the average number of boundary and route edges in the searchgraph, see equation (3.1). This leads to the following approximation of the average number of evaluated edges by a route planning algorithm, denoted by $EE(G,C_1,\ldots,C_k)$.

The estimated number of evaluated edges, $EE(G,C_1,\ldots,C_k)$, is equal to

$$EE(G,C_1,\ldots,C_k) = \sum_{i=1}^{k} \left\{ \frac{n_i}{n}(2-\frac{n_i}{n})m_i \right\} + \alpha(\sum_{i=1}^{k} \left\{ (1-\frac{n_i}{n})^2 r_i \right\} + m_B). \qquad (3.2)$$

So we can conclude that in order to obtain the best possible partition, we have to solve the following optimization problem, as was confirmed by Van der Horst [2003], see also Section 4.3.

PARTITION_ESTIMATED_EDGES
GIVEN: Roadgraph $G = (N,E,w_e,w_r)$.
QUESTION: Partition G into a number of subgraphs $\{C_1,\ldots,C_k\}$ that minimizes $EE(G,C_1,\ldots,C_k) = \sum_{i=1}^{k} \left\{ \frac{n_i}{n}(2-\frac{n_i}{n})m_i \right\} + \alpha(\sum_{i=1}^{k} \left\{ (1-\frac{n_i}{n})^2 r_i \right\} + m_B)$.

Note that choosing $\alpha = 1$ leads to the minimization of the average number of edges in the searchgraph. Choosing $\alpha = 0$ leads to a partition consisting of cells that contain only a single node. Flinsenberg, Van der Horst, Lukkien & Verriet [2004] compare partitions created by using different values of α. They conclude that smaller values of α lead to partitions consisting of smaller cells.

3.3 Complexity

In this section we prove that PARTITION_ESTIMATED_EDGES is strongly *NP*-hard. To this end, we first define a roadgraph called a *loopclique.*

Definition 3.8. Consider a set of positive integers $\{a_1,\ldots,a_n\}$. Create a roadgraph $LK_n(a_1,\ldots,a_n) = (N,E,w_e,w_r)$ such that $N = \{a_1,\ldots,a_n\}$. Let e_{ij} be an edge such that $\delta_1(e_{ij}) = a_i$ and $\delta_2(e_{ij}) = a_j$. Furthermore, let e'_{ij} be an edge such that $\delta_1(e'_{ij}) = \delta_2(e'_{ij}) = a_i$ for $1 \leq j \leq a_i$. The set of edges of $LK_n(a_1,\ldots,a_n)$ is given by $E = \{e_{ij} | i,j \in \{1,\ldots,n\}, i \neq j\} \cup \{e'_{ij} | i \in \{1,\ldots,n\}, 1 \leq j \leq a_i\}$. We call $LK_n(a_1,\ldots,a_n)$ a *loopclique.* □

Roadgraph $LK_n(a_1,\ldots,a_n)$, see also Figure 3.8, consists of a directed clique of n nodes. So there are two directed edges between every pair of nodes. Furthermore, there are a_i edges from node a_i to itself for every node a_i in the roadgraph.

In order to prove that PARTITION_ESTIMATED_EDGES is strongly *NP*-hard, we recall the strongly *NP*-complete 3-PARTITION problem [Garey & Johnson, 1979].

3-PARTITION
GIVEN: Integers z and B and a set of positive integers $A = \{a_1,\ldots,a_{3z}\}$ such that $B/4 < a_i < B/2$ and $\sum_{i=1}^{3z} a_i = zB$.
QUESTION: Does there exist a collection of sets $\{A_1,\ldots,A_z\}$ with $A_i \subseteq A$, $A_i \cap A_j = \emptyset$ for all $i \neq j$, and $\cup_i A_i = A$, such that $\sum_{a_i \in A_j} a_i = B$, and $|A_j| = 3$ for all j?

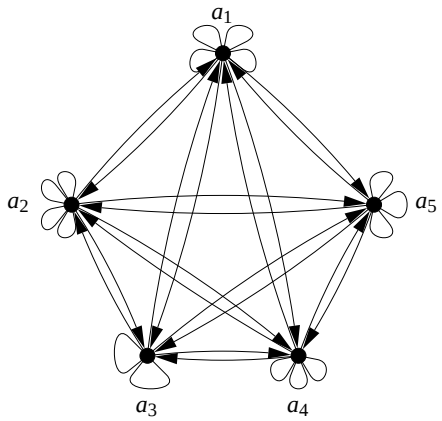


Figure 3.8. *Loopclique* $LK_5(4,4,2,3,3)$.

We define the decision version of PARTITION_ESTIMATED_EDGES for a given number of subgraphs, named *k*-PARTITION_DECISION.

k-PARTITION_DECISION
GIVEN: Roadgraph $G = (N, E, w_e, w_r)$, a positive integer k and rational number s.
QUESTION: Can $G = (N, E, w_e, w_r)$ be partitioned in k subgraphs $\{C_1, \ldots, C_k\}$ such that $EE(G, C_1, \ldots, C_k) \leq s$?

Now we prove that PARTITION_ESTIMATED_EDGES is strongly *NP*-hard.

Theorem 3.1. PARTITION_ESTIMATED_EDGES *is strongly NP-hard.*
Proof. If *k*-PARTITION_DECISION is (strongly) *NP*-complete for any given k, then PARTITION_ESTIMATED_EDGES is (strongly) *NP*-hard for any given k. By solving the problem PARTITION_ESTIMATED_EDGES for different values of k, the problem can be solved for unknown k. Therefore, if PARTITION_ESTIMATED_EDGES is *NP*-hard for a given k, it is also *NP*-hard for unknown k.

For any proposed solution of *k*-PARTITION_DECISION we can check in polynomial time whether it consists of k subgraphs and has $EE(G, G_1, \ldots, G_k) \leq s$. Thus *k*-PARTITION_DECISION is in *NP*. Note that 3-PARTITION is also strongly *NP*-hard without the restriction that $B/4 < a_i < B/2$. In the remainder of this proof, when referring to 3-PARTITION, we consider the problem without the restriction $B/4 < a_i < B/2$. To prove *NP*-completeness we reduce 3-PARTITION to *k*-PARTITION_DECISION. Given an instance $\{a_1, \ldots, a_{3z}\}$ and B of 3-PARTITION, let $G = LK_{3z}(a_1, \ldots, a_{3z})$. Let function $h : \mathbb{R} \longrightarrow \mathbb{R}$ be a continuous function such that $h(x)$ for $x \in \mathbb{N}$ gives the number of route edges in cell C_i given the number of boundary nodes of cell C_i. Furthermore, we set $k = z$ and $s = (2 - \frac{1}{z})(B + 6 -$

$\alpha h(3)) + \alpha 3z(3z-3+\frac{1}{3}h(3))$.

Let $\{A_1,\ldots,A_z\}$ be a collection of node sets with $A_i \cap A_j = \emptyset$ for all $i \neq j$ and $\cup_i A_i = A$. Let $\{C_1,\ldots,C_z\}$ be a partition with C_i the graph induced by the node set A_i. The number of route edges r_i depends solely on the number of boundary nodes of cell C_i.

We have $n_i = |A_i|$, $m_i = |A_i|(|A_i|-1) + \sum_{a_j \in A_i} a_j$, $n = 3z$, $r_i = h(|A_i|)$ and $m_B = \sum_{i=1}^{z} |A_i|(3z-|A_i|)$ for all $i \in \{1,\ldots,z\}$. Define $y_i = \sum_{a_j \in A_i} a_j$, $x_i = |A_i|$ and $c = \frac{|A_i|}{3z}$. We have

$$\begin{aligned}
&EE(G,C_1,\ldots,C_z)\\
&= \sum_{i=1}^{z}\{c(2-c)\{x_i(x_i-1)+y_i\} + \alpha(1-c)^2 h(x_i) + \alpha x_i(3z-x_i)\}\\
&= \sum_{i=1}^{z}\{c(2-c)y_i + (2c-c^2)(x_i^2-x_i) + \alpha h(x_i)(1-2c+c^2) - \alpha x_i^2 + \alpha 3z x_i\}\\
&= \sum_{i=1}^{z}\{(2c-c^2)y_i + (2c-c^2)(x_i^2-x_i) + \alpha h(x_i)(1-2c+c^2) - \alpha x_i^2\} + \alpha 9z^2.
\end{aligned}$$

"$\Rightarrow$" Suppose there exists a solution $\{A_1,\ldots,A_z\}$ to 3-PARTITION. We prove that there also exists a solution to k-PARTITION_DECISION. We create a partition $\{C_1,\ldots,C_z\}$ such that C_i is the graph induced by the nodes of set A_i. Because $x_i = |A_i| = 3$, $y_i = \sum_{a_j \in A_i} a_j = B$, for all i, and $c = \frac{|A_i|}{3z} = \frac{1}{z}$, there exists a partition with

$$\begin{aligned}
&EE(G,C_1,\ldots,C_k)\\
&= \sum_{i=1}^{z}\{(2c-c^2)y_i + (2c-c^2)(x_i^2-x_i) + \alpha h(x_i)(1-2c+c^2) - \alpha x_i^2\} + \alpha 9z^2\\
&= \sum_{i=1}^{z}\{\frac{2B}{z} - \frac{B}{z^2} + \frac{12}{z} - \frac{6}{z^2} + \alpha h(3) - \alpha h(3)\frac{2}{z} + \alpha h(3)\frac{1}{z^2} - \alpha 9\} + \alpha 9z^2\\
&= (2-\frac{1}{z})B + 12 - \frac{6}{z} + z\alpha h(3) - 2\alpha h(3) + \alpha h(3)\frac{1}{z} - \alpha 9z + \alpha 9z^2\\
&= (2-\frac{1}{z})(B+6-\alpha h(3)) + \alpha 3z(3z-3+\frac{1}{3}h(3))\\
&= s.
\end{aligned}$$

"$\Leftarrow$" Assume there exists a partition $\{C_1,\ldots,C_z\}$ of $G = LK_n(a_1,\ldots,a_{3z})$ such that $EE(G,C_1,\ldots,C_z) \leq s$. We prove that there also exists a solution to 3-PARTITION. We construct a set of positive integers $A = \{a_1,\ldots,a_{3z}\}$, and a collection of sets $\{A_1,\ldots,A_z\}$ such that $A_i \subseteq A$ for all $1 \leq i \leq z$, $\cup A_i = A$, and $A_i \cap A_j = \emptyset$ for $i \neq j$. The partition $\{C_1,\ldots,C_z\}$ has an objective value $EE(G,C_1,\ldots,C_z)$ of at most $\sum_{i=1}^{z}\{(2c-c^2)y_i + (2c-c^2)(x_i^2-x_i) + \alpha h(x_i)(1-2c+c^2) - \alpha x_i^2\} + \alpha 9z^2$, with $x_i = |A_i|$, $y_i = \sum_{a_j \in A_i} a_j$, and $c = \frac{x_i}{3z}$. The best possible partition of $LK_n(a_1,\ldots,a_{3z})$ has an

objective value $EE(G,C_1,\ldots,C_z)$ determined by

$$\begin{aligned}\min \quad & \sum_{i=1}^{z}\{\frac{2x_iy_i}{3z}-\frac{x_i^2y_i}{9z^2}+\frac{2x_i^3}{3z}-\frac{2x_i^2}{3z}-\frac{x_i^4}{9z^2}+\frac{x_i^3}{9z^2}\\ & \quad +\alpha h(x_i)-\alpha h(x_i)\frac{2x_i}{3z}+\alpha h(x_i)\frac{x_i^2}{9z^2}-\alpha x_i^2\}+\alpha 9z^2\\ s.t. \quad & \sum_{i=1}^{z}x_i=3z\\ & \sum_{i=1}^{z}y_i=Bz\\ & x_i,y_i \text{ integer }, \forall i\end{aligned}$$

We solve the following minimization problem

$$\begin{aligned}\min \quad & \sum_{i=1}^{z}\{\frac{2x_iy_i}{3z}-\frac{x_i^2y_i}{9z^2}+\frac{2x_i^3}{3z}-\frac{2x_i^2}{3z}-\frac{x_i^4}{9z^2}+\frac{x_i^3}{9z^2}\\ & \quad +\alpha h(x_i)-\alpha h(x_i)\frac{2x_i}{3z}+\alpha h(x_i)\frac{x_i^2}{9z^2}-\alpha x_i^2\}+\alpha 9z^2\\ s.t. \quad & \sum_{i=1}^{z}x_i=3z\\ & \sum_{i=1}^{z}y_i=Bz\end{aligned}$$

Let $\frac{\partial h(x_i)}{\partial x_i}$ denote the first left or right derivative of $h(x_i)$ to x_i. Solving this by using Lagrange multipliers λ_1 and λ_2 leads to

$$\begin{aligned}\alpha(\frac{\partial h(x_i)}{\partial x_i}(1+\frac{x_i^2}{9z^2}-\frac{2x_i}{3z})+h(x_i)(\frac{2x_i}{9z^2}-\frac{2}{3z})-2x_i)+ & \\ +\frac{6x_i^2-4x_i}{3z}-\frac{4x_i^3-3x_i^2}{9z^2}+\frac{2y_i}{3z}-\frac{2x_iy_i}{9z^2}+\lambda_1 &= 0, \forall\, i \qquad (3.3)\\ \frac{2x_i}{3z}-\frac{x_i^2}{9z^2}+\lambda_2 &= 0, \forall\, i \qquad (3.4)\\ \sum_{i=1}^{z}x_i &= 3z \qquad (3.5)\\ \sum_{i=1}^{z}y_i &= Bz \qquad (3.6)\end{aligned}$$

From equation (3.4) it follows that for every i, j
$\frac{2x_i}{3z}-\frac{x_i^2}{9z^2}=\frac{2x_j}{3z}-\frac{x_j^2}{9z^2} \Leftrightarrow \frac{2(x_i-x_j)}{3z}-\frac{x_i^2-x_j^2}{9z^2}=0 \Leftrightarrow \frac{x_i-x_j}{3z}(2-\frac{(x_i+x_j)}{3z})=0 \Leftrightarrow x_i=x_j$ or $6z=x_i+x_j$. Since $\sum_{i=1}^{z}x_i=3z$, we have $x_i=3$ for all i.
Using $x_i=3$, it follows from equation (3.3) that for every i, j

$\frac{2y_i}{3z} - \frac{6y_i}{9z^2} = \frac{2y_j}{3z} - \frac{6y_j}{9z^2} \Leftrightarrow \frac{2(y_i-y_j)}{3z} - \frac{6(y_i-y_j)}{9z^2} = 0 \Leftrightarrow (y_i-y_j)(\frac{2}{3z} - \frac{6}{9z^2}) = 0 \Leftrightarrow y_i = y_j$ or $\frac{2}{3z}(1-\frac{1}{z}) = 0$. Since $z > 1$ and $\sum_{i=1}^{z} y_i = Bz$, we have $y_i = B$ for all i.

Because the optimum values of x_i and y_i also satisfy x_i, y_i integer, we know that the best possible partition of roadgraph G in z cells consists of cells that each contain exactly $x_i = 3$ nodes and $y_i + x_i(x_i-1) = B+6$ edges. Thus the only partitions of G with $EE(G, C_1, \ldots, C_k) \leq s$, are partitions in cells each containing three nodes and $B+6$ edges. So, if there exists a partition $\{C_1, \ldots, C_k\}$ with objective value at most s, then there exists a collection of sets $\{A_1, \ldots, A_z\}$ such that $|A_i| = 3$, $A_i \cap A_j = \emptyset$, for every $i \neq j$, $\cup_{i=1}^{z} A_i = A$ and $\sum_{a_j \in A_i} a_j = B$. The collection of sets $\{A_1, \ldots, A_z\}$ is thus a solution to the 3-PARTITION problem. Because $3z$ nodes and $Bz + 3z(3z-1)$ edges have to be created to construct roadgraph G, the reduction can be performed in a time polynomial in z and B. Since 3-PARTITION is strongly *NP*-complete, k-PARTITION_DECISION is *NP*-complete and therefore PARTITION_ESTIMATED_EDGES is *NP*-hard for any k, and also for unknown k. Note that the rational number s of k-PARTITION_DECISION can be transformed into an integer by multiplying function $EE(G, C_1, \ldots, C_z)$ and s by n^2. This does not influence the ranking of the partitions. Because the specific input values do not effect the problem, we know that PARTITION_ESTIMATED_EDGES is strongly *NP*-hard. □

3.4 Algorithms for Partitioning

In Section 3.1, we introduced the problem of creating a partition that minimizes the estimated expected number of edges to be evaluated by a route planning algorithm. However, because the problem of creating such a partition is strongly *NP*-hard, as we proved in Section 3.3, it is expected that it is impossible to find such a partition efficiently, because it is generally believed that $P \neq NP$. Therefore, this section describes and compares two approximation algorithms that can be used to efficiently create a partition.

The problem PARTITION_ESTIMATED_EDGES is very different from other partitioning problems studied before. First of all, the partitioning criterion is quite different. The size of the cells, the number of edges between cells and the number of boundary nodes of each cell are all taken into account when determining the quality of a partition. Other partitioning problems generally only consider the number (or weight) of edges between cells (for example [Falkner, Rendl & Wolkowicz, 1994]) or the size (or weight) of the cells. The number of boundary nodes in each cell is generally not used in determining the quality of a partition, let alone in combination with the size of the cells and the number of edges between cells. Secondly, the number of cells that have to be created by the approximation algorithm is not known beforehand, and should therefore also be a result of our approximation algorithms. Finally, the algorithms have to be suitable to partition very large real-world road networks

containing millions of nodes and edges.

We believe it is not possible to determine the best possible number of cells beforehand, and then partition the road network into this number of cells. Therefore, both approximation algorithms discussed in this section evaluate at least one partition for every possible number of cells that a partition can contain. We consider increasing and decreasing the number of cells in the partition recursively. First, it is possible to recursively partition a particular cell into a number of subgraphs. Also the reverse method can be used, i.e. starting with a partition in which every cell consists of only one node, and merging a number of cells in each step. Recursively partitioning a cell into a number of subcells forms the basic idea behind the greedy approximation algorithm presented in Section 3.4.1, called the Splitting-Algorithm. Recursively merging cells is the key element in the second greedy approximation algorithm presented in Section 3.4.2. This algorithm is called the Merging-Algorithm. Both algorithms are evaluated in Section 3.4.3.

3.4.1 Splitting-Algorithm

One method to approximate an optimum partition is to recursively split a cell in the partition into a fixed number of subgraphs. Since we do not know in how many cells to divide the selected subgraph ideally, we choose to divide a selected subgraph into a constant number of subgraphs in each step. We choose to split a selected subgraph into two cells because not all numbers of cells can be achieved with 3-partitioning unless certain cells may be empty, and 3-partitioning is more difficult than 2-partitioning. Furthermore, there exist efficient approximation algorithms for Min-Cut with two subgraphs, see [Fiduccia & Matteyses, 1982; Hoffmann, 1994; Kernighan & Lin, 1970; Karger & Stein, 1996]. This leads to a recursive 2-partitioning approximation algorithm. Similar 2-partitioning methods have been developed for Min-Cut as well. However, in Min-Cut the resulting subgraphs all need to have the same number of nodes. This is not needed in our case.

In order to determine the best partition, we recursively divide the cell with most nodes into two, not necessarily equally large, cells. At each step we store the best partition so far. The best partition is the partition with minimum value of $EE(G, C_1, \ldots, C_k)$, defined in Section 3.2. Eventually this results in a total of $|N| = n$ cells, that is, all cells consist of only one node, and the recursion stops. Because we have stored the best partition at each recursion step, we now have an approximation of both the optimum number of cells and the optimum partition.

After choosing the cell C to divide into two subcells, we start with one empty cell C_1, and a cell C_2 equal to C. If cell C_2 is the entire roadgraph, then we randomly select an internal node u from cell C_2, otherwise we randomly select a boundary node u from cell C_2, and move it to cell C_1. We then repeatedly add a boundary node v from cell C_2 adjacent to a node in cell C_1, and move it to cell C_1. The boundary node that is moved to cell C_1 is selected according to a priority function $\Pi_S : N \times N \to \mathbb{R}_0^+$.

This priority function gives for each boundary node v adjacent to a node in cell C_1, the desirability of moving this node to cell C_1. Consequently, we move the node v that has highest priority. Let $m_i(v)$ denote the number of internal edges of cell C_i adjacent to node v. We have $m_i(v) = |\{e \in E(C_i) | \delta_1(e) = v \vee \delta_2(e) = v\}|$. Let $m_B(v)$ denote the number of boundary edges adjacent to node v. We know that $m_B(v) = |\{e \in E_B | \delta_1(e) = v \vee \delta_2(e) = v\}|$. Furthermore, let $\Delta(u,v)$ denote the Euclidean distance between node u and v, and let $random(a_1, a_2)$ be a random real number between a_1 and a_2. We use the following priority function

$$\Pi_S(u,v) = \frac{m_B(v)}{m_i(v)} \frac{1}{(\Delta(u,v))^2} random(1, 1.01).$$

Note that node u is the node which was the first node to be moved from C_2 to C_1.

This priority function is designed such that boundary nodes with relatively many adjacent boundary edges are given a high priority. Note that internal nodes are given a priority equal to 0. Furthermore, in order to obtain cells that are somewhat compact, we divide by the squared Euclidean distance between this node and the node that was first moved to cell C_1. As a result the cells have the tendency to be circularly shaped.

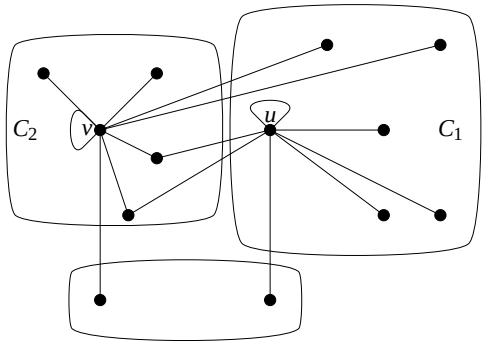


Figure 3.9. *A partition before moving node u to C_2.*

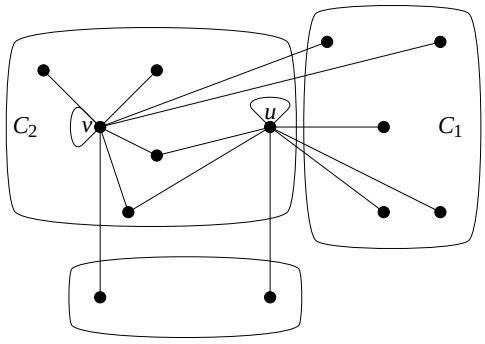


Figure 3.10. *The partition after moving node u to C_2.*

Now, we take a closer look at moving a single node from one cell to another cell. Consider a partition $\{C_1,\ldots,C_k\}$ of graph G with corresponding boundary graph $B = (N_B, E_B, w_e, w_r)$, and cells $C_i = (N_i, E_i, w_e, w_r)$. We denote the set of boundary edges adjacent to a node in cell C_i by $E_B(C_i) = \{e \in E_B | \delta_1(e) \in C_i \vee \delta_2(e) \in C_i\}$. The set of adjacent edges of node u, connecting node u to another node in graph G, is denoted by $\varepsilon(u,G) = \{e \in E(G) \mid (\delta_1(e) = u \wedge \delta_2(e) \neq u) \vee (\delta_2(e) = u \wedge \delta_1(e) \neq u)\}$. It is also possible that edges exist from a node u to itself. These edges in a graph G are denoted by $\gamma(u,G)$, in road networks usually $\gamma(u,G) = \emptyset$. We have $\gamma(u,G) = \{e \in E(G) \mid (\delta_1(e) = u \wedge \delta_2(e) = u)\}$. The set of neighboring nodes of node u in graph G, not including u, is denoted by $\eta(u,G) = \{\delta_1(e), \delta_2(e) \mid \delta_1(e) = u \vee \delta_2(e) = u, e \in E(G)\} \setminus \{u\}$. Let $W_j(u)$ denote the set of boundary nodes of cell C_j that have only node u as neighbor in the boundary graph, so $W_j(u) = \{w \in N_j \mid \eta(w,B) = \{u\}\}$. Let $X_{ij}(u)$ be the set consisting of node u, if u is adjacent to at least one other node in C_i or if not all adjacent boundary nodes of u except node u are part of one single cell. We have

$$X_{ij}(u) = \begin{cases} \{u\} & \text{if } |\varepsilon(u,C_i)| > 0 \vee \eta(u,B) \not\subseteq N_j; \\ \emptyset & \text{otherwise.} \end{cases}$$

Let node u and node v be the two end nodes of a boundary edge $e \in E_B$. Without loss of generality we assume that $u \in N_B(C_1)$ and $v \in N_B(C_2)$. Moving node $u \in C_1$ to graph C_2 of the partition $\{C_1,\ldots,C_k\}$ results in partition $\{\tilde{C}_1, \tilde{C}_2, C_3, \ldots, C_k\}$ with (see Appendix B)

$\tilde{N}_1 = N_1 \setminus \{u\}$,
$\tilde{N}_2 = N_2 \cup \{u\}$,
$\tilde{E}_1 = (E_1 \setminus \gamma(u,C_1)) \setminus \varepsilon(u,C_1)$,
$\tilde{E}_2 = E_2 \cup \gamma(u,C_1) \cup (\varepsilon(u,B) \cap E_B(C_2))$,
$\tilde{E}_B = (E_B \cup \varepsilon(u,C_1)) \setminus (\varepsilon(u,B) \cap E_B(C_2))$,
$N_B(\tilde{C}_1) = (N_B(C_1) \setminus \{u\}) \cup \eta(u,C_1)$,
$N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.

Figure 3.9 illustrates a partitioning before node u is moved from cell C_1 to C_2 and Figure 3.10 gives the resulting partitioning after node u has been moved.

Figure 3.11 gives the pseudo-code of the Splitting-Algorithm. Note that function *Update_partition*(v, B_{i+1}, C_{i+1}, C) moves node v from cell C to cell C_{i+1} and updates cells C and C_{i+1} as well as boundary graph B_{i+1} of the partition. Because the nodes that have to be moved between cells are selected at random, the entire procedure can be repeated several times, selecting the overall best found partition. We call one such iteration a *run*.

From experiments we have seen that the best found partition generally consists of a number of cells which is much smaller than the number of nodes in the graph. So, in order to reduce the running time of the algorithm, a stopping criterium could be added to prevent the algorithm from exploring partitions consisting of cells with only a few nodes.

Splitting-Algorithm

Input: graph $G = (N, E)$.

$B_i = (N_B, E_B)$	:	Boundary graph in step i of the algorithm.
$\{C_1, \ldots, C_i\}$	:	List of constructed cells in step i, with $C_i = (N_i, E_i)$.
$B^\star$	:	Best found boundary graph.
$G^\star = \{C_1^\star, \ldots, C_k^\star\}$	:	Best found matching partition.
$B_i^\star$	:	Best found boundary graph during iteration i.
$G_i^\star = \{C_{i1}^\star, \ldots, C_{ii}^\star\}$	:	Best found matching partition during iteration i.
Obj	:	Best found the objective value during a single iteration.
Opt	:	Overall best found objective value.

Initialization step
$B^\star = B_1^\star = B_1 = (\emptyset, \emptyset)$
$G^\star = G_1^\star = \{C_1\} = \{G\}$
$Opt = E(G)$

Main body of the algorithm

for $i = 1$ **to** $i = N(G) - 1$ **do**
  Select a cell $C \in G_i^\star$ such that $N(C)$ is maximum
  Set $C_{i+1} = (\emptyset, \emptyset)$ and $B_{i+1} = B_i^\star$
  Set $Obj = \infty$
  if $i = 1$ **then**
    Select a node $u \in N$ at random
  else
    Select a boundary node $u \in B_i^\star \cap N(C)$ at random
  end if
  Update_partition(u, B_{i+1}, C_{i+1}, C)
  if $EE(G, C_1, \ldots, C_{i+1}) \leq Obj$ **then**
    $(B_{i+1}^\star, G_{i+1}^\star) = (B_{i+1}, \{C_1, \ldots, C_{i+1}\})$
    $Obj = EE(G, C_1, \ldots, C_{i+1})$
  end if
  while $|N(C \cap B_{i+1})| > 0$
    Select node $v \in C \cap B_{i+1}$ with maximum priority $\Pi_S(u, v)$
    Update_partition(v, B_{i+1}, C_{i+1}, C)
    if $EE(G, C_1, \ldots, C_{i+1}) \leq Obj$ **then**
      $(B_{i+1}^\star, G_{i+1}^\star) = (B_{i+1}, \{C_1, \ldots, C_{i+1}\})$
      $Obj = EE(G, C_1, \ldots, C_{i+1})$
    end if
  end while
  if $Obj \leq Opt$ **then**
    $(B^\star, G^\star) = (B_{i+1}^\star, G_{i+1}^\star)$
    $Opt = Obj$
  end if
end for

Output: $B^\star$, $G^\star$ and Opt.

Figure 3.11. *Pseudo-code of the Splitting-Algorithm.*

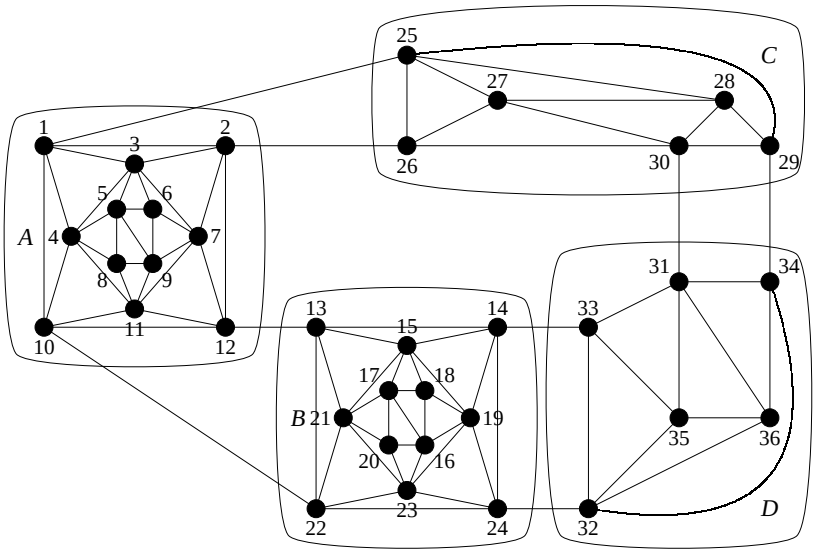


Figure 3.12. *Example of a road network.*

Example 3.1. We now give a specific example to illustrate the Splitting-Algorithm. In order to simplify the discussion of this algorithm, we use $\Delta(u,v) = 1$, for every u and v. Consider the graph G given in Figure 3.12. The graphs within the rounded squares of Figure 3.12 denote possible cells of a cell partition. The algorithm starts by taking a random node as part 1, and the rest of the graph as part 2. Then it determines the resulting boundary graph and selects from the boundary nodes of part 2 the node with maximum priority.

Suppose the algorithm selects node 25 as part 1. Part 2 is then given by $G \setminus \{25\}$. The boundary nodes of part 2 are nodes 1, 26, 27, 28 and 29. All these nodes have one adjacent boundary edge, but the number of adjacent internal edges of part 2 is different for boundary node 1 compared to the other nodes. Node 1 namely has 4 adjacent internal edges in part 2, while all other boundary nodes have only 3 adjacent internal edges in part 2. Therefore, node 1 is certainly not selected to be added to part 1. Suppose node 26 is selected to add to part 1. In the next step nodes 1, 2, 27, 28, 29 and 30 are boundary nodes of part 2. Node 27 has now two adjacent boundary edges, and all other boundary nodes in part 2 have only one. Since node 27 also has only two adjacent internal edges in part 2, this node is selected to add to part 1. Now part 1 consists of three nodes, nodes 25, 26 and 27. In the following iterations node 28, node 30 and node 29 are added to part 1 respectively. Part 1 then consists of C and part 2 of $A \cup B \cup D$. This is a (possibly the first) candidate for the result of the

first iteration of the Splitting-Algorithm.

For other initial nodes we can demonstrate in a similar fashion that if the random initial node u that is selected belongs to part X in Figure 3.12 with $X \in \{A,B,C,D\}$, but node $u \notin \{1,2,10,12,13,14,22,24,29,34\}$ then the Splitting-Algorithm first constructs a partitioning consisting of part X and $G \setminus X$. So as we already argued, if the algorithm selects node 25 as part 1 then it constructs parts C and $A \cup B \cup D$.

The mentioned partitions are candidates for an optimum 2-partition because they are clearly better than any partitioning splitting A, B, C or D in two parts, if we use $\alpha = 1$, and $r_i = b_i(b_i - 1)$. Note that our objective function favors a partitioning consisting of parts of approximately the same number of nodes and edges. The objective values of all promising partitions using $\alpha = 1$ and $r_i = b_i(b_i - 1)$, are listed in Table 3.2. The table also gives the probabilities that a given partition is the best partition of at least two cells that is found by our Splitting-Algorithm. Note that the partitioning consisting of cells A and D and $B \cup C$ is not incorporated into the table because parts B and C are not connected.

Cells	Objective value	Probability
$A \cup B \cup C \cup D$	88.00	-
A, $B \cup C \cup D$	95.67	0
B, $A \cup C \cup D$	95.67	0
C, $A \cup B \cup D$	101.22	0
D, $A \cup B \cup C$	101.22	0
$A \cup B$, $C \cup D$	97.33	0
$A \cup C$, $B \cup D$	73.00	0.0175
A, B, $C \cup D$	67.56	0.3705
A, C, $B \cup D$	71.81	0.0085
B, D, $A \cup C$	71.81	0.0085
C, D, $A \cup B$	100.39	0
A, B, C, D	70.61	0.5950

Table 3.2. *Objective values of partitions of the graph in Figure 3.12.*

As one can see from Table 3.2, the optimum partitioning consisting of 2 cells of the graph in Figure 3.12 consists of the cells $A \cup C$ and $B \cup D$. This partition has objective value 73. The optimum partitioning according to the used objective function consists of cells A, B, and $C \cup D$. □

When the Splitting-Algorithm is applied to the graph in Figure 3.12, it returns, after the first iteration, the partitioning consisting of $A \cup B$ and $C \cup D$ (with probability $\frac{10}{27} \approx 0.3705$), or $A \cup C$ and $B \cup D$ (with probability $\frac{17}{27}$), depending on the selected start node and the nodes selected to add to part 1. If the optimum 2-partitioning is returned, the overall optimum can no longer be found because part C and D are

split in the first iteration and cannot be re-united by the Splitting-Algorithm. If the other (non-optimum) 2-partition is returned, the overall optimum remains achievable. Recursively applying an optimum 2-partitioning algorithm does therefore not necessarily lead to the overall optimum. So, using a non-optimal 2-partitioning algorithm in each iteration can actually outperform applying a 2-partitioning algorithm that is guaranteed to give an optimum result. Therefore, we think a local search algorithm can lead to good results.

After the first partitioning iteration, the algorithm selects the cell with most nodes and recursively tries to partition this cell into two new cells. The best found partition among all iterations is stored. In this example only five possible partitions other than the initial graph are candidates for the best partition found by the Splitting-Algorithm. From Table 3.2 it can be seen what the probability is that a partition is found by the Splitting-Algorithm. Notice that in this example the optimum partition is found with probability $\frac{10}{27}$. All other found partitions are worse than the partition consisting of only a single cell and an empty boundary graph. This is caused by the small number of nodes and edges in the graph in combination with the large number of adjacent edges of all nodes. As a result a partition has relatively many boundary nodes in each cell and the reduction that can be achieved by dividing the graph into several cells is therefore limited.

Note that the Splitting-Algorithm does not necessarily produce an optimum partition if the number of runs goes to infinity, not even for priority function $\Pi_S(u,v) = random(0,1)$. To see this consider graph $LK_9(1,1,1,2,2,2,2,3,4)$ and assume $r_i = 0$ and $\alpha = 1$. In order to reach the optimum partition induced by the nodesets $s_1 = \{1,1,4\}$, $s_2 = \{1,2,3\}$ and $s_3 = \{2,2,2\}$, the Splitting-Algorithm first has to create a cell induced by one of these three nodesets. Note that such a partition has an objective value of 80. The partition induced by nodesets $s_i \cup \{x\}$ and $s \setminus \{s_i \cup \{x\}\}$ has an objective value of $78\frac{10}{81} - \frac{1}{9}x$, with $x \in s \setminus s_i$ and $s = \cup_i s_i$. This means that the Splitting-Algorithm is not able to find the optimum partition in this example.

Let the maximum number of boundary nodes of any cell during a run be denoted by M_B, the maximum degree of a node by M_D, and the maximum number of nodes that should be kept together because of the restriction that rules may only be formulated on pairs of adjacent internal edges by M_R. We implemented the Splitting-Algorithm with a worst-case time-complexity of $O(n^2 \cdot M_D \cdot \log(M_B) + n \cdot M_R)$. Here n and m denote the number of nodes and edges in roadgraph G respectively. Note that for real-world road networks, M_D and M_R are generally small constants. If function $\Pi_S(u,v)$ that determines which node is moved is chosen properly, then the number of boundary nodes of a cell remains limited, and also M_B is a small constant. In this case, the complexity of the Splitting-Algorithm becomes $O(n^2)$.

3.4.2 Merging-Algorithm

In the previous section, we discussed an algorithm that creates a partition by recursively splitting a graph in two parts. An alternative approach is doing the opposite, recursively merging two cells to form a new cell. This forms the basis of our second approximation algorithm, called the Merging-Algorithm, see also [Flinsenberg, Van der Horst, Lukkien & Verriet, 2004].

The Merging-Algorithm is a greedy algorithm that starts by creating n cells that each consist of a single node. Then in each step, it repeatedly selects two cells and merges these two cells into a new cell. This process continues until only a single cell remains, containing the entire graph. This process is called a *run*. During a single run the best found partition is stored. We have randomized the selection of the two cells that are merged together in each step, thereby covering more of the state space with each run. To find a good partition the Merging-Algorithm consists of several runs.

In order to create good partitions, the two cells to merge are selected according to priority function Π_M. A good partition typically contains only a few boundary nodes per cell. We also like to keep the cells roughly equal in size, so that the partition does not contain one very large cell, and a lot of very small cells. This is convenient for a car navigation system because it means there is not too much difference in handling different cells.

Define the *priority* of merging cells C_i and C_j by $\Pi_M(i,j)$. We use

$$\Pi_M(i,j) = \frac{m_B(i,j)(1+b_i+b_j-b_{ij})}{n_i n_j} random(1,1.01),$$

with $m_B(i,j)$ the number of boundary edges between cell C_i and C_j, b_{ij} the number of boundary nodes of the merged cell, and $random(a_1,a_2)$ a random (real) number between a_1 and a_2. This priority function is based on the observation that cells that have many edges between them should be merged. Furthermore, if we can choose between merging two small cells or two large cells with the same number of boundary edges between the two cells, we should merge the two small cells, because compared to the number of internal edges of both cells combined they have relatively the most connecting boundary edges. So, we divide by the number of nodes in each cell to favor small cells. As a result, we are also more likely to end up with cells of approximately equal sizes. Furthermore, we multiply by the reduction in the number of boundary nodes (plus 1) to favor partitions with only a few boundary nodes. Note that the number of boundary nodes of the merged cell can never be larger than the total number of boundary nodes of cells C_i and C_j. So the priority value cannot be negative. By adding 1, we prevent the priority-value from becoming zero for adjacent cells, so $\Pi_M(i,j) > 0$ for adjacent cells. The multiplication with a random number is done to randomize the algorithm and obtain an efficient implementation, see [Van der Horst, 2003]. Note that many alternative priority functions are possible, for example the change in objective value by merging two cells could also be used as priority

function. However, Van der Horst [2003] showed that this priority function does to produce very good results. Several other priority functions have also been tested and evaluated by Van der Horst [2003], which were all shown to lead to partitions of lesser quality.

We consider merging cells C_i and C_j, that is replacing cells C_i and C_j by one new larger cell $C_{ij} = (N_{ij}, E_{ij})$ with boundary nodes B_{ij}. Let $E_B(i,j)$ be the set of (boundary) edges between cells C_i and C_j, and let $N_B(i,j)$ be the set of boundary nodes of cells C_i and C_j that have only adjacent nodes in C_i or C_j. Recall that $\eta(u,G)$ denotes the set of neighboring nodes of node u in G, not including u. Formally
$E_B(i,j) = \{e \in E_B(C_i) \cup E_B(C_j) | \delta_1(e), \delta_2(e) \in N_i \cup N_j\}$,
$N_B(i,j) = \{u \in B_i \cup B_j | \eta(u,G) \subseteq N_i \cup N_j\}$.

This leads to the following changes in the node and edge sets, where $\tilde{N}_B$, $\tilde{E}_B$ denote the new sets of boundary nodes and edges respectively
$N_{ij} = N_i \cup N_j$,
$E_{ij} = E_i \cup E_j \cup E_B(i,j)$,
$B_{ij} = (B_i \cup B_j) \setminus N_B(i,j)$,
$\tilde{N}_B = N_B \setminus N_B(i,j)$,
$\tilde{E}_B = E_B \setminus E_B(i,j)$.

The resulting partition after merging the cells C_i and C_j from Figure 3.13 into cell C_{ij} can be found in Figure 3.14.

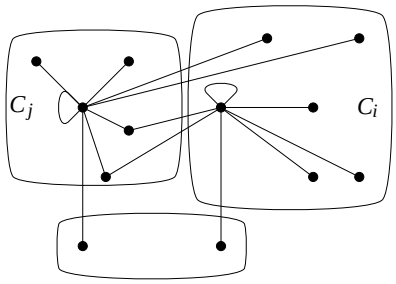


Figure 3.13. *A partition before merging cell C_i and cell C_j.*

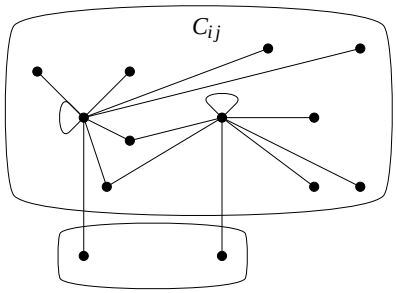


Figure 3.14. *The partition after merging cell C_i and cell C_j into cell C_{ij}.*

Comparing these changes with the changes in node and edge sets caused by moving a node from one cell to another cell, leads us to conclude that merging two cells seems easier than moving a node between cells. Specifically, the change in the value of $EE(G,C_1,\ldots,C_k)$ due to the merging of cells C_i and C_j is given by $\Delta_1(C_i,C_j)$ where r_{ij} denotes the number of route edges of the cell resulting from merging cells C_i and C_j. We have

$$\begin{aligned}\Delta_1(C_i,C_j) &= -\frac{n_i}{n}(2-\frac{n_i}{n})m_i - \frac{n_j}{n}(2-\frac{n_j}{n})m_j \\ &\quad + \frac{n_i+n_j}{n}(2-\frac{n_i+n_j}{n})(m_i+m_j+|E_B(i,j)|) \\ &\quad -\alpha(1-\frac{n_i}{n})^2 r_i - \alpha(1-\frac{n_j}{n})^2 r_j + \alpha(1-\frac{n_i+n_j}{n})^2 r_{ij} - \alpha|E_B(i,j)|.\end{aligned}$$

The pseudo-code of the Merging-Algorithm is given in Figures 3.15 and 3.16. A partition P can be created by calling $P = \mathit{Merging}(G,\Delta_1(C_i,C_j),\mathit{opt})$, with initial objective value $\mathit{opt} = \alpha|E|$, which creates a partition stored in structure P with objective value opt. Note that the function $\mathit{Create\_CellGraph}(CL^\star)$ creates a partition and the matching boundary graph and stores this in P for a given list of cells $CL^\star$. Furthermore, in function $\mathit{Update\_data\_cell}(C_i,C_j,f,\mathit{opt},CL)$, we compute the objective value of the new partition after the merging of cells C_i and C_j from the old objective value to increase the speed of the algorithm by using function $f = \Delta_1(C_i,C_j)$.

Consider again as an example the graph in Figure 3.12. We start by choosing a node pair with highest priority. Because the graph has no multi-edges we randomly choose two adjacent nodes and merge them into one cell. If the two chosen nodes belong to the same part $X \in \{A,B,C,D\}$ in Figure 3.12, there exists a node $u \in X$ that has two adjacent cells in the next iteration, and subsequently a cell containing three nodes is created. Then we again randomly select two adjacent nodes and merge them into one cell. If the two selected nodes do not belong to the same part, we immediately randomly choose two nodes and merge them into one new cell. This process continues, and depending on the cells chosen, the algorithm will in general form cells consisting of 2, 3 or 4 nodes. When the number of cells with a few nodes increases, the algorithm will generally also start creating larger cells. As a result, the Merging-Algorithm is capable of finding much more partitions than the Splitting-Algorithm. In this example, the Merging-Algorithm finds the optimum partition for $r_i = b_i(b_i-1)$ and $\alpha = 1$, which consists of three cells containing part A, B and $C \cup D$ respectively, if it does not select a cell from part A and B, and not from A and C, and not from B and D to merge until this partition is found. In general, unlike the Splitting-Algorithm, the Merging-Algorithm finds an optimum partition if the number of runs goes to infinity for priority function $\Pi_M(i,j) = \mathit{random}(0,1)$.

Example 3.2. To illustrate the Merging-Algorithm, Table 3.3 gives details of a possible run of the Merging-Algorithm for the graph in Figure 3.12. Note that the iteration

Merging-Algorithm

Input: graph $G = (N, E)$, a function f and initial objective value opt.

$u.C$	:	Cell of node $u \in N$.	$u.\eta$	:	Adjacent boundary nodes of node u.
$C.N$	:	Number of nodes in cell C.	$C.E$	:	Number of internal edges in C.
$C.N_B$	:	Number of boundary nodes in C.	$C.E_B$	:	Number of boundary edges of C.
$C.r$	:	Number of route edges of C.	$C.\varepsilon(C_i)$	:	Number of edges between C and C_i.
$C.\Delta$	:	Adjacent cells of C.	$C.\delta$	:	Set of boundary edges of C.
CL	:	List of all non-empty cells in the current iteration.	$CL^\star$	:	Best found list of non-empty cells.
opt	:	Best found objective value so far.	obj	:	Current objective value.
P	:	The found partition.	f	:	Function to determine the increase in objective value by partitioning a cell.

Initialization step
for every $u \in N$ **do**
    $C.N = 1$
    $C.E = 0$
    $C.N_B = 1$
    $C.E_B =$ Number of edges with one end node equal to u
    $C.r = 0$
    $C.\Delta =$ Adjacent nodes of u
    $u.\eta = C.\Delta$
    $u.C = C$
    for every adjacent node v of u **do**
        $C.\varepsilon(v.C) =$ Number of edges between u and v
    end for
    add C to CL
end for
$CL^\star = CL$
$k = |N|$

Main body of the algorithm
while $k \geq 2$ do
    Choose the pair of cells C_i, C_j with maximum priority $\Pi_M(i, j)$
    $obj = Update\_data\_cell(C_i, C_j, f, obj, CL)$
    delete C_j from CL
    if $obj < opt$ **then**
        $opt = obj$
        $CL^\star = CL$
    end if
    $k = k - 1$
end while
$P = Create\_CellGraph(CL^\star)$

Output: Partition P with objective value opt.

Figure 3.15. *Pseudo-code of the Merging-Algorithm.*

Update_data_cell(C_i, C_j, f, obj, CL)

Input: cells C_i, C_j, a function f, objective value obj and list of cells CL.

$u.C$	: Cell of node $u \in N$.	$u.\eta$	: Adjacent boundary nodes of node u.
$C.N$	: Number of nodes in cell C.	$C.E$	: Number of internal edges in C.
$C.N_B$	: Number of boundary nodes in C.	$C.E_B$	: Number of boundary edges of C.
$C.r$	: Number of route edges of C.	$C.\varepsilon(C_i)$	: Number of edges between cells C and C_i.
$C.\Delta$	: Adjacent cells of C.	$C.\delta$	: Set of boundary edges of C.
$r_e(b)$	: Number of route edges for a cell with b boundary nodes.	b	: Number of boundary nodes of the new cell.
		r	: Number of route edges of the new cell.

for every u with $u.C = C_j$ **do** set $u.C = C_i$
for every u with $u.C = C_i$ **do**
    for every $v \in u.\eta$ **do**
        if $v.C = C_i$ **then**
            remove v from $u.\eta$
        end if
    end for
end for
$b = 0$
for every u with $u.C = C_i$ **do**
    if $u.\eta \neq \emptyset$ **then**
        $b = b + 1$
    end if
end for
$r = r_e(b)$
$obj = obj + f(C_i, C_j)$
$C_i.N_B = b$
$C_i.r = r$
$C_i.N = C_i.N + C_j.N$
$C_i.E = C_i.E + C_j.E + C_i.\varepsilon(C_j)$
$C_i.E_B = C_i.E_B + C_j.E_B - 2 * C_i.\varepsilon(C_j)$
$C_i.\varepsilon(C_j) = 0$
$C_j.\varepsilon(C_i) = 0$
for every $C_k \in C_j.\Delta$ with $k \neq i$ **do**
    $C_i.\varepsilon(C_k) = C_i.\varepsilon(C_k) + C_j.\varepsilon(C_k)$
    $C_k.\varepsilon(C_i) = C_i.\varepsilon(C_k)$
end for
for every $C \in C_j.\Delta$ **do**
    if $C \neq C_i$ **then**
        add C to $C_i.\Delta$
    end if
end for
if $C_j \in C_i.\Delta$ **then**
    remove C_j from $C_i.\Delta$
end if

Output: The new cell C_i and the new objective value obj.

Figure 3.16. *Pseudo-code of the updating of data in the Merging-Algorithm.*

number is denoted by "It." and the objective value of the partition by "Obj". Let a node in the graph be denoted by u_i, where i is the number in Figure 3.12. A cell is denoted by its collection of node-numbers, for example the cell consisting of nodes u_1 and u_4 is denoted by $\{1,4\}$.

It.	Cells to merge	Obj	It.	Cells to merge	Obj
1	$\{7\}+\{9\}$	88.89	19	$\{34\}+\{36\}$	110.84
2	$\{11\}+\{7,9\}$	90.52	20	$\{2\}+\{12\}$	111.73
3	$\{22\}+\{23\}$	91.41	21	$\{20\}+\{17\}$	112.63
4	$\{21\}+\{22,23\}$	93.04	22	$\{3,5,6\}+\{7,9,11\}$	114.53
5	$\{30\}+\{31\}$	93.93	23	$\{3,5,6,7,9,11\}+\{8\}$	112.13
6	$\{26\}+\{27\}$	94.83	24	$\{3,5,\ldots,9,11\}+\{1,4,10\}$	104.02
7	$\{25\}+\{26,27\}$	96.45	25	$\{1,3,\ldots,11\}+\{2,12\}$	96.62
8	$\{32\}+\{35\}$	97.35	26	$\{24\}+\{21,22,23\}$	99.63
9	$\{33\}+\{32,35\}$	98.98	27	$\{17,20\}+\{16,18,19\}$	105.85
10	$\{1\}+\{4\}$	99.87	28	$\{16,\ldots,20\}+\{21,22,23,24\}$	97.43
11	$\{10\}+\{1,4\}$	101.50	29	$\{16,\ldots,24\}+\{13,14,15\}$	80.61
12	$\{3\}+\{6\}$	102.39	30	$\{32,33,35\}+\{34,36\}$	86.83
13	$\{5\}+\{3,6\}$	104.02	31	$\{30,31\}+\{32,\ldots,36\}$	81.49
14	$\{13\}+\{14\}$	104.91	32	$\{28,29\}+\{25,26,27\}$	87.72
15	$\{15\}+\{13,14\}$	106.54	33	$\{25,\ldots,29\}+\{30,\ldots,36\}$	67.56
16	$\{18\}+\{19\}$	107.43	34	$\{13,\ldots,24\}+\{25,\ldots,36\}$	75.67
17	$\{16\}+\{18,19\}$	109.06	35	$\{13,\ldots,36\}+\{1,\ldots,12\}$	88.00
18	$\{28\}+\{29\}$	109.95			

Table 3.3. *A detailed run of the Merging-Algorithm for the graph in Figure 3.12.*

In the first iteration two adjacent nodes are selected randomly. Suppose the selected nodes are nodes 7 and 9. After they have been merged to form one cell, there exist two nodes that have 2 adjacent boundary edges. Merging one of these nodes to this cell creates a cell of 3 nodes. The priority of merging these cells is $\frac{2(1+1+2-3)}{2\cdot 1}random(1,1.01) = random(1,1.01)$. The priority of merging two adjacent nodes is $\frac{1(1+1+1-2)}{1\cdot 1}random(1,1.01) = random(1,1.01)$. Suppose we create a cell containing three nodes. Subsequently merging another node with two adjacent edges to this cell does not outperform merging two random adjacent nodes because the new cell would get value $\frac{2(1+3+1-4)}{3\cdot 1}random(1,1.01) = \frac{2}{3}random(1,1.01)$. After iteration 21 in our example, it becomes impossible to merge two single nodes or merge a cell consisting of two nodes with a single node. However, it is possible to remove a boundary node by merging two cells. This now becomes the best possible action. In iteration 22 in our example boundary node 6 becomes an internal node by merging cell $\{3,5,6\}$ and $\{7,9,11\}$. Then it is optimal to remove another boundary

node from the cell $\{3,5,6,7,9,11\}$ by merging this cell with node 8. In iteration 25 part A is created. For the run in Table 3.3, the optimum partition for $r_i = b_i(b_i - 1)$ and $\alpha = 1$ consists of three cells and is found after 33 iterations. □

Note that Table 3.3 gives just one possible run of the Merging-Algorithm. Of course many alternative runs are possible. We do know however, that the Merging-Algorithm finds partitions consisting of k cells in iteration $n-k$, where n denotes the total number of nodes in the input graph.

Let the maximum number of boundary nodes of any cell during a run be denoted by M_B and the maximum number of neighboring cells of any cell during a run by M_C. The Merging-Algorithm was implemented with a worst-case time-complexity of $O(m + n \cdot M_C \cdot (M_B + \log(m))$, see [Van der Horst, 2003]. Here n and m denote the number of nodes and edges in roadgraph G respectively. Note that for real-world road networks, M_C is generally a small constant. If function $\Pi_M(i,j)$ that determines which cells are merged is chosen properly, then the number of boundary nodes of a cell remains limited, and also M_B is a small constant. In this case, the complexity of the Merging-Algorithm becomes $O(m + n\log(m))$.

3.4.3 Computational Evaluation

In order to test the Splitting- and Merging-Algorithm, we use some maps that are used by Siemens VDO I IS DCE. Specifically, we tested the algorithms on the maps of Sophia Antipolis (France), Eindhoven (Netherlands), Siegen (Germany), Giessen/Wetzlar (Germany), Antwerp (Belgium) and the Netherlands. The details of the used maps can be found in Table 3.4. Sophia Antipolis is a small town located in the south of France. Eindhoven is one of the largest cities in the Netherlands, and it is located in the south of the Netherlands. The map of Siegen contains not only the city of Siegen which is located in the west of Germany, but also an area surrounding the city. Giessen/Wetzlar contains the cities Giessen and Wetzlar, as well as some surrounding area. Both Giessen and Wetzlar are also located in the west of Germany. The map of Antwerp contains not only the city of Antwerp but the entire province of Antwerp, which is located in the north of Belgium. The map of the Netherlands contains the entire road network of the Netherlands.

We let each algorithm execute 5 runs. In order to determine the proper value of α, we use the area of the map which is also put in Table 3.4, and the average route length of planned routes by the average user of a car navigation system using that particular map. The value of δ is set to 1.3 after consultation with route planning experts working at Siemens VDO Automotive in Eindhoven. The resulting values for α can be found in Table 3.4. We generated the best possible partition using these values of α, for different values of r_i. As indicated in Section 3.2, we use three different route graphs. These three route graphs for a cell with b_i boundary nodes have $r_i = b_i(b_i - 1)$, $r_i = 2b_i$ and $r_i = 0$ route edges respectively. Because $b_i(b_i -$

Map	n	m	SM (km^2)	AD (km)	α	Disjoint parts
Sophia	1,705	2,015	38	5	0.56	7
Eindhoven	11,087	14,696	111	5	0.19	3
Siegen	19,681	23,952	1,290	25	0.41	8
Giessen/Wetzlar	69,460	88,177	4,625	43	0.34	67
Antwerp	76,608	98,564	2,948	32	0.29	49
Netherlands	894,200	1,135,280	41,525	122	0.30	23

Table 3.4. *Available input graphs.*

$1) \geq 2b_i$ if $b_i \geq 3$, and $2b_i \geq 0$, we use $r_i = b_i(b_i-1)$, $r_i = \min\{b_i(b_i-1), 2b_i\}$ and $r_i = 0$ to create a partition. We denote $r_i = b_i(b_i-1)$ by r_i^C, $r_i = \min\{2b_i, b_i(b_i-1)\}$ by r_i^S and $r_i = 0$ by r_i^P. We discuss and motivate the chosen values of r_i in more detail in Section 4.2. Because none of the used maps is connected, the number of cells of the found partition is somewhat misleading. Therefore, the number of unconnected parts besides the major part is added in column 7 of Table 3.4. All these parts consist of only a few, typically between 1 to 4, nodes. The main connected subgraph is not included in the listed number of unconnected parts. As a result, the number of connected cells in the partition can be calculated by subtracting the number of unconnected parts in Table 3.4 from the total number of cells in a partition (see Table 3.5).

The results of the partitioning algorithms can be found in Table 3.5. The values of the found partitions, $EE(G, C_1, \ldots, C_k)$, are put together in columns 3 and 4 of Table 3.5. Columns 5 and 6 give the number of boundary edges of the found partitions. The number of cells of the best found partition can be found in columns 7 and 8. Columns 9 and 10 show the computation time in seconds needed to execute 5 runs.

Consider for example Figure 3.17. This figure shows the map of the city of Eindhoven, partitioned by the Merging-Algorithm in 16 cells, using $r_i = b_i(b_i-1)$ and $\alpha = 1$. Two of these 16 cells are not connected to the rest of the city as can be seen from Table 3.4. Each of the cells is given its own color. Furthermore, all boundary edges have the same color and route edges are not displayed. Figures of all partitions of both the Splitting- and Merging-Algorithm for all maps from Table 3.4, using the values of α from Table 3.4 can be found in Appendix C. Flinsenberg [2003a] presents partitions found by the Merging-Algorithm for $\alpha = 1$ for all maps except the one from Antwerp.

Both partitioning methods discussed in Sections 3.4.1 and 3.4.2 iterate on the number of cells of the partition. The difference lies in the fact that the Merging-Algorithm decreases the number of cells in the partition by 1 in each step, while the Splitting-Algorithm increases the number of cells in the partition by 1 in each step. As mentioned in Section 3.4.2, merging two cells seems easier than moving a node

Map (Edges)	r_i	$EE(G,C_1,\ldots,C_k)$		$\lvert E_B \rvert$		k		Computation time (s)	
		Splitting	Merging	Splitting	Merging	Splitting	Merging	Splitting	Merging
Sophia	r_i^C	678.92	621.04	88	82	38	42	0.22	0.06
(2.015)	r_i^S	529.16	463.79	92	80	33	40	0.20	0.07
	r_i^P	283.84	263.47	165	168	71	85	0.19	0.08
Eindhoven	r_i^C	5,406.08	3,467.43	6,531	608	3,526	102	1.98	0.85
(14.696)	r_i^S	1,429.17	1,155.88	958	724	127	131	1.83	0.88
	r_i^P	677.41	591.41	1,438	1,101	269	246	1.72	0.74
Siegen	r_i^C	7,248.60	4,746.82	545	389	83	86	3.87	1.47
(23.953)	r_i^S	2,602.22	2,080.10	746	538	129	128	3.62	1.51
	r_i^P	1,363.74	1,141.29	1,142	803	242	223	3.40	1.39
Giessen/	r_i^C	20,153.16	10,883.02	1,800	834	294	206	15.68	5.82
Wetzlar	r_i^S	5,157.57	3,882.64	1,952	1,218	334	321	15.47	7.12
(88.177)	r_i^P	2,540.90	2,059.67	2,624	1,801	520	512	14.00	7.82
Antwerp	r_i^C	43,594.45	19,459.15	1,526	712	108	104	17.68	6.89
(98.564)	r_i^S	6,629.07	4,865.35	2,971	1,889	278	261	16.60	7.14
	r_i^P	3,114.72	2,486.96	4,265	2,847	463	439	15.60	7.75
Netherlands	r_i^C	417,441.53	98,329.39	16,202	2,958	1,351	189	279.82	104.10
(1.135.280)	r_i^S	34,276.90	19,893.94	15,376	7,382	1,419	668	245.06	116.31
	r_i^P	15,301.45	10,058.06	21,183	11,172	2,145	1,254	232.93	132.27

Table 3.5. *Results of the partitioning algorithms.*

Figure 3.17. *Partition of Eindhoven with the Merging-Algorithm* ($\alpha = 1$ *and* r_i^C).

from one cell to another cell. Merging two cells requires less information on nodes, edges and cells of the partition than moving a node between two cells. Furthermore, the Splitting-Algorithm generally has to move more than one node between cells to split a cell in two subcells while the Merging-Algorithm only needs to merge two cells once. Therefore, we expect that the Merging-Algorithm is faster than the Splitting-Algorithm, which is confirmed by the results in Table 3.5. As can be seen from Table 3.5 the Merging-Algorithm finds a partition faster than the Splitting-Algorithm for all available road networks and all used values of r_i. For all but one of the tested combinations, the Merging-Algorithm is more than twice as fast as the Splitting-Algorithm.

From the results in Table 3.5 it becomes clear that in general, in order to determine a partition with only a few cells, the Splitting-Algorithm needs more computation time than to find a partition with many cells. This can be explained from the fact that the algorithm is implemented in such a way that the best found partition is stored only once by reverting from the last partition, i.e. the partition with only single-node cells, to the best found partition. This requires more time for partitions consisting of only a few cells. On the other hand, the situation for the Merging-Algorithm is exactly the other way around. The Merging-Algorithm needs more computation time to find a partition with many cells than to find a partition with only a few cells. This can be explained from the fact that we did not use this approach for the Merging-Algorithm. The Merging-Algorithm stores each newly best found partition, which takes more time for a partition consisting of many cells.

From the results in Table 3.5 we can see that if the number of route edges per cell increases then the number of cells in the best found partition decreases. This can be explained by the fact that $EE(G, C_1, \ldots, C_k)$ seeks a balance between the num-

Map	Splitting			Merging		
	r_i^C	r_i^S	r_i^P	r_i^C	r_i^S	r_i^P
Sophia	20	36	47	26	31	56
Eindhoven	7	51	102	16	45	100
Siegen	45	68	137	44	81	138
Giessen/Wetzlar	73	136	283	149	197	315
Antwerp	7	133	252	82	143	244
Netherlands	2,752	709	1,408	93	332	624

Table 3.6. *Number of cells for* $\alpha = 1$.

ber of route edges of a cell on one hand and the number of boundary and internal edges of a cell on the other hand. For a cell with many route edges, the number of internal edges has to be large in order to outweigh the number of route edges. This means that if the number of route edges of a cell is large, an optimum partition contains, in general, a few large cells, connected by only a few boundary edges. If the number of route edges is small though, an optimum partition contains, in general, many small cells, connected by many boundary edges. This means that if the value of α is increased, the number of cells in the best found partition decreases. In order to verify this observation, we created partitions using $\alpha = 1$, which indeed lead to partitions containing fewer cells, see Table 3.6. We can also see from the results in Table 3.5 that the Splitting-Algorithm generally constructs partitions containing more cells than the Merging-Algorithm does.

We noted in Section 3.4.2 that the Merging-Algorithm is more flexible in creating partitions than the Splitting-Algorithm. We therefore expect that the Merging-Algorithm finds better partitions than the Splitting-Algorithm. This is confirmed by the results in Table 3.5 from which we can see that the Merging-Algorithm clearly constructs partitions with a lower objective value than the Splitting-Algorithm. Note that the objective value is lower for partitions with $r_i = 0$ than for $r_i = \min\{2b_i, b_i(b_i - 1)\}$, and similarly for partitions with $r_i = \min\{2b_i, b_i(b_i-1)\}$ and $r_i = b_i(b_i-1)$. This is caused by the difference in the definition of the objective value. In Section 4.2, we discuss the different values of r_i, and which value should be chosen.

3.5 Multi-Level Partitioning

The expected number of edges evaluated to plan an optimum route using the partitions from Section 3.5 is not small enough for a car navigation system. Therefore, the expected number of evaluated edges has to decrease. From the partitioning results in Table 3.5, we can see that even for the partitions created using $r_i = 0$, the expected number of evaluated boundary edges is generally small compared to the total expected number of evaluated edges. Specifically, for the partition of the Netherlands

created with the Merging Algorithm using $r_i = 0$, the expected number of evaluated boundary edges is equal to $\alpha \cdot 11,172 = 3,351.60$. The total expected number of evaluated edges is 10,058.06. So most evaluated edges are internal edges. Therefore, the best way to further reduce the number of evaluated edges is to reduce the size of the cells.

In Section 3.4, we discussed two algorithms for creating a partition of a roadgraph. These algorithms partitioned a given roadgraph into an unknown number of cells. Of course, it is also possible to partition a particular cell into subcells, thus creating a *multi-level* partition. This section discusses the issues involved in creating a multi-level partition. Note that multi-level graph partitioning is also frequently used to refer to a multi-level strategy of creating a single-level partition, for example by Preis [1999] and Monien, Preis & Diekmann [2000].

3.5.1 Representing Multi-Level Partitions

An example of a multi-level partition is given in Figure 3.18. The roadgraph C_0 in Figure 3.18 is divided into three cells C_1, C_2 and C_3, and two of these cells are again divided into two subcells, cell C_1 is divided into subcells C_4 and C_5 and cell C_2 is divided into subcells C_6 and C_7. This is a so-called 2-level partition. It contains at least one cell that is partitioned into subcells. We denote the number of levels in a multi-level partition by λ, so in this example we have $\lambda = 2$.

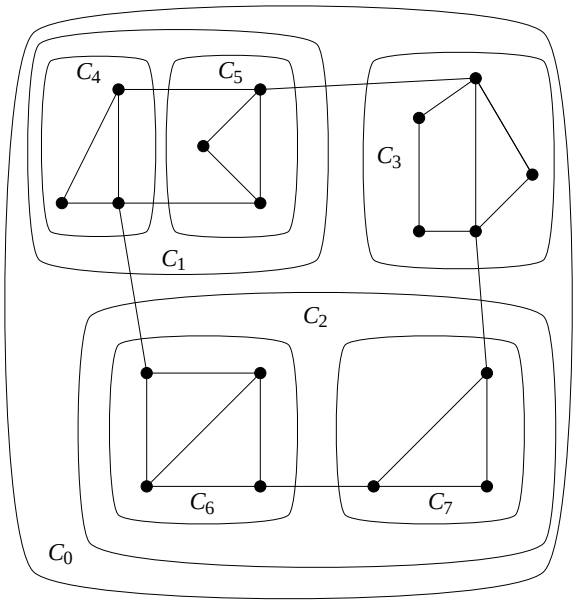


Figure 3.18. *A multi-level partition.*

Definition 3.5.1 gives a formal description of a multi-level partition.

Definition 3.9. Let $C_0 = (N_0, E_0, w_e, w_r)$ be a roadgraph, then a *multi-level partition* of C_0 is given by a set of cells $\{C_1, \ldots, C_k\}$, and a function $\varphi : \{0, \ldots, k\} \rightarrow \mathcal{P}(1, \ldots, k)$, where $\mathcal{P}(S)$ denotes the collection of all subsets of set S. Here $C_i = (N_i, E_i, w_e, w_r)$ is a roadgraph induced by the node set N_i. The *cell-hierarchy function* φ describes the ordering of cells in a multi-level partition. The set of subcells of a cell $C_i = (N_i, E_i, w_e, w_r)$ is denoted by $\varphi(i)$. We have

$$\varphi(i) = \{j | N_j \subset N_i \wedge \nexists q : N_j \subset N_q \subset N_i\}.$$

The collection of cells that contain at least one subcell is denoted by Φ. We have

$$\Phi = \{i | \varphi(i) \neq \emptyset\}.$$

The set of cells $\{C_1, \ldots, C_k\}$ of a multi-level partition should satisfy $N_i \cap N_j = \emptyset$, for every $i, j \in \varphi(q)$, $i \neq j$, $q \in \{0, \ldots, k\}$. Furthermore, $\cup_{j \in \varphi(i)} N_j = N_i$ for all $i \in \Phi$. Finally, for all rules in C_0, such that $w_r(e_1, e_2) > 0$, there may not exist a cell $C_i \in \{C_1, \ldots, C_k\}$ such that $e_1 \in C_i$ and $e_2 \notin C_i$. The edge and rule costs of *cell* C_i are equal to the costs of the corresponding edges and rules in C_0. □

Consider the multi-level partition in Figure 3.18. Here we have $\varphi(0) = \{1, 2, 3\}$, $\varphi(1) = \{4, 5\}$, $\varphi(2) = \{6, 7\}$ and $\varphi(i) = \emptyset$ otherwise. Just as for single-level partitions, the cells of a multi-level partition are connected by *boundary edges*, and the nodes with adjacent edges to other cells are called *boundary nodes*. Unlike the single-level partition, more than one boundary graph exists in a multi-level partition. For every cell C_i that is divided into subcells, these subcells are connected by a set of *boundary edges* that forms the boundary graph belonging to cell C_i. Definition 3.10 gives the definition of a boundary edge of cell C_i for a multi-level partition.

Definition 3.10. Given a cell $C_i = (N_i, E_i, w_e, w_r)$ and a cell-hierarchy function φ, the set of boundary edges E_B^i of the partition of C_i is given by $E_B^i = E_i \setminus \{\cup_{j \in \varphi(i)} E_j\}$. □

The boundary edges connect boundary nodes, just as for single-level partitions. Definition 3.11 gives the formal definition of a boundary node of the partition of a cell C_i.

Definition 3.11. Given a cell $C_i = (N_i, E_i, w_e, w_r)$ and a cell-hierarchy function φ, the set of boundary nodes of a cell C_j, $j \in \varphi(i)$ of the partition of cell C_i is given by $N_B^i(C_j) = \{u \in N(C_j) | \exists e \in E_B^i : u = \delta_1(e) \vee u = \delta_2(e)\}$. □

Definition 3.12 defines the set of boundary graphs for a multi-level partition.

Definition 3.12. Given a multi-level partition $\{C_0, \ldots, C_k\}$ and a cell-hierarchy function φ, the set of *boundary graphs* $\{B_0, \ldots, B_k\}$ of a multi-level partition is defined as $B_i = (N_B^i, E_B^i, w_e, w_r)$ with $N_B^i = \cup_{j \in \varphi(i)} N_B^i(C_j)$ the collection of *boundary nodes* and E_B^i the collection of *boundary edges*. The costs of the edges of boundary graph B_i are equal to the costs of the corresponding edges in C_0, and all rule costs are equal to 0. □

We introduce some additional notation. We denote the number of the cell containing cell C_i as subcell, by $\omega(i)$. We know that $\omega(i) = j$, for all $i \in \varphi(j)$. We call $C_{\omega(i)}$ the *parent* of C_i. The collection of cells that do not contain any subcells is denoted by Φ^c. We have $\Phi^c = \{1,\ldots,k\} \setminus \Phi = \{i|\varphi(i) = \emptyset\}$. Note that the multi-level graphs as studied by [Schulz, Wagner & Zaroliagis, 2002; Holzer, 2003; Holzer, Schulz & Willhalm, 2004] are defined differently and have the disadvantage that additional edges between levels are created. These additional edges can cause route planning with multi-level (or even single-level) partitions to become slower than with the original graph. Using our definition of a multi-level partition, this is not the case. Furthermore, the partitions are created by computing a shortest path tree for every node in the graph, which is unsuitable for large road networks. Note that the partitions defined by Jung & Pramanik [2002] are very similar to ours but do not take turn restrictions into account.

For a multi-level partition, creating the searchgraph is more complicated than for a single-level partition. Consider for example the multi-level partition in Figure 3.18. The searchgraph for start node s and destination node d can be found in Figure 3.19, assuming we create a single edge between every pair of boundary nodes to represent the optimum route between those two boundary nodes (i.e. $r_i = b_i(b_i - 1)$). This searchgraph contains the lowest-level cells containing the start and destination node, the route edges of the other subcells of the parents of the cells containing the start and destination nodes, and the boundary edges of the partitions of the parents of the start and destination nodes.

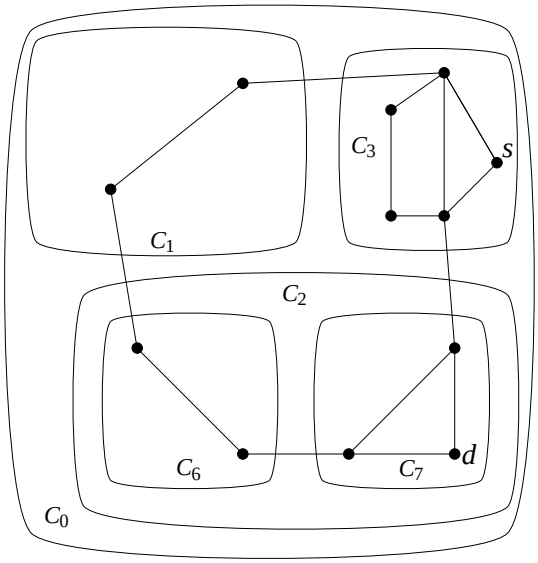


Figure 3.19. *Searchgraph for start node s and destination d for Figure 3.18.*

Mathematically, the average number of edges of the searchgraph of a multi-level partition is equal to

$$AE(\varphi, C_0, \ldots, C_k) = \sum_{i \in \Phi^c} \frac{n_i}{n}(2 - \frac{n_i}{n})m_i + \sum_{i \in \Phi} \frac{n_i}{n}(2 - \frac{n_i}{n})|E_B^i| + \sum_{i=1}^{k} \frac{n_{\omega(i)} - n_i}{n}(1 - \frac{n_i}{n})r_i.$$

Here the term $\sum_{i \in \Phi^c} \frac{n_i}{n}(2 - \frac{n_i}{n})m_i$ represents the contribution of the edges in the lowest-level cells, or equivalently of the cells that do not have any subcells, i.e. all cells in Φ^c, to the average size of the searchgraph. Note that a cell C_i containing n_i nodes contains the start or destination node with probability $\frac{n_i}{n}(2 - \frac{n_i}{n})$. The term $\sum_{i \in \Phi} \frac{n_i}{n}(2 - \frac{n_i}{n})|E_B^i|$ represents the contribution of the boundary edges to the average size of the searchgraph. Only boundary edges of the partition of cells that contain the start or destination node contribute to the size of the searchgraph. In Figure 3.19 the boundary edges of cell C_2 and C_0 are added to the searchgraph. Finally, the term $\sum_{i=1}^{k} \frac{n_{\omega(i)} - n_i}{n}(1 - \frac{n_i}{n})r_i$ represents the contribution of the route edges to the searchgraph. The route edges of a cell C_i are added to the searchgraph if the start or destination node is located in the parent of cell C_i, i.e. in $C_{\omega(i)}$, but not in C_i itself. This happens with probability $\frac{n_{\omega(i)} - n_i}{n}(1 - \frac{n_i}{n})$ for a cell C_i.

Just as for single-level partitions, we would like to minimize the expected number of edges to be evaluated by a route planning algorithm, instead of the average number of edges in the searchgraph. The $A^\star$-algorithm generally evaluates edges in an ellipse-shaped area. This ellipse-shaped area contains the cells containing the start and destination node, a part of the boundary edges of B_0, and a part of the route edges of all highest-level cells. Note that we assume that not all route edges of the highest-level cells are evaluated because the highest-level cells span the entire road network. Because the route edges of lower-level cells are only part of the searchgraph for specific start or destination nodes, we assume that all route edges of lower-level cells contained in the searchgraph are evaluated. For the same reason, we assume all boundary edges in $B_1, \ldots, B_k$ that are contained in the searchgraph are evaluated.

As a result, the expected number of edges evaluated by the route planning algorithm can be expressed as

$$\begin{aligned} EE(\varphi, C_0, \ldots, C_k) &= \sum_{i \in \Phi^c} \frac{n_i}{n}(2 - \frac{n_i}{n})m_i + \alpha(|E_B^0| + \sum_{i=1,\ldots,k:\omega(i)=0} (1 - \frac{n_i}{n})^2 r_i) \\ &+ \sum_{i \in \Phi \setminus \{0\}} \frac{n_i}{n}(2 - \frac{n_i}{n})|E_B^i| + \sum_{i=1,\ldots,k:\omega(i)\neq 0} \frac{n_{\omega(i)} - n_i}{n}(1 - \frac{n_i}{n})r_i. \end{aligned}$$

So, in order to obtain an optimum multi-level partition, we have to find a multi-level partition that minimizes $EE(\varphi, C_0, \ldots, C_k)$. Note that for a single-level partition, the values of $EE(G, C_1, \ldots, C_k)$ and $EE(\varphi, C_0, \ldots, C_k)$ coincide. For a single-level partition, we have $G = C_0$, $\varphi(0) = \{1, \ldots, k\}$, $\varphi(i) = \emptyset$, $i = 1, \ldots, k$, $\Phi = \{0\}$ and

$\Phi^c = \{1,\ldots,k\}$ which gives $E_B^0 = E \setminus \cup_{i=1}^k E_i$ and $\omega(i) = 0$, $i = 1,\ldots,k$.

Determining a multi-level partition can be done by first creating the highest-level partition of the roadgraph, and subsequently partitioning individual cells to obtain a multi-level partition. We call this *Level-Splitting.* The opposite approach can also be used. First create the lowest-level partition, and then repeatedly merge cells to obtain a higher-level partition. This is called *Level-Merging*. Both approaches are discussed in the following section.

3.5.2 Algorithms for Multi-Level Partitions

The easiest way to obtain a multi-level partition, is to create a single-level partition, and then partition its individual cells to obtain a multi-level partition. We call this approach *Level-Splitting.* For partitioning a cell (or the original roadgraph), either the Merging- or the Splitting-Algorithm can be used.

In order to obtain the best possible multi-level partition, $EE(\varphi, C_0,\ldots,C_k)$ should be minimized. However, during the creation of the single-level partition, knowledge of subcells is not available yet. Therefore, we first try to create an optimum single-level partition minimizing $EE(\varphi, C_0,\ldots,C_k)$. Afterwards we try to create each new level optimally. Obviously, creating an optimum single-level partition not necessarily leads to an optimum multi-level partition.

The first-level partition can be created by minimizing $EE(G, C_1,\ldots,C_k)$ as usual, because we have

$$\begin{aligned} EE(\varphi, C_0,\ldots,C_k) &= EE(G, C_1,\ldots,C_k) \\ &+ \sum_{i\in\Phi\setminus\{0\}} \frac{n_i}{n}(2-\frac{n_i}{n})|E_B^i| + \sum_{i=1,\ldots,k:\omega(i)\neq 0} \frac{n_{\omega(i)}-n_i}{n}(1-\frac{n_i}{n})r_i. \end{aligned}$$

Once the first-level partition is known, we determine a lower-level partition of cell C_i by computing the partition such that the expected number of evaluated edges is minimized. We denote the partition of cell C_i in k_2 subcells by $\{C_{k+1},\ldots,C_{k+k_2}\}$. Note that after partitioning cell C_i, we have that $i \in \Phi$. We need to minimize the increase in $EE(\varphi, C_0,\ldots,C_{k+k_2})$, which we denote by $\Delta(C_i, C_{k+1},\ldots,C_{k+k_2})$. This leads to

$$\begin{aligned} \Delta(C_i, C_{k+1},\ldots,C_{k+k_2}) &= \sum_{j=k+1}^{k+k_2} \frac{n_j}{n}(2-\frac{n_j}{n})m_j + \frac{n_i}{n}(2-\frac{n_i}{n})m_B^i \\ &+ \sum_{j=k+1}^{k+k_2} \frac{n_i-n_j}{n}(1-\frac{n_j}{n})r_j - \frac{n_i}{n}(2-\frac{n_i}{n})m_i. \end{aligned}$$

We iteratively partition every cell C_i in the current partition $\{C_1,\ldots,C_k\}$ until $\Delta(C_i, C_{k+1},\ldots,C_{k+k_2}) \geq 0$ for every C_i. If $\Delta(C_i, C_{k+1},\ldots,C_{k+k_2}) < 0$, then partitioning cell C_i into $\{C_{k+1},\ldots,C_{k+k_2}\}$ leads to a partition for which the expected number of evaluated edges decreases, in which case the partitioning of cell C_i into

$\{C_{k+1},\ldots,C_{k+k_2}\}$ is accepted.

The change $\Delta_2(C_i,C_j)$ in the objective value $EE(\varphi,C_0,\ldots,C_k)$ of a multi-level partition when merging two lowest-level cells C_i and C_j with $\omega(i)=\omega(j)$ is equal to

$$\begin{aligned}\Delta_2(C_i,C_j) &= -\frac{n_i}{n}(2-\frac{n_i}{n})m_i-\frac{n_j}{n}(2-\frac{n_j}{n})m_j\\ &+ \frac{n_i+n_j}{n}(2-\frac{n_i+n_j}{n})(m_i+m_j+|E_B(i,j)|)\\ &- \frac{n_{\omega(i)}-n_i}{n}(1-\frac{n_i}{n})r_i-\frac{n_{\omega(j)}-n_j}{n}(1-\frac{n_j}{n})r_j\\ &+ \frac{n_{\omega(i)}-n_i-n_j}{n}(1-\frac{n_i+n_j}{n})r_{ij}\\ &- \frac{n_{\omega(i)}}{n}(2-\frac{n_{\omega(i)}}{n})|E_B(i,j)|.\end{aligned}$$

As indicated before, the number of evaluated boundary edges is generally small compared to the total number of evaluated edges. The best way to further reduce the number of edges evaluated by a route planning algorithm is thus further reducing the size of the cells. The Level-Splitting approach as described above can be used to achieve this. The pseudo-code of the Level-Splitting algorithm can be found in Figure 3.20. By calling $C = \mathit{Level\_Splitting}(G,f,opt)$ with G the original roadgraph, $f=\Delta_1(C_i,C_j)$ and $opt=\alpha|E|$, a multi-level partition stored in the structure C is determined, as well as the matching objective value opt. As can be seen from Figure 3.20, the Level-Splitting Algorithm is a recursive algorithm.

We now discuss an alternative approach, called *Level-Merging* to create multi-level partitions. The alternative approach to Level-Splitting is to first create the lowest-level partition, and then merge cells to create a multi-level partition. This can be achieved by regarding the cells of the lowest-level partition as single nodes. The graph formed by these "cell nodes" can be partitioned using either the Splitting- or Merging-Algorithm.

Unlike in the Level-Splitting algorithm, with Level-Merging we do not know when the highest-level cells are created. Therefore, weighting the number of boundary and route edges of the highest-level cells is more difficult than for the Level-Splitting algorithm. We choose to create the lowest-level partition by minimizing $AE(G,C_1,\ldots,C_k)$, because we assume that more than one level will be created.

Once the lowest-level partition is known, we determine a higher-level partition by computing the partition such that the expected number of edges in the searchgraph is minimized. We denote a partition of graph G in k subcells by $C^2=\{C_1,\ldots,C_k\}$. A higher-level partition of G in k_2 cells is denoted by $C^1=\{C_{k+1},\ldots,C_{k+k_2}\}$.

In order to obtain an optimal multi-level partition, we need to minimize the in-

Algorithm *Level_Splitting*(G, f, opt)

Input: roadgraph G, a function f and initial objective value opt.

$C.Cell$	:	The graph of cell C.
$C.k$	:	Number of cells in the partition of cell C.
$C.B$	:	The boundary graph of the partition of cell C.
$C.Cells$	:	The list of k cells in partition of cell C.
$Cells \rightarrow C[i]$	:	The i'th cell in the partition $Cells$.
obj	:	Best objective value of a multi-level partition during one iteration.
opt	:	Overall best found objective value.
f	:	Function to determine the increase in the objective value of a multi-level partition by partitioning a cell C.

Initialization step

$C^\star.k = 1$
$C^\star.B = (\emptyset, \emptyset)$
$C^\star.Cells = NULL$
$C^\star.Cell = G$

Main body of the algorithm

$C = Merging(C^\star.Cell, f, obj)$.
If $obj < opt$ **then**
    $opt = obj$
    $C^\star = C$
    for $i = 1$ to $C^\star.k$ **do**
        $C_2 = C^\star.Cells \rightarrow C[i]$
        $C = Level\_Splitting(C_2.Cell, \Delta_2, obj)$.
    end for
end if

Output: $C^\star$ and opt.

Figure 3.20. *Pseudo-code of the Level-Splitting Algorithm.*

crease in $EE(\varphi, C_0, \ldots, C_{k+k_2})$, denoted by $\Delta(C^1, C^2)$ with

$$\begin{aligned}\Delta(C^1,C^2) &= \alpha \sum_{i=1}^{k_2} \frac{n_{k+i}}{n}(2-\frac{n_{k+i}}{n})|E_B^{k+i}| - \sum_{i=1}^{k}(1-\frac{n_{\omega(i)}}{n})(1-\frac{n_i}{n})r_i \\ &\quad +\alpha \sum_{i=1}^{k_2} \frac{n_{\omega(k+i)}-n_{k+i}}{n}(1-\frac{n_{k+i}}{n})r_{k+i}.\end{aligned}$$

For Level-Merging, the expected number of evaluated edges $EE(\varphi, C_0, \ldots, C_k)$ decreases only if $\Delta(C^1, C^2) < 0$. So only if $\Delta(C^1, C^2) < 0$, cells are merged to create higher-level cells.

The change in the objective value $EE(\varphi, C_0, \ldots, C_k)$ by merging two highest-level cells C_i and C_j denoted by $\Delta_3(C_i, C_j)$ is equal to

$$\begin{aligned}\Delta_3(C_i,C_j) &= -\frac{n_i}{n}(2-\frac{n_i}{n})|E_B^i| - \frac{n_j}{n}(2-\frac{n_j}{n})|E_B^j| \\ &+ \frac{n_i+n_j}{n}(2-\frac{n_i+n_j}{n})(|E_B^i|+|E_B^j|+|E_B(i,j)|) \\ &- \alpha(1-\frac{n_i}{n})^2 r_i - \alpha(1-\frac{n_j}{n})^2 r_j + \alpha(1-\frac{n_i+n_j}{n})^2 r_{ij} - \alpha|E_B(i,j)|.\end{aligned}$$

The pseudo-code of the Level-Merging algorithm can be found in Figure 3.21. By using the function call $C = Level\_Merging(G, f, opt)$ with G the original roadgraph, $f = \Delta_1(C_i, C_j)$ and $opt = \alpha|E|$ the initial objective value, a multi-level partition stored in structure C is determined, as well as the matching objective value opt. Note that function $Merging\_Partition(C_1, \Delta_3(C_i, C_j), obj)$ is very similar to function $Merging(C_1, \Delta_3(C_i, C_j), obj)$. The difference between the two functions lies in the fact that function $Merging\_Partition(C_1, \Delta_3(C_i, C_j), obj)$ starts from a partition while function $Merging(C_1, \Delta_3(C_i, C_j), obj)$ first creates a partition consisting of single nodes from a graph.

The different order in which the levels are created by Level-Merging and Level-Splitting has important consequences. First of all, for Level-Merging, the cells in the lowest-level are determined first, and the number of lowest-level cells serves as an upperbound on the number of levels in the multi-level partition. Specifically, in worst-case every level in the multi-level partition could contain only one cell that contains subcells. So, if the number of lowest-level cells is denoted by k, a multi-level partition constructed by Level-Merging contains at most $k-1$ levels, where the cell containing all nodes in the roadgraph is not considered to be a level in the multi-level partition. This leads to a worst-case complexity of the Level-Merging algorithm of order $O(n \cdot m + n^2 \cdot M_C \cdot (M_B + \log m))$. For real-world road networks and realistic values of k, it is very unlikely that $k-1$ levels are created. We expect the number of levels to be created by Level-Merging to be rather small. If we assume that the number of levels is a constant, then the worst-case complexity of the Level-Merging Algorithm is given by the complexity of the Merging-Algorithm $O(m + n \cdot M_C \cdot (M_B + \log m))$.

Level-Merging Algorithm

Input: graph $G = (N, E)$, a function f and initial objective value $opt = \alpha|E|$.

$C.k$	:	Number of cells in the partition of cell C.
$C.Cells$	:	The list of k cells in partition of cell C.
$C.B$	:	The boundary graph of the partition of cell C.
$C.Cell$	:	The cell C.
$Cells \rightarrow C[i]$	:	The i'th cell in the partition.
u_i	:	Node i of graph G, $i = 1, \ldots, \|N\|$.
obj	:	Best found the objective value during a single iteration.
opt	:	Overall best found objective value.
f	:	Function to determine the increase in the objective value of a multi-level partition by partitioning a cell C.

Initialization step
$C_1.k = 0$
$C_1.B = (\emptyset, \emptyset)$
$C_1.Cell = G$
$C_1.Cells = NULL$

Main body of the algorithm
$C_2 = Merging(G, f, obj)$
while $obj < opt$ **do**
    for $i = 1$ to $C_1.k$ **do**
        determine which cell C_j in partition C_2 is the parent of C_i in partition C_1
        add C_i to the collection of subcells of C_j
    end for
    for every cell C_i in partition C_2 that has more than one subcell **do**
        Create the boundary graph of the partition of cell C_i
    end for
    $C_1 = C_2$
    $opt = obj$
    $C_2 = Merging\_Partition(C_1, \Delta_3, obj)$.
end while
$C = C_1$

Output: C and opt.

Figure 3.21. *Pseudo-code of the Level-Merging Algorithm.*

For Level-Splitting, the cells in the highest-level of the multi-level partition are determined first. The number of levels in a partition created by Level-Splitting is also limited, in worst-case, but by the maximum number of nodes in a highest-level cell of the partition. This also leads to a worst-case complexity of the Level-Splitting Algorithm of order $O(n \cdot m + n^2 \cdot M_C \cdot (M_B + \log m))$. If we assume the number of levels is given by a small constant then the complexity of the Level-Splitting is also given by the complexity of the Merging-Algorithm. A result of the different order in which the levels of the multi-level partition are constructed is that, if the same function is minimized, the highest-level cells with Level-Splitting correspond to the lowest-level cells with Level-Merging. Finally, Level-Merging does not reduce the size of the lowest-level cells compared to the single-level partition, like Level-Splitting. Because we have shown that it is most useful to decrease the size of the cells, we only consider the Level-Splitting approach in Section 3.5.3.

3.5.3 Computational Evaluation

We use the Level-Splitting algorithm to determine a multi-level partition. Each cell is partitioned with the Merging-Algorithm because this has demonstrated to be the better algorithm for partitioning a graph. We use five runs to partition each cell. The used values of α can be found in Table 3.4. The partitioning results can be found in Table 3.7. For every road network in Table 3.4, a multi-level partition is determined that minimizes the expected number of evaluated edges for three possible values of r_i. For the three multi-level partitions of each road network, the expected number of evaluated edges is determined for three possible values of r_i. The results can be found in columns 3, 4 and 5 of Table 3.7. Column 6 gives for every partition the number of levels of the multi-level partition. Column 7 gives the total number of cells in the multi-level partition, k. The total number of optimum route costs that need to be stored for a particular partition can be found in column 8.

As can be seen from the results in Table 3.7, between 3 and 10 levels are created for each of these partitions. The cost of creating these additional levels can also be found in Table 3.7. The number of cells in the multi-level partition and the number of costs that need to be stored increase drastically compared to a single-level partition. For example, for the single-level partition of the Netherlands created using the Merging-Algorithm using r_i^C we have 189 cells and 249,900 costs to store. For the multi-level partition using the same settings, we have 107,097 cells and 2,205,506 costs to store. The number of cells in a single-level partition can be found in Table 3.5. Because of the increase in the number of cells, many more optimum route costs need to be stored to create a multi-level partition. Specifically, the multi-level partition of the Netherlands using $r_i = b_i(b_i - 1)$ requires storage of about 2,200,000 additional optimum route costs, see also Table 3.7. If we assume that every cost can be stored in 24 bits (this is possible if every cost is at most 16,777,215), this leads to 6.6 MB of data for every cost function. Fortunately, experiments [Van der Horst,

Map (Edges)	r_i	$EE(\varphi, C_0, \ldots, C_k)$			Levels	Cells	Costs to store
		r_i^C	r_i^S	r_i^P			
Sophia (2.015)	r_i^C	432.44	291.95	94.32	4	272	1,846
	r_i^S	451.19	291.08	81.63	3	305	2,584
	r_i^P	645.48	436.70	102.62	3	441	3,442
Eindhoven (14.696)	r_i^C	3,823.30	783.64	290.05	8	1,144	34,400
	r_i^S	4,405.89	865.88	220.51	6	1,508	30,172
	r_i^P	5,583.33	1,257.10	277.23	4	2,649	62,150
Siegen (23.953)	r_i^C	3,646.47	1,040.93	267.49	6	2,140	31,500
	r_i^S	4,404.91	1,260.99	295.39	6	2,394	39,186
	r_i^P	5,374.56	1,789.52	383.88	4	4,416	54,860
Giessen/Wetzlar (88.177)	r_i^C	8,673.94	1,958.24	477.72	7	7,205	123,280
	r_i^S	9,701.29	2,332.73	523.55	10	8,568	171,918
	r_i^P	12,508.19	3,507.93	741.38	5	15,854	244,416
Antwerp (98.564)	r_i^C	15,498.54	2,146.52	618.77	9	9,378	234,572
	r_i^S	21,156.87	3,055.46	668.10	8	11,547	330,514
	r_i^P	26,321.24	4,655.05	967.84	5	20,300	388,506
Netherlands (1.135.280)	r_i^C	73,916.98	6,117.55	1,398.34	9	107,097	2,205,506
	r_i^S	106,798.60	10,866.19	2,229.80	10	123,177	2,935,432
	r_i^P	138,709.28	17,580.06	3,566.75	6	211,502	3,668,006

Table 3.7. *Results of multi-level partitioning.*

2003] have shown that it is possible to reduce the number of route costs that need to be stored considerably (to about 1,000,000 for example), with only a small decrease in quality of the multi-level partition.

Whether it is worth storing that many additional costs can be determined by considering the decrease in the expected number of evaluated edges in Table 3.7. For the three multi-level partitions of each road network, the expected number of evaluated edges is determined for three different values of r_i. The results can be found in Table 3.7. As one can see, optimizing the multi-level partition for $r_i = 0$ produces partitions of worse quality for $r_i = 0$ than optimizing the multi-level partition for $r_i = b_i(b_i - 1)$. This can be explained by the fact that for multi-level partitions it is beneficiary to have large high-level cells connected by only a few edges, and small cells on the lowest level. This is caused by the fact that edges connecting high-level cells are contained in the searchgraph for any start and destination node, while the edges in lower-level cells are only contained in the searchgraph for a limited number of start and destination nodes. Therefore, it is beneficiary to create large high-level cells connected by only a few boundary edges, and small lower-level cells connected by more boundary edges. However, the highest-level cells are first created optimally, and since $r_i = 0$ results in the creation of small cells in a single level (as was shown in Section 3.4.3), this results for $r_i = 0$ in a partition that contains relatively small high-level cells, and even smaller lower-level cells. Using $r_i = b_i(b_i - 1)$ leads to relatively large cells in a single level, and therefore, partitions created by using $b_i(b_i - 1)$ generally have a lower objective value. Of course, the expected number of evaluated edges is smaller for $r_i = 0$ than for $r_i = b_i(b_i - 1)$. Note that the motivation for the different values of r_i is given in Section 4.2.

3.6 Conclusion

In this chapter, we have defined a partitioned roadgraph, and we have shown that creating a partition with the future route planning process in mind leads to a non-standard partitioning problem. Specifically, we want to create a partition that leads to the minimum estimated expected number of edges that will be evaluated by a route planning algorithm. This partitioning problem was shown to be strongly *NP*-hard in Section 3.3. We presented and discussed two approximation algorithms for determining a partition in Section 3.4, the Splitting-Algorithm and the Merging-Algorithm. We have constructed partitions for a number of real-world road networks, and compared the performance of the two algorithms. We have seen that the Merging-Algorithm constructs partitions of higher quality twice as fast as the Splitting-Algorithm for the considered road networks, see also Table 3.5. Although only six road networks were considered, we conclude that the Merging-Algorithm is better suited for creating a partition than the Splitting-Algorithm.

The Splitting- and Merging-Algorithm described in this chapter are not the only

algorithms that can be used to create a partition. For instance, these two algorithms can be combined into a hybrid approximation algorithm that alternates between merging and splitting cells. Specifically, an algorithm can start merging (or splitting) cells until a certain condition is satisfied. Then it can switch to splitting (or merging) cells each time a condition is satisfied. Of course, this process can be repeated several times. Such a hybrid algorithm may be better suited to avoid local optima. Other partitioning algorithms are also possible however. Improvement algorithms which improve a found partition, such as the Kernighan-Lin heuristic [Kernighan & Lin, 1970], and the Fiduccia-Mattheyses heuristic [Fiduccia & Matteyses, 1982], could be modified such that they can be used to further improve the quality of the partition.

In order to determine the use of trying to further improve the quality of partitions, it is necessary to determine the value of an optimum partition. Determining this optimum value is an interesting problem because it not only measures the quality of the partitioning algorithms, but it also tells us how fast routes could be planned using a partitioning approach, assuming the objective value of the partition accurately represents the average speed of the route planning algorithm. Determining the optimum value of a partition is *NP*-hard however, as we proved in Section 3.3. If we can determine a (good) lowerbound of the quality of a partition $EE(G, C_1, \ldots, C_k)$, then we may be able to determine the usefulness of further increasing the quality of a partition.

Furthermore, we assume that the probability that a node is selected as start or destination node is the same for every node. This is not necessarily true. For example, a node could have a larger probability to be selected as start or destination node if it is the end node of an edge with many houses or offices along the corresponding road segment. Also, instead of selecting a start and destination node, a start and destination edge could be chosen. We can assume that every edge is selected with an equal probability. Alternatively, this probability can also depend on characteristics of the edge, such as the number of house numbers, road type, whether it is part of an industrial or a residential area for example. Taking this kind of information into account will most likely lead to a partition with smaller cells in densely populated areas.

From the results of the two partitioning algorithms, we can see that the majority of the expected number of edges evaluated by a route planning algorithm are internal edges. In order to further increase the route planning speed, we should thus reduce the size of the cells in a partition. For this reason we introduced the concept of a multi-level partition in Section 3.5. We also presented the Level-Splitting algorithm for determining a multi-level partition. From the results of this algorithm which can be found in Table 3.7, we can see that the expected number of edges evaluated by a route planning algorithm when planning a route through the Netherlands decreases considerably for the best found multi-level partition, compared to the best

found single-level partition. So, we conclude that the use of multi-level partitions is very useful to increase the route planning speed in a car navigation system. However, because of the limited storage capacity available in car navigation systems, attention should be paid to the number of optimum route costs that need to be stored to be able to plan an optimum route using a multi-level partition.

We have seen that for multi-level partitions, the partitions with least expected number of evaluated edges are created if $r_i = b_i(b_i - 1)$ is used for creating a partition, and $r_i = 0$ is actually used during planning. This is caused by the fact that it is beneficial to create large high-level cells and small lower-level cells, while the highest-level cells are created by creating the best possible single-level partition. In order to create better multi-level partitions, better objective values for creating highest (and also lower) level cells can be developed. Note that in creating a multi-level partition, also a balance should be found between storing edge costs and improving the quality of the partition. In order to practically implement the developed algorithms and use it for planning optimum routes fast in car navigation systems, it is necessary to find data structures to efficiently store and use multi-level partitions.

4

Time-Independent Planning

In Chapter 3 we have discussed an approach to partition a roadgraph into several subgraphs called cells. We presented two algorithms for determining a partition, and compared their performance. In this chapter, we discuss planning optimum time-independent routes in partitioned roadgraphs. Section 4.1 discusses the representation of three possible route graphs in detail, and we show that optimum time-independent routes can be planned using partitions. Section 4.2 focusses on planning optimum routes in single-level partitions. In Section 4.3 we evaluate the efficiency of planning time-independent optimum routes for different partitions. Section 4.4 discusses the planning of optimum routes in multi-level partitions. Conclusions are presented in Section 4.5.

4.1 Representing Route Graphs

After a partition has been created, we need to create route graphs to store the route edges and their costs as discussed in Section 3.2. This section discusses three possible route graphs in Sections 4.1.1, 4.1.2 and 4.1.3, and shows that optimum routes in single-level partitions can be planned using each of these route graphs. Note that the results of this section also apply to multi-level partitions.

4.1.1 Cliques

As stated in Section 3.2, the searchgraph needed to plan a route from a start node to a destination node consists of the cells containing these nodes, the boundary graph of the partition, and the route graphs of all cells not containing the start or destination node. These route graphs are used to represent the minimum-cost route between every pair of boundary nodes in a cell. The most intuitive way of creating these route graphs is to create an edge between every pair of boundary nodes of a single cell. This leads to a so-called *Clique*. Note that the minimum-cost route from boundary node u to v is not always the same as the minimum-cost route from node v to u, because of one-way streets and turn restrictions. Therefore, we create a directed edge between every pair of boundary nodes of a single cell, leading to $b_i(b_i-1)$ route edges for a cell with b_i boundary nodes. Figure 4.1 shows an example of a Clique.

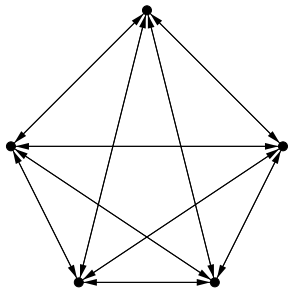


Figure 4.1. *A Clique (with $r_i = b_i(b_i-1)$) for storing the route edges of a cell.*

We still have to define the cost of a route edge, in order to be able to plan a route in the searchgraph. The cost of a route edge e of cell C is set equal to the cost of the optimum route from $\delta_1(e)$ to $\delta_2(e)$ in cell C. Because only the local optimum cost of a route between two boundary nodes of one cell is stored, it is possible that there exists another route between these two nodes that has a lower cost. Such a route has to leave the cell via a boundary node of the cell, and enter the cell again possibly via another boundary node of the cell. An optimum route may even enter and leave the cell in which both boundary nodes are contained several times.

The presence of turn restrictions in a roadgraph complicates the planning of optimum routes in partitioned roadgraphs. We now explain why we have defined a partition such that it contains only rules on pairs of internal edges in Definition 3.1. We assume that an edge e_i in the graph of Figure 4.2 has cost i. A cell is given by the subgraph induced by the nodes contained in a rounded square. Assume the route $\langle e_6, e_1\rangle$ corresponds to a forbidden turn, i.e. $w_{\mathrm{r}}(e_6, e_1) = \infty$. So, in this case this partition does not satisfy the requirements of a partition in Definition 3.1. In this example, the optimum route from node 4 to node 9 is the route $\langle e_6, e_3, e_4, e_{13}\rangle$, with cost 26. The optimum route from node 5 to node 8 is the route $\langle e_1, e_2\rangle$, so the searchgraph contains an edge from node 5 to node 8 with cost 3. Consequently, a standard

route-planning algorithm such as Dijkstra's algorithm [Dijkstra, 1959], planning the optimum route from node 4 to 9 in the searchgraph, finds a route with cost 22. So in this case, planning an optimum route with the searchgraph does not lead to a route with the correct minimum route cost. These problems arise because the optimum route between two nodes contains a route different from the optimum route between two boundary nodes. This is caused by the turn restrictions imposed on a pair of edges, for which not both edges are contained in a single cell or in the boundary graph.

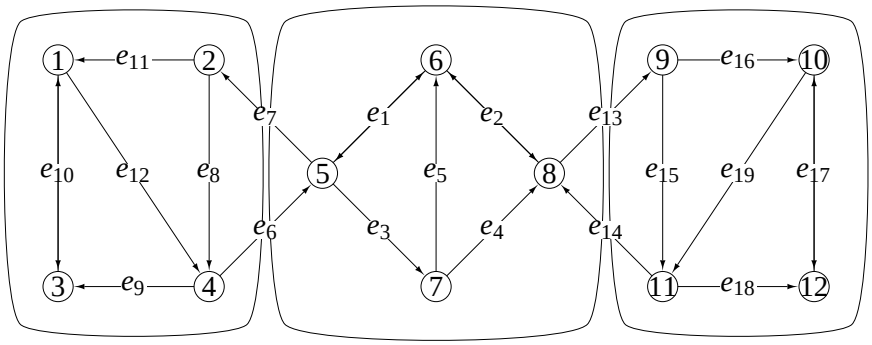


Figure 4.2. *Turn restrictions in a partition.*

Now assume that the turn from edge e_6 to edge e_7 is also forbidden, i.e. $w_r(e_6, e_7) = \infty$, which is also a violation of Definition 3.1. As a result the optimum route from node 4 to node 2, is the (cyclic) route $\langle e_6, e_3, e_5, e_1, e_7 \rangle$ with cost 22. The searchgraph consists of the cell that contains node 4, the boundary graph and the route edges of the other cells. As a result, G_S does not contain nodes 6 and 7, and a route-planning algorithm can only find a route via node 8. To overcome all difficulties with turn restrictions in a partition, we defined a partition such that rules only occur on pairs of internal edges. As a result, every rule of the roadgraph is completely contained in a single cell. This is practically feasible because the number of rules is small compared to the number of edges and nodes in a real-world roadgraph, as can be seen from Table 4.1.

Map	Nodes	Edges	Rules
Sophia	1,705	2,015	89
Eindhoven	11,087	14,696	665
Siegen	19,681	23,952	752
Giessen/Wetzlar	69,460	88,177	2,340
Antwerp	76,608	98,564	3,524
Netherlands	894,200	1,135,280	35,342

Table 4.1. *Number of nodes, edges and rules of the available input graphs.*

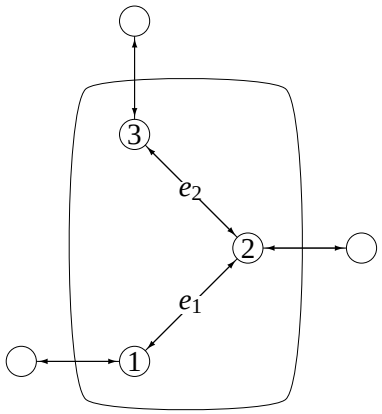


Figure 4.3. *Turn restrictions between route edges.*

We also have to introduce rules between route edges in order to guarantee that optimum routes can be planned. Consider the graph in Figure 4.3 and assume there is a turn restriction between edges e_1 and e_2, i.e. $w_{\mathrm{r}}(e_1, e_2) = \infty$. The routes from node 1 to node 2 and from node 2 to node 3 are feasible, but the route from node 1 to node 3 is infeasible. As a result, we need to create turn restrictions between the route edges that start and end in node 2, to prevent the route from node 1 via node 2 to node 3 from becoming feasible.

Therefore, we also introduce rules $w_{\mathrm{r}}(e_1, e_2) = \infty$, for all route edges e_1 and e_2 in a Clique such that $\delta_2(e_1) = \delta_1(e_2)$. Due to these rules, a feasible route never contains two adjacent route edges because two route edges are only adjacent if they are contained in a single Clique. Definition 4.1 defines a Clique graph that can be used to store the route edges of a single cell.

Definition 4.1. Let $C_i = (N_i, E_i, w_{\mathrm{e}}, w_{\mathrm{r}})$ be a cell of a partition $\{C_1, \ldots, C_k\}$, with boundary graph $B = (N_B, E_B, w_{\mathrm{e}}, w_{\mathrm{r}})$. Define the *Clique graph* K_i of cell C_i as $K_i = (N_i \cap N_B, \{e | \delta_1(e), \delta_2(e) \in N_B \cap N_i\}, w_{\mathrm{e}}, w_{\mathrm{r}})$, with $w_{\mathrm{e}}(e) = w^{\star}_{\mathrm{p}}(C_i, \delta_1(e), \delta_2(e))$, $e \in K_i$ and $w_{\mathrm{r}}(e_1, e_2) = \infty$ if $\delta_2(e_1) = \delta_1(e_2)$, and $w_{\mathrm{r}}(e_1, e_2) = 0$ otherwise. □

An optimum route can be found in a partition that only contains rules between internal edges, despite the only locally optimal routes between boundary nodes of one cell. This is due to the fact that a global optimum route between two boundary nodes (of one cell) consists of boundary edges, route edges, and internal edges of the cell containing both boundary nodes. We first prove that a route between two boundary nodes with cost at most c exists in G if and only if a route between those two boundary nodes with cost at most c exists in B.

Lemma 4.1. *Let $\{C_1, \ldots, C_k\}$ be a partition of G with boundary graph B, and let $s, d \in B$ and $R_i = K_i$, $i = 1, \ldots, k$. There exists a route p from s to d with cost at most c in G if and only if there exists a route p' from s to d with cost at most c in $B \cup \bigcup_{i=1}^{k} R_i$.*

Proof. "⇒" We prove this by induction on the number of boundary edges in the minimum-cost route p between two boundary nodes, which we denote by $m \geq 0$. If $m = 0$ then node s and d are part of one cell, say C_i, and route p only contains edges of cell C_i. Because there exists an edge $e \in R_i$ from s to d with cost $w^\star_{\mathrm{p}}(C_i, s, d)$, we are done. Assume the theorem is satisfied for $m = t$ and let $m = t+1$. Let node $s \in C_i$ and $d \in C_j$. We know that route p leaves cell C_i and enters C_j so let $p = \langle p_1, e, p_2 \rangle$ where p_1 is the sub-route of p from s to node $\delta_1(e)$ with e the last boundary edge leaving C_i in route p and p_2 is the sub-route of p from $\delta_2(e)$ to d. It is possible that $p_1 = \langle\rangle$ and/or $p_2 = \langle\rangle$. Route p_1 now contains at most $m \leq t$ boundary edges, so there exists a route p_1' from s to $\delta_1(e)$ in $B \cup \bigcup_{i=1}^{k} R_i$ with cost $w^\star_{\mathrm{p}}(B \cup \bigcup_{i=1}^{k} R_i, s, \delta_1(e))$. Similarly, route p_2 contains at most $m \leq t$ boundary edges, so there exists a route p_2' from $\delta_2(e)$ to d in $B \cup \bigcup_{i=1}^{k} R_i$ with cost $w^\star_{\mathrm{p}}(B \cup \bigcup_{i=1}^{k} R_i, \delta_2(e), d)$. Because no rules are formulated on edge e, route p can be written as a sequence of minimum-cost routes in $B \cup \bigcup_{i=1}^{k} R_i$ with cost $w_{\mathrm{p}}(p) = w^\star_{\mathrm{p}}(B \cup \bigcup_{i=1}^{k} R_i, s, \delta_1(e)) + w(e) + w^\star_{\mathrm{p}}(B \cup \bigcup_{i=1}^{k} R_i, \delta_2(e), d) = w^\star_{\mathrm{p}}(B \cup \bigcup_{i=1}^{k} R_i, s, d) \leq c$.
"⇐" Let p' be a route in $B \cup \bigcup_{i=1}^{k} R_i$ from s to d with cost at most c. This route consists of boundary and route edges. Every boundary edge also exists in graph G and every route edge represents a minimum-cost route between two boundary nodes in G. Consequently there also exists a route p from s to d in G with cost at most c. □

Now that we have proven optimum routes can still be found between boundary nodes in $B \cup \bigcup_{i=1}^{k} R_i$, it is easy to prove that optimum routes between any two nodes in G can still be found in $G_S(s,d)$.

Theorem 4.1. *Let $\{C_1, \ldots, C_k\}$ be a partition of G with boundary graph B, and let $R_i = K_i$ for $i = 1, \ldots, k$. Let $s \in C_i$ and $d \in C_j$. There exists a route p from s to d in G with cost at most c if and only if there exists a route p' from s to d in $G_S(s,d)$ with cost at most c.*
Proof. Trivial. □

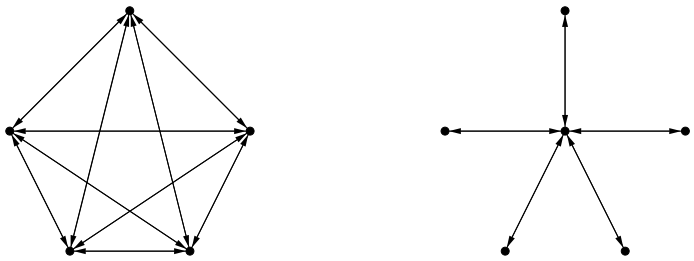
Consequently, if the cost of an optimum route from s to d in graph G is equal to c, then the cost of an optimum route from s to d in graph G_S is also equal to c and vice versa.

4.1.2 Stars

As stated in Section 4.1.1 the most intuitive way of storing the minimum route costs through a cell is by creating two directed edges between all pairs of boundary nodes of a cell with as cost the minimum route cost within that cell. This leads to a quadratic number of edges between these nodes. In particular $b_i(b_i - 1)$ route edges are created for a cell with b_i boundary nodes. In this section, we show that this can be reduced to a linear number of edges, specifically to $\min\{2b_i, b_i(b_i-1)\}$ route edges for a cell with

b_i boundary nodes, see also [Flinsenberg, 2003b]. First, we introduce the structure of the new route graph and then we demonstrate that using this structure, the optimum route costs between every pair of boundary nodes remains the same. Finally, we argue how planning with this new structure relates to planning with Cliques.

To create a Clique, an edge is added between every pair of boundary nodes of a cell, which leads to a graph as in Figure 4.4(a). The $b_i(b_i-1)$ route edges can be reduced to $2b_i$ route edges by storing the optimum route costs in a *Star*. We create a Star by adding a *dummy* node v, and two directed edges between the dummy node and every boundary node, see Figure 4.4(b). This leads to $2b_i$ route edges. Note that for the Clique graph we have b_i boundary nodes per cell, and for the Star graph we have b_i+1 (boundary) nodes per cell.



(a) *A Clique, with* $r_i = b_i(b_i-1)$. (b) *A Star, with* $r_i = 2b_i$.

Figure 4.4. *Two ways of storing route edges of a cell with 5 boundary nodes.*

What remains to be done, is establishing the edge costs of the edges in the Star graph such that the costs of the routes between every pair of boundary nodes remain the same as in the Clique graph. We do that by introducing rules between every pair of adjacent edges in the Star graph. Consider a cell $C_i = (N_i, E_i)$ with b_i boundary nodes. We create a Star graph $S_i = ((N_B \cap N_i) \cup \{w_i\}, \{e | \delta_{j+1}(e) = u, \delta_{2-j}(e) = w_i, u \in N_B \cap N_i, j = 0,1\}, w_e, w_r)$. The cost of each edge in the Star graph is set to half the minimum route cost between all pairs of boundary edges rounded down, i.e. $w_e(e) = \min_{u_1,u_2 \in S_i} \lfloor \frac{1}{2} w^\star_p(C_i, u_1, u_2) \rfloor$ for all $e \in S_i$. We introduce rules $w_r(e_1,e_2) = w^\star_p(C_i,u_1,u_2) - 2w_e(e_1)$, for every pair of boundary nodes u_1 and u_2 with $u_1 = \delta_1(e_1)$, $u_2 = \delta_2(e_2)$, $\delta_2(e_1) = \delta_1(e_2) = w_i$, and $u_1 \neq u_2$. For each boundary node we introduce two more rules to forbid routes consisting of two edges with the same start as destination node. Specifically, we introduce rules $w_r(e_1,e_2) = \infty$, for route edges e_1 and e_2 in S_i such that $u = \delta_1(e_1) = \delta_2(e_2)$, and such that $u = \delta_2(e_1) = \delta_1(e_2)$ for every boundary node $u \in S_i$. These last rules, which forbid certain edge combinations, are introduced for the same reason as the rules in the Clique graph. That is, to prevent forbidden routes from becoming feasible, see Figure 4.3. Definition 4.2 gives the formal definition of a *Star graph*.

Definition 4.2. Let $C_i = (N_i, E_i, w_e, w_r)$ be a cell of a partition $\{C_1, \ldots, C_k\}$, with boundary graph $B = (N_B, E_B, w_e, w_r)$. We define for edges e_1 and e_2 in E_B, $u_1 = \delta_1(e_1)$, $u_2 = \delta_2(e_1)$, $v_1 = \delta_2(e_2)$ and $v_2 = \delta_2(e_2)$. The node set, edge set, edge costs and rules of the *Star graph* S_i of cell C_i are defined as follows

$$\begin{aligned} N(S_i) &= (N_i \cap N_B) \cup \{w_i\}, \\ E(S_i) &= \{e | \delta_{j+1}(e) = u, \delta_{2-j}(e) = w_i, u \in N_B \cap N_i, j = 0, 1\}, \\ w_e(e) &= \min_{u_1, u_2 \in N_i \cap N_B} \lfloor \frac{1}{2} w_p^\star(C_i, u_1, u_2) \rfloor, e \in E(S_i), \end{aligned}$$

and $\forall \langle e_1, e_2 \rangle \in P^{(2)}(S_i)$

$$w_r(e_1, e_2) = \begin{cases} w_p^\star(C_i, u_1, v_2) - 2w_e(e_1) & \text{if } u_2 = v_1 = v \wedge u_1 \neq v_2; \\ \infty & \text{if } u_2 = v_1 = v \wedge u_1 = v_2; \\ \infty & \text{if } u_2 = v_1 \in N(S_i) \setminus \{w_i\}; \\ 0 & \text{otherwise.} \end{cases}$$

A *Star graph* S_i of cell C_i is then defined as $S_i = (N(S_i), E(S_i), w_e, w_r)$. □

Using $R_i = S_i$, the searchgraph for start node s and destination d is given by $G_S(s, d)$. With these edge costs and rules, we have that between every pair of boundary nodes u, v, the minimum route cost is equal to $w_p^\star(C_i, u, v)$ as it is in the Clique graph. This is formally stated in the next lemma.

Lemma 4.2. *Let u and v be two boundary nodes of cell C_i. There exists a route p from u to v in K_i with cost at most c if and only if there is a route p' from u to v in S_i with cost at most c.*

Proof. "⇒" Assume there exists a route p in K_i from $u \in K_i$ to $v \in K_i$ with cost c. If $u = v$ then p is the empty route and the lemma is trivial so assume $u \neq v$. Since S_i contains an edge e_1 from u to dummy node $w \in S_i$ and an edge e_2 from w to v, there exists a route from u to v in S_i with cost $2w_e(e_1) + w_r(e_1, e_2) = w_p^\star(C_i, u, v) \leq c$.
"⇐" Let w_i denote the dummy node of S_i. Assume there exists a route p' from $u \in S_i \setminus \{w_i\}$ to $v \in S_i \setminus \{w\}$ in S_i with cost c. If $u = v$ then p' is the empty route and the lemma is trivial, so assume $u \neq v$. We know $c = 2w_e(e_1) + w_r(e_1, e_2) = w_p^\star(C_i, u, v)$. So there exists a route from u to v in K_i with cost c. □

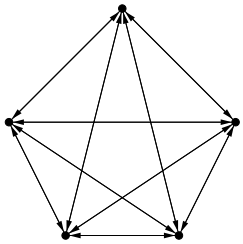
From Lemma 4.2, it follows that Theorem 4.1 remains valid for $R_i = S_i$ instead of $R_i = K_i$.

4.1.3 Points

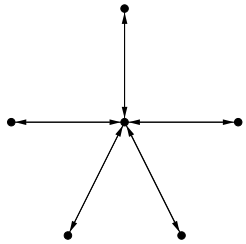
In the previous section, we showed that it is possible to reduce the number of route edges and still plan optimum routes by taking advantage of the presence of turn restrictions in the road network. In this section, we discuss how the costs of optimum routes between boundary nodes can be stored without creating any route edges. We

show that the costs of optimum routes between boundary nodes can be stored completely in turn restrictions between boundary edges, thereby completely eliminating the creation of route edges.

The cost of optimum routes between two boundary nodes can also be stored in a so-called *Point graph*, see Figure 4.5(c). For comparative reasons, we have included the Clique (Figure 4.5(a)) and Star (Figure 4.5(b)) graph as well.



(a) *A Clique, with* $r_i = b_i(b_i - 1)$.



(b) *A Star, with* $r_i = 2b_i$.

(c) *A Point, with* $r_i = 0$.

Figure 4.5. *Three ways of storing route edges of a cell with 5 boundary nodes.*

Once again, we need to define the costs of the Point graph such that the cost of the route between two boundary nodes of cell C_i using Point graphs remains the same as for Clique or Star graphs. This is achieved by creating rules between boundary edges entering and leaving a cell C_i. The formal definition of a Point graph is given in Definition 4.3.

Definition 4.3. Let $C_i = (N_i, E_i, w_e, w_r)$ be a cell of a partition $\{C_1, \ldots, C_k\}$, with boundary graph $B = (N_B, E_B, w_e, w_r)$. Let w_i be a virtual *dummy* node belonging to cell C_i. The node and edge set of the *Point graph* of cell C_i are defined as follows

$$\begin{aligned} N(P_i) &= \{w_i\}, \\ E(P_i) &= \emptyset. \end{aligned}$$

A *Point graph* P_i of cell C_i is then defined as $P_i = (N(P_i), E(P_i))$. We define a function

$d_n : N_B \to N$ that gives for each node u in N_B the corresponding dummy node w_i. We set $d_n(u) = w_i$ for every $u \in N_i \cap N_B$. Furthermore, we define some additional rules on boundary graph B using $u_1 = \delta_1(e_1)$, $u_2 = \delta_2(e_1)$, $v_1 = \delta_1(e_2)$ and $v_2 = \delta_2(e_2)$

$$w_{\mathrm{r}}(e_1,e_2) \quad = \quad \begin{cases} w^{\star}_{\mathrm{p}}(C_i \cup B, u_1, v_2) - w_{\mathrm{e}}(e_1) - w_{\mathrm{e}}(e_2) & \text{if } d_n(u_2) = d_n(v_1) = w_i; \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, end nodes u of boundary edges with $u \in C_i$ need to be modified. First, we define two functions $\delta^o_1 : E_B \to N$ and $\delta^o_2 : E_B \to N$ that give the original start and end node of a boundary edge e respectively. We set $\delta^o_1(e) = \delta_1(e)$ and $\delta^o_2(e) = \delta_2(e)$ for every $e \in E_B$. Now, we can modify the end nodes u of boundary edges with $u \in C_i$. We set $\delta_j(e) = d_n(\delta_j(e))$ for all $e \in E_B$ with $\delta_j(e) \in C_i$, $j = 1,2$. □

(a) *A cell.* (b) *The cell contracted into a Point.*

Figure 4.6. *Contracting a cell into a single node.*

By using Point graphs, we contract a cell into a single node, see Figure 4.6. Using $R_i = P_i$ and the modified boundary graph, the searchgraph for start node s and destination d is given by $G_S(s,d)$. Again, we have that between every pair of boundary nodes u, v, the minimum route cost is equal to $w^{\star}_{\mathrm{p}}(C_i,u,v)$ as it is in the Clique and Star graph. This is formally stated in the next lemma.

Lemma 4.3. *Let u and v be two boundary nodes with $u,v \in N_B \setminus N_i$ adjacent to nodes in C_i. There exists a route p from u to v in $B \cup K_i$ with cost at most c if and only if there is a route p' from u to v in $B \cup P_i$ with cost at most c.*

Proof. "⇒" Assume there exists a route $p = \langle e_1, p_1, e_2 \rangle$ from $u = \delta_1(e_1)$ to $v = \delta_2(e_2)$ in $B \cup K_i$ with cost c. We have $c = w_{\mathrm{e}}(e_1) + w_{\mathrm{e}}(e_2) + w^{\star}_{\mathrm{p}}(C_i \cup B, u, v) - w_{\mathrm{e}}(e_1) - w_{\mathrm{e}}(e_2) = w^{\star}_{\mathrm{p}}(C_i \cup B, u, v)$. Since $d_n(\delta_2(e_1)) = d_n(\delta_1(e_2)) \in C_i$ there exists a route $p = \langle e_1, e_2 \rangle$ in $B \cup P_i$ from u to v. This route has cost $w_{\mathrm{e}}(e_1) + w_{\mathrm{e}}(e_2) + w_{\mathrm{r}}(e_1,e_2) = w_{\mathrm{e}}(e_1) + w_{\mathrm{e}}(e_2) + w^{\star}_{\mathrm{p}}(C_i \cup B, u, v) - w_{\mathrm{e}}(e_1) - w_{\mathrm{e}}(e_2) = w^{\star}_{\mathrm{p}}(C_i \cup B, u, v) = c$.
"⇐" We know that every feasible route p' in $B \cup P_i$ consists of only two edges. Let $p' = \langle e_1, e_2 \rangle$, with $\delta_1(e_1) = u$, $\delta_2(e_2) = v$, and $\delta_2(e_1) = \delta_1(e_2) = w_i$ from u to v with cost c in $B \cup P_i$. We have $c = w_{\mathrm{e}}(e_1) + w_{\mathrm{e}}(e_2) + w_{\mathrm{r}}(e_1,e_2) = w_{\mathrm{e}}(e_1) + w_{\mathrm{e}}(e_2) + w^{\star}_{\mathrm{p}}(C_i \cup B, u, v) - w_{\mathrm{e}}(e_1) - w_{\mathrm{e}}(e_2) = w^{\star}_{\mathrm{p}}(C_i \cup B, u, v)$. So there exists a route from u to v in $B \cup K_i$ with cost c. □

From Lemma 4.3, it follows that Theorem 4.1 remains valid for $R_i = P_i$. It is no longer necessary to create partitions in such a way that rules are only formulated on internal edges, in order to guarantee that optimum routes can be planned. This is due to the fact that we store the optimum route cost of the route from $\delta_1(e_1)$ to $\delta_2(e_2)$ as $w_r(e_1, e_2)$ for $d_n(\delta_2(e_1)) = d_n(\delta_1(e_2))$. However, we do not change our partitioning algorithm to take this into account in order to be able to change the route graphs created for a partition from one type to another. This means that for all partitions, only rules on internal edges exist. If we let go of the requirement that rules are only formulated on internal edges, then the partitioning algorithms can be simplified, the computation time reduces and the quality of the found partitions increases.

Table 4.2 gives for all three route graphs the number of rules that need to be created. In the second column the number of rules with a positive finite cost is given. In the third column the total number of rules with a positive cost is given. Note that it is possible to store only the rules with a positive finite cost and still plan optimum feasible routes. All rules with infinite costs (in a route graph) can be taken into account by letting a route planning algorithm only return edges that lead to a feasible route when it determines all adjacent edges of an edge e. Note that the last row indicates the number of rules for Point graphs for partitions that do not require that rules are only formulated on pairs of internal edges.

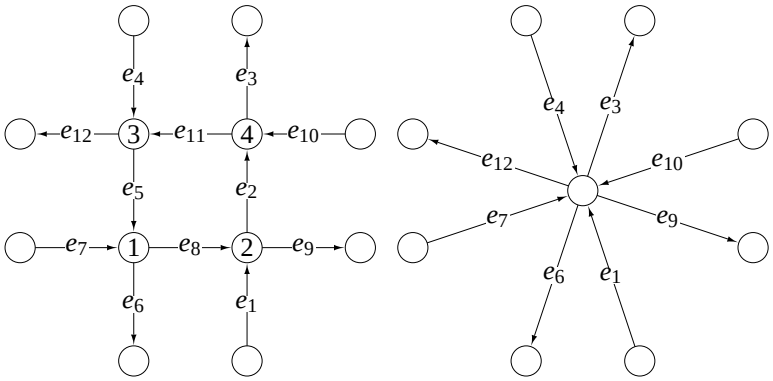
Route graph	Number of finite rules	Total number of rules
Clique	0	$b_i(b_i-1)^2$
Star	$b_i(b_i-1)$	$b_i(b_i+1)$
Point	$\|E_B(C_i)\|(\|E_B(C_i)\|-1)$	$\|E_B(C_i)\|(\|E_B(C_i)\|-1)$
Point$'$	$\|E_B(C_i)\|^2$	$\|E_B(C_i)\|^2$

Table 4.2. *Number of rules for different route graphs.*

Note that from Table 4.2 we can see that the storage requirements (edges and finite rules combined) are of the same order for each of the three route graphs.

In real-world road networks also turn restrictions on pairs of non-adjacent edges exist, which we can formulate by using $w_r : P(G) \to \mathbb{R}_0^+ \cup \infty$. Consider for example the intersection in Figure 4.7(a). A restriction to make a U-turn (for instance a turn from edge e_1 to edge e_6) can be formulated as $w_r(e_1, e_2, e_{11}, e_5, e_6) = \infty$. Note that the introduction of $w_r(e_2, e_{11}) = \infty$ would also cause routes that make a left turn from e_1 to e_{12} to be illegal. Such complex turn restrictions on non-adjacent edges can be handled by creating Points. If the destination node is a node other than nodes $1, \ldots, 4$ then we can replace nodes $1, \ldots, 4$ and edges e_2, e_5, e_8, e_{11} by a Point, see Figure 4.7(b). The rules between edges entering and leaving the intersection (or Point) can be used to guarantee that only legal routes are planned, and that the proper costs are used.

Note that also other route graphs are possible, for example a *Ring* as in Figure 4.8.



(a) *An intersection.* (b) *An intersection as Point graph.*

Figure 4.7. *Formulating an intersection as a Point.*

This structure has the advantage that the degree of all nodes is three. However, it requires the introduction of rules formulated on non-adjacent edges, which complicates the route planning algorithm.

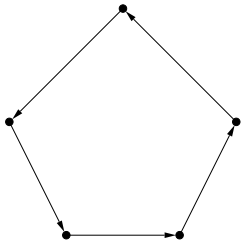


Figure 4.8. *An alternative way of storing the route edges of a cell: the Ring.*

4.2 Algorithms for Time-Independent Optimum Route Planning

This section discusses the planning of optimum time-independent routes in single-level partitions. In Section 2.1 we motivated and explained the use of rules. Rules are used to represent for instance forbidden turns in a roadgraph. These rules influence the planning of optimum routes. Because forbidden turns may be present in a roadgraph, an optimum route in a roadgraph may contain a node more than once. Consider the example in Figure 4.9, and assume that $w_{\mathrm{r}}(e_1, e_2) = \infty$. The optimum route from node 1 to node 6 is the route through all six nodes, which passes through node 2 twice. Note that this cyclic route is the only feasible route from node 1 to node 6.

Because an optimum route may contain a node more than once, a standard node

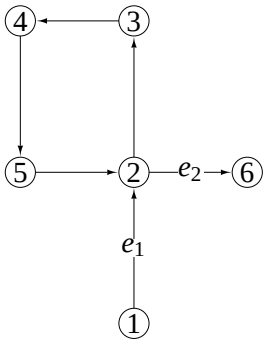


Figure 4.9. *An optimum route with a cycle.*

labelling algorithm such as the standard $A^\star$-algorithm [Hart, Nilsson & Raphael, 1968] cannot be used to determine the optimum route. However, an optimum route does not contain an edge more than once. In order to plan optimum routes in a graph with rules, a modified $A^\star$-algorithm can be used that evaluates edges instead of nodes. Schmid [2000] and Winter [2002] consider turn restrictions on pairs of adjacent edges. Both papers present polynomial-time algorithms that construct optimum routes that may contain nodes more than once. Winter [2002] uses that an optimum route does not contain an edge more than once by constructing the line graph of the roadgraph to plan optimum routes. Szeider [2003] considers the problem of determining whether a simple path exists between two nodes in an undirected simple graph with turn restrictions, where a simple path is a route in which each node appears at most once. He proves that this problem is *NP*-complete if the graph satisfies certain conditions.

We use algorithm $C^\star$ defined in Figure 4.10 to determine an optimum route in a (partitioned) roadgraph G with rules. Algorithm $C^\star$ is a modified $A^\star$-algorithm. The $A^\star$-algorithm is a shortest path algorithm that is based on Dijkstra's algorithm [Dijkstra, 1959]. In order to reduce the number of evaluated edges, the $A^\star$-algorithm [Hart, Nilsson & Raphael, 1968] uses an estimation of the cost from a node to the destination, which is called the h-value. The minimum cost from the start node to a node is called the g-value. The $A^\star$-algorithm selects a node with minimum expected cost from the start node to the destination, i.e. a node with minimum $g+h$-value. Subsequently, all the adjacent nodes of the selected node are determined and their minimum expected costs are updated. The process is repeated until no better route from start node s to destination node d can be found. As explained a modified $A^\star$-algorithm [Hart, Nilsson & Raphael, 1968] can be used to plan an optimum route in a roadgraph with turn restrictions. Because of these turn restrictions, we select the edge with minimum expected cost from the start node to the destination.

Our $C^\star$-algorithm starts by setting all costs equal to infinity, the set of unevaluated edges equal to the edge set of the input graph, and the route equal to the empty route. No edge has a predecessor yet. Furthermore, the cost of the route from start node s to start node s is set to 0, the costs of the edges adjacent to start node s are determined, and their predecessors are set. Subsequently, the algorithm repeatedly selects the edge e from the set of unevaluated edges H with minimum expected cost to the destination (i.e. minimum $g+h$-value), and determines all allowed adjacent edges of edge e. The costs and predecessors of these edges are updated if the minimum found route cost changes. Then edge e is removed from the set of unevaluated edges H. This process continues until the minimum route cost from start node s to destination d is found or until the set of evaluated edges H is empty. If the optimum route cost is found the algorithm determines the matching route by using the stored predecessors, otherwise the algorithm returns the empty route and an infinite cost.

The $A^\star$-algorithm can be used to plan optimum routes if the h-value underestimates the real cost from the current node to the destination, and if the h-value is a so-called dual feasible estimator. Algorithm $C^\star$ also uses a dual feasible estimator h to direct the planning process to the destination. Recall the definition of a dual feasible estimator, see Pearl [1984].

Definition 4.4. A function $h : N \to \mathbb{R}$ is called a dual feasible estimator for roadgraph G if $w_\mathrm{e}(e) \geq h(\delta_1(e)) - h(\delta_2(e))$ for every edge e of roadgraph G. □

This definition can be easily generalized to estimators for edges, as follows.

Definition 4.5. A function $h : E \to \mathbb{R}$ is called a dual feasible estimator for roadgraph G if $w_\mathrm{e}(e_2) \geq h(e_1) - h(e_2) - w_\mathrm{r}(e_1, e_2)$ for all edges e_1, e_2 of roadgraph G such that $\delta_2(e_1) = \delta_1(e_2)$. □

Pearl [1984] proves that a permanent label set by a node labelling algorithm is never modified if the expected remaining cost from a node u to destination d is a dual feasible estimator. This can be easily generalized to roadgraphs with turn restrictions. Note that in our $C^\star$-algorithm a label $g(e)$ of edge e becomes permanent when edge e is selected from set H.

Lemma 4.4. *Let G be a roadgraph. A permanent label set by $C^\star$ is never modified if a dual feasible estimator is used.*

Proof. Note that because roadgraph G contains rules, we have to evaluate edges instead of nodes. So suppose an edge e_2 with a permanent label $\ell(e_2)$ has been found by the algorithm that now receives a lower cost. Then the new label of e_2 is lower than the old one, so $\ell(e_1) + w_\mathrm{e}(e_2) + w_\mathrm{r}(e_1, e_2) < \ell(e_2)$, with $e_1 = \lambda(e_2)$ in algorithm $C^\star$. Because edge e_2 was permanent, we know $\ell(e_2) + h(e_2) \leq \ell(e_1) + h(e_1)$. Because $h(e)$ is a dual feasible estimator, $\ell(e_2) - \ell(e_1) \leq h(e_1) - h(e_2) \leq w_\mathrm{e}(e_2) + w_\mathrm{r}(e_1, e_2)$. This is a contradiction with $w_\mathrm{e}(e_2) + w_\mathrm{r}(e_1, e_2) < \ell(e_2) - \ell(e_1)$. So the label of e_2 cannot have been permanent. Therefore, a permanent label is never modified. □

Algorithm $C^\star$

Input: A roadgraph G, start node s, destination node d.

H : The set of unexpanded edges.
$g(e)$: The minimum found cost of a route from node s to edge e.
$g_{\mathrm{n}}(u)$: The minimum found cost of a route from node s to node u.
$\lambda(e)$: The adjacent edge leading to edge e of the minimum-cost route from s to e.
p : The minimum-cost route from s to d.
$w_{\mathrm{p}}^\star(p)$: The cost of route p.
$h(u)$: The estimated cost from node u to node d.
$h(e)$: The estimated cost from edge e to node d.

Initialization step
$H = E$, $p = \langle\rangle$, $w_{\mathrm{p}}^\star(p) = \infty$
for every $e \in H$ **do**
    $g(e) = \infty$
    $\lambda(e) = \emptyset$
end for
for every $u \in N$ **do** $g_{\mathrm{n}}(u) = \infty$ **end for**
$g_{\mathrm{n}}(s) = 0$
for every $e \in H$ such that $\delta_1(e) = s$ **do**
    $g(e) = w_{\mathrm{e}}(e)$
    $\lambda(e) = e$
end for

Main body of the algorithm
while there is an edge $e \in H$ with $g_{\mathrm{n}}(d) > g(e) + h(e)$ **do**
    Let e be an edge from H with $g(e) + h(e)$ minimum
    for every allowed adjacent edge e_1 of e in G **do**
        if $g(e_1) > g(e) + w_{\mathrm{e}}(e_1) + w_{\mathrm{r}}(e, e_1)$ **then**
            $g(e_1) = g(e) + w_{\mathrm{e}}(e_1) + w_{\mathrm{r}}(e, e_1)$
            $g_{\mathrm{n}}(\delta_2(e_1)) = \min\{g_{\mathrm{n}}(\delta_2(e_1)), g(e_1)\}$
            $\lambda(e_1) = e$
        end if
    end for
    $H = H \setminus \{e\}$
end while
if $g_{\mathrm{n}}(d) < \infty$ **then** $w_{\mathrm{p}}^\star(p) = g(e)$ **and** construct optimum route p from edge e using $\lambda(e)$ **end if**

Output: Optimum route p, and the minimum route cost $w_{\mathrm{p}}^\star(p)$.

Figure 4.10. *Algorithm $C^\star$.*

In algorithm $C^\star$, we use that the used estimator is dual feasible. Note that algorithm $C^\star$ can also be used for unpartitioned roadgraphs, by regarding the roadgraph as a single cell, and the boundary graph as the empty graph. The time-complexity of algorithm $C^\star$ does not depend on the number of rules, but only on the number of edges. Specifically, algorithm $C^\star$ can be implemented in $O(n+m\log m)$ similarly to algorithm $A^\star$, as stated in Theorem 4.2.

Theorem 4.2. *Algorithm $C^\star$ can be implemented with a worst-case time-complexity of order $O(n+m\log m)$, where n and m denote the number of nodes and edges in the searchgraph respectively.*

Proof. Algorithm $C^\star$ is identical to algorithm $A^\star$ up to the fact that edges are selected in each step, and up to the determination of the cost $w_{\mathrm{r}}(e_1,e_2)$ for every two adjacent edges e_1 and e_2. Note that we can choose $w_{\mathrm{r}}(e_1,e_2)$ in such a way that there exists at most one rule cost between every pair of adjacent edges. The selection of an adjacent edge e_2 of edge e_1 and the determination of the cost $w_{\mathrm{r}}(e_1,e_2)$ can be done in a single operation. This can be achieved by storing for each edge e_1 for every adjacent edge a data structure containing (a reference to) the adjacent edge e_2 and (a reference to) the cost $w_{\mathrm{r}}(e_1,e_2)$. Therefore, the time-complexity of algorithm $C^\star$ does not depend on the number of rules, but only on the number of edges. The proof now follows immediately from the fact that the $A^\star$-algorithm can be implemented with a worst-case time-complexity of order $O(n+m\log n)$, see [Johnson, 1972], where n and m denote the number of nodes and edges in the searchgraph respectively. □

To plan an optimum route with the $C^\star$-algorithm, we use a h-value based on the Euclidean distance from a node to the destination. For a node with a geographical location associated with it, the Euclidean distance to the destination can be determined. Because we are concerned with real-world road networks, every node in the roadgraph has a geographical location. We can use the Euclidean distance as h-value if the edge costs are equal to the length of the edge. For edge costs equal to the driving time, the Euclidean distance divided by the overall maximum speed can be used as h-value to plan optimum routes. Note that this applies to the h-value for both edges and nodes.

If a Clique is used as route graph, every node in the searchgraph has a geographical location, so algorithm $C^\star$ in Figure 4.10 can be used to plan optimum routes. When we need to plan a route in a searchgraph that is constructed with Star or Point graphs, an added dummy node in a Star or Point graph does not exist in reality and therefore has no geographic location associated with it. Without a geographic location, the remaining Euclidean distance is not defined, and therefore the h-value cannot be computed for these dummy nodes. We first define the h-value for an edge ending with a dummy node, and then we show that this can be used to plan optimum routes in a searchgraph that contains Star graphs, see also [Flinsenberg, 2003b].

Definition 4.6. Let $G_S = (N, E, w_e, w_r)$ be a searchgraph that contains Star graphs and dummy nodes U. Let $h(u)$ denote the expected remaining cost from node $u \in N \setminus U$ to destination $d \in N \setminus U$.

Define $h^\star(e) = \begin{cases} h(\delta_1(e)) - w_e(e) & \text{if } \delta_2(e) \in U; \\ h(\delta_2(e)) & \text{if } \delta_2(e) \notin U. \end{cases}$ □

The h-value of edge e is equal to the expected remaining cost from the end node of the edge to the destination. If $h(u)$ under-estimates the remaining cost then $h^\star(e)$ also under-estimates the remaining cost because the remaining cost from the end-node of e with $\delta_2(e) \in U$, is equal to the remaining cost from $\delta_1(e)$ minus the cost of edge e, $w_e(e)$. Furthermore, we can show that the estimator defined in Definition 4.6 is a dual feasible estimator.

Lemma 4.5. *Let G be a roadgraph with dummy nodes U. If $h(u)$ is a dual feasible estimator for roadgraph $G \setminus U$, then $h^\star(e)$ is a dual feasible estimator for roadgraph G.*

Proof. Assume estimator $h(u)$ is dual feasible for $G \setminus U$, which means that $h(\delta_1(e)) - h(\delta_2(e)) \leq w_e(e)$ for every $e \in G \setminus U$. Because $h^\star(e) = h(\delta_2(e))$ for every $\delta_2(e) \notin U$, we know that for every adjacent $e_1, e_2 \in G \setminus U$ the condition in Definition 4.5 is satisfied. Now let e be such that $\delta_1(e) \in G \setminus U$ and $\delta_2(e) \in U$. Because $\delta_2(e) \in U$, let e_1 be an edge with $\delta_2(e_1) = \delta_1(e)$ and $\delta_1(e_1) \in G \setminus U$. We need to prove that $w_e(e) + w_r(e_1, e) + h^\star(e) \geq h^\star(e_1)$. Because no rules exist between edges e_1 and e, we have $w_e(e) + w_r(e_1, e) + h^\star(e) - h^\star(e_1) = w_e(e) + h(\delta_1(e)) - w_e(e) - h(\delta_1(e)) \geq 0$. Now let e be such that $\delta_1(e) \in U$ and $\delta_2(e) \in G \setminus U$. Because $\delta_1(e) \in U$, let e_1 be an edge with $\delta_2(e_1) = \delta_1(e)$ and $\delta_1(e_1) \in G \setminus U$. We need to prove that $w_e(e) + w_r(e_1, e) + h^\star(e) \geq h^\star(e_1)$. We have $w_e(e) + w_r(e_1, e) + h^\star(e) - h^\star(e_1) = w_e(e) + w_r(e_1, e) + h(\delta_2(e)) - h(\delta_1(e_1)) + w_e(e_1)$. Because $h(u)$ is a dual feasible estimator, we know that $h(\delta_1(e_1)) - h(\delta_2(e)) \leq w_e(e_1) + w_e(e)$. This gives $w_e(e) + w_r(e_1, e) + h(\delta_2(e)) - h(\delta_1(e)) + w_e(e_1) \geq w_e(e) + w_r(e_1, e) - w_e(e_1) - w_e(e) + w_e(e_1) \geq 0$. This proves that $h^\star(e)$ is a dual feasible estimator. □

From Lemma 4.4 and Lemma 4.5 it follows that we can again use algorithm $C^\star$ in Figure 4.10 to plan optimum routes in a searchgraph, by using $h(e)$ in Definition 4.6, where route edges are stored in Star graphs.

When we need to plan a route in a searchgraph that is constructed with Point graphs, we can again use algorithm $C^\star$ in Figure 4.10. Note that an adjacent edge e_1 of edge $e \in B$ in algorithm $C^\star$ is an edge with $d_n(\delta_2(e)) = d_n(\delta_1(e_1))$. The value of $h(e)$ can be determined for $e \in B$ by considering the end node of edge e in the boundary graph, $\delta_2^o(e)$, and not by looking at $d_n(\delta_2(e))$. Definition 4.7 gives the formal definition of the estimated remaining cost to the destination for a searchgraph containing Points.

Definition 4.7. Let $G_S = (N, E, w_e, w_r)$ be a searchgraph that contains Point graphs and dummy nodes U. Let $h(u)$ denote the expected remaining cost from a node $u \in N \setminus U$ to destination $d \in N \setminus U$.
Define $h'(e) = \begin{cases} h(\delta_2^o(e)) & \text{if } \delta_2(e) \in U; \\ h(\delta_2(e)) & \text{if } \delta_2(e) \notin U. \end{cases}$ □

The estimator defined in Definition 4.7 is dual feasible as stated in Lemma 4.6.

Lemma 4.6. *Let P be a partition $\{C_1, \ldots, C_k\}$ with boundary graph B of roadgraph G. Let G_S be a searchgraph of P containing dummy nodes U and Points. If $h(u)$ is a dual feasible estimator for searchgraph G'_S of P containing Cliques, then $h'(e)$ is a dual feasible estimator for searchgraph G_S.*

Proof. Assume estimator $h(u)$ is dual feasible for G'_S containing Cliques. We know from Definition 4.4
$h(\delta_1(e)) - h(\delta_2(e)) \leq w_e(e)$ for every $e \in G_S \setminus U$,
$h(\delta_1^o(e)) - h(\delta_2^o(e)) \leq w_e(e)$ for every e with $\delta_1(e) \in U$ or $\delta_2(e) \in U$,
$h(\delta_2^o(e_1)) - h(\delta_1^o(e_2)) \leq w_r(e_1, e_2) - w_e(e_1) - w_e(e_2)$, for every adjacent e_1, e_2 with $\delta_2(e_1), \delta_1(e_2) \in U$.
Let e_1, e_2 be two (adjacent) edges with $\delta_1(e_2) = \delta_2(e_1)$. We need to prove that $w_e(e_2) + w_r(e_1, e_2) + h'(e_2) \geq h'(e_1)$. Because $h'(e) = h(\delta_2(e))$ for every $\delta_2(e) \notin U$, the condition in Definition 4.5 is satisfied for every adjacent $e_1, e_2 \in G \setminus U$.
For $\delta_1(e_2) = \delta_2(e_1) \in U$ and $\delta_2(e_2) \in U$ we have
$w_e(e_2) + w_r(e_1, e_2) + h'(e_2) - h'(e_1) = w_e(e_2) + w_r(e_1, e_2) + h(\delta_2^o(e_2)) - h(\delta_2^o(e_1)) =$
$w_e(e_2) + w_r(e_1, e_2) + h(\delta_2^o(e_2)) - h(\delta_1^o(e_2)) + h(\delta_1^o(e_2)) - h(\delta_2^o(e_1)) \geq$
$w_e(e_2) + w_r(e_1, e_2) - w_e(e_2) - w_r(e_1, e_2) + w_e(e_1) + w_e(e_2) \geq 0.$
For $\delta_1(e_2) = \delta_2(e_1) \notin U$ and $\delta_2(e_2) \in U$ we have
$w_e(e_2) + w_r(e_1, e_2) + h'(e_2) - h'(e_1) = w_e(e_2) + w_r(e_1, e_2) + h(\delta_2^o(e_2)) - h(\delta_2(e_1)) =$
$w_e(e_2) + w_r(e_1, e_2) + h(\delta_2^o(e_2)) - h(\delta_1^o(e_2)) \geq w_e(e_2) + w_r(e_1, e_2) - w_e(e_2) \geq 0.$
For $\delta_1(e_2) = \delta_2(e_1) \in U$ and $\delta_2(e_2) \notin U$ we have
$w_e(e_2) + w_r(e_1, e_2) + h'(e_2) - h'(e_1) = w_e(e_2) + w_r(e_1, e_2) + h(\delta_2(e_2)) - h(\delta_2^o(e_1)) =$
$w_e(e_2) + w_r(e_1, e_2) + h(\delta_2^o(e_2)) - h(\delta_1^o(e_2)) + h(\delta_1^o(e_2)) - h(\delta_2^o(e_1)) \geq$
$w_e(e_2) + w_r(e_1, e_2) - w_e(e_2) - w_r(e_1, e_2) + w_e(e_1) + w_e(e_2) \geq 0.$
This proves that $h'(e)$ is a dual feasible estimator. □

From Lemma 4.4 and Lemma 4.6 it follows that we can again use algorithm $C^\star$ in Figure 4.10 to plan optimum routes in a searchgraph, by using estimator h' in Definition 4.7, where route edges are stored in Point graphs.

4.3 Computational Evaluation

In this section, we compare the planning of optimum routes in roadgraphs for all three methods of storing route edges, discussed in Section 4.1.1, 4.1.2 and 4.1.3. Observe that the number of edges in a Clique graph is larger than the number of edges in a Star

graph if $b_i \geq 4$. For $b_i = 3$ the number of edges in a Star graph is equal to the number of edges in a Clique graph, while for $b_i \leq 2$ the number of edges in a Star graph is larger than the number of edges in a Clique graph. We choose to create a Star graph for cells with $b_i \geq 4$ and a Clique graph for cells with $b_i \leq 3$. For $b_i = 3$, we create a Clique graph because this structure is intuitively clearer, and requires fewer nodes and finite turn restrictions to be stored.

We used the partitions created by the Merging-Algorithm. Data on these partitions can be found in Table 3.5, and in Appendix C. We have planned 2,000 routes through the Netherlands between start and destination nodes that are selected in such a way that they are believed to be representative for the routes driven by a typical commuter. Table 4.3 gives some additional information on the routes that were planned. For every selected node (4,000 start or destination nodes) Table 4.3 indicates if it is located in an urban or rural area. We determined the set of 2,000 start and destination nodes by selecting edges and choosing one of the end nodes of the selected edge as start or destination node.

Distribution of	Elements			Number	Percentage
Number of routes			-	2,000	100%
Urban/Rural	Urban			2,063	51%
	Rural			1,937	49%
Euclidean	0	to	1,000	2	0%
distance (m)	1,000	to	2,500	5	0%
	2,500	to	5,000	10	0%
	5,000	to	10,000	49	2%
	10,000	to	25,000	285	14%
	25,000	to	50,000	808	40%
	50,000	to	100,000	306	15%
	100,000	to	250,000	509	25%
	250,000	to	500,000	26	1%
	500,000	to	1,000,000	0	0%
	more than		1,000,000	0	0%

Table 4.3. *Selected start and destination pairs in the Netherlands.*

The set of selected start and destination nodes covers the entire country, as can be seen from Figure 4.11. So we can safely base our evaluation of algorithm $C^\star$ on the selected pairs of start and destination nodes. We will also refer to this set as our *route batch*.

Tables 4.4 and 4.5 give the results for planning these $2,000$ routes when the edge costs are given by the time needed to traverse the edge at the maximum allowed travel speed and by the edge length respectively. Note that the maximum allowed travel speed is different for different types of edges, for example the maximum allowed

Figure 4.11. *Start and destination nodes in the Netherlands from Table 4.3.*

travel speed of a motorway edge is assumed to be 120 km/h. We refer to a route with minimum travel time as a fastest route, and to a route with minimum route length as a shortest route. Column 1 in Tables 4.4 and 4.5 gives the road network used for planning the routes. Column 2 gives the number of edges that is selected from the set of unexpanded edges H of algorithm $C^\star$ in Figure 4.10 for unpartitioned road networks, or equivalently, the number of *evaluated* edges. Column 3 gives the value of r_i that was used to create a partition. Columns 4, 5 and 6 give the number of evaluated edges for a partitioned roadgraph, using Clique, Star and Point graphs respectively to store the route edges. In the "Star"-column, Clique graphs are created for cells with $b_i \leq 3$.

Furthermore, we have also gathered data on the type of the evaluated edges. Specifically, we determined for each evaluated edge whether this edge was an internal edge, or a boundary or route edge. For each internal edge we determined whether it was located in the cell containing the start node or the destination node. This data can be found in Tables 4.6 and 4.7. Again column 1 gives the used road network. Column 2 gives the value of r_i that was used to create a partition. The columns denoted by $C(s)$ present the percentage of evaluated edges, for which the evaluated edge is located in the cell containing the start node. Similarly, the columns denoted by $C(d)$ present the percentage of evaluated edges, for which the evaluated edge is located in the cell containing the destination node. Finally, the columns denoted by B present the percentage of evaluated edges, for which the evaluated edge is a boundary or route edge. All percentages are rounded to the nearest integer.

As can be seen from Table 4.4 and Table 4.5, the number of evaluated edges is lowest if the route edges are stored in Point graphs. The number of evaluated edges is highest if the route edges are stored as Clique graphs. This can be explained by looking at the number of edges in the route graph. The searchgraph used for planning

Map (Edges)	$A^\star$	r_i	Clique	Star	Point
Sophia	1,261	r_i^C	294	277	200
(2,015)		r_i^S	300	277	195
		r_i^P	413	330	167
Eindhoven	6,822	r_i^C	2,346	1,159	643
(14,696)		r_i^S	2,553	1,229	617
		r_i^P	3,235	1,593	655
Siegen	14,409	r_i^C	2,518	1,545	1,107
(23,953)		r_i^S	2,905	1,520	889
		r_i^P	3,285	1,739	824
Giesen/Wetzlar	43,220	r_i^C	5,540	3,052	2,249
(88,177)		r_i^S	5,697	2,770	1,605
		r_i^P	6,892	3,167	1,455
Antwerp	42,459	r_i^C	8,665	5,463	4,852
(98,564)		r_i^S	11,096	3,656	2,037
		r_i^P	13,725	4,321	1,892
Netherlands	340,420	r_i^C	36,588	19,004	17,404
(1,135,280)		r_i^S	42,824	10,673	6,683
		r_i^P	51,951	11,768	5,676

Table 4.4. *Average number of evaluated edges for planning fastest routes.*

Map (Edges)	$A^\star$	r_i	Clique	Star	Point
Sophia	1,179	r_i^C	276	263	192
(2,015)		r_i^S	282	262	188
		r_i^P	386	308	158
Eindhoven	6,680	r_i^C	2,311	1,114	626
(14,696)		r_i^S	2,516	1,178	600
		r_i^P	3,185	1,518	637
Siegen	13,564	r_i^C	2,391	1,479	1,079
(23,953)		r_i^S	2,749	1,433	860
		r_i^P	3,097	1,604	786
Giesen/Wetzlar	42,318	r_i^C	5,481	2,960	2,215
(88,177)		r_i^S	5,627	2,644	1,578
		r_i^P	6,798	2,990	1,425
Antwerp	40,183	r_i^C	8,308	5,283	4,720
(98,564)		r_i^S	10,522	3,444	1,975
		r_i^P	12,971	3,991	1,813
Netherlands	335,170	r_i^C	36,328	18,662	17,114
(1,135,280)		r_i^S	42,520	10,451	6,576
		r_i^P	51,486	11,490	5,585

Table 4.5. *Average number of evaluated edges for planning shortest routes.*

Map (Edges)	r_i	Clique			Star			Point		
		$C(s)$	B	$C(d)$	$C(s)$	B	$C(d)$	$C(s)$	B	$C(d)$
Sophia	r_i^C	44	43	13	44	42	14	56	26	17
(2.015)	r_i^S	43	44	13	44	43	13	56	26	17
	r_i^P	20	74	6	22	72	6	34	57	9
Eindhoven	r_i^C	26	67	7	33	58	9	48	39	13
(14.696)	r_i^S	21	73	6	27	66	7	42	47	11
	r_i^P	12	85	3	15	82	4	26	68	6
Siegen	r_i^C	37	52	11	47	39	14	61	21	18
(23.953)	r_i^S	26	66	7	35	55	10	50	35	14
	r_i^P	17	78	5	22	72	6	36	54	10
Giesen/Wetzlar	r_i^C	37	52	10	50	35	14	64	18	18
(88.177)	r_i^S	25	68	7	35	55	10	51	35	14
	r_i^P	16	81	4	22	73	6	36	54	9
Antwerp	r_i^C	51	37	13	67	17	16	74	8	18
(98.564)	r_i^S	22	73	5	35	57	8	51	37	12
	r_i^P	13	84	3	21	74	5	36	56	8
Netherlands	r_i^C	54	30	16	69	10	20	74	4	22
(1.135.280)	r_i^S	31	60	9	48	38	13	60	23	17
	r_i^P	21	73	6	35	55	10	48	38	13

Table 4.6. *Distribution of the evaluated edges for planning fastest routes.*

Map (Edges)	r_i	Clique			Star			Point		
		$C(s)$	B	$C(d)$	$C(s)$	B	$C(d)$	$C(s)$	B	$C(d)$
Sophia	r_i^C	45	42	13	46	41	13	58	26	16
(2.015)	r_i^S	45	43	13	46	41	13	58	25	16
	r_i^P	22	73	6	23	71	6	36	56	9
Eindhoven	r_i^C	26	67	7	34	58	8	49	39	12
(14.696)	r_i^S	22	73	6	28	65	7	43	47	10
	r_i^P	12	85	3	15	81	4	27	67	6
Siegen	r_i^C	38	51	11	49	37	14	61	21	18
(23.953)	r_i^S	27	65	7	36	54	10	52	34	14
	r_i^P	17	78	5	23	70	6	37	53	10
Giesen/Wetzlar	r_i^C	38	52	10	51	35	14	64	18	17
(88.177)	r_i^S	26	68	7	36	54	9	52	34	13
	r_i^P	16	80	4	23	72	6	37	54	9
Antwerp	r_i^C	51	36	12	67	17	16	75	8	18
(98.564)	r_i^S	22	73	5	36	56	8	52	36	12
	r_i^P	13	84	3	22	73	5	37	55	8
Netherlands	r_i^C	54	30	16	70	10	20	74	4	21
(1.135.280)	r_i^S	31	60	9	49	38	13	61	23	16
	r_i^P	21	73	6	35	55	10	49	38	13

Table 4.7. *Distribution of evaluated edges for planning shortest routes.*

an optimum route contains the most edges in the "Clique"-column, and the least in the "Point"-column.

Furthermore, we see that for all road networks, except for that of Eindhoven, it is preferable to adjust the partitioning criterium to the type of route graphs that will be created for that partition. Specifically, the number of evaluated edges for partitions generated with $r_i = b_i(b_i - 1)$ is lowest if Clique graphs are used to store the route edges. Similarly, if the partition is generated with $r_i = \min\{2b_i, b_i(b_i - 1)\}$ it is best to use Star graphs (for cells with $b_i > 3$) to store the route edges. For partitions generated with $b_i = 0$, Point graphs should be used to store the route edges. This can be explained from the fact that the partitions that are generated using a particular value of r_i, try to optimize the partition with the future route planning process in mind. If the partition is generated using a different value for r_i than the one used later on, the number of route edges in the searchgraph does not correspond to the expected average number of route edges that was computed during the creation of the partition. As a result, other partitions result in faster route planning, specifically those partitions that are created using the proper number of route edges per cell.

The overall best results are obtained with partitions generated with $r_i = 0$, and Point graphs for storing route edges. As can be seen from Table 4.4 and Table 4.5, planning an optimum route between two random nodes in the road network of the Netherlands, with an average route length of 122 kilometers, requires the evaluation of about 5,500 edges.

From Tables 4.6 and 4.7, we can see that for partitions created with a decreasing value of r_i, a higher percentage of the evaluated edges consists of boundary and route edges. This is caused by the fact that those partitions tend to have smaller cells and more boundary and route edges. Furthermore, for partitions with Cliques a higher percentage of the evaluated edges consists of boundary and route edges than for partitions with Stars, for which in turn, a higher percentage of the evaluated edges consists of boundary and route edges than for partitions with Points. This is because a Clique contains more route edges than a Star, which has more route edges than a Point graph. Actually, the percentage of evaluated boundary edges can be found in the sub-column *B* of the column denoted by Point. The effect of creating a Star instead of a Clique can be deducted from comparing the percentage of evaluated boundary and route edges in columns Clique and Star.

Furthermore, we see that a relatively large part of the evaluated edges are part of the cell containing the start node. Note that the $C^\star$-algorithm searches from the start node to the destination node. The $C^\star$-algorithm apparently has some trouble leaving the start cell. This can be explained from the fact that the costs of the route edges are relatively high compared to the costs of the internal edges of the start cell. The $C^\star$-algorithm evaluates edges in the start cell and at some point evaluates a boundary edge leaving this start cell. This boundary edge then becomes a candidate in the

list of unexpanded edges H (see algorithm $C^\star$ in Figure 4.10). When this boundary edge is evaluated, the costs of a number of route edges are updated in list H if the searchgraph contains Cliques or Stars. For Point graphs, the cost of another boundary edge is updated. These updated edge costs are relatively high compared to the costs of internal edges of the start cell, because the cost of an entire route is included in the cost of such a route or boundary edge. At the same time, the expected cost from the end node of an internal edge of the start cell to the destination is usually relatively low, because for small cells, this is comparable to the cost of a route following a straight line from the start node to the destination node. So, these route edges in list H are not selected until a lot of the internal edges of the start cell have been evaluated.

Finally, we see that the percentage of the evaluated edges that is located in the cell containing the destination node, is relatively low. This is because the use of a h-value causes the search process to be drawn straight to the destination, especially for edges located closely to the destination.

Flinsenberg [2003a] presents more details on the comparison between the $A^\star$-algorithm and the $C^\star$-algorithm for a searchgraph containing Cliques. Flinsenberg [2003b] presents more results on the comparison between route planning using Cliques and Stars.

Verification of the Partitioning Criterion

Now that we have discussed the route planning speed or, equivalently, the number of edges evaluated by the $C^\star$-algorithm, we are interested in whether this corresponds to the estimated number of evaluated edges which was used to determine the quality of a partition. In order to verify the used partitioning objective function, we created additional partitions by using the Merging-Algorithm and verified whether the estimated number of evaluated edges corresponds to the actual number of edges evaluated when planning 2,000 routes through each partition.

First, we saved the first partition of the Netherlands generated by the Merging-Algorithm using $r_i = r_i^S$ that has an objective value $EE(G, C_1, \ldots, C_k)$ that differs less than 50 from an element in $\{28,500, 29,000, 29,500, \ldots, 60,000\}$. Subsequently, we created Star graphs for every cell with $b_i > 3$ and Clique graphs for cells with $b_i \leq 3$. For each of these partitions, $2,000$ fastest and shortest routes were planned. The results can be found in Figure 4.12. In this figure we see the results for partitions saved from 5 different runs of the Merging-Algorithm displayed in different shades of grey, although only two shades of grey can be clearly distinguished. The results for fastest and shortest routes are denoted by diamonds and squares respectively.

As we can see from Figure 4.12, a lower objective value $EE(G, C_1, \ldots, C_k)$ leads to a decrease in the number of edges that is actually evaluated to plan an optimum route. The relation between $EE(G, C_1, \ldots, C_k)$ and the actual number of evaluated edges AE can be expressed as $AE \approx 1.13 \cdot EE(G, C_1, \ldots, C_k) + 1,853.6$, with $R^2 = 0.9996$. The regression coefficient $R^2 \in [0,1]$ denotes the quality of the approx-

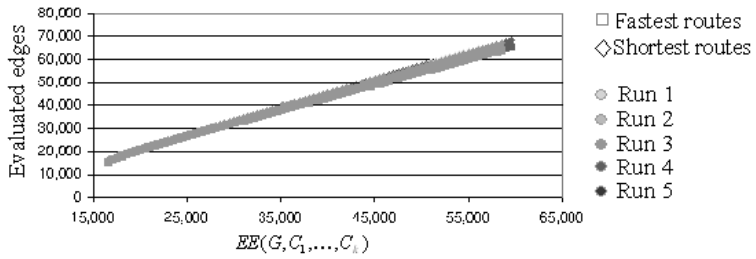


Figure 4.12. *Relation between number of evaluated edges and* $EE(G,C_1,\ldots,C_k)$.

imation. Note that $R^2 = 1$ denotes a perfect approximation, and $R^2 = 0$ denotes that $EE(G,C_1,\ldots,C_k)$ gives no information about the value of AE at all. So we conclude that $AE = 1.13 \cdot EE(G,C_1,\ldots,C_k) + 1,853.6$ is a good approximation.

Secondly, we used the Merging-Algorithm using $r_i = r_i^S$ and saved a partition if the value of $\sum_{i=1}^{k} \frac{n_i}{n}(2-\frac{n_i}{n})m_i$, $\sum_{i=1}^{k}(1-\frac{n_i}{n})^2 r_i$, m_B, or k differed at least $1,000$ from the previously stored partition. For each saved partition, we planned 2,000 fastest and shortest routes using Star graphs and computed the average number of edges evaluated to plan these routes. The results can be found in Figure 4.13.

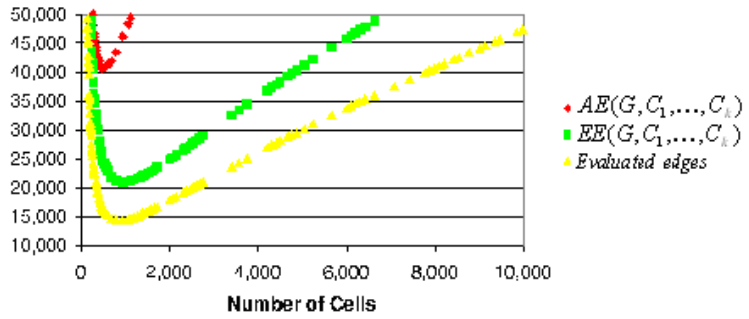


Figure 4.13. *Average number of evaluated edges required to plan 2,000 routes.*

From Figure 4.13 we can see that the value of $EE(G,C_1,\ldots,C_k)$ resembles the average number of edges evaluated to plan 2,000 routes between start and destination nodes in the Netherlands more closely than the value of $AE(G,C_1,\ldots,C_k)$. Furthermore, if $0.28 \leq \alpha \leq 0.40$ then the best partition of the Netherlands according to the partitioning objective $EE(G,C_1,\ldots,C_k)$ corresponds to the partition with the lowest average number of edges evaluated to plan 2,000 shortest and fastest routes. The same holds for partitions of Wetzlar/Giessen if $0.31 \leq \alpha \leq 0.62$. Note that $\alpha = 0.30$ was chosen for the Netherlands, and $\alpha = 0.34$ for Wetzlar/Giessen, so the best partition among the saved partitions is selected by using $EE(G,C_1,\ldots,C_k)$, see also [Van der Horst, 2003].

Although we have only considered partitions generated with $r_i = r_i^S$ to validate the use of $EE(G, C_1, \ldots, C_k)$, we think that these results can be generalized to other values of r_i. So, from Figures 4.12 and 4.13, we conclude that if a partition has a lower value of $EE(G, C_1, \ldots, C_k)$, then the expected number of evaluated edges needed to plan an optimum route is also lower. Note that many alternative route planning algorithms exist. As long as the search area of the planning algorithm can be described by an ellipse with the start and destination node as its foci, we expect that $EE(G, C_1, \ldots, C_k)$ still provides a good measure of the quality of a partition (i.e. the actual average number of evaluated edges). The value of α may need to be adjusted though. For differently shaped search areas the formula of $EE(G, C_1, \ldots, C_k)$ may also need to be adjusted. For example, factors α_1, α_2, α_3 and α_4 may be used to properly adjust the estimated number of edges evaluated in the start cell, destination cell, boundary graph and route graphs respectively.

4.4 Extension to Multi-Level Partitions

In Section 4.2, we showed that optimum routes can be planned for single-level partitions with algorithm $C^\star$ in Figure 4.10. For multi-level partitions, optimum routes can also be planned with algorithm $C^\star$. It just requires an adjustment of the searchgraph. In order to define the searchgraph for a particular start node s and destination node d, we need to define some additional sets. Let $C_{\ell_1} = C(s)$ be the cell containing the start node and $C_{\ell_2} = C(d)$ the cell containing the destination node. Let $\eta_1 = \{\omega(\ell_1), \omega(\ell_2)\}$ and $\eta_{\kappa+1} = \{i \in \{0, \ldots, k\} | \exists j \in \eta_\kappa : j \in \varphi(i)\}$, for $\kappa = 1, \ldots, \lambda - 1$, with λ the number of levels of the partition. Define $\eta = \cup_{i=1}^{\lambda} \eta_i$. Mathematically, the searchgraph can now be expressed as

$$G_S(G, \{C_1, \ldots, C_k\}, s, d) = C(s) \cup C(d) \cup \bigcup_{i \in \eta} \bigcup_{\{j \in \varphi(i) \setminus \eta\}} R_j \cup \bigcup_{i \in \eta} E_B^i.$$

An example of a searchgraph for a multi-level partition (see Figure 3.18) is given in Figure 3.19. The searchgraph of a multi-level partition contains the lowest-level cells containing the start and destination node (cells C_3 and C_7 in Figure 3.19), the route edges of the other subcells of the parents of the cells containing the start and destination nodes (i.e. the route edges of cells C_1 and C_6 in Figure 3.19), and the boundary edges of the partitions of the parents of the start and destination nodes (i.e. the boundary edges of the partition of cells C_0 and C_2 in Figure 3.19).

Giving this searchgraph as input to algorithm $C^\star$ together with start node s and destination node d, results in an optimum route from s to d.

To compare planning with multi- and single-level partitions, we have planned the same routes with multi-level partitions from Section 3.5.3 and with single-level partitions created with the Merging-Algorithm from Section 3.4.3. Because Section 4.3 showed that creating Point graphs gives the fastest route planning results, we have

only created Point graphs for the multi-level partitions. The results can be found in Table 4.8.

		fastest routes		shortest routes	
Map (Edges)	r_i	$A^\star$	Point	$A^\star$	Point
Sophia	r_i^C	1,261	113	1,179	108
(2,015)	r_i^S		96		92
	r_i^P		120		113
Eindhoven	r_i^C	6,822	449	6,680	436
(14,696)	r_i^S		433		420
	r_i^P		593		576
Siegen	r_i^C	14,409	390	13,564	372
(23,953)	r_i^S		442		419
	r_i^P		575		543
Giesen/Wetzlar	r_i^C	43,220	660	42,318	644
(88,177)	r_i^S		734		718
	r_i^P		1,053		1,029
Antwerp	r_i^C	42,459	803	40,183	772
(98,564)	r_i^S		952		906
	r_i^P		1,376		1,308
Netherlands	r_i^C	340,420	1,356	335,170	1,332
(1,135,280)	r_i^S		2,225		2,188
	r_i^P		3,409		3,352

Table 4.8. *Average number of evaluated edges using multi-level partitions.*

As can be seen from Table 4.8 in combination with Table 3.7, the multi-level partition with the lowest objective value leads to the lowest average number of evaluated edges for all partitions except for the road network of Eindhoven. For this road network, the partition for r_i^P leads to more evaluated edges on average than for r_i^C, although its objective value is lower.

Furthermore, the number of evaluated edges is generally lowest for multi-level partitions generated using r_i^C. This is caused by the fact that it is beneficial to have large high-level cells with only a few boundary nodes, because this leads to a large reduction in the searchgraph size. Using r_i^C leads to partitions with the largest high-level cells among the three possible options. Furthermore, we can see from Tables 4.6 and 4.7 that for partitions with Point graphs, the number of evaluated boundary edges is especially low for partitions created using r_i^C. The creation of multiple levels reduces the number of evaluated internal edges, but the number of highest-level boundary edges in the searchgraph remains the same. This also explains why using r_i^C for creating a multi-level partition leads to the best results.

Finally, the number of evaluated edges is clearly much lower for good multi-level partitions than for single-level partitions. In particular on average about 1,350 edges have to be evaluated to plan a route in the Netherlands. Therefore, we conclude that it is worthwhile creating multi-level partitions. However, the number of additional route edges or costs to be stored should be considered, when determining a multi-level partition.

4.5 Conclusion

In this chapter, we have discussed three possible route graphs. We have shown that each of these route graphs can be used to plan optimum time-independent routes in a roadgraph. Furthermore, we presented a polynomial-time algorithm, the $C^\star$-algorithm, which can be used to determine an optimum route. In order to evaluate the speed of the $C^\star$-algorithm, we have planned 2,000 fastest and shortest routes through the single-level partitions from Chapter 3. The results confirm the conclusion from Chapter 3 that single-level partitions created with $r_i = 0$ and Point graphs lead to the fastest route planning. We also saw a large reduction in the average number of edges evaluated by the $C^\star$-algorithm compared to the standard $A^\star$-algorithm. We have also planned 2,000 fastest and shortest routes through the multi-level partitions from Chapter 3. The results show a large decrease in the number of edges evaluated by the $C^\star$-algorithm compared to using a single-level partition. In particular the average number of edges evaluated by the $C^\star$-algorithm using the best multi-level partition of the Netherlands is about 1,350. For comparison, the standard $A^\star$-algorithm requires on average about 340,000 evaluations to plan the same 2,000 fastest or shortest routes.

For many partitions of the Netherlands created with $r_i = \min\{b_i(b_i-1), 2b_i\}$ and Star graphs, we compared the value of the average and estimated number of evaluated edges and the real number of edges evaluated by the $C^\star$-algorithm. This showed that the estimated number of evaluated edges closely resembles the real number of evaluated edges, and that partitions with a lower value of $EE(G, C_1, \ldots, C_k)$ also lead to faster route planning. The relation between the objective value $EE(G, C_1, \ldots, C_k)$ and the real number of edges evaluated by the route planning algorithm can also be determined for Clique and Point structures; we expect the found relation for Star structures to hold for Clique and Point structures as well. For multi-level partitions we also expect this relation to hold.

Note that the $C^\star$-algorithm still evaluates a lot of edges in the first cell for partitions with relatively large cells, as can be seen from Tables 4.6 and 4.7. This confirms the motivation of Chapter 3 for constructing multi-level partitions. Especially for large cells, the estimated cost via a route edge is much larger than the estimated cost of a route via an internal edge of the start cell. This is caused by the h-value that is (too) low, and the high cost of a route edge compared to the cost of an internal edge of the start cell. For partitions with small cells, most evaluated edges are route edges

or boundary edges. Consequently, the number of evaluated edges can be decreased by decreasing the size of the cells, by increasing the h-value, and by removing edges from the set of unevaluated edges H in algorithm $C^\star$. The size of the cells can be decreased by using $r_i = 0$ to partition the graph or by creating an additional level. The h-value can be increased in several ways, for example by exploiting the structure of the searchgraph, or by splitting the planning process in two phases where the first phase is used to determine a better h-value, see Kaindl & Kainz [1997]. There are also several ways to remove edges from the set H with and without sacrificing the optimality of the planned route. For example, as soon as an optimum route to every boundary node of the start cell is found, all internal edges of the start cell can be removed from set H without sacrificing the optimality of the planned route.

5

Time-Dependent Planning

In Chapter 4 we discussed the planning of optimum routes in time-independent partitioned road networks. In this chapter, we study optimum route planning in time-dependent road networks. In Section 5.1, we first introduce the issues involved in planning optimum routes in unpartitioned time-dependent road networks. In Section 5.2, we discuss the criteria that need to be satisfied so that a modified standard route planning algorithm can be used to plan optimum time-dependent routes in polynomial time. In Section 5.3, we present an algorithm for planning time-dependent routes. Section 5.4 presents data and results on planning time-dependent routes in unpartitioned road networks, specifically in the Netherlands. In Section 5.5, we discuss planning time-dependent optimum routes in partitioned road networks. Section 5.6 presents the conclusion.

5.1 Introduction

In Section 2.2, we defined an optimum route in a time-dependent roadgraph as a feasible time-route that has minimum cost. This minimum cost typically depends on time. It is possible that a time-route is feasible for one departure time and not feasible for another departure time. In Figure 5.1 time-route $\pi_1 = (\langle e_1, e_2 \rangle, 1)$ has cost $w_p(\pi_1) = 2$, and time-route $\pi_2 = (\langle e_1, e_2 \rangle, 2)$ has cost $w_p(\pi_2) = \infty$. Therefore time-route π_2 is not feasible (see Definition 2.14), while time-route π_1 is. This is the

desired situation because the cost of edge e_2 is equal to ∞ for $t > 2$, which means that the edge is closed during the time we use it.

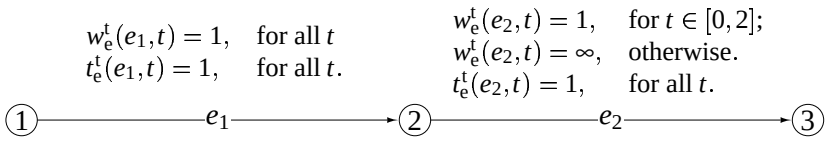


Figure 5.1. *A time-route can be feasible or infeasible for different departure times.*

$$
\begin{array}{lll}
t_e^t(e,t) & = 1, & \text{for all } e, t. \\
w_e^t(e_i,t) & = 3, & \text{for } t \in [7,9], \text{ and } i = 1,2. \\
w_e^t(e_i,t) & = 1, & \text{otherwise.}
\end{array}
$$

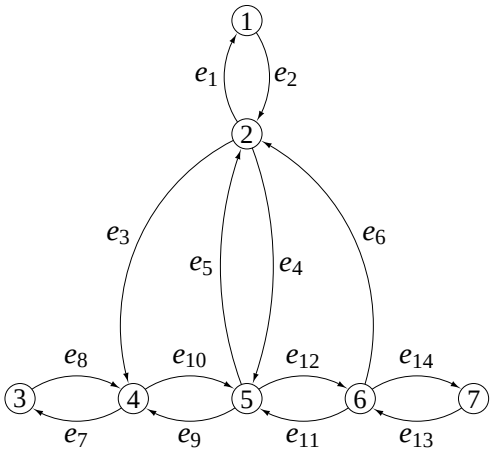


Figure 5.2. *Example with time-dependent edge costs.*

Because the costs in a time-dependent roadgraph are not constant, the cost of a particular time-route in a roadgraph can differ for different departure times. Consider the roadgraph in Figure 5.2. Note that only the edge costs of the edges between nodes 1 and 2 are time-dependent. The travel time is constant for all t. When the driver departs from his home at node 7 at 7:00h ($t = 7$) and travels to node 1, the cost of the time-route $\langle e_{13}, e_6, e_1 \rangle$ is equal to 5. However, when the driver departs at 21:00h ($t = 21$), the cost of the time-route $\langle e_{13}, e_6, e_1 \rangle$ is 3. So the cost of a time-route between two nodes can be different at different departure times.

Moreover, during the planning, the arrival time at a node has to be calculated in order to be able to determine the cost of edges adjacent to that node at that time. In order to calculate the cost of time-route $\langle e_{13}, e_6, e_1 \rangle$ we need to determine the arrival

time at node 2, which is 9:00h if the driver departs at 7:00h and 23:00h if the driver departs at 21:00h.

Furthermore, for a fixed departure time from the start node, the cost of a particular edge can still differ for different time-routes that lead to that node or edge. Imagine that the time-route to a node calculated so far makes a detour. That influences the cost of the edge leading away from that node, because the arrival time of the driver following this time-route depends on the time-route towards this node. If later on a different time-route is found to that node, the costs of the same departing edge and all succeeding edges change, because the arrival time of the driver at that node has changed. In our example, the driver can also take time-route $\langle e_{13}, e_{11}, e_5, e_1 \rangle$. In that case the arrival time at node 2 is 10:00h if the driver departs at 7:00h. This leads to a time-route with cost 4. If the driver departs at 21:00h and takes time-route $\langle e_{13}, e_{11}, e_5, e_1 \rangle$ the time-route has also cost 4. So if there are no restrictions on the time-dependent edge costs, it may be optimal to delay the arrival at a node. In our example, the optimum time-route for the driver starting at 7:00h, is time-route $\langle e_{13}, e_{11}, e_5, e_1 \rangle$ with cost 4 and an arrival time at node 2 of 10:00h, although there also exists a time-route $\langle e_{13}, e_6, e_1 \rangle$ that arrives at node 2 at 9:00h, but has a total cost of 5.

It is optimal to delay the arrival at a node u if the cost of the time-route from node u to the destination decreases more than the cost of the time-route from the starting point to node u increases by delaying the arrival. In situations like this, it is theoretically possible that it is optimal to (infinitely) delay the arrival at a node. So it can (theoretically) be optimal to create cycles or large detours. In Figure 5.3 the edge costs are put next to the edges. The driving time for each edge is 1. If the driver departs from node 1 at time 0, and wants to go to node 3, it is optimal delay the arrival at node 3 by creating 2 cycles. Naturally, this greatly complicates the search for an optimum time-route.

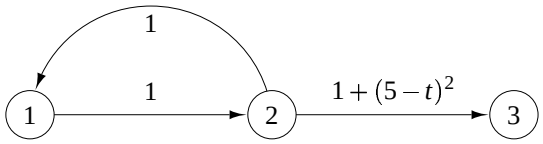


Figure 5.3. *A time-route with multiple cycles is optimal.*

Another example of the complexity of our problem is the presence of ferries. Sometimes, taking a ferry is one possible way to get across a river. In many cases, also a bridge is present, somewhere along the riverbanks. Ferries usually have time-tables that determine when cars are transported to the other riverbank. If a driver arrives just before the ferry sets off, it can be beneficial to take the ferry. However, if the driver arrives just after the ferry has left, the driver is forced to wait until the next

ferry departs. In that case it could be beneficial not to wait for the ferry, but to drive to the nearest bridge to cross the river. Again the optimum time-route, and the cost of the time-route using the ferry, depend on the time the driver arrives at the ferry, and the ferry's time-table. Notice that the time that the driver arrives at the ferry depends on the time-route that leads to the ferry, and the driver's departure time.

5.2 Consistency

This section discusses requirements that have to be met in order to be able to plan an optimum time-dependent route in polynomial time using a modified version of the standard $A^\star$-algorithm. We discuss these requirements for several possible realizations of the edge cost functions of a time-dependent road network. In particular, Section 5.2.1 discusses what happens if the edge cost of an edge equals the driving time of that edge. In Section 5.2.2 we handle more general cost functions.

5.2.1 Time Consistency

In this section, we assume that the edge cost of edge e is given by the travel time of edge e, i.e., $w^{\mathrm{t}}_{\mathrm{e}}(e,t) = t^{\mathrm{t}}_{\mathrm{e}}(e,t)$.

Kaufman & Smith [1993] showed that for time-dependent edge costs, standard shortest path algorithms such as Dijkstra's algorithm [Dijkstra, 1959] can be applied to determine a minimum-cost time-route if the roadgraph does not contain turn restrictions and if it satisfies a *consistency condition.* If this condition is satisfied then every sub-time-route starting from the start node of a minimum-cost time-route is again a minimum-cost time-route. In the example in Figure 5.2 the minimum-cost time-route from node 7 to node 1 $\langle e_{13}, e_{11}, e_5, e_1\rangle$ does not consist of only minimum-cost sub-time-routes. Specifically, the sub-time-route from node 7 to node 2 $\langle e_{13}, e_{11}, e_5\rangle$ of the minimum-cost time-route does not have minimum cost. There is another time-route from node 7 to node 2 that has a lower cost, namely time-route $\langle e_{13}, e_6\rangle$. This example therefore should not satisfy the consistency condition of Kaufman & Smith [1993].

Let t_1 and t_2 be departure times and let $c_{uv}(t)$ be the minimum time needed for travelling from node u to node v departing at time t from node u. A roadgraph without rules is called *consistent* if and only if for all nodes u and v and times $t_1 \leq t_2$,

$$t_1 + c_{uv}(t_1) \leq t_2 + c_{uv}(t_2).$$

This condition states that leaving node u later can perhaps reduce the duration of traversing an edge, but it cannot decrease the arrival time at node v. Taking the minimum time is sufficient because we want to give a condition under which the sub-time-route from the start node of a minimum-cost time-route to another node of a minimum-cost time-route is again a minimum-cost time-route. So $t_1 + c_{uv}(t_1)$ has to denote the time of a minimum-cost time-route, which is only the case if $c_{uv}(t)$ gives the minimum time needed to drive from node u to node v starting at time t.

Note that if Bellman's principle of optimality [Bellman, 1957] holds then the graph is consistent.

Kaufman & Smith [1993] also argue that if for every e, t and $\Delta t > 0$ we have $t + t_e^t(e,t) \leq t + \Delta t + t_e^t(e, t+\Delta t)$ then G^t without rules is consistent. However, the reverse need not be true. Consider the graph in Figure 5.4. Note that when the driver departs from node 1 at time t, he arrives at node 3 at time $2t$ and at node 2 at $2t + t_e^t(e_2, 2t) = 4t$. The minimum cost of a route from node 1 to node 2 departing at time t is $\min\{4t, 300-2t\} = 4t$, which is at least the minimum route cost between these two nodes departing at time $t + \Delta t$. The other cases are easy, and it follows that this graph is consistent. But $t + t_e^t(e_3,t) = 300 - t > 300 - t - \Delta t = t + \Delta t + 300 - 2(t+\Delta t) = t + \Delta t + t_e^t(e_3, t+\Delta t)$. Therefore, the reverse does not hold in this case. So the consistency of routes is more general than the consistency of edges, but also more difficult to verify.

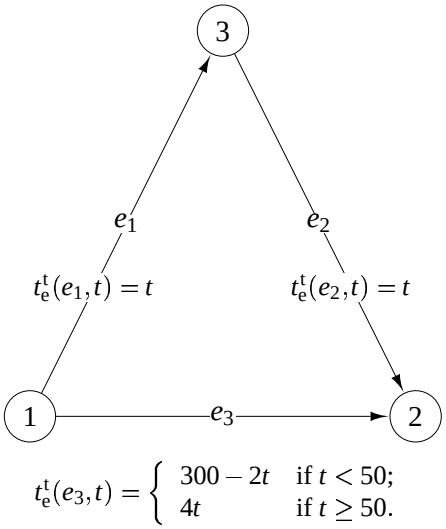


Figure 5.4. *The reverse of the consistency condition need not be true.*

Just as for planning optimum time-independent routes we need to evaluate edges instead of nodes to plan a time-dependent route in a roadgraph with rules. If every sub-time-route of a optimum time-route is again an optimum time-route (i.e. Bellman's principle of optimality [Bellman, 1957] holds), we can use a modified version of a standard route planning algorithm to plan optimum time-dependent routes. Because a time-route consisting of a single edge has only the empty route or the time-route itself as sub-time-route, this condition is satisfied for time-routes consisting of only a single edge. As a result, we only have to consider time-routes containing at least two edges. The consistency condition can be adapted for roadgraphs with rules as formulated in Definition 5.1.

Definition 5.1. Let $T(e,d,t)$ denote the minimum travel time of a time-route in a time-dependent roadgraph G^{t} from $\delta_1(e)$ starting with edge e to node d departing from node $\delta_1(e)$ at time t. A roadgraph G^{t} is *time consistent* if and only if for every time $t_1 \leq t_2$, and every pair of adjacent edges e_1 and e_2 in G^{t} we have

$$t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1) + T(e_2,d,t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1)) \leq t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2) + T(e_2,d,t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2)).$$

□

Just as for roadgraphs without rules, we can show that it is sufficient to consider pairs of adjacent edges to determine if a roadgraph is time consistent. This is formulated in Lemma 5.1.

Lemma 5.1. *A roadgraph G^{t} is* time consistent *if for every time $t_1 \leq t_2$, and every pair of adjacent edges e_1 and e_2 in G^{t} we have*

$$t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1) + t_{\mathrm{e}}^{\mathrm{t}}(e_2,t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1)) \leq t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2) + t_{\mathrm{e}}^{\mathrm{t}}(e_2,t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2)).$$

Proof. Follows easily by using induction on the number of edges of the time-route from e_2 to d. □

The time consistency condition can be verified fast by checking for every pair of adjacent edges in a roadgraph if the condition in Lemma 5.1 is satisfied. If a roadgraph is time consistent, then starting later cannot lead to an earlier arrival time at the destination. This leads to Lemma 5.2.

Lemma 5.2. *Let s and d be two nodes in roadgraph G^{t}. Let π_i be a time-route in G^{t} departing from s at time t_0 ending with edge e_1 in $\delta_2(e_1) \neq d$ arriving at $t_i > t_0$, $i = 1,2$. Let π_{i+2} be an optimum time-route in G^{t} departing from $\delta_2(e_1) = \delta_1(e_2)$ at t_i starting with edge e_2 arriving at node d at time $t_{i+2} > t_i$, $i = 1,2$. If G^{t} is time consistent and $t_1 \leq t_2$ then $t_1 \leq t_3 \leq t_4$.*

Proof. We have that $t_3 = t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1) + T(e_2,d,t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1)) \geq t_1$. Furthermore, $t_4 = t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2) + T(e_2,d,t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2))$. Because G^{t} is time consistent we know that for every nodes s, d, adjacent edges e_1, e_2, and times t_1, t_2, given that $t_1 \leq t_2$ we have $t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1) + T(e_2,d,t_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_1)) \leq t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2) + T(e_2,d,t_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t_2))$, which leads to $t_4 \geq t_3$. □

The time consistency condition above assumes that waiting is not possible. If waiting is possible and costless, meaning that only the time increases by the amount of waiting time and the costs remain constant, then the time consistency condition is satisfied. The time consistency condition implies that if waiting is not allowed, introducing a time-window in which the time needed to traverse an edge is long, and short

otherwise, could violate time consistency. If waiting is not allowed, it would be possible to depart later and arrive earlier if, due to the late departure, the time-window in which it takes a lot of time to traverse an edge is missed. The roadgraph is then no longer time consistent. A time-window will generally be given by $t_e^t(e,t) = b$, $t \in [t_1, t_2]$ and $t_e^t(e,t) = a$ otherwise, with $b \gg a$. For such a time-function the difference between the driving time at time t_2 and time $t_2 + \varepsilon$ can be too large to sustain time consistency. Suppose there exists an edge e with $t_e^t(e,t) = 4$ if $t \in [6,9]$ and $t_e^t(e,t) = 2$ otherwise. Postponing the departure from the start node of edge e from $t = 9$ to $t = 10$ leads to an earlier arrival at the end node of edge e at time $t = 12$ instead of $t = 13$.

5.2.2 Cost Consistency

In a roadgraph, edge costs are not necessarily equal to the travel time. Orda & Rom [1990] minimize the arrival time at the destination. In car navigation this only applies to planning a minimum-cost time-route with edge costs equal to the time needed to traverse an edge. However, other objectives such as minimum distance are also supported by a car navigation system. The cost of such a time-route can also depend on time but the objective is not to minimize the arrival time but to minimize the cost of the time-route, which depends on the entire time-route. Therefore, in this section we let go of the requirement that edge costs are equal to the driving time, and we define a *cost consistent* roadgraph in Definition 5.2. For a roadgraph that is cost consistent, using a sub-time-route with a higher cost cannot lead to a minimum-cost time-route, regardless of the arrival time at node u.

Definition 5.2. Roadgraph G^t is *cost consistent* if and only if for every minimum-cost time-route $\pi = (\langle e_1, \ldots, e_\ell \rangle, t)$ in G^t, for every $i = 1, \ldots, \ell$ there does not exist a time-route $\pi' = (\langle a_1, \ldots, a_j \rangle, t)$ in G^t with $\delta_1(a_1) = \delta_1(e_1)$, $a_j = e_i$ and $w_p(\pi') < w_p((\langle e_1, \ldots, e_i \rangle, t))$ or $t'_j < t_i$ and $w_p(\pi') = w_p(\langle e_1, \ldots, e_i \rangle, t))$. □

This condition says that for a minimum-cost time-route $(\langle e_1, \ldots, e_\ell \rangle, t)$ departing at time t from $s = \delta_1(e_1)$ to $d = \delta_2(e_\ell)$, the time-route $(\langle e_1, \ldots, e_i \rangle, t)$ is also a minimum-cost time-route departing at time t from s to e_i for every $i = 1, \ldots, \ell$. Note that if there are two minimum-cost time-routes from node s to edge e_i each with cost $w_p((\langle e_1, \ldots, e_i \rangle, t))$ then the time-route $(\langle e_1, \ldots, e_i \rangle, t)$ is the time-route with the earliest arrival time at node $\delta_2(e_i)$. Note that we cannot verify easily if a roadgraph is cost consistent or not. To verify whether a roadgraph is cost consistent, we need to check for every minimum-cost time-route in roadgraph G^t whether every starting sub-time-route is again a minimum-cost time-route. This requires the planning of a minimum-cost time-route between every pair of nodes in graph G^t for every possible departure time.

The cost consistency condition above assumes that waiting is not possible. If waiting is possible and costless, which means that only the time increases by the

amount of waiting time, then Definition 5.2 is not necessarily satisfied. If the cost of an optimum route departing at time t_0 from node s ending with edge e arriving at node $\delta_2(e)$ at time t_1 is always lower than the cost of an optimum route departing at time t_0 from node s ending with edge e arriving at node $\delta_2(e)$ at time t_2 for $t_1 \leq t_2$, then the driver can always use the former optimum route to edge e and wait at no additional cost at $\delta_2(e)$ before continuing his journey to destination d. Consequently, the leading sub-time-route of a minimum-cost route is a minimum-cost route. Otherwise, it may be beneficial for the driver to use a route from s to e with a higher cost so that the cost of the remainder of his route decreases due to the earlier arrival time. Therefore, the introduction of costless waiting does not necessarily satisfy the cost consistency condition.

Notice that apart from the time consistency condition, also the cost consistency assumption can be violated if we introduce a time-window with an edge cost equal to infinity inside the time-window and equal to some finite number otherwise. An increase (or decrease) of the arrival time at node $\delta_2(e_i)$ can cause the cost of the edge from $\delta_1(e_{i+1})$ to $\delta_2(e_{i+1})$ to become finite instead of infinite for certain arrival times. As a result, an optimum time-route may consist of a time-route ending with edge e_i that has a cost that is higher than the minimum time-route cost, so that the driver can start traversing edge e_{i+1} at a time at which edge e_{i+1} has a finite cost. Therefore the roadgraph is not necessarily cost consistent if time-windows are present.

For a discrete time-scale, we can plan optimum time-dependent routes in roadgraphs with time-windows by creating a time-expanded network, see for example [Aronson, 1989; Chabini & Lan, 2002; Mohajer, Mutapcic & Emami, 2004; Orlin, 1984]. This means that we create an edge e for every possible departure time, and compute its matching time-dependent cost. Note that this can drastically increase the number of nodes and edges in a roadgraph depending on the discretization of the time-scale. As a result, the planning time can also increase substantially. Planning optimum time-dependent routes in time-expanded networks for different waiting policies is studied by Dean [2004].

Our time-dependent model from Section 2.2 is more general than the problem with time-dependent edge lengths and waiting at nodes presented by Orda & Rom [1990], because in our model the objective is not to minimize the arrival time, but the time-dependent cost of the time-route. Furthermore, turn restrictions are included in our model.

5.3 Algorithms for Time-Dependent Planning

In this section, we discuss the planning of optimum time-routes in a time-dependent roadgraph G^{t}. As argued by Orda & Rom [1990], planning an optimum time-route in a time-dependent roadgraph and not allowing waiting at nodes is *NP*-hard. However, if the time-dependent roadgraph is cost consistent, planning an optimum time-route

can be done in polynomial time. This follows from Definition 5.2 which says that for a cost consistent roadgraph, an optimum time-route consists of optimum sub-routes. As a result, only (the cost of) the optimum time-route to a certain edge needs to be stored. Therefore, we can use a modification of our $C^\star$-algorithm (see Figure 4.10) to plan an optimum time-route in a cost consistent time-dependent roadgraph. We call this slightly modified algorithm the $T^\star$-algorithm (see Figure 5.5) to distinguish it from the time-independent algorithm $C^\star$. The difference between the $T^\star$-algorithm and the $C^\star$-algorithm is that the $T^\star$-algorithm keeps track of the arrival time at the end node of an edge e and uses this arrival time to determine the (time-dependent) cost of the adjacent edges of edge e. Note that algorithm $T^\star$ in Figure 5.5 can also be implemented in order $O(n + m \log m)$, just as algorithm $C^\star$.

If the time-dependent graph G^{t} is not cost consistent, algorithm $T^\star$ can still be used to determine a time-route in G^{t}. However, the resulting time-route is not guaranteed to be optimum. Specifically, it may be better to take a time-route with higher cost to a certain edge in the presented time-route to change the arrival time at this edge, so that the time-route from this edge to the destination node has a lower cost. In practice this not only very unlikely, it is also very hard to explain to the driver. For real-world road networks, it is possible that certain roads are closed during a certain time-period, for example some mountain passes are closed during winter and ferries depart only at certain time points. Also, sometimes it is only allowed to make a certain turn during non-peak hours, to avoid congestion. These situations usually cause the graph to become non-consistent. However, because the turns are usually forbidden for several hours, and edges are even closed for days or weeks, the time at which the turn becomes forbidden or the edge is closed is usually not defined very precisely. Therefore, slightly modifying the arrival time at such a turn or edge is not recommended. First of all, the driver may arrive ahead of or behind schedule because his driving speed differed from the expected speed (see also Chapter 6). Secondly, an edge may already be closed ahead of schedule or open behind schedule. Also a turn may be considered illegal even though it is supposed to be legal according to the stored data in the database. As a result, the time-route presented to the driver should not try to arrive at certain edges in time to take advantage of an edge that is about to close or open, or of a rule that is about to become (il)legal. Therefore, we prefer time-routes that consist of optimum sub-routes, and we assume that the time-dependent graph is cost consistent, even though this may not be the case.

5.4 Computational Evaluation

This section discusses the data available for planning time-dependent routes as well as the results of planning time-dependent routes through the unpartitioned road network of the Netherlands.

Algorithm $T^\star$ from Figure 5.5 can be used to plan optimum time-dependent

Algorithm $T^\star$

Input: A roadgraph G^{t}, start node s, destination node d and departure time t_0.

H : The set of unexpanded edges.
$g(e)$: The minimum cost found of a time-route from s to edge e.
$g'(e)$: Alternative cost of edge e.
$a(e)$: Arrival time at the end node of edge e of the time-route to edge e with cost $g(e)$.
$a'(e)$: Arrival time at the end node of edge e of the time-route to edge e with cost $g'(e)$.
$g_{\mathrm{n}}(u)$: The minimum time-route cost found from node s to node u.
$\lambda(e)$: The adjacent edge leading to edge e in the best time-route found from s to e.
π : The best time-route found from s to d.
$w_{\mathrm{p}}(\pi)$: The cost of time-route π.
$h(u)$: The estimated cost from node u to node d.
$h(e)$: The estimated cost from edge e to node d.

Initialization step
$H = E$, $\pi = (\langle\rangle, t_0)$, $w_{\mathrm{p}}(\pi) = \infty$
for every $e \in H$ **do**
    $g(e) = \infty$
    $a(e) = \infty$
    $\lambda(e) = \emptyset$
end for
for every $u \in N$ **do** $g_{\mathrm{n}}(u) = \infty$ **end for**
$g_{\mathrm{n}}(s) = 0$
for every $e \in H$ such that $\delta_1(e) = s$ **do**
    $g(e) = w_{\mathrm{e}}^{\mathrm{t}}(e, t_0)$
    $a(e) = t_0 + t_{\mathrm{e}}^{\mathrm{t}}(e, t_0)$
    $\lambda(e) = e$
end for

Main body of the algorithm
while there is an edge $e \in H$ with $g(e) + h(e) < g_{\mathrm{n}}(d)$ **do**
    Let e be an edge from H with $g(e) + h(e)$ minimum
    for every (allowed) edge e_1 with $\delta_2(e) = \delta_1(e_1)$ in G^{t} **do**
        $g'(e_1) = g(e) + w_{\mathrm{e}}^{\mathrm{t}}(e_1, a(e) + t_{\mathrm{r}}^{\mathrm{t}}(e, e_1, a(e))) + w_{\mathrm{r}}(e, e_1, a(e))$
        $a'(e_1) = a(e) + t_{\mathrm{r}}^{\mathrm{t}}(e, e_1, a(e)) + t_{\mathrm{e}}^{\mathrm{t}}(e_1, a(e) + t_{\mathrm{r}}^{\mathrm{t}}(e, e_1, a(e)))$
        if $g(e_1) > g'(e_1)$ **or** $((g(e_1) = g'(e_1)$ **and** $a(e_1) > a'(e_1))$ **then**
            $g(e_1) = g'(e_1)$
            $g_{\mathrm{n}}(\delta_2(e_1)) = \min\{g_{\mathrm{n}}(\delta_2(e_1)), g(e_1)\}$
            $a(e_1) = a'(e_1)$
            $\lambda(e_1) = e$
        end if
    end for
    $H = H \setminus \{e\}$
end while
if $g_{\mathrm{n}}(d) < \infty$ **then** $w_{\mathrm{p}}(\pi) = g(e)$ **and** construct time-route π from edge e using $\lambda(e)$ **end if**

Output: Time-route π, and the time-route cost $w_{\mathrm{p}}(\pi)$.

Figure 5.5. *Algorithm $T^\star$.*

routes through the Netherlands using an unpartitioned road network, because, as we will show in this section, our time-dependent roadgraph is time consistent when we assume that all ferries in the road network are always available and can be treated as "normal" road edges. Because the set of selected start and destination nodes covers the entire country, as can be seen from Figure 4.11, we can safely base our evaluation of time-dependent route planning on the selected pairs of start and destination nodes.

5.4.1 Speed Profiles

In this section, we present the data that was available for time-dependent planning. First of all, we used the road network of the Netherlands to test time-dependent planning, and for this road network all rule costs are time-independent and either 0 or infinite. Also, we set the turning time of a turn restriction equal to the matching rule cost. First, we present the data obtained from Van Woensel [2003]. It concerns travel speeds in Antwerp on the Belgium motorway A1/E19, section Mechelen North-Rumst. Subsequently, we present data obtained from "Rijkswaterstaat", on travel speeds on most major motorways in the Netherlands.

Antwerp

The data set available consists of a simulation of the number of vehicles per hour and the resulting average travel speed determined by Van Woensel [2003], for 1 week of the Belgium motorway A1/E19, section Mechelen North-Rumst, headed for Brussels (i.e. counter 19012). The simulation is based on data collected in 1995. The average travel speed for this motorway section is presented in Table 5.1, and Figure 5.6.

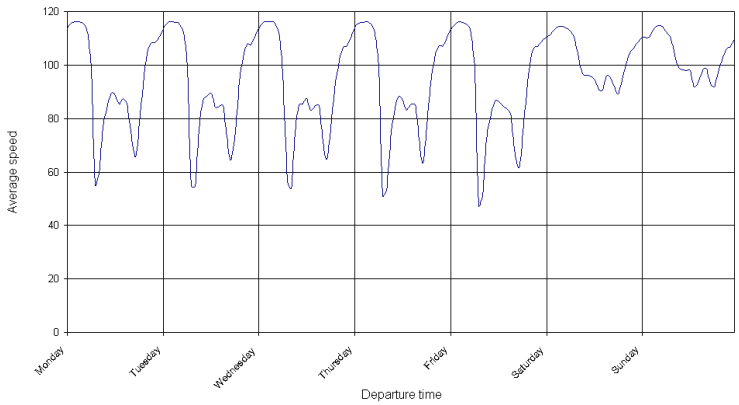


Figure 5.6. *Average speed (km/h) from Mechelen North to Rumst, 1995 (A1/E19).*

As can be seen from Table 5.1, data is available on average travel speeds per hour for each day of the week. We can see from Figure 5.6 that the average speed is clearly different in the weekends compared to working days. The average travel speed pattern for working days is comparable though, and so are the patterns for Saturday

Hour	Average speed						
	Mon.	Tue.	Wed.	Thu.	Fri.	Sat.	Sun.
0	113.8	113.7	113.5	113.8	113.4	110.3	110.3
1	115.5	115.6	115.8	115.5	115.2	111.6	110.2
2	116.2	116.3	116.1	115.9	116.2	113.4	111.4
3	116.3	115.9	116.2	116.2	115.9	114.4	113.9
4	115.3	115.6	115.5	115.4	115.1	114.5	114.7
5	112.2	111.8	112.1	112.1	112.2	114.0	114.5
6	98.3	98.7	97.2	98.8	96.9	112.4	112.4
7	55.6	54.7	56.6	51.2	47.6	109.5	109.9
8	59.9	54.6	53.6	52.9	51.5	103.0	103.6
9	76.1	75.2	74.2	74.8	72.2	96.7	99.2
10	85.3	86.6	85.0	83.7	81.7	96.1	98.3
11	89.4	88.4	85.4	88.2	86.5	96.0	98.0
12	88.5	89.1	87.5	86.8	86.4	94.3	97.8
13	85.5	84.2	83.1	83.1	85.0	90.6	91.8
14	87.5	84.6	84.9	85.2	83.7	90.7	94.3
15	85.5	84.8	84.7	85.2	82.0	95.9	98.0
16	75.0	73.2	71.8	72.2	68.7	95.3	98.5
17	65.8	64.5	64.8	63.3	61.7	92.1	93.2
18	77.8	75.9	76.0	76.7	74.3	89.2	91.9
19	92.5	93.2	90.6	92.6	89.8	95.9	97.3
20	104.4	102.7	101.7	102.4	101.9	101.2	102.9
21	108.0	107.7	106.7	106.9	106.7	104.9	106.0
22	108.3	107.6	107.2	107.3	107.4	106.7	107.0
23	110.7	110.3	110.1	110.5	109.5	108.7	109.7

Table 5.1. *Average speed (km/h) from Mechelen North to Rumst, 1995 (A1/E19).*

and Sunday. For working days, we see that the average travel speed is drastically reduced during peak hours from about 7:00h to 9:00h, and from about 16:00h to 18:00h. We can also see that even during the day, the travel speed is not even near the maximum allowed travel speed of 120 km/h. This may be caused by freight traffic that also contributed to the computed average speeds.

We decided to use only the time-dependent travel speeds of Monday because we use the road network of the Netherlands to evaluate time-dependent planning, and in the Netherlands even an extremely long time-route does not take longer than 4 to 5 hours. Note that $t_e^t(e,t)$ is a continuous function and we have only data on travel speeds available for every hour. To obtain a continuous function $t_e^t(e,t)$, the travel speed of an edge when departing at time t, with for example 5:00h $\leq t \leq$ 6:00h, is

computed by using linear interpolation. For example, for $t = 5{:}25$h, the travel speed is equal to $112.2 + 25\frac{98.3-112.2}{60} = 106.4$.

From a given driving speed for edge e at time t, we compute the driving time for edge e at time t by assuming that the driving speed of the driver for the entire edge is the same and equal to the driving speed with which the driver departs from the start node of edge e at time t. So, if the driving speed of edge e at time t is denoted by $v(e,t)$, then the driving time of edge e for a driver starting from the start node of edge e at time t is equal to $\frac{\ell(e)}{v(e,t)}$, with $\ell(e)$ the length of edge e. It is also possible to assume that the driving speed changes while the driver is traversing the edge. In this case it would be useful to compute the driving time of an edge beforehand and store this information instead of performing a more involved computation to determine the real driving time while planning a time-route. However, the driving time of every edge can be different, while multiple edges have the same driving speed.

Because we minimize driving time, we should check whether for our data, the roadgraph is time consistent. Note that there are only time-independent turn restrictions with infinite costs. Checking the time consistency of our roadgraph thus comes down to determining whether $t_1 + t^{\mathrm{t}}_{\mathrm{e}}(e,t_1) \leq t_2 + t^{\mathrm{t}}_{\mathrm{e}}(e,t_2)$ for every edge e and $t_1 \leq t_2$, see Lemma 5.1. Consider the driving speeds $v(e,t)$ in Table 5.1. We have $t^{\mathrm{t}}_{\mathrm{e}}(e,t) = \frac{\ell(e)}{v(e,t)}$, for edge e and time t. In order to satisfy the time consistency constraint, we need $t^{\mathrm{t}}_{\mathrm{e}}(e_2,t_1) - t^{\mathrm{t}}_{\mathrm{e}}(e_2,t_2) \leq t_2 - t_1$ for $t_1 \leq t_2$. Or equivalently, $\frac{t^{\mathrm{t}}_{\mathrm{e}}(e_2,t_2) - t^{\mathrm{t}}_{\mathrm{e}}(e_2,t_1)}{t_2-t_1} \geq -1$, for every $t_1 < t_2$. Because we use linear interpolation, function $t^{\mathrm{t}}_{\mathrm{e}}(e,t)$ is continuous. Taking the limit of $t_1 \uparrow t_2$ or $t_2 \downarrow t_1$, this leads to $\frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} \geq -1$, where $\frac{\partial f(t)}{\partial t}$ denotes the first left or right derivative of function $f(t)$ to t. Since $t^{\mathrm{t}}_{\mathrm{e}}(e,t) = \frac{\ell(e)}{v(e,t)}$, we need to show that $\frac{-\ell(e)}{(v(e,t))^2} \cdot \frac{\partial v(e,t)}{\partial t} \geq -1$ for all edges e and times t, in order to prove that our time-dependent roadgraph is time consistent.

The largest decrease in driving speed in one hour in Table 5.1 is 49.3 km/h (Friday from 6:00h to 7:00h). Due to the existence of very short edges, both departure times and driving times of edges are measured in seconds, and distances are measured in meters. Because we use linear interpolation, the left derivative for this hour is equal to $\frac{-49{,}300}{3{,}600 \cdot 3{,}600}$. The largest increase in driving speed in one hour is 21.9 km/h (Thursday from 8:00h to 9:00h). The left derivative for this hour is equal to $\frac{21{,}900}{3{,}600 \cdot 3{,}600}$. The driving speed is at most 116.3km/h, and at least 47.6km/h. This means that $\frac{-\ell(e)}{(v(e,t))^2} \cdot \frac{\partial v(e,t)}{\partial t} \geq \frac{-\ell(e)}{(47{,}600/3{,}600)^2} \cdot \frac{21{,}900}{3{,}600 \cdot 3{,}600} \geq -1e^{-5}\ell(e) > -1$, for $\ell(e) < 100{,}000m$, which is realistic. Furthermore, we have $\frac{-\ell(e)}{(v(e,t))^2} \cdot \frac{\partial v(e,t)}{\partial t} \leq \frac{-\ell(e)}{(116{,}300/3{,}600)^2} \cdot \frac{-49{,}300}{3{,}600 \cdot 3{,}600} \leq 4e^{-6}\ell(e) \leq 1e^{-5}\ell(e)$.

The Netherlands

We have also obtained data from the AVV Transport Research Center of the Ministry of Transport, Public Works and Water Management on average travel speeds for a large number of motorways in the Netherlands. The data is based on measurements done on working days in 2002. Specifically, for every working day in 2002, the average realized travel speed per 15 minutes has been recorded, for the motorways in Table 5.2 in the Netherlands for both directions. Figure 5.7 gives a typical example of the average travel speed during a working day in 2002 for a motorway in the Netherlands.

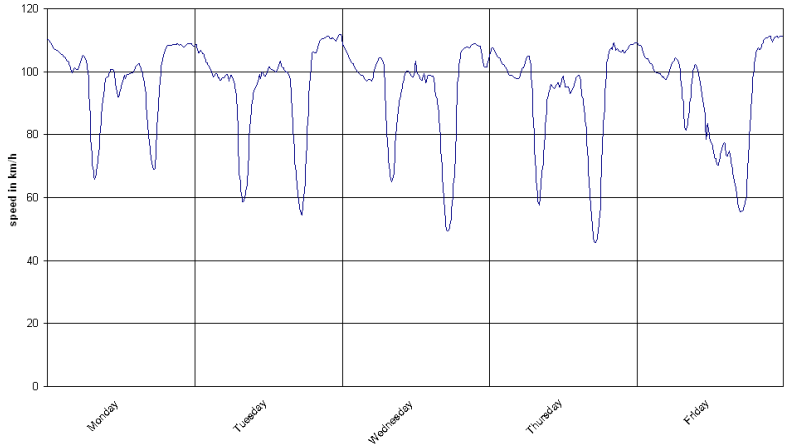


Figure 5.7. *Average speed (km/h) from Eindhoven to Den Bosch in 2002.*

The average travel speed for all motorways, computed by taking the inverse of the average of the inverse of the travel speed for all motorway sections in Table 5.2 in both directions, is given in Figure 5.8.

Just as for the data from Antwerp we compute the travel speed for a time t using linear interpolation. Note that the time-dependent roadgraph is also time consistent for this data-set. Furthermore, we can see that the data obtained from Van Woensel [2003], which was determined by simulation using actual traffic data, quite accurately represents the average speeds on Dutch motorways. Obviously, differences exist, because some motorways are less congested than others. However, the data from Antwerp can be used to represent the travel speeds on all motorways in the Netherlands, in order to test time-dependent planning. General trends and results will be similar for both data sets. If time-dependent route planning is actually implemented in a real car navigation system, the most recent data should be used.

In the next sections, we consider route planning with time-dependent travel speeds. The data from Antwerp, and specifically the data on travel speeds on Monday is used to plan time-dependent routes in the Netherlands.

Dutch motorways				
A76	:	Heerlen	↔	Maasbracht
A12	:	The Hague	↔	Gouda
A2	:	Den Bosch	↔	Eindhoven
A20	:	Gouda	↔	Vlaardingen
A58	:	Eindhoven	↔	Tilburg
A29	:	Oud-Beijerland	↔	Westland
A58	:	Breda	↔	Tilburg
A4	:	The Hague	↔	Rijnmond
A27	:	Breda	↔	Gorinchem
A15	:	Europoort	↔	Papendrecht
A16	:	Breda	↔	Dordrecht
A15	:	Gorinchem	↔	Rotterdam
A2	:	Culemborg	↔	Den Bosch
A16	:	Dordrecht	↔	Rotterdam
A2	:	Beesd	↔	Utrecht
A13	:	The Hague	↔	Rotterdam
A2	:	Gorinchem	↔	Utrecht
A1	:	Barneveld	↔	Bussum
A28	:	Amersfoort	↔	Utrecht
A1	:	Amersfoort	↔	Amsterdam
A12	:	Arnhem	↔	Ede
A6	:	Almere	↔	Amsterdam
A325	:	Arnhem	↔	Nijmegen
A2	:	Amsterdam	↔	Utrecht
A12	:	Ede-Wageningen	↔	Utrecht
A8	:	Burgerveen	↔	Zaanstad
A50	:	Arnhem	↔	Deventer
A7	:	Amsterdam	↔	Purmerend
A27	:	Hilversum	↔	Utrecht
A9	:	Alkmaar	↔	Haarlem
A12	:	Bodegraven	↔	Utrecht
A5	:	Amsterdam	↔	Haarlem
A12	:	Gouda	↔	Utrecht

Table 5.2. *Motorways for which average speed data is available.*

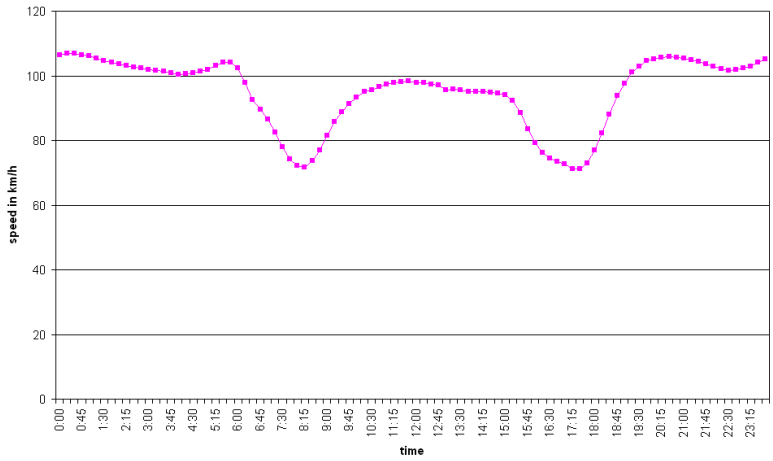


Figure 5.8. *Average speed (km/h) on motorways in the Netherlands in 2002.*

5.4.2 General Results

In this section, we present general results of planning time-dependent routes with algorithm $T^\star$ (see Figure 5.5) using the unpartitioned road network of the Netherlands. When we plan a time-dependent route, the time-dependent travel times from Antwerp in Section 5.4.1 are used. The importance of a road is indicated by the so-called road class: the less important a road, the higher the road class. Note that edges with road class 0 are generally part of a motorway. Edges with road class 1 are generally part of a highway. Specifically, we assume that all roads of road class 0 have a time-dependent travel time equal to the time-dependent travel time from Antwerp in Section 5.4.1 on Monday. If a driver departs at 23:59h on Monday, we assume that when the driver arrives at the end node of the edge after for example 2 minutes, the time-dependent travel time of Monday 00:01h applies. Furthermore, because edges with road class 1 (with an assumed time-independent speed of 100 km/h) are also important roads that are subject to congestion, we assume the travel speeds on these roads are equal to $\frac{100}{120}$ times the time-dependent travel time from Section 5.4.1 on Monday. Of course, also less important edges are subject to daily congestion. However, there is no data available on travel speeds on less important roads. Therefore, we assume that the travel speeds on these roads are time-independent. Alternatively, we could also determine a time-dependent speed for these roads ourselves.

As was already noted, time-dependent planning results in different optimum time-routes for different departure times. This is illustrated by Figure 5.9, which displays the optimum time-dependent route from Eindhoven to The Hague for three different departure times. As can be seen from this picture, departing at 3:00h in the morning leads to a time-route that contains a lot of motorway sections. Departing at

9:00h in the morning, leads to a different time-route because daily congestion still influences the travel time of the time-route. As a result, an alternative time-route becomes the optimum time-route which uses fewer motorways especially near the beginning of the journey since at that time delays still occur. If the driver departs at 6:00h, the optimum time-route is again different. For this departure time the driver experiences daily congestion during most of his journey. As a result, this time-route uses only a few motorway sections, especially at the start of the time-route because the delays are the least severe at that time.

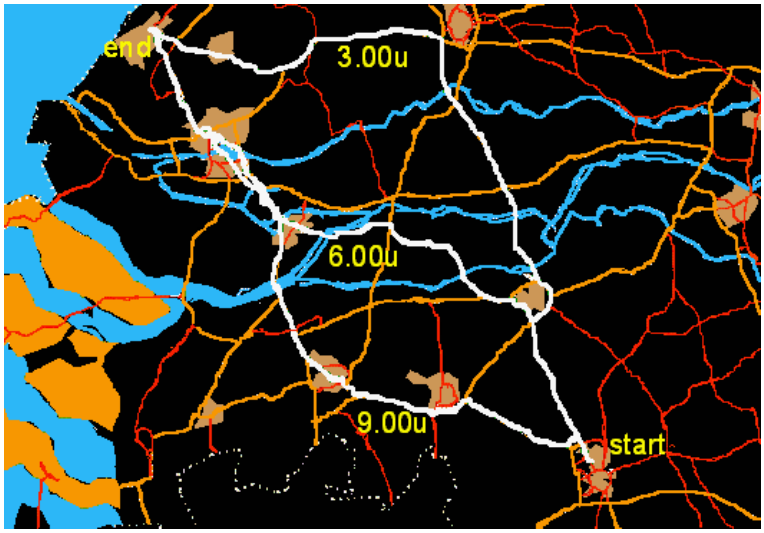


Figure 5.9. *Three optimum time-routes from Eindhoven to The Hague.*

In general, we see that planning a time-dependent route leads to different time-routes for different departure times. Usually, for departure times that lead to a large part of the time-route being driven in peak hours, the optimum time-dependent route for that departure time becomes shorter than the time-independent route. Of course, sometimes also the opposite occurs. However, we generally see that during peak hours, the optimum time-dependent route uses short-cuts to avoid serious congestion. These short-cuts tend to consist of secondary roads, i.e. the behavior of time-dependent route planning corresponds to human behavior in the sense that also drivers who are very familiar with a certain part of the road network tend to use short-cuts to avoid congestion. So, using time-dependent route planning also drivers that are not familiar with the road network can avoid congestion. Note that the more cars are equipped with a car navigation system offering time-dependent route planning, the higher the importance to use time-dependent travel times for secondary roads becomes. Note that we assume the travel times are not influenced by the time-routes presented by the navigation system.

We also see that during most of the day, the time-dependent routes are different from the time-independent routes. So, taking time-dependent travel speeds into account really influences the optimum time-route. Again, time-dependent routes use fewer motorways because of the congestion. Furthermore, also in the middle of the night, the real travel speeds are not equal to the maximum allowed speeds of 120 km/h or 100 km/h, which is probably caused by the relatively low travel speed of freight traffic which is also taken into account when determining the average time-dependent travel speed. The fact that the maximum time-dependent travel speed and the maximum allowed travel speed differ is another reason why time-dependent routes differ from time-independent routes. With time-independent route planning, we assume that the maximum allowed speed is realized, which is not the case even if no congestion is present as can be seen from Table 5.1.

Furthermore, the results show that despite the fact that different time-routes may be optimum for different departure times, the number of different time-routes that is an optimum time-route for a particular departure time is limited compared to the number of departure times that we used to test time-dependent planning. In particular, 96 different departure times were used, and on average the number of distinct optimum time-routes was about 5, and at most 10. This means that there are only a limited number of interesting alternative time-routes during a day, which seems logical.

In the next section, we discuss the usefulness of time-dependent route planning for a driver.

5.4.3 Time-Dependent versus Time-Independent Planning

In the previous section, we generally discussed the impact of time-dependent route planning. In this section, we compare the results of time-dependent and time-independent route planning.

We can also use time-independent travel speeds in order to compute the optimum routes in the route batch from Table 4.3. This generally results in different routes for time-dependent route planning as was already discussed in Section 5.4.2. In order to evaluate the use of time-dependent planning compared to time-independent planning, we compare both approaches in this section. Specifically, we determine the time-dependent optimum time-route $\pi = (\langle e_1, \ldots, e_\ell \rangle, t)$ for different departure times t as well as the time-independent optimum route $p = \langle a_1, \ldots, a_n \rangle$. Then we determine for both the time-dependent time-route and the time-independent route the arrival times $T_\pi^t(t)$ and $T_p(t)$ respectively. Secondly, we determine for the time-independent route, the time-dependent arrival time assuming time-dependent travel speeds. So, we take route $p = \langle a_1, \ldots, a_n \rangle$, and determine for edge a_i the time-dependent travel time, using that the driver departs from node $\delta_1(a_1)$ at time t. This again results in an arrival time at the destination, which we denote by $T_p^t(t)$. Finally, we compute the arrival time of the time-dependent route assuming that the time-independent travel speeds are

realized. So, we compute the travel time using the time-independent travel speed for each edge e_i in route $\langle e_1, \ldots, e_\ell \rangle$. The resulting arrival time is denoted by $T_\pi(t)$. So, summarizing, we have for departure time t

$T_p(t)$: Arrival time of time-independent route p;
$T_p^t(t)$: Arrival time of time-independent route p for time-dependent travel times;
$T_\pi(t)$: Arrival time of time-dependent route π for time-independent travel times;
$T_\pi^t(t)$: Arrival time of time-dependent route π.

Note that we decided to measure the (departure) time in seconds, and $t = 0$ denotes a departure at midnight (from Sunday to Monday). Now, we compute for each of the 2,000 start and destination nodes in the route batch, the value of $T_p(t)$, $T_p^t(t)$, $T_\pi(t)$, $T_\pi^t(t)$ for $t = 0, 900, \ldots, 86,400$, i.e. we determine the optimum time-dependent route departing every 15 minutes (on Monday). The averages, standard deviations and maxima of $T_p^t(t) - T_\pi^t(t)$ and $T_\pi(t) - T_p(t)$ can be found in Figures 5.10, 5.11 and 5.12 respectively. All realizations of $T_p^t(t) - T_\pi^t(t)$ and $T_\pi(t) - T_p(t)$ are displayed in Figures 5.13 and 5.14 respectively. Figure 5.15 and Figure 5.16 give the distribution of $T_p^t(t) - T_\pi^t(t)$ and $T_\pi(t) - T_p(t)$ respectively for all 2,000 routes for departure times 7:00h, 10:30h and 16:30h.

Because the optimum time-dependent route can be different for every possible departure time t, also the value of $T_\pi(t)$ can be different for different departure times although the route costs are independent of that departure time. Therefore, Figures 5.10 to 5.16 all show different results for different departure times. Note that the average of $T_p(t)$ is about 54 minutes.

When we consider the results in Figures 5.10, 5.12 and 5.13, we see that if the time-dependent travel speeds are accurate, i.e. the actual travel speeds correspond exactly to the used time-dependent travel speeds, then using these time-dependent travel speeds results in an earlier arrival time at the destination, especially during rush hours. Specifically, for our route batch, the driver can arrive at his destination up to 48 minutes earlier (see Figure 5.12), if the time-dependent travel speeds correspond to the actual travel speeds and are used during route planning. The decrease in average total travel time is lower, but during rush hours the decrease is about 3 to 12 minutes, see Figure 5.10, and the effort of using time-dependent travel times can still be considered worthwhile. During the night and in the middle of the day, the gain of using these time-dependent travel times is much lower, as could be expected. During these hours, the decrease in travel speeds is much lower, and the time-dependent travel speeds are more closely resembled by the time-independent (maximum) travel speeds. Furthermore, the standard deviation of the difference in arrival time is relatively high compared to the average difference in arrival time as can be seen from Figures 5.10 and 5.11. This means there is a lot of variation in the difference between the time-dependent travel time and the time-independent travel time. We can also see

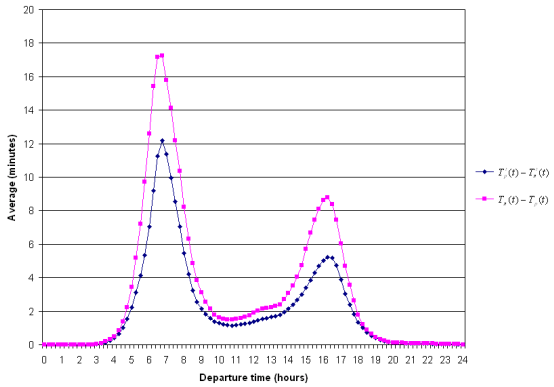


Figure 5.10. *Average of* $T_p^t(t) - T_\pi^t(t)$ *and* $T_\pi(t) - T_p(t)$.

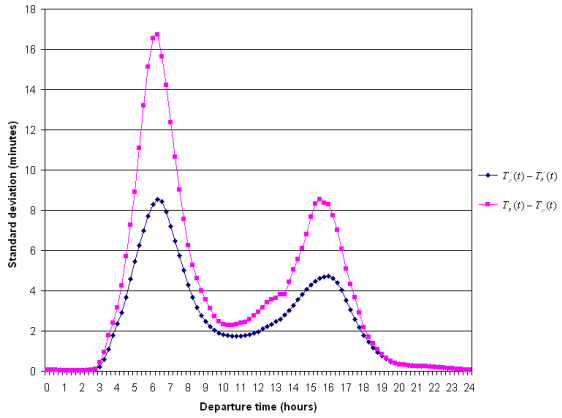


Figure 5.11. *Standard deviation of* $T_p^t(t) - T_\pi^t(t)$ *and* $T_\pi(t) - T_p(t)$.

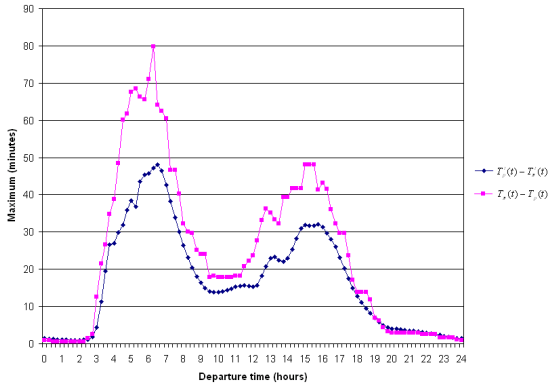


Figure 5.12. *Maximum of* $T_p^t(t) - T_\pi^t(t)$ *and* $T_\pi(t) - T_p(t)$.

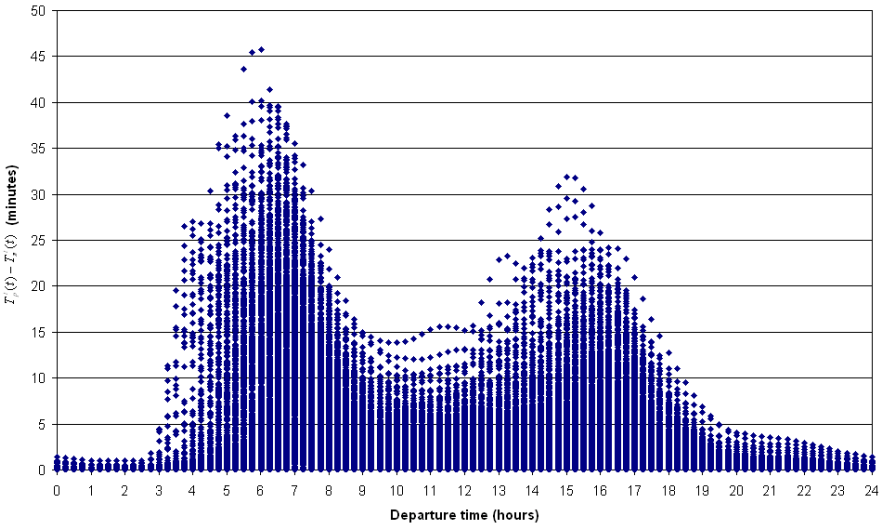


Figure 5.13. *All realizations of* $T_p^t(t) - T_\pi^t(t)$.

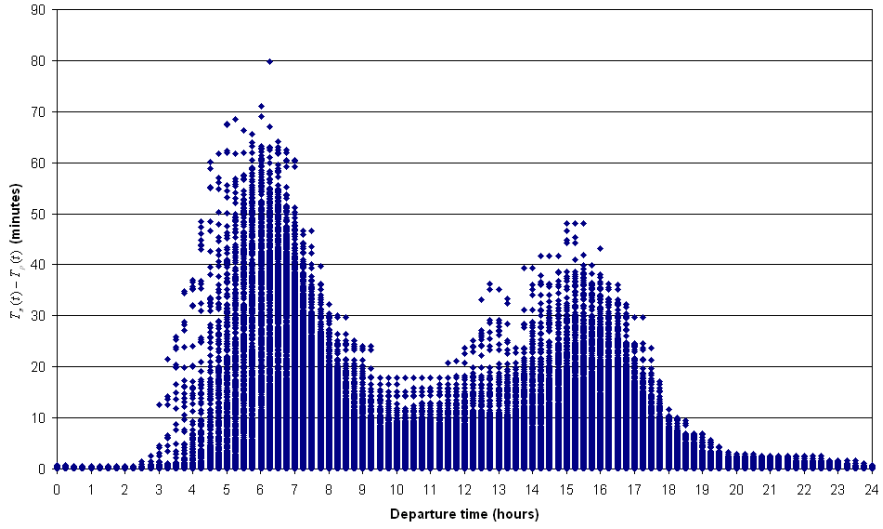


Figure 5.14. *All realizations of* $T_\pi(t) - T_p(t)$.

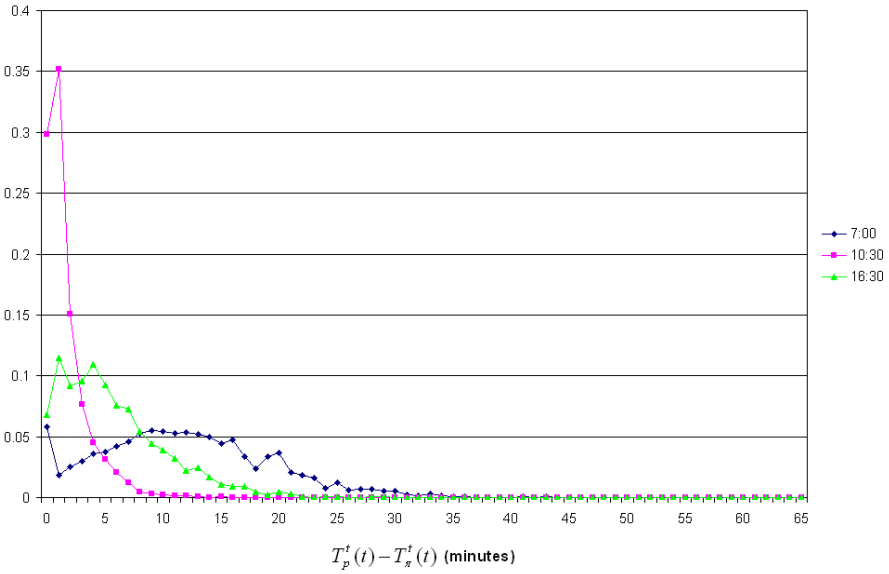


Figure 5.15. *Distribution of $T_p^t(t) - T_\pi^t(t)$ for 7:00h, 10:30h and 16:30h.*

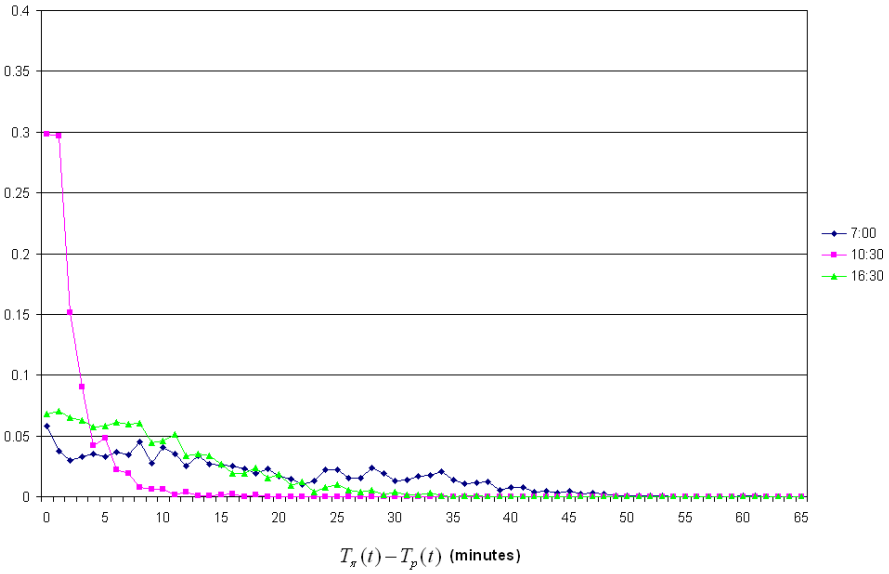


Figure 5.16. *Distribution of $T_\pi(t) - T_p(t)$ for 7:00h, 10:30h and 16:30h.*

that the standard deviation of the decrease in total travel time for time-independent travel speeds compared to time-dependent travel speeds is higher during rush hours, as can be seen from Figure 5.11. From Figures 5.15 and 5.16 we can also see that the deviation in $T_p^t(t) - T_\pi^t(t)$ and $T_\pi(t) - T_p(t)$ is much smaller for departure time 10:30h than for departure times 7:00h and 16:30h as could be expected.
As can be seen from Figures 5.13 and 5.14, also routes exist that are not influenced by the use of time-dependent instead of time-independent travel speeds. These routes do not use any roads of road class 0 and 1, and therefore do not contain any edges that have a time-dependent travel time, or no good alternative route exists. As a result, the arrival times of the routes using time-dependent and time-independent travel times are equal. This also explains the relatively low average decrease in travel time. Note that the more edges of the time-independent route have a time-dependent cost, (i.e. edges with road class 0 or 1 in our case), the greater is the impact of time-dependent route planning.

Figures 5.10, 5.12 and 5.14 consider what happens if the used time-dependent travel speeds are not according to reality, and the actual travel speeds are given by the time-independent (maximum) speeds instead. We can see from these figures that in that case, using time-dependent travel speeds leads to a delayed arrival at the destination. This can be explained from the fact that the travel speeds that are used to plan the route are too low. As a result, the planned optimum time-route is no longer optimum. As can be seen from the figures, the delay is largest for rush hours, up to 80 minutes, which is caused by the larger error in the used travel speeds compared to the real actual travel speeds.

Note that because route p is optimum for time-independent travel times, we have $T_p(t) \leq T_\pi(t)$. Furthermore, time-route π is optimum for time-dependent travel times, so $T_\pi^t(t) \leq T_p^t(t)$. Because the time-independent travel speeds are at least as high as the time-dependent travel speeds, we also have $T_p(t) \leq T_p^t(t)$ and $T_\pi(t) \leq T_\pi^t(t)$. This leads to $T_p(t) \leq T_\pi(t) \leq T_\pi^t(t) \leq T_p^t(t)$. This also means that $T_\pi^t(t) - T_\pi(t) \leq T_p^t(t) - T_p(t)$. This can be explained from the fact that time-dependent route π contains fewer edges with a time-dependent travel speed than the time-independent route p because these edges are more "expensive" in a time-dependent planning.

Furthermore, we can see that in general $T_\pi(t) - T_p(t)$ is larger than $T_p^t(t) - T_\pi^t(t)$, so $T_\pi^t(t) - T_p^t(t) \leq T_p(t) - T_\pi(t)$. This applies to the average, the standard deviation, the maximum and to individual routes as can be seen from Figure 5.10, Figure 5.11, Figure 5.12 and from Figure 5.13 compared to Figure 5.14 respectively. In order to explain this phenomenon, consider the following example, in which we temporarily use hours and kilometers to measure the travel time and length of the route. Suppose there exists a time-independent route p with a length of 120 km, and an average speed of 120 km/h, i.e. the route uses only motorways. The travel time of this route is one hour, $T_p(t) = 1$. The travel time $T_p^t(t)$ for this route is higher because the

time-dependent travel speeds are lower. Assume that the time-dependent average speed is 60km/h for this route p. This leads to $T_p^t(t) = 2$. Now a time-dependent route π is planned. Route π generally uses fewer motorways and is usually a bit shorter. We assume that route π does not contain any time-dependent edges and is 100 km long and has average speed v. This leads to a travel time of $T_\pi^t(t) = T_\pi(t) = \frac{100}{v}$. Because $T_p(t) \leq T_\pi(t) = T_\pi^t(t) \leq T_p^t(t)$, there apparently exists a time-route π with an average speed $v \in (50, 100)$. For average speeds v close to 100 km/h we have $T_\pi^t(t) - T_p^t(t) \leq T_p(t) - T_\pi(t)$. For average speeds close to 50 km/h the opposite applies, $T_\pi^t(t) - T_p^t(t) \geq T_p(t) - T_\pi(t)$. The results in Figures 5.10 to 5.14 show that we have $T_\pi^t(t) - T_p^t(t) \leq T_p(t) - T_\pi(t)$, which means that for our road network there exist time-routes π that are considerably shorter and use much fewer time-dependent edges and have an average speed that is not much lower than the time-independent speed of the time-dependent edges. The Netherlands has a very dense road network with many possible alternative routes. As a result, a considerably shorter route can usually be found, with only a limited reduction in the average travel speed. If the maximum travel speeds on time-dependent edges (i.e. motorways and highways) is much higher than for less important other roads (like in Germany for example) and the road network is not as dense (like in France for instance) then it is more difficult to find good alternatives for route p and it is more likely that we find the opposite effect, i.e. $T_\pi^t(t) - T_p^t(t) \geq T_p(t) - T_\pi(t)$. This can be seen most clearly from Figure 5.12. We see that $T_\pi(t) - T_p(t) > T_p^t(t) - T_\pi^t(t)$ for almost all t. Especially during non-peak hours, also $T_\pi^t(t) - T_p^t(t) \geq T_p(t) - T_\pi(t)$ occurs. Moreover, we see from Figures 5.10 to 5.14 that the difference between $T_p(t) - T_\pi(t)$ and $T_\pi^t(t) - T_p^t(t)$ increases for peak hours when the travel time of route p increases most strongly.

Note that it is not according to reality to assume that travel speeds are time-independent. As can be seen from the data in Section 5.4.1, travel speeds are time-dependent. However, the stored time-dependent travel speeds are not necessarily equal to the actual travel speeds at a particular time, day and location. Therefore, the results of $T_\pi(t) - T_p(t)$ are given to indicate the possible delay in arrival time if the used time-dependent travel speeds are lower than the actual speeds. Note that if the actual speeds are even below the used time-dependent travel speeds, then it is definitely preferable to use the time-dependent travel speeds (that are still too high), because using time-independent (maximum) travel speeds, would lead to even larger delays. However, in estimating the time-dependent travel speeds (for the Netherlands), overestimating travel speeds is preferable to an underestimation because underestimating travel speeds leads to optimum time-routes with short-cuts via unimportant roads, as can be concluded from the found relation $T_\pi(t) - T_p(t) > T_p^t(t) - T_\pi^t(t)$.

Because there is no data available on travel speeds on non-motorways, we have assumed that the travel speeds on these roads are time-independent. However, all roads are more or less time-dependent. An alternative approach is to also proportion-

ally reduce the time-dependency of these roads depending on the maximum allowed speed. So a time-dependent travel time for an edge with maximum speed of 80 km/h could be determined by $\frac{80}{120}v(e,t)$, with $v(e,t)$ the time-dependent travel speed for an edge of road class 0. A possible extension to this approach is to determine the time-dependent travel speed proportional to the number of (more important) adjacent edges. For example, first the time-dependent travel speeds of all edges adjacent to edges of road class 0 can be determined. If the number of adjacent edges leading to edge e with a road class at most i is denoted by $\alpha_i^1(e)$, and similarly the number of adjacent edges departing from edge e with a road class at most i is denoted by $\alpha_i^2(e)$, we could determine the time-dependent travel speed of edge e_1 with road class i for example by $\frac{\textit{maximum speed } e_1}{120}\frac{\alpha_i^1(e_1)}{\alpha_i^2(e_1)}v(e,t)$. Needless to say, there are many more alternatives.

5.5 Combination with Partitions

Section 5.4 focussed on time-dependent route planning using the entire unpartitioned road network of the Netherlands. As was shown in Chapter 4 optimum time-independent routes can be planned fast by using a partition. In this section we discuss time-dependent planning using a partition. In Section 5.5.1 we discuss how time-dependent route planning and the use of partitions can be combined theoretically. In practice, this may not always turn out to be a viable approach. Therefore, we show in Section 5.5.2 a more practical way to integrate time-dependent planning with the use of partitions. The quality of planned time-dependent routes using this approach is discussed in Section 5.5.3.

5.5.1 Optimum Planning

As defined in Chapter 2, a time-dependent roadgraph G^{t} is represented by a tuple $G^{\mathrm{t}} = (N, E, w_{\mathrm{e}}^{\mathrm{t}}, w_{\mathrm{r}}^{\mathrm{t}}, t_{\mathrm{e}}^{\mathrm{t}}, t_{\mathrm{r}}^{\mathrm{t}})$. The time-independent roadgraph G is represented by a tuple $G = (N, E, w_{\mathrm{e}}, w_{\mathrm{r}})$. Although the cost functions for both graphs can be totally different, the node and edge sets of both roadgraphs are the same for a particular road network. As a result, the partition of a road network as discussed in Chapter 3, can be used for route planning in both a time-independent and a time-dependent roadgraph.

As shown in Chapter 4, we can plan optimum time-independent routes using partitions, if the cost of the optimum route between every pair of boundary nodes is stored in a route graph. We discussed three route graphs that can be used to store these costs. This section discusses what is necessary to plan optimum time-dependent routes using partitions.

For now we assume that Clique graphs are used to store the optimum time-route cost between every pair of boundary nodes of a cell. As explained earlier in this chapter, the cost of a time-dependent route depends on the departure time of the driver. As a result, the minimum time-route cost between pairs of boundary nodes

also depends on the time of departure from the boundary start node. So, the most logical course of action is to store for every possible departure time the optimum time-route cost between every pair of boundary nodes of a cell. This leads to the creation of time-dependent route edges. Furthermore, the arrival time of the driver at a boundary node for a particular departure time is also important in determining the cost of a time-route containing time-dependent route edges. Therefore, we need to store not only the time-dependent minimum route cost between each pair of boundary nodes of a single cell, but also the matching driving time.

In Section 5.2 we discussed under which assumptions the optimum time-route can be planned using a (slightly) modified standard shortest path algorithm. We showed that if the roadgraph is cost consistent, then there exists an optimum time-route $\langle e_1, \ldots, e_\ell \rangle$ for which the time-route from $\delta_1(e_1)$ to edge e_i, with $i = 1, \ldots, \ell$, is also a minimum-cost time-route. As a result, time-route $\langle e_i, \ldots, e_j \rangle$ is also a minimum-cost time-route for $1 \leq i \leq j \leq \ell$. This means that, because only turn restrictions are present between internal edges, we can plan time-dependent optimum routes in cost consistent roadgraphs if we store both the optimum route cost and matching driving time for every possible departure time between every pair of boundary nodes of a single cell in a Clique graph. For Star graphs the same arguments hold. If a partition using Point graphs is created such that only between internal edges turn restrictions exist, then the same reasoning can be followed for Point graphs as well. Furthermore, if not all turn restrictions are formulated on internal edges, then we can still plan optimum time-dependent routes using partitions, because (as for time-independent planning) the costs (and driving time) of the optimum time-routes between every pair of boundary edges of a cell needs to be stored. These costs (and driving times) include the costs (and turning times) of possible turn restrictions.

So, in theory we can plan optimum time-dependent routes in cost consistent roadgraphs by storing for every possible departure time, the minimum time-route cost and matching time-route driving time between every pair of boundary nodes of a single cell.

As has been shown in Chapter 3, depending on the partition, a lot of costs may need to be stored. Now that costs have to be stored for every departure time, the amount of data that needs to be stored increases drastically especially if we measure time in seconds. It is obviously not realistic to store a cost and driving time for every second of the day between every pair of boundary nodes of a single cell. Section 5.5.2 discusses a way to substantially reduce the amount of data that needs to be stored.

5.5.2 Approximate Planning

In order to study the possibilities to reduce the amount of data that has to be stored to plan time-dependent routes, we consider planning fastest time-dependent routes using partitioned road networks of the Netherlands. For fastest routes, only the driving time needs to be stored for each departure time between every pair of boundary nodes

of a single cell.

We use a *profile* χ to store the optimum route costs and travel times between a pair of boundary nodes. Specifically, we only store the travel time and route costs for every T_χ departure time units. All route costs and travel times for other departure times are computed from this profile using linear interpolation. Note that in general also the time-dependent costs and travel times are not stored for every possible departure time. Also these costs are only stored for a selected number of departure times, usually for every T_e departure time units. In our case, we use the data from Antwerp which is known for every hour. Because the data for the Netherlands is known for every 15 minutes, and for implementation in a car navigation system this data is very likely to be used, we choose to use $T_\chi = 15$ minutes. Definition 5.3 gives the formal definition of a profile.

Definition 5.3. Let $T_{min}, T_{max} \in \mathbb{R}_0^+$ be the minimum and maximum departure time respectively. Let $T_\chi \in \mathbb{R}^+$ be such that $\exists i \in \mathbb{N}$ with $T_{max} - T_{min} = T_\chi \cdot i$. Define $n(\chi) = \frac{T_{max}-T_{min}}{T_\chi}$. A *profile* χ is defined as $\chi = \langle \chi_0, \ldots, \chi_{n(\chi)} \rangle$ with $\chi_j \in \mathbb{R}_0^+ \cup \{\infty\}$. We denote the cost at departure time $T_{min} + j \cdot T_\chi$ by χ_j, and the set of all profiles by X. □

This already drastically reduces the amount of information that needs to be stored because costs are only stored for departure times $T_{min} + j \cdot T_\chi$. However, for a car navigation system, it may still be too much. Therefore, we further reduce the amount of information that needs to be stored. Note that for every route edge we need to store at most two profiles, one for the route cost and one for the driving time. In some cases, for example if the costs are equal to the driving distance, the cost profile usually contains just a few different costs. In this case it is beneficial to store a cost and a (collection of) intervals for which this cost is the optimum route cost. The driving time is only the same for many departure times for routes that do not contain any time-dependent edges, in our case edges of road class 0 or 1.

For every pair of boundary nodes of a single cell and for every possible departure time, we compute an optimum time-dependent route $\pi = (\langle e_1, \ldots, e_\ell \rangle, t)$. Subsequently, we compute the time-dependent cost $c(p,i)$ of route $p = \langle e_1, \ldots, e_\ell \rangle$ for every departure time $t = T_{min} + i \cdot T_\chi$, $i = 0, \ldots, n(\chi)$. These costs lead to the cost profile $\langle c(p,0), \ldots, c(p,n(\chi)) \rangle$ for route p. Two example profiles are given in Figure 5.17. We can rewrite a profile as $\chi = \langle \chi_0, \ldots, \chi_{n(\chi)} \rangle$ with $\chi_i = \frac{c(p,i)}{\min_i c(p,i)}$ by storing the minimum time-dependent cost of route p, $b = \min_i c(p,i)$. Both profiles in Figure 5.17 can be represented by the profile in Figure 5.18, by using $b_1 = 14.6$ and $b_2 = 21.9$. Computing a profile with matching minimum time-dependent cost b_i for every pair of boundary nodes of a single cell and for every possible departure time leads to a set of profiles $X = \{\chi^1, \ldots, \chi^m\}$ with $\chi^i_k \geq 1$ for every i,k, and a minimum time-dependent cost b_i for every pair of boundary nodes. We then store for each edge the matching profile number.

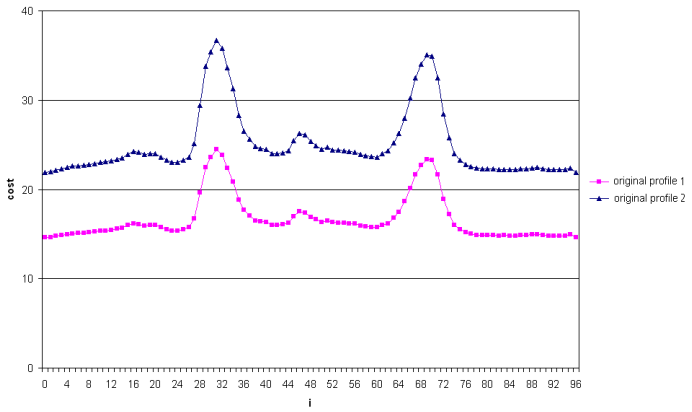


Figure 5.17. *Two (identical) profiles.*

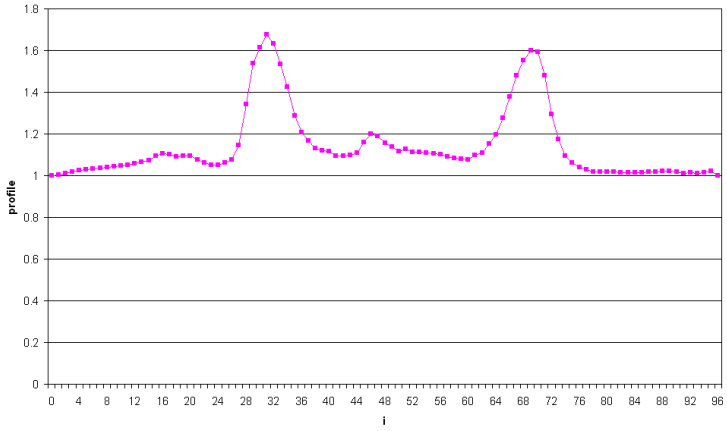


Figure 5.18. *Profile describing the profiles in Figure 5.17 with* $b_1 = 14.6$, $b_2 = 21.9$.

We can further reduce the number of profiles that needs to be stored for a particular partition by using an approximation. Definition 5.4 defines the necessary variables.

Definition 5.4. Let $b_i \in \mathbb{R}_0^+$ denote the minimum time-dependent cost of profile χ_i of a time-route π. Let $f_i \in \mathbb{R}_0^+$ denote a *multiplication factor* for profile χ_i of time-route π. Let $i \in \mathbb{N}$ denote the *profile number* of time-route π. The *profile cost* of a time-route π is denoted by tuple (i, b_i, f_i). The set of all profile costs is denoted by Γ. Note that $|\Gamma| = |X|$. □

From a profile cost we can compute the cost of a time-route π for a particular departure time $t = T_{min} + j \cdot T_\chi$ with $j = 0, \ldots, n(\chi)$ as in Definition 5.5.

Definition 5.5. Given T_χ, T_{min}, T_{max} and departure time $t = T_{min} + j \cdot T_\chi$ with $j = 0, \ldots, n(\chi)$, the cost of a time-dependent edge e with profile χ^i and profile cost (i, b_i, f_i) at departure time t is given by

$$w_e(e,t) = b_i \cdot (1 + f_i \cdot (\chi^i_j - 1)).$$

□

The cost of a time-route π for a different departure time can be computed by using linear interpolation as in Definition 5.6. For example, if the cost at time $t = 10$ is equal to 4 and at time $t = 14$ the cost is equal to 8, then the cost at time $t = 13$ is equal to $4 + (8-4)\frac{13-10}{14-10} = 7$.

Definition 5.6. Given T_χ, T_{min}, T_{max} and $T_j = T_{min} + j \cdot T_\chi$ with $t \in [T_j, T_{j+1}]$, $j = 0, \ldots, n(\chi) - 1$, the cost of a time-route π represented by route edge e with profile χ^i and cost (i, b_i, f_i) at departure time t is given by

$$w_e(e,t) = w_e(e,T_j) + \frac{t - T_j}{T_{j+1} - T_j}(w_e(e,T_{j+1}) - w_e(e,T_j)).$$

□

Note that we currently use $f_i = 1$ for the profile of every time-route π. If two time-routes between a pair of boundary nodes have the same profile and profile cost, of course only one profile needs to be stored instead of two. The number of profiles (but not the number of profile costs) can be further reduced by approximating the cost of a route edge. This can be achieved by using a single profile for more than one time-route. If two time-routes between a pair of boundary nodes have the same profile (but not the same profile cost), only one profile needs to be stored instead of two. No approximation is necessary in this case. However, it is also possible to use only one profile if the profiles of two time-routes are very much alike. In this case the time-dependent cost of a route is approximated.

For example, let $\chi^1, \chi^2 \in X$ be two profiles for which there exists a $k \in \{0, \ldots, n(\chi)\}$ such that $\chi^i_k > 1$, $i = 1, 2$. Note that all profiles with $\chi^i_k = 1$ for $k \in \{0, \ldots, n(\chi)\}$ are equal. Profiles χ^1, χ^2 correspond to routes p_1 and p_2 respectively and have profile costs $(1, b_1, 1)$ and $(2, b_2, 1)$. Let $f(2,1) = \frac{1}{|\{i|\chi^1_i > 1\}|} \sum_{i:\chi^1_i > 1} \frac{\chi^2_i - 1}{\chi^1_i - 1}$. We can approximate the cost of route p_2 by setting the profile cost of route p_2 equal to $(1, b_2, f(2,1))$. The time-dependent cost of the route edge e corresponding to route p_2 for departure time $t = T_{min} + T_\chi \cdot i$ is thus approximated by $\hat{w}_e(e,t) = b_2 \cdot (1 + f(2,1) \cdot (\chi^1_i - 1))$. The relative error of this approximation is $\frac{|w_e(e,t) - \hat{w}_e(e,t)|}{w_e(e,t)} = \frac{|b_2 \cdot (1 + f(2,1) \cdot (\chi^1_i - 1)) - b_2 \cdot \chi^2_i|}{b_2 \cdot \chi^2_i} = \frac{|1 + f(2,1) \cdot (\chi^1_i - 1) - \chi^2_i|}{\chi^2_i}$. For example profile 2 in Figure 5.19 can be approximated by profile 1 in Figure 5.19 with a maximum relative error of 5%, for $k = 0, \ldots, n(\chi)$ using $f(2,1) = 1.105898$.

In general, the cost of a route p with profile χ^j and profile cost $(j, b_j, 1)$ can be

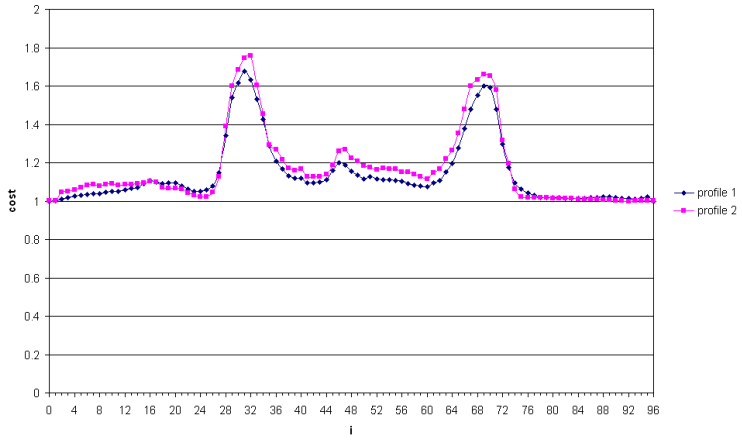


Figure 5.19. *Approximating profile 2 by profile 1 with a maximum relative error.*

approximated by profile χ^i and profile cost $(i, b_j, f(j,i))$ with

$$f(j,i) = \begin{cases} \frac{1}{|\{k|\chi^i_k>1\}|} \sum_{k:\chi^i_k>1} \frac{\chi^j_k - 1}{\chi^i_k - 1} & \text{if } \exists k : \chi^i_k > 1; \\ 0 & \text{otherwise.} \end{cases}$$

The error of this approximation is given by

$$err(\chi^j, \chi^i, k) = \frac{|1 + f(j,i) \cdot (\chi^i_k - 1) - \chi^j_k|}{\chi^j_k}, \text{ for every } k = 0, \ldots, n(\chi).$$

Of course, the error of the approximation has to be as small as possible. In order to minimize the amount of data that has to be stored, we determine a minimum-size subset of X such that the relative error in the stored route cost is at most ε, i.e. $err(\chi^j, \chi^i, k) \leq \varepsilon$ for every $k = 0, \ldots, n(\chi)$. In Section 5.5.3 we consider several possible values of ε. So, we have to solve the following problem.

PROFILES_SUBSET
GIVEN: A set of profiles X with costs Γ and a maximum error $\varepsilon > 0$.
QUESTION: Find a minimum-size subset $\hat{X} \subseteq X$ such that for every profile $\chi^i \in X$ with profile cost $(i, b_i, 1)$ there exists a profile $\chi^j \in \hat{X}$ with profile cost $(j, b_i, f(i,j))$ such that $err(\chi^i, \chi^j, k) \leq \varepsilon$ for all $k = 0, \ldots, n(\chi)$.

We expect this problem to be *NP*-hard as stated in Conjecture 5.1.

Conjecture 5.1. PROFILES_SUBSET *is NP-hard.* □

We decided not to determine the minimum-size subset of profiles required to obtain a specified maximum error, but to use a simple approximation algorithm, called *X*-

MATCH, to determine the subset of profiles and the profile costs because we expect the problem PROFILES_SUBSET to be *NP*-hard. Note that this leads to an overestimated number of profiles that need to be stored. The pseudo-code of the X-MATCH algorithm can be found in Figure 5.20.

Algorithm X-MATCH

Input: X and Γ with $|X| = |\Gamma|$.

χ^0 : The constant profile.
$\hat{X}$: Set of profiles to approximate the set X.
$\hat{\Gamma}$: Set of profile costs for profiles $\hat{X}$, with $|\hat{\Gamma}| = |\Gamma|$.
ε : The maximum error in a time-dependent cost.

Initialization step
$j = 0$
$\chi^0 = \{1, \ldots, 1\}$
$\hat{X} = \{\chi^0\}$

Main body of the algorithm
for $i = 1$ to $i = |X|$ **do**
    Profile_matched = FALSE
    while *not* Profile_matched **do**
        if $err(\chi^i, \chi^j, k) \leq \varepsilon$ for every $k = 1, \ldots, n(\chi)$ **then**
            add $(j, b_i, f(i, j))$ to $\hat{\Gamma}$
            Profile_matched = TRUE
        else if $j = |\hat{X}|$ **then**
            add χ^i to $\hat{X}$
            add $(i, b_i, 1)$ to $\hat{\Gamma}$
            Profile_matched = TRUE
        else
            $j = j + 1$
        end if
    end while
    $j = 0$
end for

Output: $\hat{X}$ and $\hat{\Gamma}$ with $|\hat{\Gamma}| = |\Gamma|$.

Figure 5.20. *Algorithm X-MATCH for problem PROFILES_SUBSET.*

5.5.3 Computational Evaluation

In the previous section, we described how the amount of profile data that needs to be stored can be reduced by using an approximation of each profile. In this section, we test this approach and determine how many profiles are actually needed for a particular maximum error ε.

We first planned for a partition an optimum time-dependent route (again using data from Antwerp in Section 5.4.1) between every pair of boundary nodes of a single cell for every $T_\chi = 900$ seconds (15 minutes). For every distinct time-dependent route

between two boundary nodes, we determined the travel time for this particular time-route for departure times $t = 0, 900, \ldots, 86,400$ (i.e. every 15 minutes). For each of these time-routes we determined its profile and profile cost. This lead to a set of profiles X and profile costs Γ with $|X| = |\Gamma|$. Given this input we determined the subset of profiles that needs to be stored by using algorithm X-MATCH in Figure 5.20.

The results for different values of ε can be found in Table 5.3. We used three different single-level partitions and three multi-level partitions of the Netherlands. The single-level partitions were created with the Merging-Algorithm using $r_i = b_i(b_i - 1)$, $r_i = \min\{b_i(b_i - 1), 2b_i\}$ and $r_i = 0$ as is listed in Table 5.3. The multi-level partitions were created with the Level-Splitting algorithm and the Merging-Algorithm using the same three values of r_i, see again Table 5.3. The values of $|X|$ for the different partitions can also be found in Table 5.3. In order to determine the necessity of the multiplication factor f_i, we also determined the subset $\hat{X}$ using $f(i, j) = 1$ for all i, j. The results can be found in Table 5.4.

λ	r_i	$\lvert X\rvert$	ε					
			5%	1%	0.5%	0.1%	0.05%	0.01%
1	r_i^C	973,494	13	148	554	10,362	29,908	143,091
1	r_i^S	1,014,131	10	74	237	4,151	13,819	102,744
1	r_i^P	1,068,068	8	77	216	3,209	10,899	91,752
9	r_i^C	1,883,691	8	69	227	4,097	13,232	49,229
10	r_i^S	2,418,230	9	68	262	4,400	13,946	41,598
6	r_i^P	2,455,245	8	70	257	4,191	11,979	26,335

Table 5.3. *Required number of profiles.*

λ	r_i	$\lvert X\rvert$	ε					
			5%	1%	0.5%	0.1%	0.05%	0.01%
1	r_i^C	973,494	52	1,494	6,144	84,538	157,010	287,356
1	r_i^S	1,014,131	39	802	2,981	44,075	96,807	217,642
1	r_i^P	1,068,068	37	686	2,384	34,619	78,611	192,128
9	r_i^C	1,883,691	42	522	1,702	17,433	34,280	59,851
10	r_i^S	2,418,230	37	444	1,564	15,183	30,681	48,562
6	r_i^P	2,455,245	38	425	1,459	12,135	20,716	29,916

Table 5.4. *Required number of profiles with $f(i, j) = 1$ for all i, j.*

From Tables 5.3 and 5.4 we can see that the number of profiles that needs to be stored decreases drastically. Specifically, the lower the required accuracy of the stored time-dependent cost, the fewer profiles need to be stored to achieve that accuracy, as could be expected. Furthermore, the multiplication factor $f(i, j)$ is very useful in reducing the number of profiles $|\hat{X}|$ that needs to be stored. If we compare the number of

profiles in X for partitions that are created with different values of r_i, we can see that partitions created with $r_i = b_i(b_i - 1)$ lead to the lowest number of original profiles $|X|$. Partitions created with $r_i = 0$ result in the highest number of original profiles $|X|$, as can be seen from column 3 in Tables 5.3 and 5.4. This is caused by the larger number of pairs of boundary nodes for which the optimum time-dependent route cost is computed.

We can also see from the results that although more original profiles exist for $r_i = 0$, the number of profiles needed to achieve a pre-specified maximum error ε is generally lowest for $r_i = 0$. In particular, the number of needed profiles becomes smaller for smaller values of r_i. This can be explained by looking at the partitions for $r_i = b_i(b_i - 1)$, $r_i = \min\{b_i(b_i - 1), 2b_i\}$ and $r_i = 0$. The partitions created with $r_i = 0$ have the smallest cells. For smaller cells it is more likely that the optimum time-dependent route does not contain any time-dependent edges, because in our case only major roads are time-dependent and optimum time-routes between nodes that are located relatively close together are less likely to contain major roads. As a result, for smaller cells more optimum time-routes between boundary nodes are time-independent or contain fewer time-dependent edges. All routes without time-dependent edges get the constant profile χ^0. Furthermore, time-routes with only a few time-dependent edges are more likely to closely resemble the time-dependent profile of other time-routes that also contain just a few time-dependent edges. Note that we use only a single time-dependent profile for all motorways. In reality multiple profiles exist, although many of these profiles closely resemble each other. Therefore, we expect that if multiple edge profiles are used to determine optimum time-routes, the number of profiles that needs to be stored to guarantee a maximum error increases, but not substantially.

As can be seen from the results, the number of profiles $|\hat{X}|$ is very small compared to the original number of profiles $|X|$. For a single-level partition the partition created with $r_i = 0$ leads to the fastest route planning process (see Section 4.3). For a multi-level partition, the partition created with $r_i = b_i(b_i - 1)$ leads to the fastest route planning process (see Section 4.4). The number of profiles required to obtain a maximum error of 0.05% for these two partitions is about 11,000 and 13,000 for the single and multi-level partition respectively. Since this number is limited, we conclude that a maximum error per edge of only 0.05% is still practically achievable. To achieve a maximum error of 0.1% per route edge, storing about 3,200 or 4,100 profiles suffices for a single or multi-level partition respectively.

When we compare the results of the single and multi-level partitions, we notice that the number of profiles needed for the single-level partitions is generally higher than for the multi-level partitions, although many more profiles need to be approximated in the latter case. This is caused by the difference in partitions. Although the multi-level partitions are created by iteratively creating single-level partitions, the

highest level of the multi-level partitions does not correspond to the single-level partitions in our case. In particular the number of boundary nodes per cell seems to be significantly lower for the highest level of the multi-level partitions. As a result, there are much fewer long time-dependent route profiles that need to be approximated and more constant time-independent profiles. Furthermore, because our algorithm is an approximation algorithm, the order in which the profiles are approximated matters. In particular, if the major distinct characteristic profiles are encountered early in the approximation algorithm then the constructed set of profiles is more likely to accurately approximate most of the remaining profiles. Otherwise, (many) more profiles may be created. The number of profiles may be further reduced by randomizing the algorithm and using several runs.

So far, we discussed the accuracy of the stored route cost compared to the optimum route cost between a pair of boundary nodes of a single cell. An optimum time-route in a searchgraph consists of multiple time-dependent route edges with minimum accuracy of $1-\varepsilon$. Note that it is also possible to approximate the profile of individual edges. However, because we only use a single profile, we only approximate route edges. For a time-dependent route π with n time-dependent edges with a positive maximum error ε_t in the travel time of the edge, the maximum error in the travel time of time-route π, denoted by $err(\pi)$, is larger than ε_t. Theorem 5.1 expresses $err(\pi)$ in terms of ε_t for fastest routes.

Theorem 5.1. *Assume* $|\frac{\partial t_e^t(e,t)}{\partial t}| \leq \partial_t$ *for all edges e. Let* ε_t *be the maximum error in the travel time of a single time-dependent edge e. Let* π *be a time-route in a searchgraph containing* $L^t(\pi)$ *time-dependent edges. An upperbound on the error* $err(\pi)$ *in the travel time of time-route* π *is given by*

$$err(\pi) = \varepsilon_t(1+(1+\varepsilon_t)\partial_t)^{L^t(\pi)-1}.$$

Proof. We prove this by induction on the number of time-dependent edges $L^t(\pi)$ in time-route π. Assume $L^t(\pi)=1$, then $err(\pi) \leq \varepsilon_t = \varepsilon_t \cdot 1 = \varepsilon_t(1+(1+\varepsilon_t)\partial_t)^0$. Let time-route π contain k time-dependent edges $a_1,\ldots,a_k$ with matching departure times $\tau_1,\ldots,\tau_k$ and $m-k$ time-independent edges $e_1,\ldots,e_{m-k}$. Route π can be expressed without loss of generality as $(\langle p_1, e_{\ell+1}, p_2\rangle, t)$ where p_1 is a time-route containing $k-1$ time-dependent edges and $\ell-k+1$ time-independent edges $e_1,\ldots,e_{\ell-k+1}$, edge $e_{\ell+1}$ adjacent to the last edge of route p_1 and with $\delta_2(e_{\ell+1}) = \delta_1(e_{\ell+2})$ is time-dependent, and time-route $p_2 = \langle e_{\ell-k+2},\ldots,e_{m-k}\rangle$ is time-independent. Let the optimum route travel time of time-route p_2 be denoted by t_2. Because all edges in time-route p_2 are time-independent the travel time t_2 is also time-independent. Furthermore, the travel time of the time-independent edges $e_1,\ldots,e_{\ell-k+1}$ is denoted by t_1. The total travel time of all time-dependent edges in time-route p_1 is denoted by $t(k)$.

We have $t(2) \leq (1+\varepsilon_t)t^{\mathrm{t}}_{\mathrm{e}}(a_1,\tau_1) + (1+\varepsilon_t)t^{\mathrm{t}}_{\mathrm{e}}(a_2,\tau_2) + (1+\varepsilon_t)\varepsilon_t\partial_t t^{\mathrm{t}}_{\mathrm{e}}(a_1,\tau_1)$ for $k=2$. Here $(1+\varepsilon_t)t^{\mathrm{t}}_{\mathrm{e}}(a_i,\tau_i)$ is the maximum extra travel time due to the error in $t^{\mathrm{t}}_{\mathrm{e}}(a_i,\tau_i)$, $i=1,2$. Furthermore, $(1+\varepsilon_t)\varepsilon_t\partial_t t^{\mathrm{t}}_{\mathrm{e}}(a_1,\tau_1)$ is the maximum extra travel time of edge a_2 due to the delayed arrival at node $\delta_1(a_2)$ because of the error in the travel time of edge a_1. We define the route travel time for a route with k time-dependent edges without errors in the travel time as $A_k = \sum_{i=1}^{k} t^{\mathrm{t}}_{\mathrm{e}}(a_i,\tau_i)$ and we define $B_k = \sum_{i=0}^{k}(1+(1+\varepsilon_t)\partial_t)^i$. Assume that for a time-route of k time-dependent edges, we have

$$t(k) \leq (1+\varepsilon_t)A_k + (1+\varepsilon_t)\varepsilon_t\partial_t A_{k-1}B_{k-2}.$$

Now, for a time-route with $k+1$ time-dependent edges we know

$$\begin{aligned}
t(k+1) &\leq t(k) + (1+\varepsilon_t)t^{\mathrm{t}}_{\mathrm{e}}(a_{k+1},\tau_{k+1}) + (1+\varepsilon_t)\partial_t(t(k)-A_k)\\
&\leq (1+\varepsilon_t)A_{k+1} + ((1+\varepsilon_t)^2 - (1+\varepsilon_t))\partial_t A_k\\
&\quad +\varepsilon_t((1+\varepsilon_t)\partial_t + (1+\varepsilon_t)^2\partial_t^2)A_{k-1}B_{k-2}\\
&= (1+\varepsilon_t)A_{k+1} + (1+\varepsilon_t)\varepsilon_t\partial_t A_k + (1+\varepsilon_t)\varepsilon_t\partial_t A_{k-1}(1+(1+\varepsilon_t)\partial_t)B_{k-2}\\
&\leq (1+\varepsilon_t)A_{k+1} + (1+\varepsilon_t)\varepsilon_t\partial_t A_k + (1+\varepsilon_t)\varepsilon_t\partial_t A_k(1+(1+\varepsilon_t)\partial_t)B_{k-2}\\
&= (1+\varepsilon_t)A_{k+1} + (1+\varepsilon_t)\varepsilon_t\partial_t A_k(1+(1+(1+\varepsilon_t)\partial_t)B_{k-2})\\
&= (1+\varepsilon_t)A_{k+1} + (1+\varepsilon_t)\varepsilon_t\partial_t A_k B_{k-1}.
\end{aligned}$$

Consider a time-route π with k time-dependent edges. We have

$$\begin{aligned}
err(\pi) &= \frac{t_1+t_2+t(k)-(t_1+t_2+A_k)}{t_1+t_2+A_k}\\
&\leq -1 + \frac{t_1+t_2+(1+\varepsilon_t)A_k}{t_1+t_2+A_k} + \frac{(1+\varepsilon_t)\varepsilon_t\partial_t A_{k-1}B_{k-2}}{t_1+t_2+A_k}\\
&\leq -1 + (1+\varepsilon_t) + (1+\varepsilon_t)\varepsilon_t\partial_t B_{k-2}\\
&= \varepsilon_t + (1+\varepsilon_t)\varepsilon_t\partial_t\frac{1-(1+(1+\varepsilon_t)\partial_t)^{k-1}}{1-(1+(1+\varepsilon_t)\partial_t)}\\
&= \varepsilon_t(1+(1+\varepsilon_t)\partial_t)^{k-1}.
\end{aligned}$$

□

The total maximum error in the route cost of a time-route π with $L^t(\pi)$ time-dependent edges is $err(\pi)$. The total maximum error $err(\pi)$ depends on the value of $(1+\varepsilon_t)\partial_t$, which determines the increase in travel time if the stored edge travel time deviates at most ε_t from the real travel time.

We can formulate a similar theorem for route costs.

Theorem 5.2. *Assume* $|\frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t}| \leq \partial_t$ *and* $|\frac{\partial w^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t}| \leq \partial_w$ *for all edges e. Let* ε_t *and* ε_w *be the maximum error in the travel time or cost of a single time-dependent edge e respectively. Let* $\pi = (\langle e_1,\ldots,e_m\rangle,t)$ *be a time-route containing* $L^t(\pi)$ *time-dependent edges* $a_1,\ldots,a_{L^t(\pi)}$ *with matching departure times* $\tau_1,\ldots,\tau_{L^t(\pi)}$ *and maximum positive errors* ε_t *and* ε_w *for the travel time and cost respectively. An upperbound on the*

error $err(\pi)$ *in the cost of time-route* π *is given by*

$$err(\pi) \leq \varepsilon_w + \varepsilon_t \frac{(1+\varepsilon_w)\partial_w}{(1+\varepsilon_t)\partial_t}((1+(1+\varepsilon_t)\partial_t)^{L^t(\pi)-1} - 1)\frac{\sum_{i=1}^{L^t(\pi)} t_e^t(a_i,\tau_i)}{\sum_{i=1}^{m} w_e^t(e_i,t_i)},$$

with $t_2 = t_1 + t_e^t(e_1,t_1)$ *and* $t_{i+1} = t_i + t_r^t(e_{i-1},e_i,t_i) + t_e^t(e_i,t_i + t_r^t(e_{i-1},e_i,t_i))$ *for* $i = 2,\ldots,m-1$.

Proof. We prove this by induction on the number of time-dependent edges $L^t(\pi)$ in time-route π. Assume $L^t(\pi) = 1$, then $err(\pi) = \varepsilon_w = \varepsilon_w + \varepsilon_t \frac{(1+\varepsilon_w)\partial_w}{(1+\varepsilon_t)\partial_t}((1+(1+\varepsilon_t)\partial_t)^0 - 1)$. Let time-route π contain k time-dependent edges $a_1,\ldots,a_k$ with matching departure time $\tau_1,\ldots,\tau_k$ and $m-k$ time-independent edges $e_1,\ldots,e_{m-k}$. Route π can be expressed without loss of generality as $(\langle p_1, e_{\ell+1}, p_2\rangle, t)$ where p_1 is a time-route containing $k-1$ time-dependent edges and $\ell-k+1$ time-independent edges $e_1,\ldots,e_{\ell-k+1}$, edge $e_{\ell+1}$ adjacent to the last edge of route p_1 and with $\delta_2(e_{\ell+1}) = \delta_1(e_{\ell+2})$ is time-dependent, and time-route $p_2 = \langle e_{\ell-k+2},\ldots,e_{m-k}\rangle$ is time-independent. Let the optimum route cost of time-route p_2 be denoted by c_2. Because all edges in time-route p_2 are time-independent the cost c_2 is also time-independent. Furthermore, the cost of the time-independent edges $e_1,\ldots,e_{\ell-k+1}$ is denoted by c_1. The total travel time and cost of all time-dependent edges in time-route p_1 is denoted by $t(k)$ and $c(k)$ respectively, were $t(k)$ was already determined in the proof of Theorem 5.1.

For $k = 2$ we have $c(2) \leq (1+\varepsilon_w)w_e^t(a_1,\tau_1) + (1+\varepsilon_w)w_e^t(a_2,\tau_2) + \varepsilon_t(1+\varepsilon_w)\partial_w t_e^t(a_1,\tau_1)$. Here $(1+\varepsilon_w)w_e^t(a_i,\tau_i)$ is the maximum extra cost due to the error in $w_e^t(a_i,\tau_i)$, $i = 1,2$. Furthermore, $\varepsilon_t(1+\varepsilon_w)\partial_w t_e^t(a_1,\tau_1)$ is the maximum extra cost of edge a_2 due to the delayed arrival at node $\delta_1(a_2)$ because of the error in the travel time of edge a_1. We again define the route travel time for a route with k time-dependent edges without errors in the travel time as $A_k = \sum_{i=1}^{k} t_e^t(a_i,\tau_i)$ and $B_k = \sum_{i=0}^{k}(1+(1+\varepsilon_t)\partial_t)^i$. Assume that for a time-route of k edges, we have

$$c(k) \leq (1+\varepsilon_w)\sum_{i=1}^{k} w_e^t(a_i,\tau_i) + \varepsilon_t(1+\varepsilon_w)\partial_w A_{k-1}B_{k-2}.$$

Now, for a time-route with $k+1$ time-dependent edges we know

$$\begin{aligned}
c(k+1) &\leq c(k) + (1+\varepsilon_w) w_{\mathrm{e}}^{\mathrm{t}}(a_{k+1}, \tau_{k+1}) + (1+\varepsilon_w)\partial_w (t(k) - A_k) \\
&= (1+\varepsilon_w) \sum_{i=1}^{k+1} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i) + \varepsilon_t (1+\varepsilon_w)\partial_w A_k \\
&\quad + \varepsilon_t (1+\varepsilon_w)\partial_w (1 + (1+\varepsilon_t)\partial_t) A_{k-1} B_{k-2} \\
&\leq (1+\varepsilon_w) \sum_{i=1}^{k+1} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i) + \varepsilon_t (1+\varepsilon_w)\partial_w A_k (1 + (1 + (1+\varepsilon_t)\partial_t) B_{k-2}) \\
&= (1+\varepsilon_w) \sum_{i=1}^{k+1} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i) + \varepsilon_t (1+\varepsilon_w)\partial_w A_k B_{k-1}.
\end{aligned}$$

Consider a time-route π with k time-dependent edges. We have

$$\begin{aligned}
err(\pi) &= \frac{c_1 + c_2 + c(k) - (c_1 + c_2 + \sum_{i=1}^{k} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i))}{\sum_{i=1}^{m} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i)} \\
&\leq -1 + \frac{\varepsilon_t (1+\varepsilon_w)\partial_w A_{k-1} B_{k-2}}{\sum_{i=1}^{m} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i)} + \frac{c_1 + c_2 + (1+\varepsilon_w)\sum_{i=1}^{k} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i)}{c_1 + c_2 + \sum_{i=1}^{k} w_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i)} \\
&= -1 + (1+\varepsilon_w) + \varepsilon_t (1+\varepsilon_w)\partial_w B_{k-2} \frac{A_{k-1}}{\sum_{i=1}^{m} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i)} \\
&= \varepsilon_w + \varepsilon_t (1+\varepsilon_w)\partial_w \frac{1 - (1 + (1+\varepsilon_t)\partial_t)^{k-1}}{1 - (1 + (1+\varepsilon_t)\partial_t)} \frac{A_{k-1}}{\sum_{i=1}^{m} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i)} \\
&\leq \varepsilon_w + \varepsilon_t \frac{(1+\varepsilon_w)\partial_w}{(1+\varepsilon_t)\partial_t} ((1 + (1+\varepsilon_t)\partial_t)^{k-1} - 1) \frac{\sum_{i=1}^{k} t_{\mathrm{e}}^{\mathrm{t}}(a_i, \tau_i)}{\sum_{i=1}^{m} w_{\mathrm{e}}^{\mathrm{t}}(e_i, t_i)}.
\end{aligned}$$

□

In a car navigation system the cost of an edge or route is generally given by a linear combination of the travel time of the edge or route and other time-independent costs such as the edge or route length or penalties for certain type of road segments such as motorway, ferry and toll road segments. As a result, we can write the cost of a route as the weighted sum of the travel time and other time-independent costs. As a result, the time-dependent cost of a route edge can be computed as the weighted sum of the time-dependent travel time of that route edge and the time-independent route costs of that route edge. Therefore, for these cost functions we only need to store profiles for travel time and a characterization of the route edge from which the time-independent route cost can be computed. Note that only the time-dependent travel time profile is a possible cause for errors in the route cost. The time-independent route costs can be computed exactly if a proper characterization of the route is stored. Therefore, we focus on time profiles in the remainder of this section.

Table 5.5 gives the value of $err(\pi)$ for different combinations of ε_t and ∂_t for a fastest time-route with 10 time-dependent edges. Because $\ell(e) = 1{,}000$ on average

∂_t	ε_t					
	5%	1%	0.5%	0.1%	0.05%	0.01%
0.00001	0.05001	0.01000	0.00500	0.00100	0.00050	0.00010
0.00010	0.05005	0.01001	0.00501	0.00100	0.00050	0.00010
0.00100	0.05047	0.01009	0.00505	0.00101	0.00051	0.00010
0.01000	0.05493	0.01095	0.00547	0.00109	0.00055	0.00011
0.10000	0.12281	0.02377	0.01184	0.00236	0.00118	0.00024
0.25000	0.40743	0.07586	0.03759	0.00746	0.00373	0.00075
0.50000	2.23048	0.39610	0.19512	0.03856	0.01925	0.00385
0.75000	9.31503	1.59977	0.78466	0.15453	0.07712	0.01540
0.85000	15.57016	2.64523	1.29564	0.25488	0.12718	0.02539
0.95000	25.31344	4.25900	2.08349	0.40946	0.20428	0.04079

Table 5.5. *Error $err(\pi)$ for a fastest time-route π with 10 time-dependent edges.*

and $\partial_t \leq \ell(e)e^{-5}$, $\partial = 1\%$ is reasonable.

Consider for example a time-route $(\langle e_1, e_2, e_3\rangle, 0)$. Suppose $\varepsilon_t = 0.05$. The real travel time of edge e_i is equal to 1, however, suppose we store a travel time equal to $(1+\varepsilon_t)\cdot 1 = 1.05$, for $i = 1,2,3$. This leads to an arrival at node $\delta_2(e_1)$ at time 1.05. The arrival time at node $\delta_2(e_2)$ is $1.05 + 1.05 + 0.05\partial_t 1.05 = 2.10 + 0.0525\partial_t$. The arrival time at node $\delta_2(e_3)$ is equal to $2.10 + 0.0525\partial_t + 1.05 + (0.10 + 0.0525\partial_t)\partial_t 1.05 = 3.15 + 0.1575\partial_t + 0.055125\partial_t^2$. For $\partial_t = 1$ this leads to an arrival time at $\delta_2(e_3)$ of 3.362625 instead of 3. The error in the travel time of this route is thus $\frac{0.362625}{3} \approx 12\%$.

For a pre-specified maximum error, we can compute the number of allowed time-dependent edges $L^t_{max}(\pi)$ in a fastest time-route for a particular value of ∂_t. Specifically we have

$$L^t_{max}(\pi) = 1 + \frac{\ln err(\pi) - \ln \partial_t}{\ln(1 + (1+\varepsilon_t)\partial_t)}.$$

Using $err(\pi) = 0.05$, this leads to the data in Table 5.6. Note that for $\varepsilon_t = 0.05$ obviously only fastest time-routes with a single time-dependent edge satisfy $err(\pi) \leq 0.05$.

As could be expected, we can see from Tables 5.5 and 5.6 that the larger the accuracy per edge, the smaller the error in the route cost and the more time-dependent route edges a route can contain to satisfy a pre-specified maximum error $err(\pi)$. Furthermore, ∂_t has a large impact on the maximum number of time-dependent route edges allowed to satisfy a maximum route cost error $err(\pi)$. For $\partial_t = 1$, a route may contain only a very small number of time-dependent route edges. We have that $\partial_t \leq 1e^{-5}\ell(e)$ for normal non-route edges of length $\ell(e)$ and the profiles in $\hat{X}$ resemble the profiles of single time-dependent edges. The cells of a single-level partition created with $r_i = b_i(b_i - 1)$ are the largest on average among the used parti-

	∂_t							
ε_t	0.0001	0.001	0.01	0.1	0.25	0.50	0.75	1
0.0500	1	1	1	1	1	1	1	1
0.0100	15,936	1,595	161	17	8	4	3	3
0.0050	22,913	2,293	231	25	11	6	5	4
0.0010	39,084	3,911	393	42	18	10	7	6
0.0005	46,031	4,606	463	49	21	12	9	7
0.0001	62,143	6,218	625	66	28	16	12	9

Table 5.6. *Maximum number of time-dependent edges for* $err(\pi) = 0.05$.

tions of the Netherlands. These cells have an average size of 220 km^2, which is less than the area of a square of 15 by 15 km. It seems reasonable to assume that the routes between boundary nodes of such cells are shorter than 50 km, which leads to $\partial_t \leq 50,000 \cdot 1e^{-5} = 0.5$. For all other used partitions, the cells are smaller on average. From Table 5.6 we can see that for $\varepsilon \leq 0.001$ and $\partial_t \leq 0.5$, a time-dependent route with at most 10 time-dependent edges has a maximum error of 5%.

In reality, this maximum error is generally not realized however. To determine this maximum error we assumed that the error in the cost and travel time of each edge is at its maximum, and the cost and travel time of each edge are overestimated. In reality also underestimated costs or times occur. This results in a computed travel time or cost that is too low. For a route with both underestimated and overestimated costs and travel times, these two factors may very well cancel each other out, leading to a much lower error for the entire route cost or travel time than predicted. This effect especially occurs for routes with many time-dependent route edges.

5.6 Conclusion

In this chapter, we have discussed consistency conditions under which Bellman's principle of optimality [Bellman, 1957] holds and a modified version of the standard $A^\star$-algorithm can be used to plan optimum time-dependent routes. We presented the $T^\star$-algorithm for planning time-dependent routes through a (partitioned) roadgraph. We have used real-world travel speeds for which the roadgraph was shown to satisfy the time consistency condition, to plan 2,000 fastest routes through the Netherlands. The results gave valuable insights in the changes in optimum time-dependent routes during the course of a day and in the usefulness of time-dependent optimum route planning for a driver. In particular, we have seen that especially during peak hours the use of time-dependent information can significantly reduce the route travel time.

In planning these time-dependent optimum routes, we have used a single speed profile for all motor- and highways in the Netherlands to plan deterministic time-dependent routes. To incorporate time-dependent route planning in car navigation

systems, we have to determine a speed profile of each individual road segment. In reality, congestion usually starts in front of a junction, and thus the decrease in travel speed generally occurs sooner at edges near that junction and the increase in travel speed generally occurs later at those edges. Determining realistic travel speed (or time) profiles of edges that are part of a route with a given travel speed profile is thus not trivial.

Furthermore, also less important roads are subject to congestion. In a city for example congestion usually occurs, but is generally not reported. Information on congestion on these roads is not readily available at this time. If congestion on these types of roads should be taken into account during route planning, then a model needs to be developed that accurately describes the congestion and travel speeds at local roads, given information about congestion at highways.

We have also shown how daily congestion patterns formulated by using time-dependent travel times can be combined with the use of partitions. In order to plan optimum time-dependent routes using a partitioned roadgraph, this requires the storage of a lot of data. However, due to the inherent uncertainty in the time-dependent travel times, an approximation of the optimum time-dependent route travel time is also valuable. We introduced the approximation algorithm X-MATCH to significantly reduce the number of time-dependent travel time profiles that need to be stored to compute a time-dependent route with a travel time that deviates at most a factor ε from the optimum travel time. Note that a different algorithm is required to determine the actual minimum number of profiles. From the results, we saw that the required number of profiles can be reduced to a level that is suitable for car navigation systems, while the route planning algorithm still provides high-quality routes. For car navigation systems the cost of a route generally depends on the time-dependent travel time of the route and a time-independent cost. With a proper characterization of the optimum routes between boundary nodes, only the travel time of route edges may differ from the optimum travel time. If different time-dependent cost functions are used, we can additionally store cost profiles for each route edge. Also the number of required cost profiles can be significantly reduced using the same procedure. Of course, to use time-dependent route planning in a commercial car navigation system, data structures have to be developed to efficiently store and use travel speed profiles.

6

Stochastic Time-Dependent Planning

In Chapter 5, we discussed planning routes in time-dependent roadgraphs which can be used to incorporate daily congestion patterns. As a result, daily congestion can be taken into account when planning a route. However, the information on daily congestion only reflects the average congestion and thus the average delays. The real delay for a particular time and day hardly ever matches the expected delay. In this chapter, we discuss *stochastic* route planning, in a stochastic time-dependent roadgraph, see Section 2.3. In a stochastic roadgraph, not only the average time-dependent costs are given but also the standard deviation of these costs. As a result, stochastic route planning has to cope with uncertainty in costs and travel times. Uncertainty is also present in the real travel times and costs due to additional or reduced congestion or a slow or fast progress of the driver (not necessarily due to congestion).

Section 6.1 gives an introduction in stochastic time-dependent planning. It discusses the specific problems that occur when a time-dependent stochastic route needs to be planned. In Section 6.2 we discuss the requirements for planning optimum stochastic routes by using a modified version of our $C^\star$-algorithm. The planning algorithm that can be used to plan a stochastic time-dependent route, is presented in Section 6.3. Section 6.4 presents the data and the results of planning stochastic routes using the unpartitioned road network of the Netherlands. How stochastic time-dependent planning can be combined with the partitions presented in Chapter 3 is discussed in Section 6.5. Finally, Section 6.6 presents the conclusions.

6.1 Introduction

We can assume without loss of generality that the cost and driving time of an edge is given by a probability density function $f(t)$. Depending on the used definition of the stochastic cost or driving time of an edge, the complexity of computing the stochastic cost of a route can vary considerably. For example, if we choose to use the expected cost and expected travel times to represent the stochastic cost of an edge, then the computation of the stochastic cost of a route boils down to adding the expected (time-dependent) costs of all edges in the route. On the other hand, if we choose to use a cumulative distribution function $F(t)$ to describe the cost of an edge at time t, the computation of the stochastic cost of a route, i.e. the cumulative distribution function of the route cost, is very complicated.

We first give an example to illustrate the complexity of the computation of the cumulative distribution function of the route cost. Consider Figure 6.1.

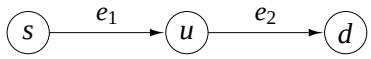


Figure 6.1. *A route consisting of two edges.*

Travel time distributions are best described by a left-centered distribution, i.e. a distribution with a larger probability of deviations to high realized travel times than to low travel times, see for example [He, Liu, Kornhauser & Ran, 2002]. Usually a log-normal distribution is chosen to represent the driving times, see also [Van Woensel, 2003]. To illustrate the impact of stochastic travel times, we use the following data

$$\begin{aligned} Pr(\tilde{t}_e^t(e_1,0)=2) &= 0.20, \\ Pr(\tilde{t}_e^t(e_1,0)=3) &= 0.50, \\ Pr(\tilde{t}_e^t(e_1,0)=4) &= 0.30, \end{aligned}$$

and for $t \geq 2$

$$\begin{aligned} Pr(\tilde{t}_e^t(e_2,t)=t-1) &= 0.10, \\ Pr(\tilde{t}_e^t(e_2,t)=t) &= 0.20, \\ Pr(\tilde{t}_e^t(e_2,t)=t+1) &= 0.30, \\ Pr(\tilde{t}_e^t(e_2,t)=t+2) &= 0.25, \\ Pr(\tilde{t}_e^t(e_2,t)=t+3) &= 0.15. \end{aligned}$$

This gives an expected travel time $\mu(\tilde{t}_e^t(e_1,0)) = 3.1$, and variance $\sigma^2(\tilde{t}_e^t(e_1,0)) = 0.49$. Furthermore, we have expected travel times $\mu(\tilde{t}_e^t(e_2,t)) = t+1.15$ for $t \geq 2$, and variance $\sigma^2(\tilde{t}_e^t(e_2,t)) = 1.4275$, for $t \geq 2$. When we use these data to compute the real, time-dependent travel time of route $p = \langle e_1, e_2 \rangle$, denoted by $\tilde{t}_e^t(p,0)$, and we assume the travel time of edge e_2 is only dependent on the departure time t and not

(directly) on the travel time of edge e_1, then we obtain the following

$$\begin{aligned}
Pr(\tilde{t}_e^{\tau}(p,0)=3) &= 0.020,\\
Pr(\tilde{t}_e^{\tau}(p,0)=4) &= 0.040,\\
Pr(\tilde{t}_e^{\tau}(p,0)=5) &= 0.110,\\
Pr(\tilde{t}_e^{\tau}(p,0)=6) &= 0.150,\\
Pr(\tilde{t}_e^{\tau}(p,0)=7) &= 0.210,\\
Pr(\tilde{t}_e^{\tau}(p,0)=8) &= 0.185,\\
Pr(\tilde{t}_e^{\tau}(p,0)=9) &= 0.165,\\
Pr(\tilde{t}_e^{\tau}(p,0)=10) &= 0.075,\\
Pr(\tilde{t}_e^{\tau}(p,0)=11) &= 0.045.
\end{aligned}$$

Furthermore, we obtain $\mu(\tilde{t}_e^{\tau}(p,0)) = 7.35$ and $\sigma^2(\tilde{t}_e^{\tau}(p,0)) = 3.3875$.

From this example we can see that the uncertainty of the travel time in one edge progresses through a route, and also influences the cost of the other edges. Specifically, the expected travel time of route p is equal to $\sum_x \{Pr(\tilde{t}_e^{\tau}(e_1,0)=x)\cdot\mu(\tilde{t}_e^{\tau}(e_2,x))\}$. The impact of the variance of the travel time of edge e_1 on the variance of the travel time of route p is even more complicated. Computing this variance exactly requires a detailed computation of all possible combinations of realized travel times for the edges in route p. For normal routes in large roadgraphs this is not practically feasible. Also, it is not practically feasible to store all possible realizations of the travel time for each possible departure time. This would require too much storage space, which is not available in a car navigation system. Therefore, we need to use approximations to compute the variance and expected travel time (and cost) of a route. As indicated in Section 2.3, we assume that the cost or driving time of an edge is given by a distribution function that is described completely by its expectation and variance. Note that the actual detailed distribution of the edge cost or driving time is generally unknown. For information on matching general distribution functions by phase-type distributions, see for example [Osogami & Harchol-Balter, 2003]. Zhu [2004] shows that if the travel time for time t is exponential or Erlang$_k$ distributed and the travel time profiles have a simple form, then the expected travel time and variance of the travel time of a stochastic time-route can be computed exactly and reasonably fast. However, we use a different approach.

As defined in Section 2.3, β tries to find a balance between the expectation and standard deviation of a stochastic cost or time $\tilde{x}$. If $\tilde{x} \prec \tilde{y}$ then $\tilde{x}$ is *preferred to* $\tilde{y}$. For two routes $\tilde{\pi}_1$ and $\tilde{\pi}_2$ with stochastic cost or driving time $\tilde{w}_p(\tilde{\pi}_1)$ and $\tilde{w}_p(\tilde{\pi}_2)$ we have

$$\begin{aligned}
&\tilde{w}_p(\tilde{\pi}_1) \prec \tilde{w}_p(\tilde{\pi}_2) \text{ if } \mu(\tilde{w}_p(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_1)) < \mu(\tilde{w}_p(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_2)).\\
&\tilde{w}_p(\tilde{\pi}_1) \preceq \tilde{w}_p(\tilde{\pi}_2) \text{ if } \mu(\tilde{w}_p(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_1)) \leq \mu(\tilde{w}_p(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_2)).\\
&\tilde{w}_p(\tilde{\pi}_1) = \tilde{w}_p(\tilde{\pi}_2) \text{ if } \mu(\tilde{w}_p(\tilde{\pi}_1)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_1)) = \mu(\tilde{w}_p(\tilde{\pi}_2)) + \beta\sigma(\tilde{w}_p(\tilde{\pi}_2)).
\end{aligned} \tag{6.1}$$

We have also defined a factor γ. If γ is high then the probability that the driver arrives before the estimated arrival time is high. The factor γ can be used to let the

driver indicate the importance of arriving no later than the estimated arrival time. As explained above the exact computation of the expectation and standard deviation of the route cost can be very complex. Because routes have to be planned fast, an exact computation is most likely unfeasible. Therefore, we first approximate the arrival time t_i at the end node of edge e_i as indicated in Section 2.3,

$$\begin{aligned} t_2 &= t_1 + \mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_1,t_1)) + \gamma\cdot\sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_1,t_1)), \\ t_{i+1} &= t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i) + \mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i))) \\ &\quad +\gamma\cdot\sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i))), i = 2,\ldots,\ell. \end{aligned}$$

Furthermore, we approximate the expectation $\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi}))$ and standard deviation $\sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi}))$ of the route cost as indicated in Section 2.3 by

$$\begin{aligned} \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \mu(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_1,t_1)) + \sum_{i=2}^{\ell}\mu(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i))) + \sum_{i=2}^{\ell} w_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i), \\ \sigma^2(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \sigma^2(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_1,t_1)) + \sum_{i=2}^{\ell}\sigma^2(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i + t_{\mathrm{r}}^{\mathrm{t}}(e_{i-1},e_i,t_i))). \end{aligned}$$

The expectation $\mu(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,t))$ and variance $\sigma^2(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,t))$ of the edge cost are assumed to be given for every edge e and every possible departure time t. Similarly, the expectation $\mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e,t))$ and variance $\sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e,t))$ of the driving time are assumed to be known for every edge e and departure time t. Furthermore, the driving time of an edge is assumed to depend only on the departure time of the edge and not (directly) on the travel time of the preceding edge.

If $t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t) = 0$ for every e_1, e_2 and t this can be simplified to

$$\begin{aligned} t_{i+1} &= t_i + \mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) + \gamma\cdot\sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)), i = 1,\ldots,\ell, \\ \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \textstyle\sum_{i=1}^{\ell}\mu(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) + \sum_{i=1}^{\ell-1} w_{\mathrm{r}}^{\mathrm{t}}(e_i,e_{i+1},t_i), \\ \sigma^2(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \textstyle\sum_{i=1}^{\ell}\sigma^2(\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)). \end{aligned} \tag{6.2}$$

So, as can be seen more clearly from the system of equations (6.2), the expectation and variance of the route cost are approximated by taking the sum of the expectations and variances of the edge costs respectively. Using these approximations leads to an approximation of the optimum route cost.

The approximated expected travel time and approximated variance of the travel time of route p in Figure 6.1 are equal to

$$\begin{aligned} &\mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 7.35 \text{ and } \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 1.9175 \text{ for } \gamma = 0, \\ &\mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 8.05 \text{ and } \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 1.9175 \text{ for } \gamma = 1, \\ &\mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 8.75 \text{ and } \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 1.9175 \text{ for } \gamma = 2, \\ &\mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 9.45 \text{ and } \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(p,0)) = 1.9175 \text{ for } \gamma = 3. \end{aligned}$$

Note that given $t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,t) = 0$ for every e_1, e_2 and t, we approximate the arrival time at the end node of an edge e_i by $t_i + \mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) + \gamma\cdot\sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i))$. Depending on the

value of γ this corresponds to a more or less pessimistic estimated arrival time. For $\gamma = 0$ the driving time is assumed to be equal to the expected driving time. For $\gamma > 0$, the driver is pessimistic about the actual driving time. Taking $\gamma < 0$ would lead to extremely optimistic driving speeds that can even be larger than the maximum time-independent travel speed, therefore we only consider $\gamma \geq 0$, see Definition 2.21. The larger γ, the more likely the driver arrives before the estimated arrival time. This means that most likely, he will also arrive at his destination before the estimated arrival time. This is especially useful for drivers that want to be sure not to arrive after the indicated arrival time.

6.2 Consistency

Just like in Section 5.2, we can formulate conditions under which an optimum stochastic route can be planned using a variant of the standard $A^\star$-algorithm. This section describes these consistency conditions. We describe the time consistency condition for planning fastest routes in stochastic time-dependent roadgraphs in Section 6.2.1. In Section 6.2.2, we extend this consistency condition for planning minimum-cost routes. These consistency conditions are formulated using a general partial ordering "$\preceq$" on the stochastic cost or travel time of a route or edge. We do not yet assume that equation (6.1) holds. Note that if we assume that the arrival times are equal to the approximated arrival times then we can use a polynomial-time algorithm to plan optimum stochastic routes using a time-expanded network.

6.2.1 Time Consistency

In this section we consider planning fastest routes in time-dependent stochastic roadgraphs. If the travel times in a roadgraph are no longer deterministic time-dependent, but stochastic instead, the time consistency condition of Section 5.2.1 can no longer be applied. However, Wellman, Ford & Larson [1995] show that it is possible to formulate a stochastic consistency condition for stochastic travel times so that stochastically dominated routes can be pruned, when searching for a minimum-cost route. This validates a modification of standard shortest path algorithms, such as Dijkstra's algorithm, to determine minimum-cost routes. In fact, such an algorithm is given by Wellman, Ford & Larson [1995]. First we give the stochastic consistency condition of Wellman, Ford & Larson [1995]. Let $c_e(t)$ denote the time-dependent travel time of edge e, where the travel time is now a random variable. A network (without traffic rules) is *stochastically consistent* if and only if for all e, $t_1 \leq t_2$ and z,

$$Pr(t_1 + c_e(t_1) \leq z) \geq Pr(t_2 + c_e(t_2) \leq z).$$

This condition says that the probability of arriving by any time z cannot be increased by leaving later. Again, waiting is assumed to be impossible. However, if waiting is costless, then the network is stochastically consistent, as was also the case for the deterministic consistency condition of Kaufman & Smith [1993]. Note that this

condition is based on Bellman's principle of optimality [Bellman, 1957].

Because turn restrictions can be present in our stochastic time-dependent model, we formulate the stochastic time consistency condition for roadgraphs with turn restrictions in Definition 6.1.

Definition 6.1. Let relation "$\preceq$" be a partial order on stochastic travel times $\tilde{t}$. Let $\tilde{T}(e,d,\tilde{t})$ denote an optimum (stochastic) travel time of a route in a stochastic time-dependent roadgraph $\tilde{G}^{\mathrm{t}}$ from $\delta_1(e)$ starting with edge e to node d departing from node $\delta_1(e)$ at (a stochastic) time $\tilde{t}$. A roadgraph $\tilde{G}^{\mathrm{t}}$ is *stochastically time consistent* if and only if for every time $\tilde{t}_1 \preceq \tilde{t}_2$, and every pair of adjacent edges e_1 and e_2 in $\tilde{G}^{\mathrm{t}}$

$$\begin{array}{c} \tilde{t}_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_1) + \tilde{T}(e_2,d,\tilde{t}_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_1)) \\ \preceq \\ \tilde{t}_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_2) + \tilde{T}(e_2,d,\tilde{t}_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_2)). \end{array}$$

□

This condition says that a stochastic route from a start node s to edge e_1 that has a preferred arrival time, cannot lead to a stochastic route that is considered to be worse. As for normal time consistency, the condition in Definition 6.1 can be simplified.

Lemma 6.1. *Let relation "$\preceq$" be a partial order on stochastic travel times $\tilde{t}$. A stochastic time-dependent roadgraph $\tilde{G}^{\mathrm{t}}$ is* stochastically time consistent *if for every time $\tilde{t}_1 \preceq \tilde{t}_2$, and every pair of adjacent edges e_1 and e_2 in $\tilde{G}^{\mathrm{t}}$*

$$\tilde{t}_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_1) + \tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_2,\tilde{t}_1 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_1)) \preceq \tilde{t}_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_2) + \tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_2,\tilde{t}_2 + t_{\mathrm{r}}^{\mathrm{t}}(e_1,e_2,\tilde{t}_2)).$$

Proof. Follows easily by induction on the number of edges on the route from e_2 to d. □

If time-windows are present in the stochastic roadgraph then the roadgraph $\tilde{G}^{\mathrm{t}}$ is not always stochastically time consistent just as for (deterministic) time consistency, as follows immediately from Section 5.2.1. Determining if a stochastic time-dependent roadgraph is stochastically time consistent can be done fast if we can determine fast whether $\tilde{t}_1$ is preferred to $\tilde{t}_2$. In particular, we need to check the condition in Lemma 6.1 for every pair of adjacent edges in roadgraph $\tilde{G}^{\mathrm{t}}$.

In this section, we only considered edge costs that represent the (stochastic) driving time of an edge. In the next section, we generalize the stochastic time consistency condition to a more general stochastic cost consistency condition.

6.2.2 Cost Consistency

In the previous section, we discussed consistency for fastest routes. In a car navigation system, also more general edge cost functions are used, and we are interested in planning the minimum-cost route. In this section, we therefore generalize the stochastically time consistent roadgraph to a stochastically cost consistent roadgraph.

Specifically, we can define a stochastically cost consistent roadgraph as follows.

Definition 6.2. Let "$\preceq$" be a partial order on stochastic route costs. Roadgraph $\tilde{G}^{\mathrm{t}}$ is *stochastically cost consistent* if and only if for every optimum stochastic route $\tilde{\pi} = (\langle e_1, \ldots, e_\ell \rangle, t)$ in $\tilde{G}^{\mathrm{t}}$ with arrival time $\tilde{t}_i$ at node $\delta_2(e_i)$, for every $i = 1, \ldots, \ell$ there does not exist a stochastic route $\tilde{\pi}' = (\langle a_1, \ldots, a_j \rangle, t)$ in $\tilde{G}^{\mathrm{t}}$ with $\delta_1(a_1) = \delta_1(e_1)$, $a_j = e_i$, arrival time $\tilde{t}'_j$ and $\tilde{w}_{\mathrm{p}}(\langle a_1, \ldots, a_j \rangle, t) \prec \tilde{w}_{\mathrm{p}}(\langle e_1, \ldots, e_i \rangle, t)$ or $\tilde{w}_{\mathrm{p}}(\langle a_1, \ldots, a_j \rangle, t) = \tilde{w}_{\mathrm{p}}(\langle e_1, \ldots, e_i \rangle, t)$ and $\tilde{t}'_j \prec \tilde{t}_i$. □

This condition says that there does not exist a stochastic route departing from node s at time t ending with edge e_i that is preferred to the subroute from node s to edge e_i of an optimum stochastic route $\tilde{\pi}$ departing from node s at time t to destination d for every edge e_i in the optimum stochastic route $\tilde{\pi}$. Note that if there are two optimum stochastic routes from node s to edge e_i each with cost $\tilde{w}_{\mathrm{p}}(\langle e_1, \ldots, e_i \rangle, t)$ then the time-route $(\langle e_1, \ldots, e_i \rangle, t)$ is the time-route with the preferred arrival time at node $\delta_2(e_i)$. Note that determining if a stochastic time-dependent roadgraph is stochastically cost consistent is so not easy. To verify if a roadgraph is stochastically cost consistent we need to check for every optimum stochastic time-route in roadgraph $\tilde{G}^{\mathrm{t}}$ whether every starting stochastic sub-time-route is again an optimum stochastic time-route. This requires the planning of an optimum stochastic time-route between every pair of nodes in graph $\tilde{G}^{\mathrm{t}}$ for every possible departure time. If a stochastic roadgraph $\tilde{G}^{\mathrm{t}}$ is stochastically cost consistent, we can use a modified version of a standard route planning algorithm such as the $A^\star$-algorithm to plan an optimum stochastic route.

6.3 Algorithms for Stochastic Time-Dependent Planning

In this section, we discuss the planning of optimum stochastic time-routes in stochastically cost consistent time-dependent roadgraphs $\tilde{G}^{\mathrm{t}}$. For a stochastically cost consistent roadgraph, an optimum route consists of optimum sub-routes, see Definition 6.2. Therefore, we can again use a variant of the standard $A^\star$-algorithm to plan optimum stochastic time-dependent routes in stochastically cost consistent roadgraphs. Figure 6.2 presents the $S^\star$-algorithm, which is very similar to the $T^\star$-algorithm in Figure 5.5 for cost consistent time-dependent roadgraphs, and to the $C^\star$-algorithm in Figure 4.10 for time-independent roadgraphs. The difference between the $S^\star$-algorithm and the $T^\star$-algorithm lies in the use of the total ordering "$\preceq$", which is used to determine if the new route to an edge e is preferred to the best found route to edge e so far. If all stochastic operations concerning adding (i.e. $\tilde{x}_1 + \tilde{x}_2$) and comparing (i.e. $\tilde{x}_1 \preceq \tilde{x}_2$) two stochastic variables as well as the determination of a stochastic or deterministic function of a stochastic variable (i.e. $g(\tilde{x})$ or $\tilde{g}(\tilde{x})$) are of order $O(1)$ then algorithm $S^\star$ can be implemented in order $O(n + m \log m)$ just as the $C^\star$- and the $T^\star$-algorithm.

The assumption that all stochastic operations are of order $O(1)$ is certainly plausi-

Algorithm $S^\star$

Input: A graph $\tilde{G}^{\mathrm{t}}$, start node s, destination node d, start time t_0 and total ordering "$\preceq$".

H : The set of unexpanded edges.
$\tilde{g}(e)$: The best stochastic cost found of a route from s to edge e.
$\tilde{g}'(e)$: Alternative stochastic cost of a route from s to edge e.
$\tilde{a}(e)$: The stochastic arrival time of the route with cost $\tilde{g}(e)$ at the end node of edge e.
$\tilde{a}'(e)$: The stochastic arrival time of the route with cost $\tilde{g}'(e)$ at the end node of edge e.
$\tilde{g}_{\mathrm{n}}(u)$: The best underestimated route cost found from node s to node u.
$\lambda(e)$: The adjacent edge leading to edge e in the best route found from s to e.
$\tilde{\pi}$: The stochastic optimum route from s to d.
$\tilde{w}_{\mathrm{p}}(\tilde{\pi})$: The stochastic cost of route $\tilde{\pi}$.
$h(u)$: The estimated cost from node u to node d.
$h(e)$: The estimated cost from edge e to node d.

Initialization step
$H = E$, $\tilde{\pi} = (\langle\rangle, t_0)$, $\tilde{w}_{\mathrm{p}}(\tilde{\pi}) = \infty$
for every $e \in H$ **do**
  $\tilde{g}(e) = \infty$
  $\tilde{a}(e) = \infty$
  $\lambda(e) = \emptyset$
end for
for every $u \in N$ **do** $\tilde{g}_{\mathrm{n}}(u) = \infty$ **end for**
$\tilde{g}_{\mathrm{n}}(s) = 0$
for every $e \in H$ such that $\delta_1(e) = s$ **do**
  $\tilde{g}(e) = \tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e, t_0)$
  $\tilde{a}(e) = t_0 + \tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e, t_0)$
  $\lambda(e) = e$
  if $\tilde{g}_{\mathrm{n}}(\delta_2(e)) \succ \tilde{g}(e)$ **then** $\tilde{g}_{\mathrm{n}}(\delta_2(e)) = \tilde{g}(e)$ **end if**
end for

Main body of the algorithm
while there is an edge $e \in H$ with $\tilde{g}_{\mathrm{n}}(d) \succ \tilde{g}(e) + h(e)$ **do**
  Let e be an edge from H such that $\tilde{g}(e) + h(e) \preceq \tilde{g}(e_1) + h(e_1)$ for all $e_1 \in H$ **end if**
  for every (allowed) edge e_1 with $\delta_2(e) = \delta_1(e_1)$ in $\tilde{G}^{\mathrm{t}}$ **do**
    $\tilde{g}'(e_1) = \tilde{g}(e) + \tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_1, \tilde{a}(e) + t^{\mathrm{t}}_{\mathrm{r}}(e, e_1, \tilde{a}(e))) + w^{\mathrm{t}}_{\mathrm{r}}(e, e_1, \tilde{a}(e))$
    $\tilde{a}'(e_1) = \tilde{a}(e) + t^{\mathrm{t}}_{\mathrm{r}}(e, e_1, \tilde{a}(e)) + \tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_1, \tilde{a}(e) + t^{\mathrm{t}}_{\mathrm{r}}(e, e_1, \tilde{a}(e)))$
    if $\tilde{g}(e_1) \succ \tilde{g}'(e_1)$ **or** $(\tilde{g}(e_1) = \tilde{g}'(e_1)$ **and** $\tilde{a}(e_1) \succ \tilde{a}'(e_1))$ **then**
      $\tilde{g}(e_1) = \tilde{g}'(e_1)$
      $\tilde{a}(e_1) = \tilde{a}'(e_1)$
      $\lambda(e_1) = e$
      if $\tilde{g}_{\mathrm{n}}(\delta_2(e_1)) \succ \tilde{g}(e_1)$ **then** $\tilde{g}_{\mathrm{n}}(\delta_2(e_1)) = \tilde{g}(e_1)$ **end if**
    end if
  end for
  $H = H \setminus \{e\}$
end while
if $\tilde{g}_{\mathrm{n}}(d) \neq \infty$ **then** $\tilde{w}_{\mathrm{p}}(\tilde{\pi}) = \tilde{g}(e)$ **and** construct route $\tilde{\pi}$ from edge e using $\lambda(e)$ **end if**

Output: Time-route $\tilde{\pi}$, and the route cost $\tilde{w}_{\mathrm{p}}(\tilde{\pi})$.

Figure 6.2. *Algorithm $S^\star$.*

ble. We use a general total order relation "$\preceq$" on stochastic costs in algorithm $S^\star$. As a result, this algorithm can be used to plan an optimum stochastic route for any total ordering for which the stochastic time-dependent roadgraph satisfies the stochastic cost consistency condition. Depending on the total ordering "$\preceq$", the computational complexity of determining whether $\tilde{c}_1 \prec \tilde{c}_2$ can vary. We intend to use equation (6.1) to determine if $\tilde{c}_1 \prec \tilde{c}_2$ is satisfied. Determining whether $\tilde{c}_1 \prec \tilde{c}_2$ is a operation of $O(1)$ for this particular total ordering. Alternatively, we could also use a total ordering such as $\tilde{c}_1 \preceq \tilde{c}_2$ if and only if $F_1^{-1}(0.95) \leq F_2^{-1}(0.95)$, with F_1 and F_2 the cumulative distribution function of $\tilde{c}_1$ and $\tilde{c}_2$ respectively. Note that this approach does not take the most extreme realizations of $\tilde{c}_1$ and $\tilde{c}_2$ into consideration when determining if $\tilde{c}_1 \preceq \tilde{c}_2$, which leads to more reliable results. In this case the computational complexity depends on the cumulative distribution functions of $\tilde{c}_1$ and $\tilde{c}_2$. For discrete distribution functions with k possible values this can be an operation of order $O(k)$. If we assume for example that c_1 and c_2 are independent and normally distributed with mean μ_1, μ_2 and standard deviation σ_1, σ_2 respectively, then this is an operation of order $O(1)$ because $F_i^{-1}(0.95) = \mu_i + 1.6449\sigma_i$, $i = 1,2$, see [Zwillinger & Kokoska, 1999].

Similarly, adding two independent stochastic costs or times $\tilde{c}_1 + \tilde{c}_2$ can vary in computational complexity, see [Zwillinger & Kokoska, 1999]. For example, if $\tilde{c}_1$ and $\tilde{c}_2$ are independent and normally distributed with mean μ_1, μ_2 and standard deviation σ_1, σ_2 respectively, then $\tilde{c}_1 + \tilde{c}_2$ is also normally distributed with mean $\mu_1 + \mu_2$ and standard deviation $\sqrt{\sigma_1^2 + \sigma_2^2}$. So in this case, adding two independent stochastic variables is an operation of $O(1)$. Adding two independent stochastic variables can thus in some cases be done in order $O(1)$.

Determining the distribution of a function $g(\tilde{t})$ with g a deterministic function of time t and $\tilde{t}$ a stochastic time can also vary in computational complexity. For several continuous distribution functions of $\tilde{t}$ and for (for example) linear functions g, we can determine the distribution of $g(\tilde{t})$ exactly in order $O(1)$, see [Zwillinger & Kokoska, 1999]. If $\tilde{t}$ has a discrete distribution function with k possible values, then determining the distribution of $g(\tilde{t})$ can be done in order $O(k)$.

Finally, also the determination of the distribution of $\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,\tilde{t})$ for example can vary in computational complexity. If we assume that $\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,\tilde{t})$ is given by $\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,\mu(t))$, that the distribution of $\tilde{w}_{\mathrm{e}}^{\mathrm{t}}(e,t)$ is known for every t, and that $\mu(\tilde{t})$ is known, then this operation can be done in order $O(1)$.

Alternatively, we can also use a partial ordering such as $\tilde{c}_1 \preceq \tilde{c}_2$ if and only if $Pr(\tilde{c}_1 \leq z) \geq Pr(\tilde{c}_2 \leq z)$ for every z for example. However, determining if $\tilde{c}_1 \preceq \tilde{c}_2$ for this partial ordering is computationally more complex than for the total ordering from equation (6.1). Furthermore, there may not exist an edge $e \in H$ such that $\tilde{c}(e) \preceq \tilde{c}(e_1)$ for all $e_1 \in H$ for a partial ordering "$\preceq$". This is because in a partially ordered set H not necessarily all edges can be compared, and therefore there may not exist an

overall preferred edge. In order to determine an optimum route using a partial order "$\preceq$", we need to use a slightly different algorithm. The difference mainly lies in the fact that more than one cost can be optimum according to a partial order. Therefore, multiple optimum route costs from start node s to edge e may need to be stored. We can select from among all optimum costs, for example the cost with minimum expected value to evaluate next, because a low expected value remains important. Wellman, Ford & Larson [1995] present such a algorithm for a particular partial ordering "$\preceq$".

If a roadgraph is not stochastically consistent then algorithm $S^\star$ can still be used to plan stochastic time-routes, but the found routes are not guaranteed to be optimum. For instance, in a stochastically inconsistent roadgraph a sub-route $\tilde{\pi}_1$ from s to u with a higher expected travel time and the same variance as sub-route $\tilde{\pi}_2$ may lead to a total route containing sub-route $\tilde{\pi}_1$ with a lower expected travel time and as well as a lower variance than the route containing sub-route $\tilde{\pi}_2$. However, presenting a route containing $\tilde{\pi}_1$ is hard to explain and it is even more difficult to credibly present such routes to a driver. Therefore, we prefer routes that consist of optimum sub-routes and assume that a roadgraph is stochastically consistent, even though this may not be the case.

6.4 Computational Evaluation

This section discusses the results of planning 2,000 stochastic time-dependent routes through the Netherlands. We minimize the estimated stochastic cost of a route because determining the real time-dependent stochastic cost of a route is too time-consuming and too complex to be used in car navigation systems. As a result, we can use algorithm $S^\star$ to plan an optimum route. Note that due to the used adaptation of the cost functions, the uncertainty in edge costs is translated into a single deterministic value. Specifically, uncertainty is taken into account by setting the travel time of an edge equal to the expected edge travel time plus γ times the standard deviation of the edge travel time. Furthermore, uncertainty in the cost of an edge is taken into account by determining the variance of the edge cost for a single departure time. So in essence route planning is no longer truly stochastic, but it does take uncertainty into account. We first present the available and the used data. Then, we discuss the general trends that we observed from the planned stochastic time-dependent routes. In order to determine if and when it is useful to incorporate stochastic data into route planning, we compare deterministic time-dependent routes and stochastic time-dependent routes. Here deterministic time-dependent routes are routes planned using algorithm $T^\star$ from Chapter 5, so the uncertainty in travel times and costs is completely ignored.

6.4.1 Speed Profiles

For regular congestion, we expect the standard deviation to be higher in peak-hours, when the expected travel speed differs most from the maximum speed. Similarly, we expect the standard deviation to be lower for off-peak hours, when the expected travel speed is closer to the maximum speed. In order to keep the computation of the standard deviation simple, we set the standard deviation of the travel time equal to $1/\varsigma$ times the absolute difference between the expected travel time and the minimum travel time, i.e. the standard deviation of the travel time of edge e at time t is approximated by $\frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$. The minimum travel time is computed by dividing the length of the edge by the maximum allowed speed for that edge. As a result, the standard deviation is larger when the expected travel speed differs more from the maximum travel speed. A lower value of ς implicates a larger uncertainty in travel times.

Antwerp

Unfortunately, no data was available on the uncertainty in the travel speeds from Antwerp in Section 5.4.1. Although the uncertainty can be determined using the methods of Van Woensel [2003], this required assumptions on travel time distributions, and complex computations. Therefore, we decided not to use these methods. We can assume however, that the expected travel time is given by the data in Section 5.4.1. The standard deviation of the travel time of edge e at time t is approximated by $\frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$, with $v(e,t)$ the travel speed of edge e at time t from Section 5.4.1, and $v(e)$ the maximum speed for edge e. We will show that this is realistic when we discuss the information available about uncertainty in travel times on motorways in the Netherlands. For planning fastest routes using $\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t) = \mu(t^{\mathrm{t}}_{\mathrm{e}}(e,t)) + \gamma\sigma(t^{\mathrm{t}}_{\mathrm{e}}(e,t))$, we can check stochastic time consistency the same way we check time consistency, because the deterministic time-dependent cost $\mu(t^{\mathrm{t}}_{\mathrm{e}}(e,t)) + \gamma\sigma(t^{\mathrm{t}}_{\mathrm{e}}(e,t))$ is used to plan a route. Therefore, the roadgraph is time consistent if $\frac{\partial(\mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t))+\gamma\sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t)))}{\partial t} = \frac{\partial(t^{\mathrm{t}}_{\mathrm{e}}(e,t)+\frac{\gamma}{\varsigma}(\frac{\ell(e)}{v(e,t)}-\frac{\ell(e)}{v(e)}))}{\partial t} = \frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} + \frac{\gamma}{\varsigma}\frac{\frac{\ell(e)}{v(e,t)}}{\partial t} = \frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} + \frac{\gamma}{\varsigma}\frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} = (1+\frac{\gamma}{\varsigma})\frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} \geq -1$. In Section 5.4.1 we have seen that $\frac{\partial t^{\mathrm{t}}_{\mathrm{e}}(e,t)}{\partial t} \geq -1e^{-5}\ell(e)$, so the roadgraph is stochastically consistent if $(1+\frac{\gamma}{\varsigma})\ell(e) \cdot e^{-5} \leq 1$ for all edges e in the roadgraph.

The Netherlands

For the Netherlands, we do have information about the uncertainty in the travel speeds from Section 5.4.1. Figure 6.3 gives the average coefficient of variation for all motorway sections from Table 5.2. Figure 6.3 also shows the coefficient of variation if we determine the standard deviation by $\frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$, with $v(e,t)$ the travel speed of edge e at time t from Section 5.4.1, and $v(e)$ the maximum speed for edge e, for

$\varsigma = 1.5$. Table 6.1 shows for several values of ς the weighted mean squared error of $\frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$ compared to the average coefficient of variation for all motorway sections from Table 5.2. Because we think it is more important to have an accurate estimated coefficient of variation during rush hours, we doubled the weight of each deviation between 6:00h and 9:00h and between 15:30h and 19:00h. As can be seen from Figure 6.3 and Table 6.1, this approximation using $\varsigma = 1.5$ is not unreasonable for testing the usefulness and general trends of stochastic time-dependent planning. Therefore, in the following sections, we will use $\sigma(\tilde{t}_e^{\tau}(e,t)) = \frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$ with $\varsigma = 1.5$.

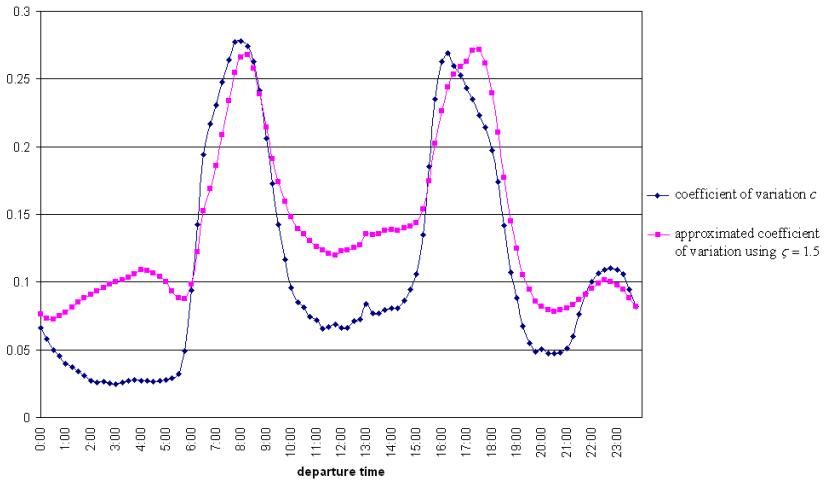


Figure 6.3. *The average coefficient of variation for motorways in the Netherlands.*

ς	WMSE
$\frac{1}{2}$	0.1270
1	0.0116
$1\frac{1}{2}$	0.0018
2	0.0023
$2\frac{1}{2}$	0.0043
3	0.0063

Table 6.1. *Weighted mean squared error (WMSE) of* $\frac{1}{\varsigma} \cdot (\frac{\ell(e)}{v(e,t)} - \frac{\ell(e)}{v(e)})$ *from* c^2.

As explained in the previous subsection, the roadgraph is stochastically time consistent if $(1 + \frac{\gamma}{\varsigma})\ell(e) \cdot e^{-5} \leq 1$ for all edges e in the roadgraph. Note that for $\varsigma = 1.5$, and $\gamma \leq 3$, the roadgraph is stochastically consistent if $\ell(e) \leq 33,333m$ for all e. For $\gamma = 1$, we can have edges with a length of $60,000m$ to sustain stochastic consistency. This is not unrealistic for a dense road network as we have in the Netherlands.

Note that the relative high variability from 21:00h to 01:00h is probably due to construction works and relatively much freight traffic. In the year 2002, many motorways were under construction or repair. In the Netherlands, it is customary to spread these activities over a long period of time, to be carried out in the evening and during the night. This is probably the main cause for the high variability in travel time in these hours.

6.4.2 General Results

We have planned the fastest stochastic route for the same 2,000 start and destination nodes as in Section 5.4, see also Table 4.3, using algorithm $S^\star$ (see Figure 6.2) with the stochastic ordering from equation (6.1). As said in Section 6.4.1 we use $\varsigma = 1.5$. Furthermore, we have set $\beta = 0$. For $\gamma = 0$ and $\beta > 0$ we expect similar results, although the optimum routes may be different for some departure times. Specifically, with $\beta > 0$ and $\gamma = 0$ the mean arrival time at the destination and the mean cost for a particular route are equal to the arrival time and cost for that particular route for deterministic time-dependent travel times. The variance of the route cost determines which route is considered optimum. However, we use $\beta = 0$, which means that the uncertainty in travel times is only taken into account by γ. Specifically, the used travel time for each edge is set to $\mu(\tilde{t}) + \gamma \cdot \sigma(\tilde{t})$ with $\tilde{t}$ the stochastic travel time of an edge. Note that planning the fastest routes using these parameters leads to $\mu(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e,t)) = \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t)) + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t))$, and $\sigma(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e,t)) = \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t))$. As a result, the travel time and cost of the planned route give an upperbound on the actual travel time and cost per edge as they most likely will be experienced by the driver. Assume for example the travel times are normally distributed with mean $\mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t))$ and standard deviation $\sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e,t))$. If we choose $\gamma = 3$ then the probability that the realized travel time for an edge is larger than the travel time for this edge used for planning the route is only $1 - \Phi(3) = 0.14\%$ (see [Zwillinger & Kokoska, 1999]), where Φ denotes the cumulative probability distribution of the standard normal distribution. Generally, the probability that the travel time of the entire route is larger than the computed travel time is (much) smaller than 0.14%. This is because the probability that the travel time is larger than the computed travel time applies to each edge separately. If we assume that the travel times are lognormally distributed with Φ^L the lognormal cumulative distribution function, then the probability that the realized travel time for an edge is larger than the travel time for this edge used for planning the route is $1 - \Phi^L(3) = 13.60\%$ if we use $\gamma = 3$.

Just as for deterministic time-dependent planning, different departure times lead to different optimum routes. After comparing the deterministic time-dependent and the stochastic time-dependent routes, we see that the stochastic optimum routes are generally also optimum for a (sometimes) different departure time for deterministic time-dependent planning. Specifically, if route p_1 is optimum in G^{t} for departure times $t \leq t_1$ and $t \geq t_3$, and route p_2 is optimum in G^{t} for departure times $t_3 > t > t_1$,

then we frequently see that route p_1 is optimum in $\tilde{G}^t$ for departure times $t \leq t_4 < t_1$, and $t \geq t_5 > t_3$ and route p_2 is optimum in $\tilde{G}^t$ for departure times $t_5 > t > t_4$. So the change from one optimum route to another for a period of increasing congestion occurs sooner, i.e. for less serious congestion, and for a period of decreasing congestion the change in optimum route occurs later, i.e. also for less serious congestion.

We can explain this by the fact that for more serious congestion, the difference between the maximum speed and the actual speed is larger, and therefore the standard deviation of the driving time is larger (which is logical because the uncertainty in travel times increases). For a route p with a higher stochastic cost, it is more likely that an alternative route with a lower cost can be found.

Planning stochastic routes using a different value of γ also leads to different results, as could be expected. This means that drivers with different attitudes towards uncertainty are presented different routes. For example, consider Figure 6.4. The optimum route from Zwolle to Apeldoorn determined with $\gamma = 1$ and $\varsigma = 3$ uses mainly motorways, while the optimum route for $\gamma = 3$ (and $\varsigma = 3$) mainly uses secondary roads. This can be explained from the fact that these secondary roads have a standard deviation equal to 0, because we assumed that only edges with road class 0 and 1 are time-dependent and thus only these edges have a positive standard deviation. In order to avoid the uncertainty in travel time, which is very important for drivers that choose to use $\gamma = 3$, these secondary roads are used. This shows that our route planning successfully searches for a trade-off between the expected travel time and the variation in travel time. Should standard deviations for all roads be known, then of course a better (and more realistic) trade-off can be found. However, the general trends probably do not change. If the value of γ is increased then more road segments with only a very small standard deviation will be contained in an optimum route.

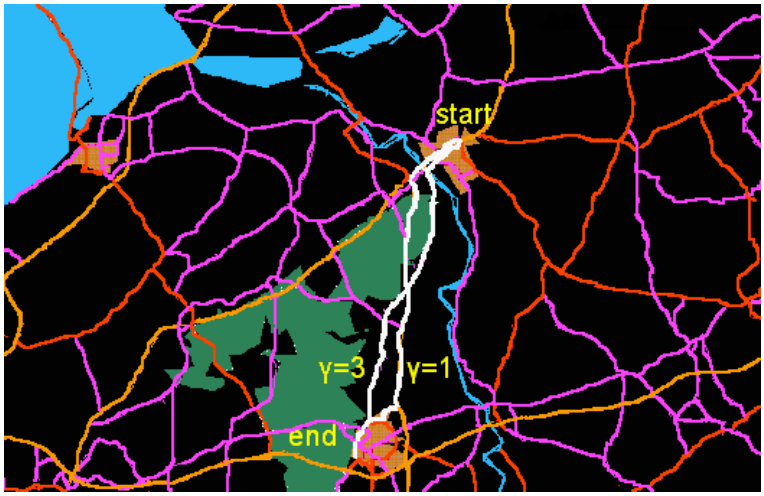


Figure 6.4. *Two optimum routes from Zwolle to Apeldoorn.*

Note that just as for deterministic time-dependent planning, the number of different optimum stochastic routes is quite small, generally between 5 and 10. Furthermore, we also see that many of the stochastic time-dependent optimum routes are also deterministic time-dependent optimum routes for particular departure times.

In the next section, we determine the added value of introducing uncertainty into our model for the driver.

6.4.3 Stochastic versus Deterministic Time-Dependent Planning

In order to determine the added value for drivers of introducing stochastic driving times, we compare the results of planning deterministic time-dependent and stochastic time-dependent routes. As was already defined in Section 5.4.3, the arrival time of a deterministic time-dependent route $\pi = (\langle a_1, \ldots, a_n \rangle, t_0)$ for departure time t_0 is denoted by $T_\pi^t(t_0)$, assuming that the deterministic time-dependent travel times from Section 5.4.1 correspond exactly to the actual realized driving times. The optimum stochastic time-dependent route $\tilde{\pi} = (\langle e_1, \ldots, e_\ell \rangle, t_0)$ can be different from the deterministic time-dependent optimum route π. We call travel times $\mu(\tilde{t}_e^t(e,t)) + \gamma \cdot \sigma(\tilde{t}_e^t(e,t))$ the estimated stochastic travel times. The arrival time of the stochastic route $\tilde{\pi}$ is denoted by $\tilde{T}_{\tilde{\pi}}^t(t_0)$, assuming that the realized travel time is equal to the estimated stochastic travel time for every edge e in route $\tilde{\pi}$.

Similar to Section 5.4.3, we also compute the arrival time of route π for departure times $t_0 = 0, 900, \ldots, 86,400$ assuming that the realized travel times are given by the estimated stochastic travel times. This leads to an arrival time denoted by $\tilde{T}_\pi^t(t_0)$ for departure time t_0. Similarly, the arrival time of route $\tilde{\pi}$ for departure time t_0, assuming the realized travel times are equal to the deterministic time-dependent travel times (i.e. equal to $\mu(\tilde{t}_e^t(e,t))$ for every edge in route $\tilde{\pi}$), is denoted by $T_{\tilde{\pi}}^t(t_0)$. So, we have for departure time t_0

- $T_\pi^t(t_0)$ Arrival time of time-dependent route π;
- $\tilde{T}_\pi^t(t_0)$ Arrival time of time-dependent route π using stochastic travel times;
- $T_{\tilde{\pi}}^t(t_0)$ Arrival time of route $\tilde{\pi}$ for deterministic time-dependent travel times;
- $\tilde{T}_{\tilde{\pi}}^t(t_0)$ Arrival time of stochastic route $\tilde{\pi}$.

We determined for each of the 2,000 start and destination nodes in the route batch (see Table 4.3) the optimum time-dependent route π and the optimum stochastic time-dependent route $\tilde{\pi}$ for departure times t with $t = 0, 900, \ldots, 86,400$, with t in seconds, i.e. for every 15 minutes (on Monday). We used $\beta = 0$ and $\gamma = 1$ (and $\varsigma = 1.5$). The averages, standard deviations and maxima of $T_{\tilde{\pi}}^t(t) - T_\pi^t(t)$ and $\tilde{T}_\pi^t(t) - \tilde{T}_{\tilde{\pi}}^t(t)$ can be found in Figures 6.5, 6.6 and 6.7 respectively. All realizations of $T_{\tilde{\pi}}^t(t) - T_\pi^t(t)$ and $\tilde{T}_\pi^t(t) - \tilde{T}_{\tilde{\pi}}^t(t)$ can be found in Figures 6.8 and 6.9 respectively. The distribution of $T_{\tilde{\pi}}^t(t) - T_\pi^t(t)$ and $\tilde{T}_\pi^t(t) - \tilde{T}_{\tilde{\pi}}^t(t)$ for departure time 7:00h, 10:30h and 16:30h can be found in Figure 6.10 and Figure 6.11 respectively. Note that the average of $T_\pi^t(t)$ is about 66 minutes. For every $(\beta, \gamma, \varsigma) \in \{(0,1,1.5), (0,2,1.5), (0,1,0.75), (0,1,3)\}$ the results can be found in Appendix D.

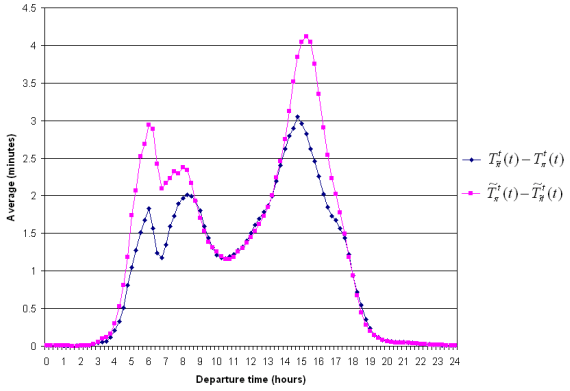


Figure 6.5. *Average of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$.

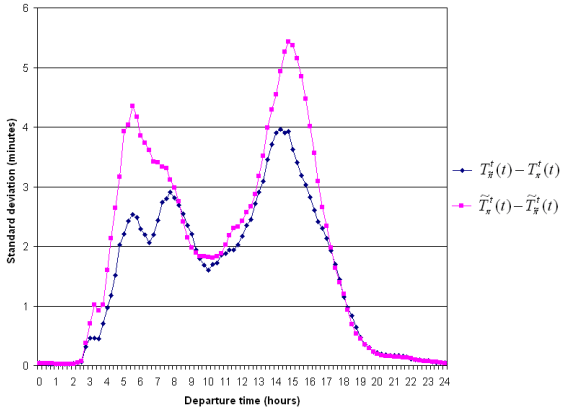


Figure 6.6. *Standard deviation of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$.

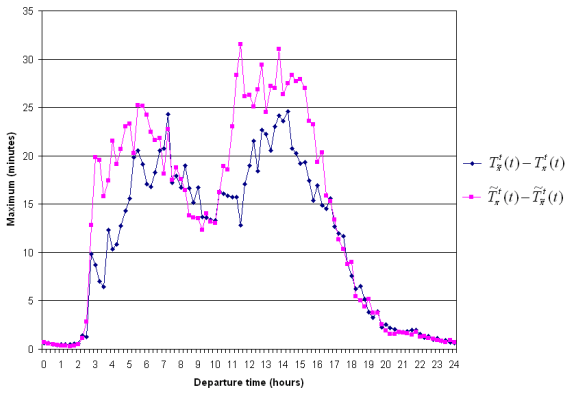


Figure 6.7. *Maximum of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$.

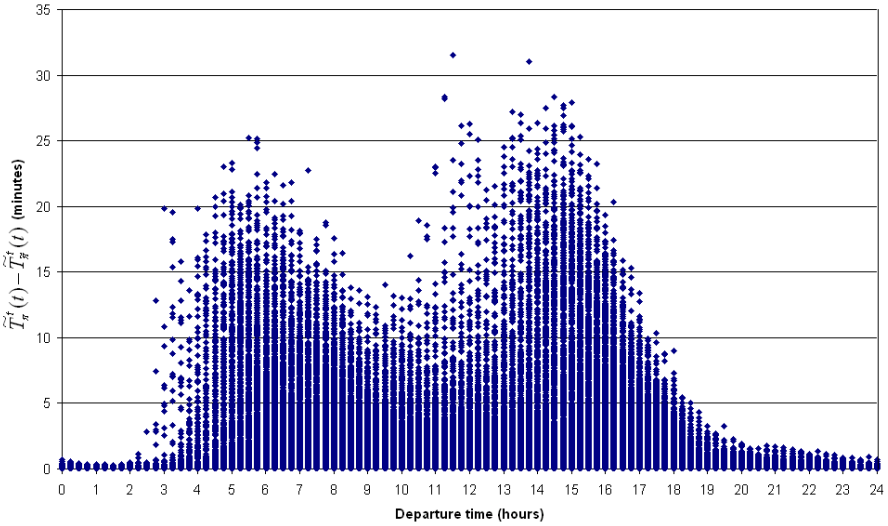


Figure 6.8. *All realizations of* $\tilde{T}_{\tilde{\pi}}^{t}(t) - \tilde{T}_{\pi}^{t}(t)$.

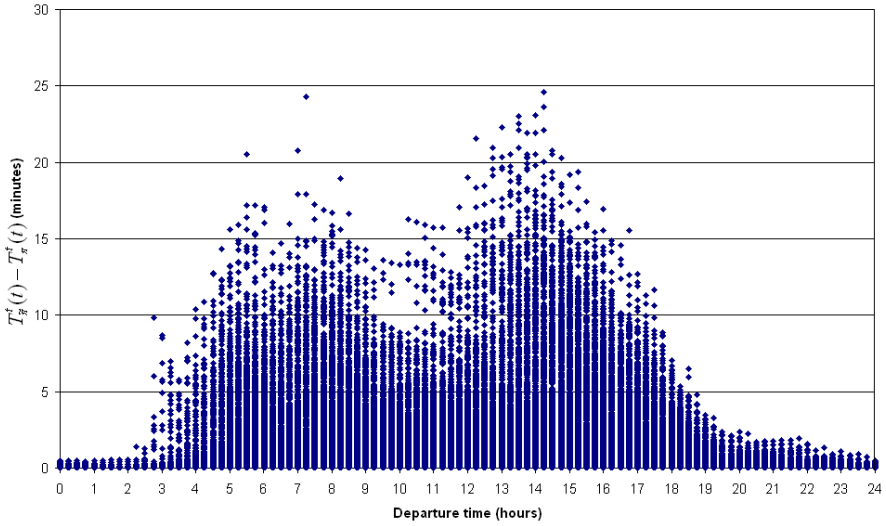


Figure 6.9. *All realizations of* $T_{\pi}^{t}(t) - T_{\tilde{\pi}}^{t}(t)$.

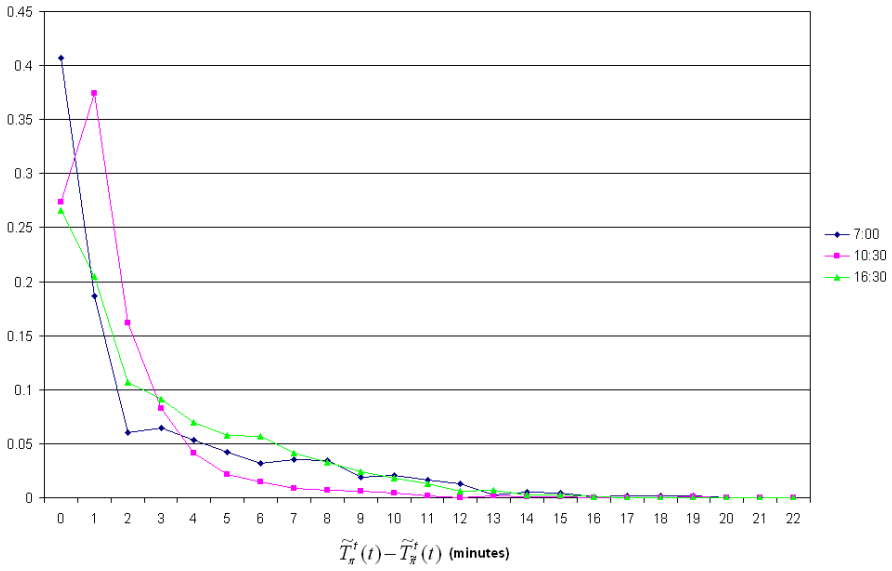


Figure 6.10. *Distribution of $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ for 7:00h, 10:30h and 16:30h.*

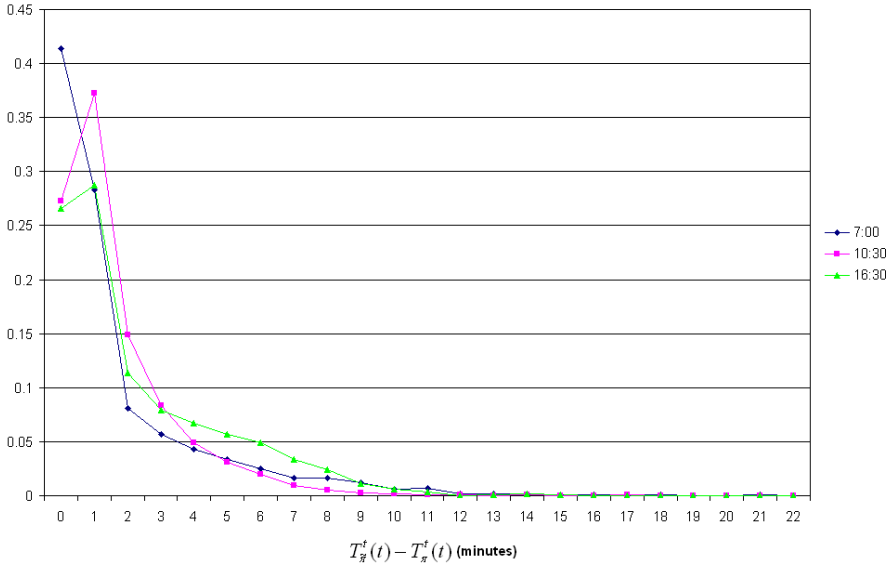


Figure 6.11. *Distribution of $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ for 7:00h, 10:30h and 16:30h.*

Unlike Figures 5.10 to 5.16, Figures 6.5 to 6.11 compare a sort of worst-case arrival time of routes π and $\tilde{\pi}$. As can be seen from these figures, a navigation system using stochastic time-dependent route planning presents a route with a lower worst-case arrival time than a navigation system with deterministic time-dependent planning. The differences during peak hours are the highest (between 1.5 and 4 minutes) because the uncertainty in the travel times during peak hours is highest. The maximum difference among our 2,000 routes is about 32 minutes. As can be seen from Figure 6.9 for many routes the difference is minimal, which results in a low average difference. From Figures 6.5, 6.7, 6.8 and 6.9 we can see that the expected travel times of routes determined by a navigation system using deterministic time-dependent planning are smaller than those of a system using stochastic time-dependent route planning. This can be explained from the fact that unlike deterministic time-dependent planning, stochastic time-dependent planning weighs uncertainty against expectation, which usually leads to routes with a possibly higher expectation and less uncertainty. Figure 6.6 shows that the standard deviation of the difference in arrival time is relatively high compared to the average difference in arrival time, so there is a lot of variation in the difference between the worst-case arrival time using stochastic time-dependent planning and the arrival time using deterministic time-dependent planning. Figure 6.6 also shows that the standard deviation is higher during rush hours. From Figures 6.10 and 6.11 we can also see that for departure time 10:30h the variation in $\tilde{T}^t_\pi(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ and in $T^t_{\tilde{\pi}}(t) - T^t_\pi(t)$ is less than for 7:00h and 16:30h as could be expected. Note that for increasing values of γ or $\frac{1}{\varsigma}$ (or β), the value of $\tilde{T}^t_\pi(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ (and of $T^t_{\tilde{\pi}}(t) - T^t_\pi(t)$) increases, see also Appendix D.

Note that because route π is optimum for deterministic time-dependent travel times, we have $T^t_{\tilde{\pi}}(t) - T^t_\pi(t) \geq 0$. Similarly, route $\tilde{\pi}$ is optimum for stochastic time-dependent travel times, so we have $\tilde{T}^t_\pi(t) - \tilde{T}^t_{\tilde{\pi}}(t) \geq 0$. Because stochastic time-dependent travel times are at least as high as deterministic time-dependent travel times ($\gamma \geq 0$), we also know that $\tilde{T}^t_\pi(t) \geq T^t_\pi(t)$ and $\tilde{T}^t_{\tilde{\pi}}(t) \geq T^t_{\tilde{\pi}}(t)$. This leads to $\tilde{T}^t_\pi(t) \geq \tilde{T}^t_{\tilde{\pi}}(t) \geq T^t_{\tilde{\pi}}(t) \geq T^t_\pi(t)$. This also means that $\tilde{T}^t_{\tilde{\pi}}(t) - T^t_{\tilde{\pi}}(t) \leq \tilde{T}^t_\pi(t) - T^t_\pi(t)$. We can explain this by observing that a stochastic optimum route $\tilde{\pi}$ generally contains fewer edges with a high uncertainty than the deterministic time-dependent optimum route π. Consequently, the difference between the stochastic travel time and deterministic time-dependent travel time is smaller for route $\tilde{\pi}$ than for route π.

From the results in Figures 6.5, 6.7, 6.8 and 6.9, we can also see that in general $T^t_{\tilde{\pi}}(t) - T^t_\pi(t) \leq \tilde{T}^t_\pi(t) - \tilde{T}^t_{\tilde{\pi}}(t)$. Using a similar argument as in Section 5.4.3, we can explain this phenomenon. Suppose there exists a route π with a length of 120 km and an average time-dependent travel speed of 60 km/h, and a stochastic time-dependent travel speed of 50 km/h. This leads to $T^t_\pi(t) = 2$, and $\tilde{T}^t_\pi(t) = 2\frac{2}{3}$. Now assume there exists an alternative route $\tilde{\pi}$ with a length of 100 km, and average time-dependent travel speed v, that does not contain any edges with an uncertain travel time. This

leads to $\tilde{T}^t_{\tilde{\pi}}(t) = T^t_{\tilde{\pi}}(t) = \frac{100}{v}$. Because $\tilde{T}^t_{\pi}(t) \geq \tilde{T}^t_{\tilde{\pi}}(t) \geq T^t_{\tilde{\pi}}(t) \geq T^t_{\pi}(t)$ the average time-dependent travel speed lies between 37.5 km/h and 50 km/h. For average speeds v close to 50 km/h we have $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t) \leq \tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$. On the other hand for average speeds close to 37.5 km/h we have $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t) \geq \tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$. We can see from our results that $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t) \leq \tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$, so apparently alternative routes exist with little uncertainty (or equivalently with only a few edges of road class 0 or 1) that have an average speed that is not much different from the deterministic time-dependent travel speeds of edges with road class 0 or 1. This can be explained from the high density of the road network of the Netherlands. Due to this high density many alternative routes exist. For less dense road networks with larger differences in travel speeds between different road classes, we expect to find the opposite effect, i.e. $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t) \geq \tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$. Furthermore, for road networks with less variability in travel times, the differences become smaller because the optimum stochastic routes resemble the optimum deterministic time-dependent routes more closely, as can also be seen from the results in Appendix D. In reality, all edges are subject to uncertainty to some extent. However, we do not expect that this changes the overall observed behavior of stochastic time-dependent route planning.

From these results, we conclude that it is only worthwhile considering stochastic effects during route planning if the driver desires a high certainty concerning his (latest) arrival time. Liu, Recker & Chen [2004] show from real-time loop data on State Route 91 in Orange County, California that commuters on this route are willing to accept a considerably higher expected travel time to avoid variability. Specifically, their tests show that the average commuter uses $\beta = 1.73$. Note that their model does not include the factor γ.

6.5 Combination with Partitions

In the previous section, we considered planning stochastic routes in unpartitioned roadgraphs. As demonstrated in Chapter 4, optimum time-independent routes can be planned fast by using partitions. We have seen in Chapter 5 that time-dependent routes of sufficiently high quality can also be planned fast by using partitions. In this section, we discuss the planning of stochastic time-dependent routes in partitioned roadgraphs. The theoretical integration of planning stochastic time-dependent routes with the use of partitions is discussed in Section 6.5.1. Just as for planning deterministic time-dependent routes, this theoretical approach is not suitable for on-board car navigation systems. Therefore, we also discuss a practical approach to combine planning stochastic routes with using partitions. Section 6.5.2 presents this approach. Section 6.5.3 deals with the quality of stochastic routes that are planned using this practical approach compared to the quality of stochastic routes that are planned using the theoretical approach from Section 6.5.1.

6.5.1 Optimum Planning

Theoretically, we can combine the use of partitions with planning stochastic time-dependent routes in a relatively straightforward manner. However, this leads to an extremely large number of costs and times that need to be stored. As defined in Chapter 2, the node and edge structure of a time-independent roadgraph G and a matching stochastic time-dependent roadgraph $\tilde{G}^t$ are identical. Therefore, the partitions constructed in Chapter 3 can also be used for stochastic time-dependent route planning.

In Chapter 4 we discussed several ways to store the optimum route costs between two boundary nodes of a single cell. For stochastic time-dependent planning these three methods can be used just as for deterministic time-dependent and time-independent planning. However, the cost of the best route between two boundary nodes of a cell is not a single number as for time-independent planning, or a travel time and route cost profile as for deterministic time-dependent planning. For stochastic time-dependent planning also the standard deviation of the route cost and travel time needs to be known in order to determine an optimum route.

As discussed in Section 6.1, in order to plan the best possible stochastic time-dependent route, the standard deviation of the travel time, as well as its mean, are needed to determine the correct arrival time at the end node of a route edge. In order to determine the correct cost of this route edge, the standard deviation of the route cost as well as its mean are also necessary. For every departure time, these expectations and standard deviations may differ. Therefore, we need to store these expectations and standard deviations for every possible departure time.

So far, we ignored the driver's wish to indicate his attitude towards uncertainty. As indicated in Section 2.3, we would like to be able to present a best route for different attitudes towards uncertainty, or equivalently, for different values of β and γ. Storing the expectation and standard deviation of the route cost is sufficient to compute the actual route cost for any value of β. However, the value of γ influences this expectation and standard deviation of the route cost, because γ determines the travel times, and thus the arrival times at nodes, and consequently the edge costs. So the expectation and standard deviation of the cost and travel time of a route between two boundary nodes of a cell is different for different values of γ.

We expect that a driver would be allowed to choose between a limited number of values for γ. Therefore, it is theoretically possible to store all expectations and standard deviations of the best route cost and matching travel time, for all possible values of γ, for all departure times, and for all pairs of boundary nodes of a single cell.

If we store all this data, then it is possible to plan the same optimum route in a partitioned roadgraph as in an unpartitioned roadgraph, because the optimum route in the unpartitioned roadgraph consists of optimum sub-routes, and the cost of these

optimum sub-routes between boundary nodes of single cells are stored in the route graphs.

For an on-board car navigation system this leads to far too much data that needs to be stored. Therefore, we discuss ways to restrict the amount of data that needs to be stored in the next section.

6.5.2 Approximate Planning

As in Section 5.5.2, we use profiles to store not only the expected route cost and travel time for a particular value of γ, but also to store the standard deviation of the route cost and travel time for a particular value of γ. Again, we only store these expectations and standard deviations for every $T_\chi = 15$ minutes. For other departure times, the expectations and standard deviations are computed using linear interpolation. This already drastically reduces the amount of data that needs to be stored.

As for deterministic time-dependent planning, the expected cost or travel time of a stochastic route $\tilde{\pi}$ are represented by route edge e with profile χ_i and profile cost (i, b_i, f_i). A profile χ of the expected cost of stochastic route $\tilde{\pi}$ is given by $\chi = \langle \chi_0, \ldots, \chi_{\eta(\chi)} \rangle$, with $\chi_i = \frac{c(\tilde{\pi},i)}{\min_i c(\tilde{\pi},i)}$ and $b_i = \min_i c(\tilde{\pi}, i)$, where $c(\tilde{\pi}, i)$ denotes the expected cost of route $\tilde{\pi}$ departing at time $t = T_{min} + i \cdot T_\chi$, $i = 0, \ldots, \eta(\chi)$.

Now that we also need to store the standard deviation or variance of the travel time or cost of a stochastic route $\tilde{\pi}$, we also construct a profile χ_i with profile cost (i, b_i, f_i) for the standard deviation or variance. Specifically, given a stochastic route $\tilde{\pi}$ we compute for every departure time $t = T_{min} + i \cdot T_\chi$, $i = 0, \ldots, \eta(\chi)$ the standard deviation or variance of the route cost or travel time, which we denote by $c^s(\tilde{\pi}, i)$. Then we create a profile $\chi = \langle \chi_0, \ldots, \chi_{\eta(\chi)} \rangle$ with $\chi_i = \frac{c^s(\tilde{\pi},i)}{\min_i c^s(\tilde{\pi},i)}$ and $b_i = \min_i c^s(\tilde{\pi}, i)$.

Given these profiles for the standard deviation or variance, we can again reduce the number of profiles that need to be stored by approximating these profiles in the same way as in Section 5.5.2. As shown in Section 5.5.3 this greatly reduces the number of profiles for the expected travel times for $\gamma = 0$. So in general, the amount of data to store will be greatly reduced.

So far, we still need to store expectations and standard deviations or variances for every possible value of γ. If the driver is allowed to choose between many values of γ, or enter a value himself, a lot of profiles and profile costs exist. To further reduce the number of profiles and profile costs, we can approximate the expectation and standard deviation of the route cost or travel time for any value of γ by the expectation and standard deviation for $\gamma = 0$ respectively.

Consider a stochastic route $\tilde{\pi} = (\langle e_1, \ldots, e_\ell \rangle, t)$. We have

$$\begin{aligned}
w_{\mathrm{p}}(\tilde{\pi}) &= \mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) + \beta \cdot \sigma(\tilde{w}_{\mathrm{p}}(\tilde{\pi})), \\
\mu(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \mu(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \sum_{i=2}^{\ell} \mu(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))) + \sum_{i=2}^{\ell} w^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i), \\
\sigma^2(\tilde{w}_{\mathrm{p}}(\tilde{\pi})) &= \sigma^2(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \sum_{i=2}^{\ell} \sigma^2(\tilde{w}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))),
\end{aligned}$$

with

$$\begin{aligned}
t_2 &= t_1 + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)) + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_1, t_1)), \\
t_{i+1} &= t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i) + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))) \\
&\quad + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i + t^{\mathrm{t}}_{\mathrm{r}}(e_{i-1}, e_i, t_i))), i = 2, \ldots, \ell.
\end{aligned}$$

First, we consider planning fastest routes, and we assume $t^{\mathrm{t}}_{\mathrm{r}}(e_i, e_j, t) = 0$ for every e_i, e_j and t. This leads to an expected route cost denoted by μ_γ and a variation of route $\tilde{\pi} = (\langle e_1, \ldots, e_\ell \rangle, t_1)$ containing n (stochastic) time-dependent edges denoted by σ^2_γ. We have

$$\begin{aligned}
t_{i+1} &= t_i + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i)) + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i)), i = 1, \ldots, \ell, \\
\mu_\gamma &= \sum_{i=1}^{\ell} \left\{ \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i)) + \gamma \cdot \sigma(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i)) \right\} + \sum_{i=1}^{\ell-1} w^{\mathrm{t}}_{\mathrm{r}}(e_i, e_{i+1}, t_i), \\
\sigma^2_\gamma &= \sum_{i=1}^{\ell} \sigma^2(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t_i)).
\end{aligned}$$

Now, we can approximate μ_γ by $\mu_\gamma \approx \mu_0 + \gamma \cdot n^\zeta \sigma_0$, and σ_γ by $\sigma_\gamma \approx \sigma_0$, where n denotes the number of time-dependent edges in route $\tilde{\pi}$, and ζ is a fixed parameter. So, we have

$$\begin{aligned}
t'_1 &= t_1, \\
t'_{i+1} &= t'_i + \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t'_i)), i = 1, \ldots, \ell, \\
\mu_\gamma &\approx \sum_{i=1}^{\ell} \mu(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t'_i)) + \sum_{i=1}^{\ell-1} w^{\mathrm{t}}_{\mathrm{r}}(e_i, e_{i+1}, t'_i) + \gamma \cdot n^\zeta \sqrt{\sum_{i=1}^{\ell} \sigma^2(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t'_i))}, \\
\sigma^2_\gamma &\approx \sum_{i=1}^{\ell} \sigma^2(\tilde{t}^{\mathrm{t}}_{\mathrm{e}}(e_i, t'_i)).
\end{aligned}$$

If we ignore that $t_i' \neq t_i$ for the moment, then we have

$$\begin{aligned}
\mu_\gamma &= \sum_{i=1}^{\ell} \left\{ \mu(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) + \gamma \cdot \sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) \right\} + \sum_{i=1}^{\ell-1} w_{\mathrm{r}}^{\mathrm{t}}(e_i,e_{i+1},t_i) \\
&= \mu_0 + \gamma \cdot \sum_{i=1}^{\ell} \sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) \\
&= \mu_0 + \gamma \cdot \sqrt{\left\{ \sum_{i=1}^{\ell} \sigma(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) \right\}^2} \\
&\leq \mu_0 + \gamma \cdot \sqrt{n \sum_{i=1}^{\ell} \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i))} \\
&= \mu_0 + \gamma \cdot n^{\frac{1}{2}} \sigma_0, \\
\sigma_\gamma^2 &= \sum_{i=1}^{\ell} \sigma^2(\tilde{t}_{\mathrm{e}}^{\mathrm{t}}(e_i,t_i)) \\
&= \sigma_0^2.
\end{aligned}$$

This suggests that $\zeta \leq \frac{1}{2}$. So, we can approximate the expectation and variance of the travel time for a driver that chooses $\gamma > 0$ by $\mu_\gamma \approx \mu_0 + \gamma \cdot n^{\zeta} \sigma_0$, and $\sigma_\gamma \approx \sigma_0$ respectively. This eliminates the necessity to store the expectation and standard deviation for every possible value of γ, leading to a drastic reduction in the amount of data that needs to be stored.

As discussed in Section 5.5.2, car navigation systems generally use a cost-function that is a weighted sum of the time-dependent travel time of an edge or route and other time-independent costs such as the route or edge length or penalties for road segments of a certain type such as motorway, ferry and toll road segments. These time-independent costs are deterministic. Let $\tilde{x} = a_1 \tilde{t} + a_2 c$ denote the cost of a route with $\tilde{t}$ the route travel time and c the time-independent deterministic route cost, for example the route length. The factors a_1 and a_2 are used to weigh both costs. We can compute the expected route cost $\mu(\tilde{x})$ as $\mu(\tilde{x}) = \mu(a_1 \tilde{t} + a_2 c) = a_1 \mu(\tilde{t}) + a_2 c$. Similarly the variation of the route cost $\sigma^2(\tilde{x})$ is given by $\sigma^2(\tilde{x}) = \sigma^2(a_1 \tilde{t} + a_2 c) = a_1^2 \sigma^2(\tilde{t})$. So, in order to compute the expected cost of a stochastic route edge we only need to store the expected travel time of the route edge and a proper characterization of the route edge such that the time-independent deterministic cost of a route edge can be computed. Furthermore, we only need to store the variance of the travel time (or the standard deviation) to determine the variance of the cost of a stochastic route edge. Therefore, we only consider approximating μ_γ and σ_γ for travel times. Note that we also need to store profiles of the expected travel time and travel time variance only in order to determine the cost of a route edge, assuming this cost is given by a weighted linear combination of the travel time and a deterministic time-independent cost.

Note that if a value for σ_γ, or equivalently $\sigma(\tilde{w}_p(\tilde{\pi}))$ is stored, then we can compute a stochastic route for every β. So by storing μ_0 and σ_0 and n, the number of time-dependent edges in the route between the two boundary nodes, we can compute a stochastic route for every β and γ.

6.5.3 Computational Evaluation

In this section, we determine the number of profiles that have to be stored to guarantee a pre-specified maximum error ε in the expected route cost and the variance or standard deviation of the route cost. These profiles are determined for a single basic type of driver that uses $\gamma = 0$. Subsequently, we consider multiple driver types. We study the possibilities to use the profiles (of the expected route cost and standard deviation of the route cost) created for drivers with $\gamma = 0$ to determine profiles for other types of drivers with $\gamma > 0$. We do this for both the expected route cost and the standard deviation of the route cost. So, we study if the suggested approximation $\mu_\gamma \approx \mu_0 + \gamma \cdot n^\zeta \sigma_0$ and $\sigma_\gamma \approx \sigma_0$ is supported by numerical evidence.

We used algorithm X-MATCH in Figure 5.20 to determine the number of profiles that is needed to achieve a specific maximum error ε in the stored expected route cost and variance or standard deviation. Note that the number of profiles needed to store the expected travel time is already determined in Section 5.5.3 (Table 5.3) because we determine these profiles for $\gamma = 0$, and in that case the expected stochastic travel time is equal to the deterministic time-dependent travel time. The number of profiles needed to store the variances for $\gamma = 0$ can be found in Table 6.2. The results for the standard deviation can be found in Table 6.3.

λ	r_i	$\lvert X \rvert$	ε					
			5%	1%	0.5%	0.1%	0.05%	0.01%
1	r_i^C	973,494	824	15,939	34,920	101,140	128,404	152,607
1	r_i^S	1,014,131	239	4,275	10,546	50,389	78,991	118,857
1	r_i^C	1,068,068	164	2,502	6,169	35,640	60,466	103,094
9	r_i^C	1,429,102	29	366	1,013	7,014	12,050	25,846
10	r_i^S	2,418,230	24	299	780	4,843	8,714	20,268
6	r_i^C	1,924,665	23	244	623	3,480	6,129	14,088

Table 6.2. *Number of profiles to store the variance σ_0^2 of the travel time.*

From Tables 6.2 and 6.3 we can see that the number of required profiles is clearly much smaller for storing the standard deviation than for storing the variance. This can be explained mainly by observing that a maximum error ε in the standard deviation corresponds to a maximum error $(1+\varepsilon)^2 - 1 \approx 2\varepsilon$ in the variance. So we should for instance compare the number of profiles for σ_0 for $\varepsilon = 0.5\%$ with the number of profiles for σ_0^2 for $\varepsilon = 1\%$. However, then we still see from Tables 6.2 and 6.3 that the required number of profiles for σ_0 is smaller than for σ_0^2. Apparently, the standard

λ	r_i	$\|X\|$	ε					
			5%	1%	0.5%	0.1%	0.05%	0.01%
1	r_i^C	973,494	195	5,976	17,093	71,530	102,524	147,888
1	r_i^S	1,014,131	65	1,620	4,647	28,754	52,526	110,085
1	r_i^C	1,068,068	52	950	2,711	18,945	37,996	93,530
9	r_i^C	1,429,102	12	159	492	4,240	8,198	21,889
10	r_i^S	2,418,230	15	151	399	2,945	5,715	16,632
6	r_i^C	1,924,665	12	125	343	2,174	4,046	11,526

Table 6.3. *Number of profiles to store the standard deviation σ_0 of the travel time.*

deviation profiles can be better approximated by our method than the variance. Note that we did not determine the required number of profiles for $f(i,j) = 1$ because we have seen in Section 5.5.3 that determining a value $f(i,j)$ for each profile results in fewer profiles that need to be stored.

The same trends apply to standard deviation and variance profiles as to the expected value profiles in Section 5.5.3. To achieve a higher accuracy, we need to store more profiles. Furthermore, for a partition with smaller cells, we need to store fewer profiles to achieve the same accuracy. This is caused by the reduced number of edges with an uncertain cost in the optimum routes between pairs of boundary nodes. The profiles of these routes are more likely to closely resemble each other. This also explains the reduced number of profiles that is required for multi-level partitions compared to single-level partitions to achieve the same accuracy. The highest-level cells in the multi-level partitions typically have (much) fewer boundary nodes than the cells of the single-level partitions. As a result, fewer profiles of routes with a high uncertainty in travel time need to be approximated. Therefore, the required number of profiles is lower than for single-level partitions.

We used identical standard deviations for edges with identical travel speeds. If more diversity in uncertainty in travel times is used, the number of required standard deviation (or variance) profiles will increase, just as for the expected travel times. However, we do not expect the increase to be very large.

From the results, we can see that for the best single-level partition and multi-level partition about 19,000 and 4,300 profiles need to be stored to achieve a guaranteed maximum error of the standard deviation of 0.1% per edge. We expect that this is still practically achievable for car navigation systems. Note that in this case the guaranteed maximum error per edge of the variance is $1.001^2 - 1 = 0.002001$, or about 0.2%.

So far, we need to store a profile for each type of driver, i.e. for each possible value of γ. In order to see if we can determine the optimum route cost for every type of driver from a single basic type (i.e. the driver with $\gamma = 0$) we determine an optimum route using $\gamma = 0$ between each pair of boundary nodes of a single cell for

departure times $0, 900, \ldots, 86,400$. We also determine the number of time-dependent edges $L^t(p)$ in each found route p for each departure time and the value of μ_γ and σ_γ for $\gamma = 1,2,3$. Given these values we determine $y_i = \ln(\frac{\mu_\gamma - \mu_0}{\gamma\sigma_0})$, and $x_i = \ln(L^t(p))$ for each route and each departure time. This leads to a collection $(y_1,x_1),\ldots,(y_\ell,x_\ell)$ for $\gamma = 1,2,3$.

As explained in Section 6.5.2, the value of μ_γ can be approximated by $\mu_0 + \gamma\cdot(L^t(p))^\zeta\sigma_0$. So we would like to determine how accurate the values of μ_γ are described by $\mu_0 + \gamma\cdot(L^t(p))^\zeta\sigma_0$, or equivalently how accurate the values of $y_i = \ln(\frac{\mu_\gamma-\mu_0}{\gamma\sigma_0})$ are described by $\zeta x_i = \zeta\ln(L^t(p))$. Note that because $\lim_{x\downarrow 0}\ln(x) = -\infty$ and because for routes with no time-dependent edges ($L^t(p) = 0$) we have $\mu_\gamma = \mu_0$ and $\sigma_0 = 0$, we only need to compute y_i and x_i for those routes with $L^t(p) > 0$ or equivalently for routes with $\mu_\gamma \neq \mu_0$. In order to determine the value of ζ such that the values of y_i are best described by x_i, we use linear regression.

We have a collection $y_1,\ldots,y_\ell$ and a collection $x_1,\ldots,x_\ell$, and we determine the value of ζ such that $y_1,\ldots,y_\ell$ is best approximated by $\zeta x_1,\ldots,\zeta x_\ell$. Using the ordinary least squares method [Montgomery, Peck & Vining, 2001], this leads to

$$\zeta = \frac{\sum_{j=1}^{\ell} y_j x_j}{\sum_{j=1}^{\ell} x_j^2},$$

and

$$R^2 = \frac{\sum_{j=1}^{\ell}(\zeta x_j - \frac{1}{\ell}\sum_{i=1}^{\ell} y_i)^2}{\sum_{j=1}^{\ell}(y_j - \frac{1}{\ell}\sum_{i=1}^{\ell} y_i)^2}.$$

Here the regression coefficient $R^2 \in [0,1]$ denotes the quality of the approximation. Note that $R^2 = 1$ denotes a perfect approximation, so $y_i = \zeta x_i$ for every $i = 1,\ldots,\ell$, and $R^2 = 0$ denotes that ζx_i gives no information at all about the value of y_i.

We also determine $y_i' = \ln(\frac{\mu_\gamma-\mu_0}{\sigma_0})$ for each route and each departure time for $\gamma = 1,2,3$. This leads to the collection $y_1',\ldots,y_\ell'$. Subsequently, we determine values ζ_0' and ζ_1' such that $(y_1',\ldots,y_\ell')$ is best approximated by $(\zeta_0' + \zeta_1' x_1,\ldots,\zeta_0' + \zeta_1' x_\ell)$. Using the ordinary least squares method this leads to

$$\zeta_1' = \frac{\ell\sum_{j=1}^{\ell} y_j' x_j - \sum_{i=1}^{\ell} x_i \sum_{i=1}^{\ell} y_i'}{\ell\sum_{j=1}^{\ell} x_j^2 - (\sum_{j=1}^{\ell} x_j)^2},$$

$$\zeta_0' = \frac{1}{\ell}\sum_{j=1}^{\ell} y_j' - \zeta_1'\frac{1}{\ell}\sum_{i=1}^{\ell} x_i,$$

and

$$(R')^2 = \frac{\sum_{i=1}^{\ell}((\ell\sum_{i=1}^{\ell} x_i y_i' - \sum_{i=1}^{\ell} x_i\sum_{i=1}^{\ell} y_i')^2}{(\ell\sum_{i=1}^{\ell} x_i^2 - (\sum_{i=1}^{\ell} x_i)^2)(\ell\sum_{i=1}^{\ell}(y_i')^2 - (\sum_{i=1}^{\ell} y_i')^2)}.$$

We have computed the values of ζ, $e^{\zeta_0'}$, ζ_2' and R^2, $(R')^2$ for different partitions, and for different values of γ, by determining the value of y_i, y_i' and x_i for each pair of

boundary nodes of a cell and for every departure time t with $t = 0, 900, \ldots, 86,400$. The results can be found in Table 6.4.

λ	r_i	γ	ζ	R^2	$e^{\zeta'_0}$	ζ'_1	$(R')^2$
1	r_i^C	1	0.384222	0.751844	0.880170	0.424964	0.918330
1	r_i^C	2	0.384131	0.751866	1.760861	0.424778	0.917996
1	r_i^C	3	0.384027	0.751448	2.642566	0.424520	0.916878
1	r_i^S	1	0.380395	0.765625	0.913523	0.413507	0.903485
1	r_i^S	2	0.380356	0.765894	1.826628	0.413551	0.904175
1	r_i^S	3	0.380308	0.765917	2.740608	0.413415	0.903838
1	r_i^C	1	0.375157	0.780584	0.931246	0.402833	0.898982
1	r_i^C	2	0.375129	0.781077	1.861843	0.402940	0.900152
1	r_i^C	3	0.375092	0.781334	2.793418	0.402812	0.900061
9	r_i^C	1	0.375819	0.817700	0.964804	0.394341	0.898868
9	r_i^C	2	0.375838	0.822487	1.929665	0.394345	0.904057
9	r_i^C	3	0.375834	0.824251	2.895213	0.394214	0.905429
10	r_i^S	1	0.347554	0.765900	0.992562	0.353245	0.791038
10	r_i^S	2	0.347688	0.783482	1.984237	0.353723	0.810742
10	r_i^S	3	0.347705	0.790632	2.977615	0.353417	0.816665
6	r_i^C	1	0.383486	0.890168	0.979860	0.392968	0.934042
6	r_i^C	2	0.383533	0.897175	1.959637	0.393034	0.941479
6	r_i^C	3	0.383532	0.900212	2.940574	0.392856	0.943840

Table 6.4. *Relation* $\mu_\gamma = \mu_0 + \gamma(L^t(p))^\zeta \sigma_0$ *and* $\mu_\gamma = \mu_0 + e^{\zeta'_0}(L^t(p))^{\zeta'_1}\sigma_0$.

As we can see from Table 6.4 we find $\zeta < 0.5$ (and also $\zeta'_1 < 0.5$) as expected. In particular, we see that $\zeta < \zeta'_1$ and $e^{\zeta'_1} < \gamma$. Furthermore, we can see that the regression coefficient $(R')^2$ is larger than R^2. This can be explained by the fact that determining two parameters to approximate the values y_i leads to better results than determining only a single value. We can see that $\mu_0 + \gamma(L^t(p))^{0.375}\sigma_0$ seems to reasonably approximate the value of μ_γ for different partitions, where 0.375 is the average of ζ in column 4. However, the quality of approximating μ_γ by $\mu_0 + e^{\zeta'_0}(L^t(p))^{0.397}\sigma_0$ is clearly better judging from R^2 and $(R')^2$, where 0.397 is the average of ζ'_1 in column 7. Note that for both relations we have $\mu_\gamma = \mu_0$ for routes with no time-dependent edges (i.e. routes with $L^t(p) = 0$) as should be the case. We can also see that we can approximate $e^{\zeta'_0}$ by $\gamma e^{\zeta'_2}$ with $e^{\zeta'_2} = 0.944$ for all tested partitions, where 0.944 is the average of $\frac{e^{\zeta'_0}}{\gamma}$ which can be computed using the data in column 6. This leads to $\mu_\gamma = \mu_0 + 0.944\gamma(L^t(p))^{0.397}\sigma_0$. So we can conclude that we are able to approximate μ_γ reasonably close by using

$$\mu_\gamma = \mu_0 + \gamma e^{\zeta'_2}(L^t(p))^{\zeta'_1}\sigma_0,$$

where ζ'_1 and $e^{\zeta'_2}$ are independent of γ. We can determine the value of ζ'_1 for a particular partition by the method described above. The value of $e^{\zeta'_2}$ can be set equal to the average of $\frac{e^{\zeta'_0}}{\gamma}$ for different values of γ, or to $e^{\zeta'_0}$ for $\gamma = 1$. As a result, we can plan stochastic time-dependent routes for different types of drivers (with different preferred γ and β) using partitioned roadgraphs by storing only the expectation and variance of the optimum travel time and cost for $\gamma = 0$ between each pair of boundary nodes of a single cell. The quality of this approximation is reasonable as can be seen from Table 6.4.

We can also approximate the value of σ_γ by σ_0 as indicated in Section 6.5.2. We can again use linear regression to determine a relation between σ_γ and σ_0. For each found route and for each departure time we compute $y_i = \sigma_\gamma$, $\gamma = 1,2,3$ and $x_i = \sigma_0$. Using the definitions of ζ, and R^2 and ζ'_0, ζ'_1, $(R')^2$ this leads to the results in Table 6.5.

λ	r_i	γ	ζ	R^2	ζ'_0	ζ'_1	$(R')^2$
1	r_i^C	1	1.001203	0.998743	-0.015166	1.001621	0.999577
1	r_i^C	2	1.002514	0.996804	-0.029633	1.003333	0.998433
1	r_i^C	3	1.003861	0.994348	-0.042689	1.005045	0.996692
1	r_i^S	1	1.000631	0.999399	-0.005697	1.000830	0.999795
1	r_i^S	2	1.001307	0.998451	-0.011253	1.001700	0.999234
1	r_i^S	3	1.002000	0.997227	-0.016378	1.002573	0.998366
1	r_i^C	1	1.000541	0.999525	-0.004187	1.000698	0.999839
1	r_i^C	2	1.001100	0.998777	-0.008197	1.001408	0.999392
1	r_i^C	3	1.001666	0.997806	-0.011886	1.002113	0.998697
9	r_i^C	1	1.000269	0.999876	-0.000726	1.000326	0.999989
9	r_i^C	2	1.000508	0.999745	-0.001359	1.000614	0.999958
9	r_i^C	3	1.000720	0.999606	-0.001907	1.000869	0.999904
10	r_i^S	1	1.000217	0.999916	-0.000501	1.000254	0.999990
10	r_i^S	2	1.000421	0.999817	-0.000970	1.000493	0.999960
10	r_i^S	3	1.000616	0.999704	-0.001414	1.000720	0.999913
6	r_i^C	1	1.000259	0.999894	-0.000582	1.000306	0.999989
6	r_i^C	2	1.000488	0.999778	-0.001090	1.000576	0.999955
6	r_i^C	3	1.000684	0.999652	-0.001519	1.000808	0.999898

Table 6.5. *Relation* $\sigma_\gamma = \zeta\sigma_0$ *and* $\sigma_\gamma = \zeta'_0 + \zeta'_1\sigma_0$.

From Table 6.5 we see again that using two parameters to approximate the value of σ_γ is better than using only a single value since $R^2 < (R')^2$. However the difference is much smaller than in Table 6.4. Furthermore, we see that the computed value of ζ'_0 is negative. Because the standard deviation of a route can never be negative, and the difference between R^2 and $(R')^2$ is very small, we prefer the relation $\sigma_\gamma = \zeta\sigma_0$ to $\sigma_\gamma = \zeta'_0 + \zeta'_1\sigma_0$. We can see that $\zeta \approx 1$ and also $\zeta'_0 \approx 0$, $\zeta'_1 \approx 1$ so approximating σ_γ

by σ_0 is reasonable. Considering the values of ζ we see that we can approximate ζ by $1+\gamma\zeta_1$ with $\zeta_1 = 0.00125$ for a single-level partition with $r_i = b_i(b_i-1)$ for example, where 0.00125 is the average of $\frac{\zeta}{\gamma}$ for single-level partitions with r_i^C. Similarly, ζ_1' can be approximated by $1+\gamma\zeta_2'$ and ζ_0' by $\gamma\zeta_3'$ with $\zeta_2' = 0.00166$ and $\zeta_3' = -0.015$ for single-level partitions with $r_i = b_i(b_i-1)$ for example, where 0.00166 and -0.015 are the averages of $\frac{\zeta_1'-1}{\gamma}$ and $\frac{\zeta_0'}{\gamma}$ respectively. As mentioned before, we prefer the relation $\sigma_\gamma = \zeta\sigma_0$. So for a particular partition, we can determine a value ζ using the method described above. Subsequently we compute ζ_1 by taking the average of $\frac{\zeta-1}{\gamma}$ for different values of γ or by setting $\zeta_1 = \zeta$ for $\gamma = 1$. Then we can approximate σ_γ by

$$\sigma_\gamma = (1+\gamma\zeta_1)\sigma_0.$$

As can be seen from the values of R^2 in Table 6.5 this approximation is quite good.

6.6 Conclusion

In this chapter we have discussed the planning of stochastic time-dependent routes. First of all, we discussed under which conditions we can plan optimum stochastic time-dependent routes by using a modified version of a standard route planning algorithm such as the $A^\star$-algorithm. Assuming these conditions are satisfied, we presented a route planning algorithm, the $S^\star$-algorithm that can be used to plan optimum stochastic time-dependent routes. Planning optimum stochastic routes, requires a definition of the optimality (or cost) of a route. We formulated a cost and optimality condition that can balance uncertainty (or equivalently the standard deviation) in travel time against the expected travel time. This can be used to represent the attitude of most drivers towards uncertainty in their travel times. Because a car navigation system has to plan routes fast, and because the determination of the exact expected travel time of a route and its variance is very complicated, we used approximations to determine the variance and expected value of the travel time of a route. It would be very interesting to determine the expected travel time and its variance exactly though. Ideally, we would like to be able to determine these values fast. Furthermore, we would like to require only the expected travel time and variance of a route $\langle e_1,\ldots,e_\ell\rangle$ together with data on edge $e_{\ell+1}$ to determine the expected travel time and variance of route $\langle e_1,\ldots,e_{\ell+1}\rangle$ exactly.

Using real-world data on uncertainty in travel times, we evaluated the usefulness of stochastic time-dependent route planning by planning 2,000 fastest routes through the Netherlands. From the results, we concluded that stochastic time-dependent planning can be very useful, but only if the driver desires a high level of certainty concerning his (latest possible) arrival time.

We used a single expected travel time profile for all motor- and highways in the Netherlands. We determined the standard deviation of the travel time by divid-

ing the difference between the maximum time-independent travel time and the time-dependent travel time by a fixed factor ς that is the same for all motor- and highways. Just as for deterministic time-dependent route planning, we can also use different stochastic profiles for different motorways. This can be done by computing a value ς for every motorway. Alternatively, we can also use variance or standard deviation profiles just as for the mean travel speed. In order to use different stochastic profiles for each motorway, we also need to determine which edges are part of a particular motorway. The problem of determining the expected travel time of a single edge is similar to the problem for deterministic time-dependent travel speeds. For stochastic time-dependent planning, we also need to determine the variance (or standard deviation) of the travel time of a single edge given the variance of the travel time of a route consisting of multiple edges. Like the expected travel time, the variance of the travel time is most likely to be larger near a junction that causes most congestion. Note that also less important roads are subject to uncertainty in their travel times, so a model could also be developed for the uncertainty in travel times on roads for which the uncertainty in travel times cannot be determined from reported travel times.

We also investigated the possibilities to combine stochastic time-dependent planning with partitions. In order to plan optimum stochastic routes with a partitioned roadgraph, we need to store a lot of information on routes between boundary nodes of single cells. If optimality of stochastic routes is not required and high-quality routes are satisfactory, which is usually the case in stochastic time-dependent route planning, then the planning of stochastic time-dependent routes in partitioned roadgraph is possible with only moderate storage requirements. We presented an approach to store the required data on stochastic routes by creating profiles of the expected travel time and standard deviation. The required number of profiles can be substantially reduced by approximating these profiles using the X-MATCH algorithm presented in Chapter 5. Like for deterministic time-dependent travel times, we need a different algorithm to determine the minimum number of profiles needed to store the expected travel times. This method can also be used to determine the minimum number of profiles needed to store the standard deviation of travel times for a pre-specified accuracy.

Finally, we have seen that the expectation and variance of the cost of a particular route for different attitudes of the driver can be approximated reasonably well by storing only the costs according to one basic attitude. As a result, high-quality stochastic time-dependent routes can be planned fast for all kinds of different types of drivers in partitioned roadgraphs requiring only limited storage space. Alternative methods to determine the route cost for different types of drivers based on one basic type can also be developed. Furthermore, an expression of the quality of the determined routes would be interesting.

7

Comparison with Product Route Planners

In the previous chapters, we have described route planning algorithms that can be used to plan optimum routes in time-independent, time-dependent and stochastic time-dependent roadgraphs. Of course, currently car navigation systems already use a route planning algorithm to determine a route from a start node s to a destination node d. Section 7.1 briefly discusses the route planning algorithm that is implemented in product versions of car navigation systems. After describing this route planning algorithm, we compare the average performance of this algorithm with our $C^\star$-algorithm for planning time-independent routes in Section 7.2, and we will see that the $C^\star$-algorithm plans optimum routes faster on average than current product route planners plan non-optimum routes. Section 7.3 discusses some exceptional cases in which our algorithm performs much better than the used product route planner. The advantages and disadvantages of our approach are discussed in Sections 7.4 and 7.5 respectively. In Section 7.6, we discuss how additional route planning functionality offered in car navigation systems, for example planning alternative routes, can be implemented using partitioned roadgraphs. Section 7.7 presents the conclusions.

7.1 Standard Route Planner Algorithms

The route planning algorithm implemented in (most) car navigation systems is an approximation algorithm that is based on the $A^\star$-algorithm. It thus is not guaranteed

that an optimum route from a start node s to a destination node d is planned. We will call this algorithm the *RP*-algorithm.

The *RP*-algorithm is based on a division of the road network into more and less important roads. As was already indicated in Section 5.4, every road segment of the road network, or equivalently, every edge in the roadgraph, has been assigned a certain road class. This road class can be used to indicate the importance of a road: the higher the road class number the less important the road. For example, edges with road class 0 are mainly motorways and edges with road class 1 are mainly highways.

The *RP*-algorithm basically evaluates edges of all road classes near the car and the destination, and the larger the distance to the car or destination, the higher the importance of an edge needs to be for the algorithm to evaluate that edge. Note that this basically corresponds to the way humans plan a route.

Specifically, the roadgraph is divided into a number of levels. The road class of an edge can for example be used to divide the road network into levels. The most detailed level is called the *detailed* network (DT), and it contains all edges for example of all road classes. Also a number of higher levels are created, called the *High-Level* networks (HL). These high-level networks contain for example all edges of at least a certain road class. For example, a HL network can be created that consists of all edges of road class 3, 2, 1 and 0. We call this network HL3. Similarly, the HL network consisting of all edges with road class 0 is called HL0. So, for a road network with 7 possible road classes (0-6) we can create for example 6 possible high level networks (HL0-HL5) and a detailed network DT. Note that the network HL6 would correspond to the network DT. The creation of the DT network and a number of HL networks is done beforehand, and these networks are stored on the CD or DVD containing the map. Usually, at most three HL networks are actually created.

When a route needs to be planned, the *RP*-algorithm first determines areas for example around the car and destination position. These areas indicate in which area the algorithm should use which network. If the car and destination are located very close together the algorithm generally only uses the DT network. However, for a car and destination position located far apart, the DT network is generally only used very close to the car and destination position.

Specifically, areas are created for every network except for the least detailed network. So, if for example a DT network and three HL networks are stored on the CD or DVD, then for example two areas (one surrounding the destination and one surrounding the car position) are created for three of these networks, in this case 6 areas. Under certain conditions multiple areas for a single network can be combined into one area, see for example Figure 7.1.

The searchgraph of the *RP*-algorithm consists of all edges of a network contained in the areas for that network. For example, if only a DT, a HL2 and a HL0 network exist, then the searchgraph consists of all edges contained in the areas of the DT

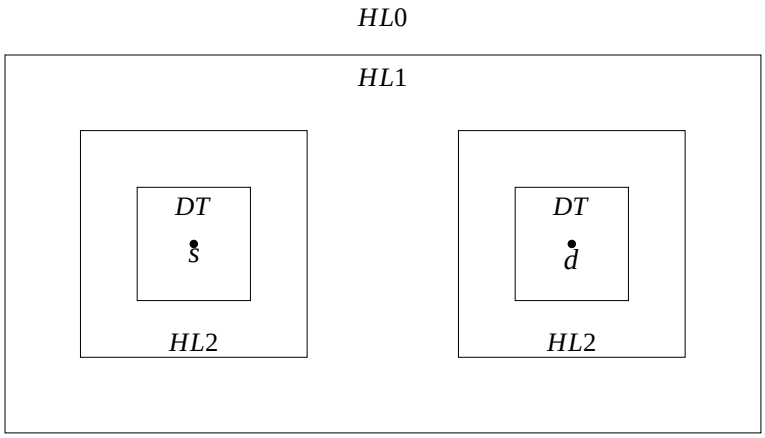


Figure 7.1. *Example of areas used by the RP-algorithm.*

network, all edges in the HL2 network that are contained in the areas of the HL2 network, and all edges in the HL0 network located outside the areas of the HL2 network.

The *RP*-algorithm then uses a modified $A^\star$-algorithm to plan a route on this searchgraph. The estimated cost from a location to the destination usually overestimates the actual cost to increase the planning speed. After an initial route has been found the driver receives advice. We call this route an *initial* route. Subsequently, the *RP*-algorithm finishes the current route, for example by trying to improve it. The resulting route is called a *final* route.

7.2 Average Performance of the $C^\star$- and *RP*-Algorithm

This section compares the average-case performance of the *RP*-algorithm[1] and the $C^\star$-algorithm. In order to compare the average performance of these two route planning algorithms, we plan the fastest and shortest routes between all 2,000 start and destination nodes routes from our route batch, see Table 4.3, using the maximum allowed travel speeds that depend on the road class. We use both algorithms to plan these routes, and we compare the number of evaluated edges and the route lengths and driving times. In order to achieve a fair comparison, we need to use the same costs for each edge, or similarly, the same lengths and driving speeds. Note that the *RP*-algorithms we used for comparison can use two different speeds, a driving speed and a planning speed which incorporates also other factors besides pure driving speeds. The driving speeds as used in Chapter 4 are not exactly the same as the used driving or

[1]We used the VDO Dayton route planning algorithm developed by Siemens VDO Automotive in Eindhoven as basis for our comparisons.

planning speeds by the *RP*-algorithm. Because the speeds used by the *RP*-algorithm are less easy to adapt, we modified the speeds used by the $C^\star$-algorithm. Therefore, the results of the $C^\star$-algorithm using these modified speeds are somewhat different from the results in Chapter 4. Note that for the driver the results using driving speeds seem most important. However, the *RP*-algorithm is specifically tuned to perform well using planning speeds. Furthermore, we modified the *RP*-algorithm such that it uses only the length of edges to determine shortest routes. We use the multi-level partition generated with $r_i = b_i(b_i - 1)$ as input for our $C^\star$-algorithm, because we have seen in Chapters 3 and 4 that this partition leads to the least number of edges evaluated by the route planning algorithm.

The data used to create our partitions has a different format than the standard databases used by the current route planner of Siemens VDO Automotive but they are both based on CEN GDF 3.0 data (Geographic Data Files), supplied by NavTeq. GDF is a European (CEN) standard, that is used to describe and transfer road networks and road related data. As a result, the roadgraph based on the standard database of Siemens VDO Automotive is slightly different from our roadgraph. Sometimes a single edge in the roadgraph used for partitioning is represented by two or more edges in the roadgraph used by the *RP*-algorithm. However, these additional edges are created so that parts of the roadgraph can be read from a CD or DVD instead of just the entire roadgraph. When a partition is used, these additional edges are not needed because the roadgraph is already partitioned into cells which can be read from the CD or DVD separately.

The results of the average-case performance of the *RP*-algorithm and the $C^\star$-algorithm can be found in Tables 7.1 and 7.2. The results for initial routes are denoted by (I) and those for final routes by (F). Columns 2 and 5 present the number of evaluated edges, columns 3 and 6 present the driving time in seconds, and columns 4 and 7 present the route length in meters. In the road network several nodes exist with identical coordinates, for example at cloverleaf junctions. If the two algorithms select different nodes, the resulting routes are quite different. These routes have been removed from the comparison between individual routes in row 4 and 5 of Tables 7.1 and 7.2.

The *RP*-algorithm requires on average more edge evaluations to plan the 2,000 routes from our route batch than the $C^\star$-algorithm. As explained in Section 7.1 the *RP*-algorithm is an approximation algorithm and does not always find an optimum route unlike the $C^\star$-algorithm. We can see from the results, that even though the $C^\star$-algorithms needs to evaluate fewer edges than the *RP*-algorithm, the $C^\star$-algorithm plans optimum routes, while the routes planned by the *RP*-algorithm may have a cost that is above optimum. Specifically, using driving (or planning) speeds, the time and distance are more than 8 (or 4.5) minutes and 7 (or 6.5) kilometers above optimum for initial fastest and shortest routes respectively. After trying to improve

Algorithm	Fastest route			Shortest route		
	Eval. Edges	Time (s)	Length (m)	Eval. Edges	Time (s)	Length (m)
$C^\star$	1,548	4,143	97,647	1,333	5,670	87,331
RP (I)	1,619	4,588	102,936	2,285	5,021	94,518
RP (F)	-	4,468	100,271	-	5,037	92,450
$RP - C^\star$ (I)	112	494	5,098	951	-631	7,325
$RP - C^\star$ (F)	-	373	2,553	-	-658	5,577
$\frac{RP-C^\star}{C^\star}$ (I)	1.61	0.16	0.10	3.00	-0.01	0.13
$\frac{RP-C^\star}{C^\star}$ (F)	-	0.11	0.04	-	-0.03	0.06

Table 7.1. *Comparison average results RP and $C^\star$ with driving speeds.*

Algorithm	Fastest route			Shortest route		
	Eval. Edges	Time (s)	Length (m)	Eval. Edges	Time (s)	Length (m)
$C^\star$	1,628	4,089	100,612	1,333	8,221	87,331
RP (I)	1,652	4,350	103,650	2,158	5,153	94,262
RP (F)	-	4,225	101,432	-	5,236	92,234
$RP - C^\star$ (I)	19	267	2,785	826	-3,061	6,993
$RP - C^\star$ (F)	-	150	704	-	-2,977	4,962
$\frac{RP-C^\star}{C^\star}$ (I)	1.33	0.08	0.07	2.43	-0.29	0.12
$\frac{RP-C^\star}{C^\star}$ (F)	-	0.04	0.01	-	-0.29	0.06

Table 7.2. *Comparison average results RP and $C^\star$ with planning speeds.*

the found routes, the final routes are still more than 6 (or 2.5) minutes and 5 (or 4.5) kilometers above optimum for fastest and shortest routes respectively using driving (or planning) speeds. The fourth and fifth row compare the results for the individual planned routes by the $C^\star$- and the *RP*-algorithm. Therefore, the difference between the first two rows, and the first and third row in Tables 7.1 and 7.2 are not equal to the results in the fourth and fifth row of Tables 7.1 and 7.2 respectively. Also the relative differences have been determined by taking the average of the relative difference for each route. Note that we did not include the number of edges evaluated to improve the initial routes because this improvement is done while the driver already receives guidance. The speed of this mechanism is thus not very important. We can also see that the *RP*-algorithm is much faster than the standard $A^\star$-algorithm from Table 4.4 and Table 4.5. Although the results in Tables 4.4 and 4.5 were generated using a roadgraph with slightly different edge costs, the number of edges evaluated by the $A^\star$-algorithm using the roadgraph from this section will be comparable.

7.3 Exceptional Cases of the *RP*-Algorithm

Section 7.2 discussed the average-case performance of the $C^\star$-algorithm compared to the *RP*-algorithm. Unlike the $C^\star$-algorithm, the *RP*-algorithm is not guaranteed to plan optimum routes. This section presents some worst-case results. Again the driving time of an edge depends on the maximum allowed driving speed which, in turn, depends on the edge's road class. The results of the worst-case comparison of the *RP*-algorithm and the $C^\star$-algorithm can be found in Tables 7.3 and 7.4. Results for initial routes are denoted by (I) and those for final routes by (F). Columns 2 and 5 present the number of evaluated edges, columns 3 and 6 present the driving time in seconds, and columns 4 and 7 present the route length in meters. Like for the average-case results in Tables 7.1 and 7.2, routes for which the two algorithms select different start or destination nodes have been removed from the comparison between individual routes in row 4 and 5 of Tables 7.3 and 7.4. Note that Tables 7.3 and 7.4 give the maximum number of evaluated edges, the maximum route travel time, and the maximum route length for both fastest and shortest routes. The routes for which these maxima are attained are not necessarily the same.

Algorithm	Fastest route			Shortest route		
	Eval. Edges	Time (s)	Length (m)	Eval. Edges	Time (s)	Length (m)
$C^\star$	6,440	32,302	418,528	6,159	35,108	386,120
RP (I)	15,888	33,867	425,856	443,390	42,551	402,726
RP (F)	-	33,869	418,816	-	36,051	399,872
$RP - C^\star$ (I)	15,508	4,011	134,455	443,094	16,098	198,266
$RP - C^\star$ (F)	-	2,639	63,371	-	2,307	36,779
$\frac{RP-C^\star}{C^\star}$ (I)	40.81	29.77	53.68	1,496.94	29.77	53.68
$\frac{RP-C^\star}{C^\star}$ (F)	-	0.77	0.90	-	1.27	0.68

Table 7.3. *Comparison worst-case results RP and $C^\star$ with driving speeds.*

From the results in Tables 7.3 and 7.4, we can see that the maximum number of edges evaluated by the $C^\star$-algorithm is much smaller than for the *RP*-algorithm. Specifically, a fastest route exists for which the *RP*-algorithm needs to evaluate 15,888 (or 18,297) edges to find an initial route using driving (or planning) speeds, while the $C^\star$-algorithm does not need more than 380 (or 388) edge evaluations for the same route. In case a shortest route needs to be planned the situation is even worse. There exists an initial route for which the *RP*-algorithm needs to evaluate 403,390 (or 100,463) edges, while the $C^\star$-algorithm only needs to evaluate 296 (or 296) edges to find the optimum route using driving (or planning) speeds. For planning fastest routes the $C^\star$-algorithm requires (for these 2,000 start and destination nodes) a maximum of about 6,400 edge evaluations, which is about 4 times the average number of evalu-

Algorithm	Fastest route			Shortest route		
	Eval. Edges	Time (s)	Length (m)	Eval. Edges	Time (s)	Length (m)
$C^\star$	6,322	17,719	432,962	6,159	35,714	386,120
RP (I)	18,297	18,930	422,916	100,463	23,397	403,486
RP (F)	-	18,726	419,984	-	23,431	402,656
$RP - C^\star$ (I)	17,909	2,835	134,522	100,167	7,460	67,553
$RP - C^\star$ (F)	-	2,347	47,107	-	2,604	36,779
$\frac{RP-C^\star}{C^\star}$ (I)	46.16	16.17	53.68	338.40	16.17	53.68
$\frac{RP-C^\star}{C^\star}$ (F)	-	0.63	0.77	-	0.71	0.67

Table 7.4. *Comparison worst-case results RP and $C^\star$ with planning speeds.*

ated edges (which is about 1,600). For shortest routes planned by the $C^\star$-algorithm the maximum number of evaluated edges is about 6,150. This is less than 5 times the average (of about 1,300). The *RP*-algorithm requires for these 2,000 fastest routes a maximum of about 16,900 edge evaluations, which is about 10 times the average number of evaluated edges (which is about 1,600). For shortest routes the *RP*-algorithm needs to evaluate a maximum of about 440,000 edges, which is even more than 45 times the average number of evaluated edges (which is about 2,200). The large difference between the results for planning fastest and shortest routes by the *RP*-algorithm is caused by the fact that a different definition of the searchgraph is used for planning fastest and shortest routes. Note that the theoretical upperbound on the maximum required number of edges evaluated by the $C^\star$-algorithm can be deduced from the constructed partitions. Specifically, the maximum number of evaluated edges is equal to the maximum number of edges in the searchgraph. For the *RP*-algorithm an upperbound on the number of evaluated edges does not exist.

In worst-case (of the 2,000 routes found by the *RP*-algorithm using driving speeds) the initial route presented by the *RP*-algorithm can have a time of more than 1 hour and a length of almost 200 kilometers above the optimum time and length of the fastest and shortest route respectively. For final routes this is still 44 minutes and more than 36 kilometers above the optimum time and length of the fastest and shortest route respectively, see row 4 and 5 of Table 7.3. Furthermore, the *RP*-algorithm is not able to find 6 fastest and 4 shortest initial routes, and 8 fastest and 4 shortest final routes, out of the 2,000 start and destination nodes. We can also see that the difference between the first two rows and between the first and third row in Tables 7.3 and 7.4 are not equal to the results in the fourth and fifth row of Tables 7.3 and 7.4. This is because the fourth and fifth row in Tables 7.3 and 7.4 compare the results for the individual routes planned by the $C^\star$- and the *RP*-algorithm. Note that the results for length and time in row $\frac{RP-C^\star}{C^\star}$ (I) in Tables 7.3 and 7.4 are somewhat misleading.

They are caused by a route that is only 25 meters long. If this route is not taken into account then the values decrease to at most 3.

One of the problems of the *RP*-algorithm is that sometimes optimum routes contain edges that are not contained in the searchgraph, see Section 7.1 (Figure 7.1). As a result, such an optimum route cannot be planned by the *RP*-algorithm. Consider for example the optimum fastest final route $p^\star$ in Figure 7.2 as planned by the $C^\star$-algorithm (using driving speeds). The *RP*-algorithm plans the route p displayed in Figure 7.3 (using driving speeds). The travel time of route $p^\star$ is 156 minutes, while the travel time of route p is 203 minutes. Obviously route p is much worse than route $p^\star$. Similarly, the optimum shortest route in Figure 7.4 is 109,879 meters long, while the shortest (final) route planned by *RP* is 146,658 meters long. Note that because only final routes can be displayed all *RP* routes displayed in Figures 7.3 to 7.9 are final routes. Figures 7.6 and 7.8 present the optimum fastest and shortest routes between Rotterdam and Nieuwkoop (using driving speeds). The fastest and shortest (final) routes planned by the *RP*-algorithm (using driving speeds) are displayed in Figures 7.7 and 7.9.

7.4 Advantages Compared to the *RP*-Algorithm

The $C^\star$-algorithm has several advantages over the *RP*-algorithm. First of all the $C^\star$-algorithm plans an optimum route between two nodes, unlike the *RP*-algorithm, as can be seen most clearly from Section 7.3. Secondly, the $C^\star$-algorithm evaluates fewer edges on average and in worst-case than the *RP*-algorithm. So if evaluating an edge takes the same amount of time in both algorithms then, by using the $C^\star$-algorithm, we can find an optimum route faster than we can find a non-optimum route with the *RP*-algorithm. Note that with efficient data structures, we expect that it is possible to evaluate an edge in the $C^\star$-algorithm in (approximately) the same time as in the *RP*-algorithm.

Another advantage of the $C^\star$-algorithm is that it uses a pre-processing result (a partition), that is independent of the map supplier. The maps containing the road network that are stored on a CD or DVD are (mainly) supplied by two companies, TeleAtlas and NavTeq. These map suppliers primarily determine the road classes of the road segments in the road network. As a result, they also indirectly determine the searchgraph used by the *RP*-algorithm. So, if a map supplier decides to adapt the road classes of road segments, then this has a direct effect on the routes planned by the *RP*-algorithm and on their quality. Note that the road classes of road segments are not defined very strictly. The definitions of especially road classes 2 and 3 are very vague. As a result, the two map suppliers frequently assign different road classes to corresponding road segments. Furthermore, because the *RP*-algorithm depends on these road classes, a differently assigned road class can cause a large difference in the planned route. Note that the dependency on the maps delivered by the map

Figure 7.2. *Optimum fastest route Emmen - Strijen: 2:36:27.*

Figure 7.3. *Fastest RP (final) route Emmen - Strijen: 3:23:27.*

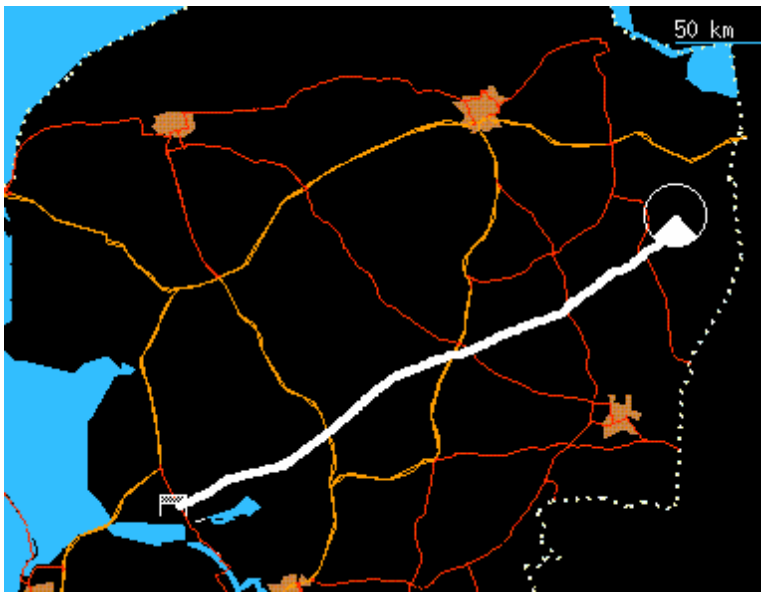


Figure 7.4. *Optimum shortest route Stadskanaal - Noordoostpolder: 109,879 m.*

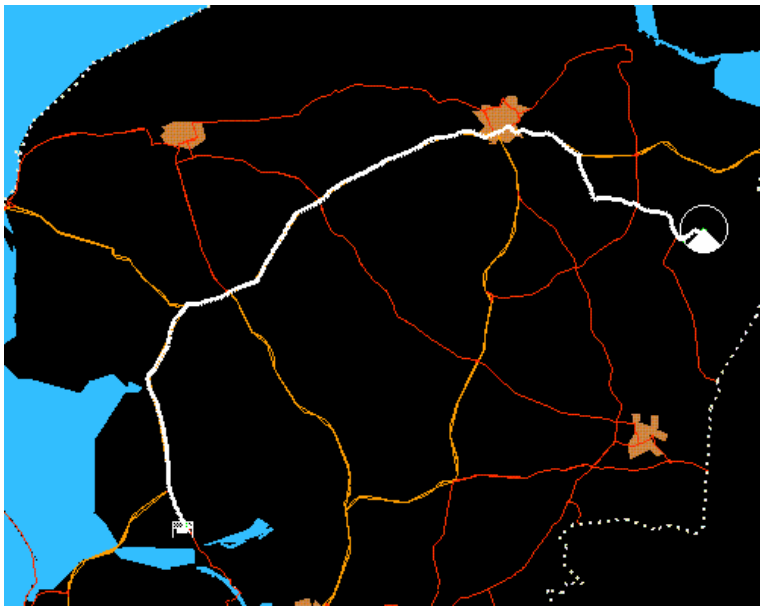


Figure 7.5. *Shortest RP (final) route Stadskanaal - Noordoostpolder: 146,658 m.*

Figure 7.6. *Optimum fastest route Rotterdam - Nieuwkoop: 0:49:23.*

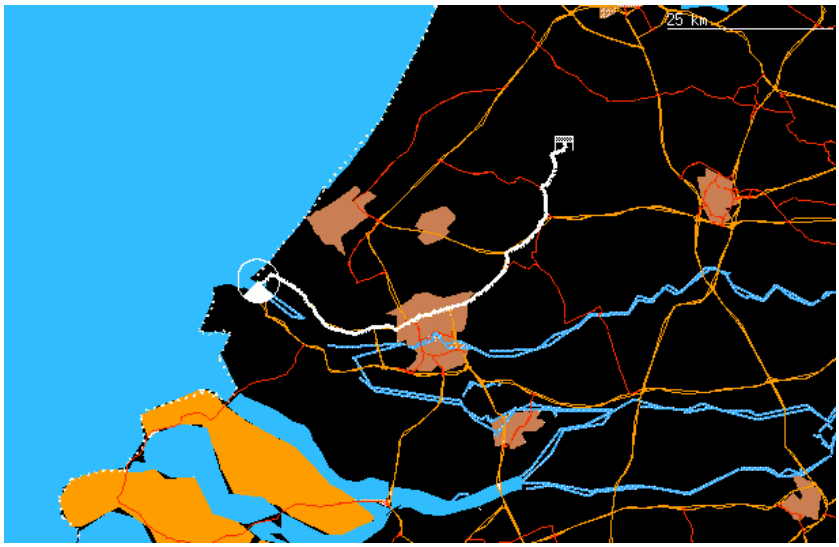


Figure 7.7. *Fastest RP (final) route Rotterdam - Nieuwkoop: 0:57:16.*

Figure 7.8. *Optimum shortest route Rotterdam - Nieuwkoop: 58,847 m.*

Figure 7.9. *Shortest RP (final) route Rotterdam - Nieuwkoop: 70,917 m.*

supplier is hard to explain to individual drivers. The $C^\star$-algorithm on the other hand does not depend directly on the assigned road classes, it only depends on the costs of the edges, and the structure of the underlying graph. Note that currently the road class of an edge is also used to determine the cost of an edge though. If travel speeds are directly assigned to edges then the dependency on the map supplier will almost disappear if the $C^\star$-algorithm is used.

As explained in Chapter 1, car navigation systems receive traffic information that is broadcasted by RDS-TMC (Radio Data System - Traffic Message Channel), or via a connected GSM phone. This information is taken into account by increasing the cost of roads that are congested when planning a route. However, traffic jams are usually only reported if they are longer than a few (typically two) kilometers, and if they cause a significant delay for drivers. However, less severe traffic jams or mild reductions in driving speed caused by everyday peak traffic are not always reported. These traffic circumstances do influence the best route however. Using traffic profiles we can take daily congestion into account while planning a route. These traffic profiles do not suffer from the problem that a traffic jam appears out of the blue. Instead the congestion increases gradually and also less severe congestion can be taken into account. Although the number of edges evaluated to plan an optimum route may change a little due to the changed edge costs, we expect this effect to be relatively small. Uncertainty is not taken into account at all in navigation systems that are currently on the market. Our method is capable of finding a balance between the expected travel time and the uncertainty in travel time. The planning speed of our route planning algorithm for planning stochastic time-dependent routes depends on the used preferences and edge cost functions. For properly selected functions for edge costs and preferences, the planning speed will probably not deteriorate much.

Finally, reading information from the CD or DVD containing the road network is a relatively time-consuming task. In our approach, as soon as the start and destination node have been determined we know which parts of the road network are needed to plan an optimum route (i.e. the searchgraph). Because the searchgraph is generally small, we can most likely load all required data into internal memory. Consequently, all data required for planning an optimum route between two nodes can be requested immediately after the start and destination node have been determined if we use the $C^\star$-algorithm. Therefore, the time spent on waiting for information to become available from the CD or DVD is greatly reduced. For the *RP*-algorithm the potential amount of data required to plan the route between the start and destination node is (generally) much larger. Furthermore, what information is actually required to plan the route is not known (exactly) until the route has been planned. As a result, more time is spent on waiting for information to become available from the CD or DVD.

7.5 Disadvantages Compared to the *RP*-Algorithm

Section 7.4 discussed the advantages of the $C^\star$-like algorithms over the *RP*-algorithm. In this section, we discuss the disadvantages of the $C^\star$-like algorithms compared to the *RP*-algorithm.

The main disadvantage of the $C^\star$-algorithm compared to the *RP*-algorithm is that in order to plan an optimum route for a cost function w_e, the cost of the optimum route between every pair of boundary nodes of a single cell needs to be stored. For the multi-level partition we used to compare the $C^\star$- and *RP*-algorithm, this means that 2,200,000 costs have to be stored for every single cost function. If we assume that every cost can be stored in 24 bits (this is possible if every cost is at most 16,777,215), this leads to 6.6 MB of data for every cost function. Note that experiments of Van der Horst [2003] show that also high-quality partitions can be created that require only about 1,000,000 costs to be stored. Currently, 32 different cost functions can be used by the *RP*-algorithm, and this number will increase in the future. Note that 30 of these cost functions only incur an additional penalty for a certain type of road segments, specifically for motorways, ferries, toll roads and tunnels, or a bonus for motorways. The special cost functions that give an additional penalty to ferries, toll roads and tunnels usually do not lead to a different optimum route between pairs of boundary nodes. Therefore, the cost of the optimum route between many pairs of boundary nodes of a single cell does not change for these cost functions. For even more boundary node pairs, the optimum route itself does not change. As a result, we do not need to store the optimum route cost for every cost function. It suffices to store a few costs and for each cost an indicator of the cost function for which that cost is the optimum route cost. Alternatively, we can also store a characterization from which the optimum route cost can be computed for all pairs of boundary nodes of a single cell.

If the optimum route cost for a certain cost function is not stored, it is still possible to plan an optimum route by inserting additional cells into the searchgraph. Specifically, all cells containing at least one pair of boundary nodes for which the optimum route cost is not stored can be added to the searchgraph. Note that an internal edge of such a cell is only evaluated if the $C^\star$-algorithm with the original searchgraph had inserted a route edge of that cell into the candidate list. As a result, an optimum route can still be planned, the amount of data that needs to be stored can be limited, but the planning time may increase for some routes for some (non-standard) cost functions. Because the cells in the used multi-level partition are very small, we expect this increase in planning time to remain limited.

Another disadvantage of the $C^\star$-algorithm is that the size of the pre-processing result needed for the $C^\star$-algorithm, which consists of a partition and a number of route graphs, increases compared to the pre-processing result of the *RP*-algorithm, which consists of several HL networks. Note that people have worked on the optimization

of the storage of the HL networks for many years already. The multi-level partition that needs to be stored probably requires more storage space. Since we have not yet considered efficient storage methods for multi-level partitions, it is very hard to say how much additional data would need to be stored.

7.6 Extending Functionality of the $S^\star$-Algorithm

So far, we have assumed that after a route has been planned, it remains optimum, and that the driver follows his route exactly. So, after a route has been planned we are done. However, sometimes accidents happen that cause a severe traffic jam, which results in a change of the costs in the roadgraph. Also the driver does not always follow the instructions and may deviate from the route, or asks for an alternative route. In all these cases we are not done after the initial route has been planned: a new route may need to be planned. Current car navigation systems offer functionality to react to these changes. In this section we discuss the approach that we recommend for handling changes in all kinds of variables: *re-planning.* We first discuss re-planning in general and indicate the situations that can occur that may require the planning of a new optimum route. Subsequently, we discuss these situations one at a time.

Re-planning is a general approach that can be used to handle all situations in which a new optimum route may need to be presented. Of course a new route can be planned using the same algorithm as was used to plan the initial route. However, usually information from the initial optimum route or planning process can be used to speed up the planning of a new optimum route. For example, in order to plan an optimum route p from a start node s to a destination node d, also other optimum routes besides route p have been determined. For route $p = \langle e_1, \ldots, e_\ell \rangle$, at least the optimum routes from edge e_i to edge e_j in p with $i \leq j$ have been determined (assuming the roadgraph is (stochastically) cost consistent).

During the route planning process, a *search tree ST* is constructed containing for every selected edge e, the minimum cost from s to e and an underestimated cost from s via e to d. Note that in algorithms $C^\star$, $T^\star$ and $S^\star$ the search tree ST is given by the set $E \setminus \{e|\lambda(e) = \emptyset\}$. The search tree also contains a number of edges that have not been selected yet. These edges form the *candidate list CL* (so $CL \subseteq ST$). For every edge $e \in CL$, an upperbound of the optimum cost from s to e is given. Note that in algorithms $C^\star$, $T^\star$ and $S^\star$ the candidate list CL is given by the set $H \setminus \{e|\lambda(e) = \emptyset\}$. Throughout this chapter, we assume that a *search tree ST* containing *candidate list CL* resulting from a planning process are kept until the beginning of a next planning process. Information from the search tree and candidate list can be extremely useful in determining a new optimum route and it may greatly reduce (re-)planning times.

Re-planning is only necessary if the current route may become sub-optimal. This only happens if

1. The costs of the roadgraph change,

2. The start or destination node changes, or
3. The driver requests a different route.

Costs in a roadgraph may change due to incidental traffic jams, for example due to accidents. Alternatively, in a car navigation system usually several planning criteria are available for the driver to choose from. For example, a driver may request the fastest route to a destination. The costs of a roadgraph also change if the driver changes his planning criterion and selects the shortest route instead. Also a change in the progress of the driver compared to what was expected may result in a change in (time-dependent) costs. Section 7.6.1 discusses re-planning after a change in costs of a roadgraph.

A changing destination is an obvious cause for re-planning and is discussed in Section 7.6.2. Also a change in the start node may occur. Imagine the driver does not follow the instructions and deviates from his route. Then a new route from his changed car position to the destination is needed in order to guide the driver to the destination. Section 7.6.2 discusses how to handle deviations.

Finally, the driver may simply not be satisfied with his route and request an alternative route. Section 7.6.3 discusses the planning of alternative routes.

7.6.1 Changing Costs

This section discusses re-planning after a change in costs of a roadgraph. As discussed in Section 7.6, the main reasons for a change in costs are the occurrence of an incidental traffic jam, a change in planning characteristic, or a change in the progress of the driver compared to what was expected. We now discuss these three causes for a change in costs one at a time.

Incidental Traffic Jams

An incidental traffic jam is one of the most obvious causes of a change in costs of a roadgraph. Note that we have already taken structural traffic jams into account by introducing (stochastic) time-dependent costs in a roadgraph. Therefore, costs can only change due to incidental traffic jams, which are usually caused by traffic accidents, road works or events like the start of the summer vacation. This section presents several methods to re-determine an optimum route if costs in the roadgraph change.

Note that the cost of turns or the duration of these turns is not likely to change. Therefore, we focus on a change in the cost and/or travel time of edges, which are more likely to be influenced by incidental congestion. Note that incidental traffic jams are reported by RDS-TMC or via a connected GSM phone. These reports generally contain information on the location, length and travel speed of the traffic jam. We assume this information is sufficient to compute the change in edge costs for all edges.

First of all, a new route needs to be planned in case of an incidental traffic jam. Normally, this can be done by starting a new planning process. However, because we store the optimum route costs between boundary nodes of a single cell, these optimum route costs may also change due to the traffic jam. As a result, we may need to modify our searchgraph in order to be able to plan an optimum route. In particular, we can include every cell that contains edges whose cost or travel time has changed in the searchgraph. Note that because the route may contain time-dependent edges, a change in travel time alone may also lead to a new optimum route, even if we do not minimize the travel time.

We have seen in Section 4.4 that multi-level partitions lead to the fastest route planning process. If we need to include an additional cell into the searchgraph of a multi-level partition, this can be done similarly to including the start or destination cell into the searchgraph. Multi-level partitions usually contain small lowest-level cells. Therefore, we can include several additional cells in the searchgraph without drastically increasing its size.

Notice that if an incidental traffic jam is known to the system before the initial planning, then this is a way to plan an optimum route. Another method is to ignore this traffic jam during the initial planning, and re-plan the optimum route afterwards should the found route contain edges whose edge cost or travel time has increased due to the traffic jam.

In case of incidental traffic jams, we consider two possible situations, the traffic jam may be appear or disappear. A new traffic jam can only lead to a non-optimum route if the costs of edges of the current route increase. A disappearing traffic jam only leads to a non-optimum route if this traffic jam influenced the current optimum route. This does not mean that the disappearing traffic jam has to lead to an increased cost of at least one edge in the current route. It is also possible that an alternative route avoiding the traffic jam was optimum. As a result, we need to re-plan the route if a new traffic jam causes the costs of edges in the route to increase, or if a traffic jam disappears that causes the cost of edges in the search tree to decrease. Let $\hat{E}$ be a set of edges for which the cost or travel time changes due to a new or disappearing traffic jam. A new optimum route can be planned as follows.

1. Adapt the cost and travel time for every edge $e \in \hat{E} \cap ST$ and all its children in the search tree.
2. Insert every edge $e \in \hat{E} \cap ST$ in the candidate list.
3. Continue the planning process with the updated search tree ST and candidate list CL.

Depending on the number of changed edge costs and travel times, the re-planning process can be much faster than a standard planning process. Pallottino & Scutellà [2003] present an algorithm to update a shortest path tree in case of a change in costs.

Change of Planning Characteristic

Unlike a traffic jam, a change in planning characteristic can change the costs and travel times of all edges in the roadgraph. For example, the costs of all edges change if the driver decides to ask for the shortest instead of the fastest route. Note that updating the search tree requires a complete re-determination of all costs in the search tree, so effectively a new route planning process is started from scratch. However, we can still increase the route planning speed of this re-planning process by using the cost of the current optimum route according to our new objective as an upperbound on the new optimum route cost. Note that the cost of the current route according to the new objective may be unknown, in which case this still has to be determined. However, because the route is known this is not expected to take too much time.

After an upperbound on the optimum route cost is known, a new optimum route can be planned according to our new objective. All routes with an (expected) cost larger than the found upperbound can be ignored, because they can never lead to an optimum route. This increases the route planning speed. Note that if the found upperbound is close to the optimum route cost, many alternative routes can be ignored during the search for an optimum route.

Change in Driver Progress

Unlike the previous two causes of a change in edge costs and travel times, a change in the progress of the driver does not actually change the cost or driving time of edges, it merely changes the arrival time of the driver compared to what was expected. Basically, the travel time used in determining an optimum route does not correspond to the realized travel time of the driver. Note that our stochastic time-dependent model takes the uncertainty in travel times into account when planning an optimum route. In practice, the driver never drives exactly as fast as expected, so the progress of the driver will always be different from what was expected. However, it is not useful or appreciated by the driver to determine a new optimum route every time the driver does not arrive at the end node of an edge on time. Therefore, a criterion has to be used to determine when re-planning may be useful. Note that if the route does not contain any time-dependent edges, then a change in the progress of the driver does not effect the route cost, so the current route remains optimum. Only for routes with time-dependent edge costs, a change in the driver's progress may cause the current route to become non-optimum.

To determine if the planning of a new optimum route is worthwhile, the cost of the remainder of the current route for the changed departure time can be computed. If the resulting cost differs substantially from the old route cost, it is advisable to re-determine an optimum route, using the just computed cost as an upperbound on the optimum route cost. For a large difference between the old and the new, adjusted route cost, the probability that another route exists with a lower cost is larger.

7.6.2 Changing Location

A second cause for re-planning is a changing location. This can be either the start node of the route or the destination node. If the start node of the route changes, then the driver is no longer at a node or edge on the computed route, and a new optimum route has to be determined to guide the driver to his destination. This is called *deviation planning.* Of course the driver can also change his destination, which causes the need for a new optimum route as well. Note that in this section we assume that the start node (or car position) is the root of the search tree. Should the destination node be the root of the search tree, then the approaches described for handling deviation planning and a change of destination can be interchanged.

Deviation Planning

A very important aspect of a car navigation system is that it presents the driver a new route automatically (and almost immediately), should the driver deviate from his route. Deviation planning has to be fast because the driver has to receive guidance at all times.

For a time-dependent roadgraph, it is recommendable to plan from the start node (or car) to the destination, because this way the arrival time at the end node of an edge can be computed exactly according to the available data. A deviation in this case corresponds to a change of the root of the search tree. Nguyen, Pallottino & Scutellà [2002] describe how a shortest path tree can be updated fast in case of a root change. Although usually a search tree is not a shortest path tree, a shortest path tree namely contains the optimum route from the root node to all other nodes in the graph, their algorithms can be used to update the search tree. The resulting updated search tree can be used, together with all leaves of the search tree as candidate list, to re-start the route planning process, which results in a new optimum route from the modified start node to the destination.

It is also possible to use an iterative procedure to plan a new optimum route. We can repeatedly choose a node u on the current route with increasing Euclidean distance from the start node, and plan an optimum route from the start node to node u. This may result in a non-optimum route if node u is not the destination node, but the re-planning process will be much faster in general for nodes u close to the new start node. Note that boundary nodes are natural node candidates to choose in this case.

Changing Destination

A change in destination can be handled easily if the search tree is rooted in the start node s. If the new destination is an end node of an edge that is contained in the search tree but not in the candidate list ($d = \delta_2(e)$, $e \in ST \setminus CL$), then the search tree already contains the optimum route from the start node to the new destination node. Note that the expected cost of an edge e in the search tree concerns the expected cost from

e to the old destination node. Should the new destination node not be contained in the search tree, then we can re-start the planning process with the current search tree and candidate list after all expected costs for edges in the candidate list are updated to represent the expected cost to the new destination.

7.6.3 Alternative Route Planning

If blockades are on a driver's route that are not known to the system or if the preferences of the driver do not correspond to the used cost function, the driver may not be satisfied with the presented optimum route. The driver can then request an alternative route. In the former case, usually a distance parameter is available to let the driver indicate the length of the blockade. An alternative route then refers to a *local* alternative or local detour. In the latter case, an alternative to the entire route needs to be planned, which is also called a *global* alternative route. An alternative route is a route that is substantially different from the current route. The number of edges or route length or travel time of route parts that occur in both routes has to be limited for the route part for which an alternative has to be found. This section discusses the planning of alternative routes.

We consider an artificial increase of edge costs in the roadgraph, followed by the planning of a new optimum route, to be the most promising way to construct an alternative route. Because we use a partition to plan routes, an alternative route can be constructed for example by introducing a penalty for (particular) route edges of cells in the middle of the route part for which an alternative has to be found. Subsequently, the cost of the penalized route edges in the current route can be increased in the search tree, and the search tree can be updated to reflect these increased costs. When the route planning process is then re-started with the adjusted search tree and candidate list, the route planning process will attempt to construct a route avoiding the penalized cells. As a result, it is very likely that the found route is sufficiently different from the original route.

We have tested this approach with an infinite penalty, and the results showed that indeed alternative routes can be planned in this way that are sufficiently different from the original routes. Note that if start and destination nodes are located closely together, or if the current optimum routes passes through a relatively small number of cells, then it is advisable not to penalize entire cells but (a number of) route edges through (usually) one cell.

Using this algorithm, we can plan an alternative route before the entire original route is known in complete detail in the sense that the route may still contain route edges that need to be replaced by a sequence of internal edges of a cell. The disadvantage is that it is more difficult to ascertain whether the alternative route sufficiently differs from the original one. Especially if the alternative route and the original route contain route edges of the same cells, it may be possible that although all route edges of both routes are different, many of the internal edges of both routes coincide.

7.7 Conclusion

In this chapter, we compared the performance, and the quality of the routes planned by the $C^\star$-algorithm for planning time-independent routes with that of the VDO Dayton route planner of Siemens VDO Automotive. Since the *RP*-algorithm is an approximation algorithm, the planned routes are sub-optimal unlike the routes planned by the $C^\star$-algorithm. On average the quality of the planned 2,000 fastest and shortest routes by the *RP*-algorithm was about 8.8% above optimum for driving speeds and 4.5% for planning speeds. We have seen that the $C^\star$-algorithm using the best multi-level partition from Chapter 3, plans routes faster than the *RP*-algorithm. Furthermore, the maximum number of edges evaluated by the $C^\star$-algorithm is much lower than for the *RP*-algorithm. Specifically, for the 2,000 fastest and shortest routes planned by the $C^\star$-algorithm, the maximum number of evaluated edges is about 4.5 times the average number of evaluated edges. For the *RP*-algorithm this is about 10 times the average for fastest routes and at least 45 times for shortest routes. Thus the planning time of the $C^\star$-algorithm is not only smaller than the planning time of the *RP*-algorithm on average, also the variation in planning time of the $C^\star$-algorithm is much smaller than for the *RP*-algorithm. Furthermore, we argued that by introducing travel time profiles, we are capable of taking daily congestion into account also when this congestion is not severe enough to be reported by RDS-TMC, or via a connected GSM phone, unlike current car navigation systems. As a result, the increase and decrease in congestion can also be taken into account during route planning. Uncertainty, which is ignored by current car navigation systems, can also be taken into account by our planning algorithms. Furthermore, current navigation systems offer a lot of additional functionality such as handling deviations and planning alternative routes. We have indicated that this functionality can be integrated with the use of partitions.

8

Conclusions

Car navigation systems are being offered as a special feature of new cars for an increasing number of car-brands. A navigation system offers the driver the possibility to be guided to his destination, by means of spoken and/or visual advices. The key components of a car navigation system are positioning, route planning and guidance. This thesis focussed on the route planning functionality of a car navigation system.

A car navigation system uses road networks that may contain millions of road segments. However, a driver expects a route to be calculated quickly, within a few seconds typically. Furthermore, he expects that the system provides him an optimum route according to a particular cost function describing his preferences. Planning optimum routes fast on very large road networks still poses a significant challenge to companies developing car navigation systems. At the same time, taking information on daily congestion patterns into account is getting more important. Drivers become increasingly demanding, and expect that traffic information is not only provided, but also used for determining the best route.

We first formulated three different route planning models in Chapter 2, one for time-independent planning, one for time-dependent planning that can be used to handle daily congestion patterns, and one for stochastic time-dependent planning in which uncertainty is taken into account. We want to plan optimum routes very fast on very large road networks. Because the road network as a whole is too large to find a minimum-cost route fast, it is necessary to use some kind of pre-processing. Therefore, we developed a pre-processing method that divides the road network into

sub-networks, that are smaller and can be searched more effectively. Specifically, in Chapter 3 we developed a partitioning approach that partitions a road network into cells. We showed that the problem of determining an optimum partition is strongly *NP*-hard, and presented two approximation algorithms to determine a partition, the Merging-Algorithm and the Splitting-Algorithm. We compared these algorithms on several real-world road networks of increasing sizes, and showed that the Merging-Algorithm outperforms the Splitting-Algorithm. Then we extended our partitioning approach to so-called multi-level partitions, where individual cells in a partition are partitioned into subcells. We showed that the expected route planning time for these multi-level partitions is much shorter than for the single-level partitions.

Once a roadgraph has been partitioned, the optimum routes between the boundary nodes of single cells have to planned and their costs have to be stored using so-called route graphs. In Chapter 4 we presented three different route graph structures that can be used to store these optimum route costs. We also proved that optimum routes can be planned using a so-called searchgraph for each of these three route graph structures and presented a route planning algorithm, the $C^\star$-algorithm, that can be used for all three route graphs. We evaluated our algorithm by planning 2,000 shortest and fastest routes through different real-world road networks. We compared the number of edges evaluated by the $C^\star$-algorithm with that of the standard $A^\star$-algorithm, because the number of evaluated edges is an objective measure of the route planning speed. The results clearly showed that our $C^\star$-algorithm plans optimum routes much faster than a standard $A^\star$-algorithm. Furthermore, we determined which route graph structure leads to the least number of evaluated edges.

Subsequently, we considered planning optimum time-dependent routes in Chapter 5. Time-dependencies can be used to plan routes that take daily congestion patterns into account. We evaluated the conditions under which time-dependent optimum routes can be planned using a modified standard $A^\star$-algorithm. These conditions are called consistency conditions. We assumed that these consistency conditions are satisfied for two reasons. First, in practical situations these conditions are usually satisfied for planning fastest or shortest routes. Secondly, the planning of optimum routes in the situation that these conditions are not satisfied results in undesirable routes. We presented the $T^\star$-algorithm that can be used to plan optimum time-dependent routes for consistent road networks. Tests with this algorithm using real-world data on time-dependent travel times showed the value of using time-dependencies for planning optimum time-dependent routes. We also investigated how time-dependent route planning can be combined with a partitioned road network. The time-dependent travel times can best be stored as travel time profiles, in order to keep the required storage space small. We presented an approximation algorithm, called X-MATCH, and showed that using this algorithm the number of travel time profiles can be drastically reduced while an upperbound on the deviation from the optimum

time-dependent costs of the planned routes can be guaranteed. Thus we showed that high-quality time-dependent routes can be planned fast on very large real-world road networks, requiring only limited storage space.

Since travel times are uncertain, we investigated planning stochastic time-dependent routes in Chapter 6. We presented conditions under which a modified $A^\star$-algorithm can be used to plan optimum stochastic time-dependent routes. These conditions are called stochastic consistency conditions. In realistic situations, these conditions were shown to be satisfied for planning fastest and shortest routes. The optimum routes in the situation that the road network is not stochastically consistent are not desirable from the driver's point of view. Therefore we assumed that the road network is stochastically consistent. We presented the $S^\star$-algorithm that can be used to plan optimum stochastic time-dependent routes if the road network is stochastically consistent. The usefulness of planning stochastic time-dependent routes was investigated by planning 2,000 stochastic time-dependent fastest routes through the Netherlands based on real-world data on the expectation and standard deviation of travel speeds. We showed that incorporating uncertainty about travel times in route planning is only useful if the driver requests a high certainty regarding his arrival time. Again, we investigated the combination of stochastic time-dependent planning with the use of partitioned road networks. In order to combine planning stochastic time-dependent routes with the use of partitions, we need to store profiles of the expected travel times, as well as profiles of the standard deviation (or uncertainty) of the travel times. We showed that the number of profiles needed to store the necessary data can be largely reduced by approximating the driving time and standard deviation profiles. We also showed that with these profiles, we can still plan high-quality stochastic time-dependent routes for drivers with different attitudes towards uncertainty. So we can also plan high-quality stochastic time-dependent routes fast on very large real-world road networks, requiring only limited storage space.

Car navigation systems that are currently on the market plan routes between start and destination nodes reasonably fast, but they are not able to plan time-dependent or stochastic time-dependent routes. So in Chapter 7, we compared the performance of the $C^\star$-algorithm and the quality of the planned routes with the performance and route quality of the VDO Dayton route planner of Siemens VDO Automotive, called the *RP*-algorithm. From this comparison, we saw that unlike our $C^\star$-algorithm, the *RP*-algorithm does not plan optimum routes. The $C^\star$-algorithm also evaluates fewer edges than the *RP*-algorithm on average and especially in worst-case. We discussed the main differences between the two algorithms and their advantages and disadvantages. The main advantages of the $C^\star$-algorithm are that it is faster, plans optimum routes, is less dependent of the map supplier and less time is spent on waiting for data to be read from the CD or DVD. The main disadvantages are the possible increase in data that needs to be stored for the partition, and that for each cost function that

leads to a different optimum route between boundary nodes of single cells, the cost of the optimum route needs to be stored. We also indicated how the most important route planning functionality besides the planning of a route between a start and destination node can be performed efficiently. We concluded that most likely all current and future route planning functionality of car navigation systems can be integrated with using partitions.

A

Determining the Average Number of Edges in a Searchgraph

In this appendix we derive the average number of edges $AE(G, C_1, \ldots, C_k)$ in a searchgraph G_S of a single-level partition $\{C_1, \ldots, C_k\}$. To mathematically formulate this criterion, we use the notation in Table A.1.

k	The number of cells in the partition of G, $\{C_1, \ldots, C_k\}$.
n	The number of nodes of roadgraph G.
n_i	The number of nodes in cell C_i.
m_i	The number of edges in cell C_i.
r_i	The number of route edges of cell C_i.
m_B	The number of boundary edges in the partition.

Table A.1. *Notation used for defining the quality of a partition.*

The average number of edges in a searchgraph G_S, see also Section 3.2 and Definition 3.7, is equal to the number of edges in the boundary graph, plus the average number of route edges in all cells except the cells containing the start and destination node for all possible start and destination node pairs, plus the average number of edges in the cells that contain the start and destination node for all possible start and destination node pairs.

We assume that every node has an equal probability of being selected as a start node or destination node. This means that the probability of choosing a node in a

cell C_i is equal to $\frac{n_i}{n}$. First, we select a start node s at random. This node is part of cell C_i with probability $\frac{n_i}{n}$. Thus the average number of internal edges of the start cell contained in the searchgraph is equal to $\sum_{i=1}^{k} \frac{n_i}{n} m_i$. Similarly the average number of internal edges in the cell containing the destination node is equal to $\sum_{i=1}^{k} \frac{n_i}{n} m_i$. However, if the start node and destination node are located in the same cell C_i, which occurs with probability $(\frac{n_i}{n})^2$, then the internal edges in this cell should be added to average number of internal edges in the searchgraph only for the start node or only for the destination node. This leads to an average number of internal edges in the searchgraph equal to $\sum_{i=1}^{k} \frac{n_i}{n} m_i + \sum_{i=1}^{k} \frac{n_i}{n} m_i - \sum_{i=1}^{k} (\frac{n_i}{n})^2 m_i = \sum_{i=1}^{k} \frac{n_i}{n} (2 - \frac{n_i}{n}) m_i$. If we select a start node s in cell C_i then the route edges of all cells except cell C_i may need to be added to the searchgraph. Therefore, the route edges of cell C_i are added to the searchgraph if we do not select a node in cell C_i as start or destination node, which occurs with probability $(1 - \frac{n_i}{n})^2$. Therefore, the average number of route edges in the searchgraph is given by $\sum_{i=1}^{k} (1 - \frac{n_i}{n})^2 r_i$. Finally, all boundary edges m_B are part of the searchgraph for any selected start and destination node. As a result, the average number of edges in the searchgraph, $AE(G, C_1, \ldots, C_k)$, is equal to

$$AE(G, C_1, \ldots, C_k) = \sum_{i=1}^{k} \left\{ \frac{n_i}{n} (2 - \frac{n_i}{n}) m_i + (1 - \frac{n_i}{n})^2 r_i \right\} + m_B.$$

B

Moving a Node between Cells

In this appendix, we give the details of moving a boundary node from a cell to an adjacent cell, which is done very frequently by the Splitting-Algorithm described in Section 3.4.1. We assume a partition $\{C_1,\ldots,C_k\}$ of roadgraph G is given, with boundary graph $B = (N_B, E_B, w_e, w_r)$. A cell $C_i = (N_i, E_i, w_e, w_r)$ contains a set of boundary nodes $N_B(C_i)$. Note that we have $k \geq 2$. We define

$E_B(C_i) = \{e \in E_B | \delta_1(e) \in C_i \vee \delta_2(e) \in C_i\}$,
$\varepsilon(u,G) = \{e \in E(G) \mid (\delta_1(e) = u \wedge \delta_2(e) \neq u) \vee (\delta_2(e) = u \wedge \delta_1(e) \neq u)\}$,
$\gamma(u,G) = \{e \in E(G) \mid (\delta_1(e) = u \wedge \delta_2(e) = u)\}$,
$\eta(u,G) = \{\delta_1(e), \delta_2(e) \mid \delta_1(e) = u \vee \delta_2(e) = u, e \in E(G)\} \setminus \{u\}$,
$W_j(u) = \{w \in N_j \mid \eta(w,B) = \{u\}\}$,
$X_{ij}(u) = \begin{cases} \{u\} & \text{if } |\varepsilon(u,C_i)| > 0 \vee \eta(u,B) \not\subseteq N_j; \\ \emptyset & \text{otherwise.} \end{cases}$

Without loss of generality, we assume we move a node $u \in N_B(C_1)$ to cell C_2 with $\varepsilon(u,B) \cap E_B(C_2) \neq \emptyset$.

Lemma B.1. *Moving node $u \in N_B(C_1)$ to cell C_2 of partition $\{C_1, C_2, \ldots, C_k\}$ with $\varepsilon(u,B) \cap E_B(C_2) \neq \emptyset$ results in partition $\{\tilde{C}_1, \tilde{C}_2, C_3, \ldots, C_k\}$ with boundary graph $\tilde{B} = (\tilde{N}_B, \tilde{E}_B)$ with*

$\tilde{N}_1 = N_1 \setminus \{u\}$,
$\tilde{N}_2 = N_2 \cup \{u\}$,
$\tilde{E}_1 = (E_1 \setminus \gamma(u,C_1)) \setminus \varepsilon(u,C_1)$,

$\tilde{E}_2 = E_2 \cup \gamma(u,C_1) \cup (\varepsilon(u,B) \cap E_B(C_2))$,
$\tilde{E}_B = (E_B \cup \varepsilon(u,C_1)) \setminus (\varepsilon(u,B) \cap E_B(C_2))$,
$N_B(\tilde{C}_1) = (N_B(C_1) \setminus \{u\}) \cup \eta(u,C_1)$,
$N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.

Proof. Choose an edge $e \in E_B(C_2) \cap \varepsilon(u,B)$. Let edge e be an edge connecting node u and v. It follows that $v \in N_B(C_2)$. The situation is illustrated by Figure 3.9. Only cells C_1 and C_2 and boundary graph B can change due to moving node u from cell C_1 to C_2. Moving node u from cell C_1 to C_2 leads to the creation of cells $\tilde{C}_1$ and $\tilde{C}_2$. Moving node u to C_2 will lead to node u no longer being part of C_1. Instead, node u is now part of $\tilde{C}_2$. By assumption, no other nodes are moved, so $\tilde{N}_1 = N_1 \setminus \{u\}$ and $\tilde{N}_2 = N_2 \cup \{u\}$.

Only adjacent edges of node u can be part of a different cell before and after node u has been moved. An edge not adjacent to u between nodes $w_1 \in N$ and $w_2 \in N$ remains in the same cell before and after node u has been moved, because both node w_1 and node w_2 remain in the same cell of the partition. In order to determine the edge sets of cells $\tilde{C}_1 = (\tilde{N}_1, \tilde{E}_1)$ and $\tilde{C}_2 = (\tilde{N}_2, \tilde{E}_2)$ and the new boundary edge set, only adjacent edges of node u have to be considered. The collection of adjacent edges of u is given by $\varepsilon(u,C_1) \cup \gamma(u,C_1) \cup (\varepsilon(u,B) \cap E_B(C_2)) \cup (\varepsilon(u,B) \setminus E_B(C_2))$.

First we consider the set of edges from u to u, $\gamma(u,C_1)$. After node u has been moved to C_2, all edges from u to u obviously become part of $\tilde{C}_2$. Secondly, edges between node u and a node $w \neq u$ in C_1, $\varepsilon(u,C_1)$, are part of C_1 before u has been moved to C_2. After the move, these edges are edges between node $u \in \tilde{N}_2$ and a node $w \neq u$ in $\tilde{C}_1$. Thus these edges connect nodes from different cells of the partition and therefore they become boundary edges and are not part of $\tilde{C}_1$. Thirdly, edges between node $u \in N_1$ and a node $w \in N_2$, $\varepsilon(u,B) \cap E_B(C_2)$, are edges between nodes in different cells of the partition and are therefore part of the boundary graph before node u has been moved to C_2. After node u has been moved to C_2 these edges are edges between two nodes u and say w in $\tilde{C}_2$. It follows that these edges are no longer boundary edges, because they connect two nodes in the same cell $\tilde{C}_2$. Instead they become edges of cell $\tilde{C}_2$. Finally, we consider edges between a node $u \in N_1$ and a node w in C_ℓ with $\ell \in \{1,\ldots,k\} \setminus \{1,2\}$. These edges are part of the boundary graph before node u has been moved to C_2 because they connect two nodes of different cells of the partition, specifically between cell C_1 and a cell C_ℓ. After node u has been moved to C_2 these edges connect nodes of cell $\tilde{C}_2$ and a cell C_ℓ so they still connect nodes of two different cells of the partition. Consequently, these edges remain in the boundary graph.

Thus we can conclude that the edge sets of $\tilde{C}_1$ and $\tilde{C}_2$ in the new partition are given by $\tilde{E}_2 = E_2 \cup \gamma(u,C_1) \cup (\varepsilon(u,B) \cap E_B(C_2))$ and $\tilde{E}_1 = (E_1 \setminus \gamma(u,C_1)) \setminus \varepsilon(u,C_1)$. The edge sets of other cells in the partition remain the same. The new edge set of the

boundary graph is given by $\tilde{E}_B = (E_B \cup \varepsilon(u,C_1)) \setminus (\varepsilon(u,B) \cap E_B(C_2))$.

What remains to be determined is the new boundary node set of the partition. By assumption, node u was part of $N_B(C_1)$ before it was moved to C_2, so after the move of node u it will no longer be part of $N_B(\tilde{C}_1)$. As noted before edges in $\varepsilon(u,C_1)$ become boundary edges between node $u \in \tilde{N}_2$ and a node w in $\tilde{C}_1$. As a consequence the adjacent nodes of u in C_1, $\eta(u,C_1)$, are connected by edges in $\varepsilon(u,C_1)$ to node $u \in \tilde{N}_2$ after node u has been moved to C_2. These nodes thus become part of $N_B(\tilde{C}_1)$. This leads to $N_B(\tilde{C}_1) = (N_B(C_1) \setminus \{u\}) \cup \eta(u,C_1)$.

In order to determine the collection of boundary nodes $N_B(\tilde{C}_2)$ of graph $\tilde{C}_2$ we need to distinguish five cases, in which we consider changes concerning the adjacent nodes of u.

1. $|\varepsilon(u,C_1)| = 0$.
   (a) $|\varepsilon(u,C_1)| = 0 \wedge \eta(u,B) \subseteq N_2$.
   The adjacent nodes of u are given by $\eta(u,C_1) \cup \eta(u,B) = \eta(u,B) \subseteq N_2$. Therefore after node u has been moved to C_2, node u is no longer a boundary node because it has no adjacent nodes outside $\tilde{C}_2$. The nodes in $W_2(u) = \{w \in N_2 | \eta(w,B) = \{u\}\}$ are, in the boundary graph, only connected to u. Because $W_2(u) \subseteq N_2$ and $u \in \tilde{N}_2$ after moving node u, the nodes of $W_2(u)$ are no longer part of the boundary graph. Using that $X_{12}(u) = \emptyset$, it follows that $N_B(\tilde{C}_2) = N_B(C_2) \setminus W_2(u) = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.
   (b) $|\varepsilon(u,C_1)| = 0 \wedge \eta(u,B) \not\subseteq N_2$.
   There exists an adjacent boundary node $w \notin N_2 \cup N_1$ of u. So after node u has been moved to C_2, there exists an edge connecting node $u \in \tilde{N}_2$ to this node $w \notin \tilde{N}_2$ so node u is a boundary node of $\tilde{C}_2$. The nodes in $W_2(u) = \{w \in N_2 | \eta(w,B) = \{u\}\}$ are, in the boundary graph, only connected to u. Because $W_2(u) \subseteq N_2$ and $u \in \tilde{N}_2$ after moving node u, the nodes of $W_2(u)$ are no longer part of the boundary graph. Using that $X_{12}(u) = \{u\}$, it follows that $N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup \{u\} = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.
2. $|\varepsilon(u,C_1)| > 0$.
   Unlike case 1, node u now has adjacent nodes in C_1. When moving node u to C_2 this means that node u has adjacent nodes in another cell of the partition and therefore node u becomes a boundary node of $\tilde{C}_2$.
   (a) $|\varepsilon(u,C_1)| > 0 \wedge |\varepsilon(u,B)| = 1$.
   Node u has only one adjacent boundary node, node v. We only have to consider if v still has adjacent nodes outside $\tilde{C}_2$ after node u has been moved to C_2.
      i. $|\varepsilon(u,C_1)| > 0 \wedge |\varepsilon(u,B)| = 1 \wedge |\eta(v,B)| > 1$.

Because $|\eta(v,B)| > 1$, v still has adjacent nodes outside $\tilde{C}_2$, so v is still a boundary node. This leads to $N_B(\tilde{C}_2) = N_B(C_2) \cup \{u\}$. Because $X_{12}(u) = \{u\}$ and $W_2(u) = \emptyset$ it follows that $N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.

ii. $|\varepsilon(u,C_1)| > 0 \wedge |\varepsilon(u,B)| = 1 \wedge |\eta(v,B)| = 1$.
Because $|\eta(v,B)| = 1$, v no longer has adjacent nodes outside $\tilde{C}_2$, so v is no longer a boundary node. This leads to $N_B(\tilde{C}_2) = (N_B(C_2) \setminus \{v\}) \cup \{u\}$. Because $X_{12}(u) = \{u\}$ and $W_2(u) = \{v\}$ it follows that $N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.

(b) $|\varepsilon(u,C_1)| > 0 \wedge |\eta(u,B)| > 1$.
The nodes in $W_2(u) = \{w \in V_2 | \eta(w,B) = \{u\}\}$ are, in the boundary graph, only connected to u. Because $W_2(u) \subseteq N_2$ and $u \in \tilde{N}_2$ after moving node u, the nodes of $W_2(u)$ are no longer part of the boundary graph. Using that $X_{12}(u) = \{u\}$, it follows that $N_B(\tilde{C}_2) = (N_B(C_2) \setminus W_2(u)) \cup \{u\} = (N_B(C_2) \setminus W_2(u)) \cup X_{12}(u)$.

□

C

Partitions

In this appendix we show the single-level partitions created by the Splitting- and Merging-Algorithm using five runs and the values of α from Table 3.4 for the road networks of Sophia, Eindhoven, Siegen, Giessen/Wetzlar, Antwerp and the Netherlands, see also Section 3.4. Each adjacent cell in a partition has been given a different color, and boundary edges are displayed in red. Route edges are not displayed.

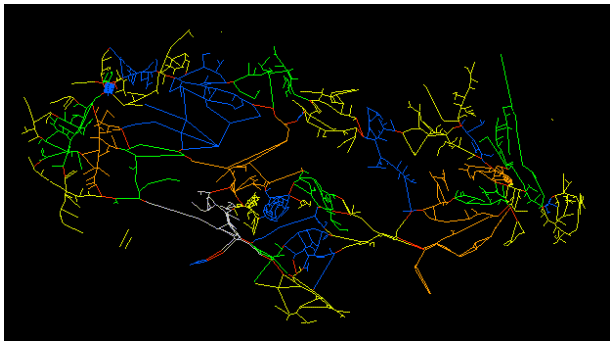


Figure C.1. *Partition of Sophia with the Splitting-Algorithm, for* r_i^C.

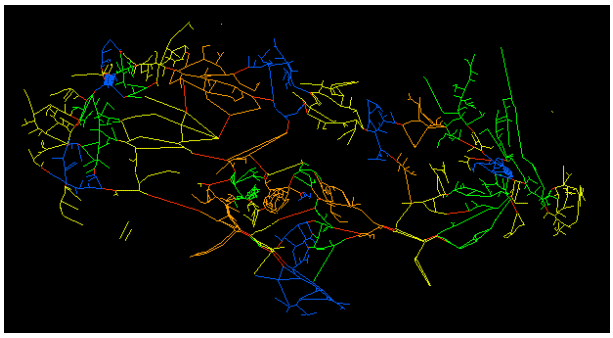


Figure C.2. *Partition of Sophia with the Splitting-Algorithm, for* r_i^S.

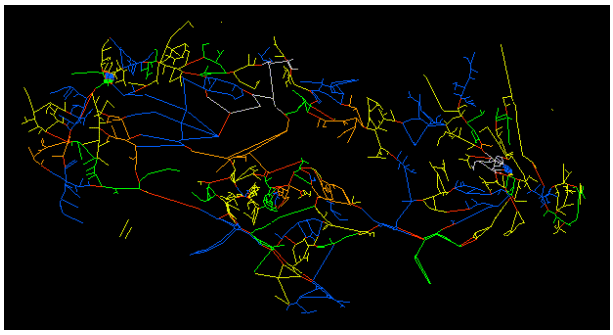


Figure C.3. *Partition of Sophia with the Splitting-Algorithm, for* r_i^P.

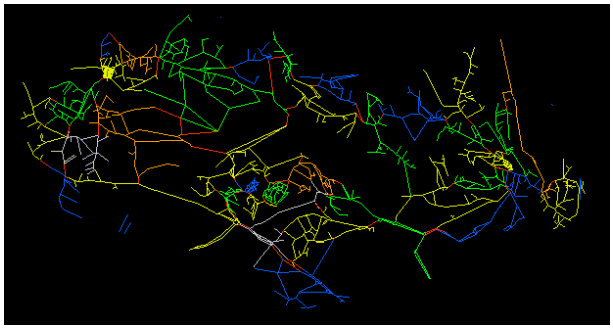


Figure C.4. *Partition of Sophia with the Merging-Algorithm, for* r_i^C.

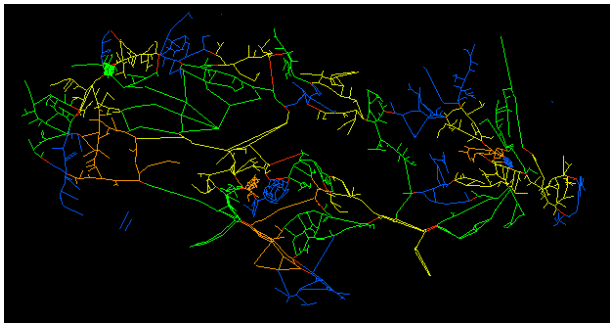


Figure C.5. *Partition of Sophia with the Merging-Algorithm, for* r_i^S.

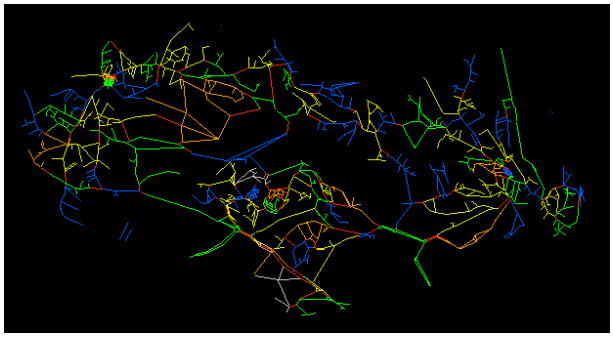


Figure C.6. *Partition of Sophia with the Merging-Algorithm, for* r_i^P.

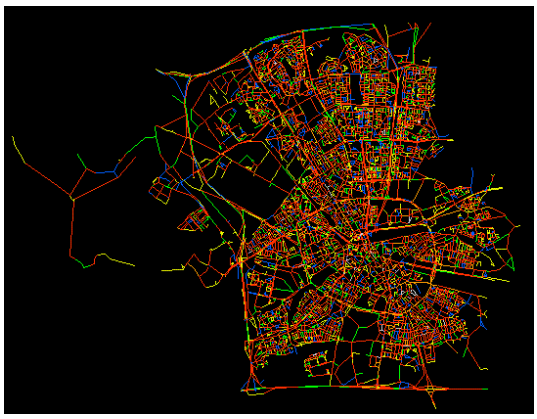


Figure C.7. *Partition of Eindhoven with the Splitting-Algorithm, for* r_i^C.

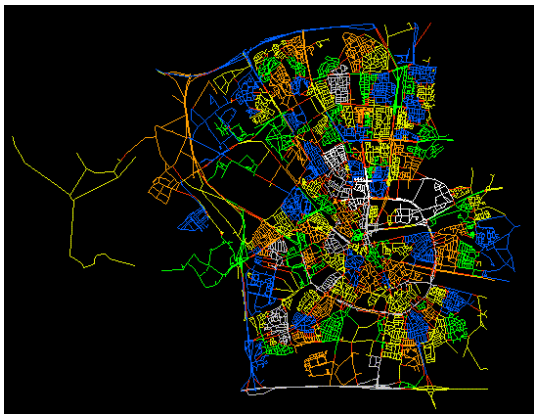


Figure C.8. *Partition of Eindhoven with the Splitting-Algorithm, for* r_i^S.

Figure C.9. *Partition of Eindhoven with the Splitting-Algorithm, for* r_i^P.

Figure C.10. *Partition of Eindhoven with the Merging-Algorithm, for* r_i^C.

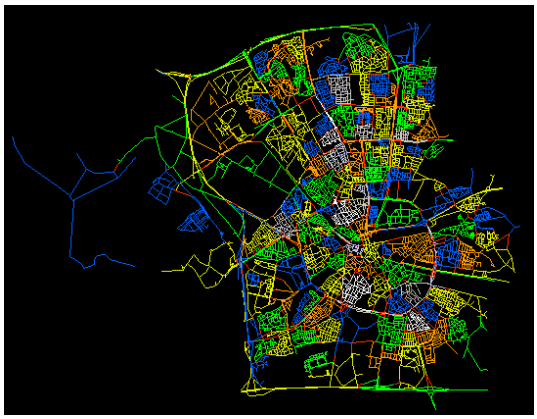


Figure C.11. *Partition of Eindhoven with the Merging-Algorithm, for* r_i^S.

Figure C.12. *Partition of Eindhoven with the Merging-Algorithm, for* r_i^P.

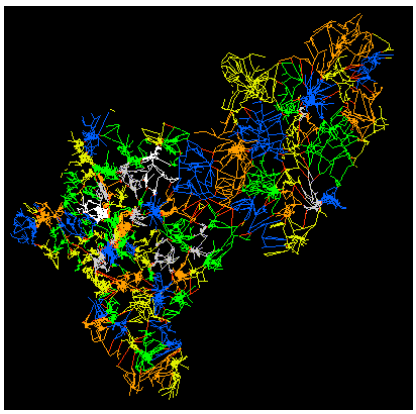


Figure C.13. *Partition of Siegen with the Splitting-Algorithm, for* r_i^C.

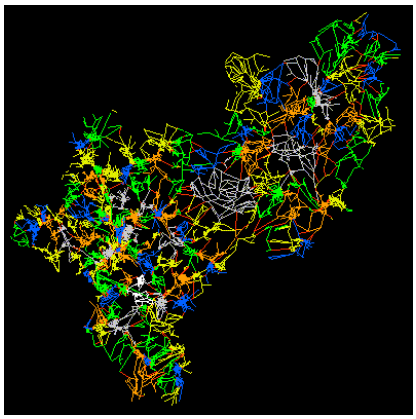


Figure C.14. *Partition of Siegen with the Splitting-Algorithm, for* r_i^S.

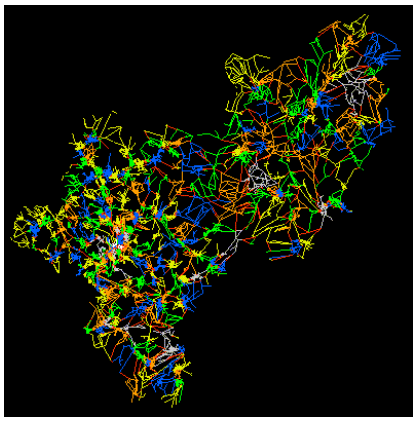


Figure C.15. *Partition of Siegen with the Splitting-Algorithm, for* r_i^P.

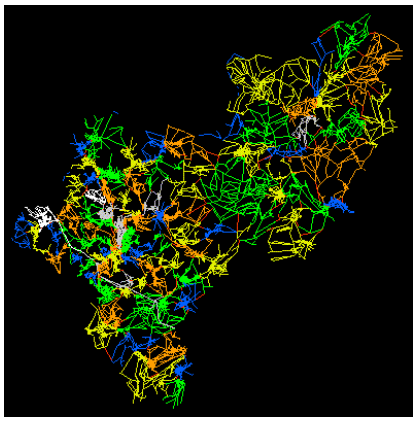


Figure C.16. *Partition of Siegen with the Merging-Algorithm, for* r_i^C.

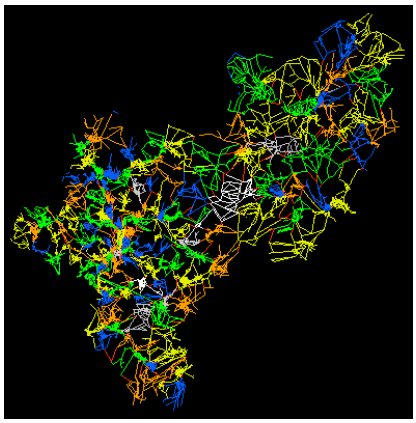


Figure C.17. *Partition of Siegen with the Merging-Algorithm, for* r_i^S.

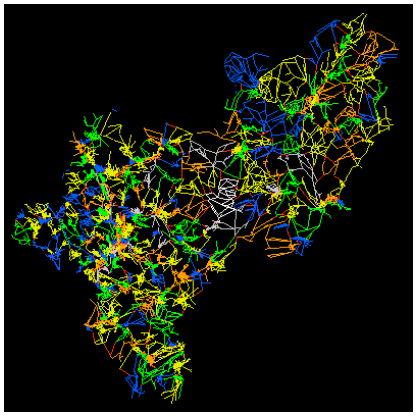


Figure C.18. *Partition of Siegen with the Merging-Algorithm, for* r_i^P.

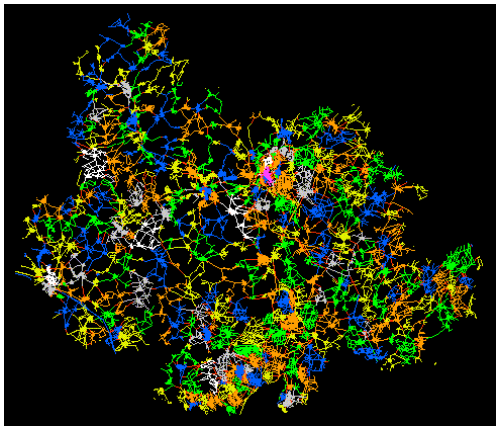


Figure C.19. *Partition of Giessen/Wetzlar with the Splitting-Algorithm, for* r_i^C.

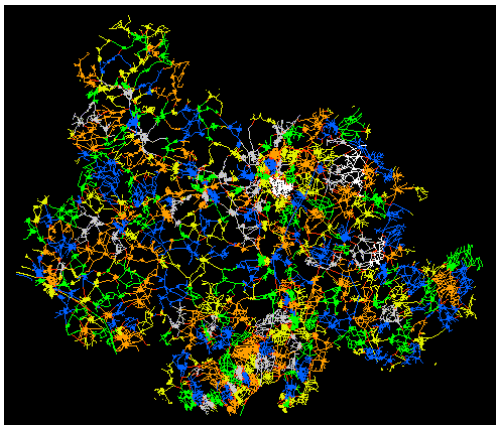


Figure C.20. *Partition of Giessen/Wetzlar with the Splitting-Algorithm, for* r_i^S.

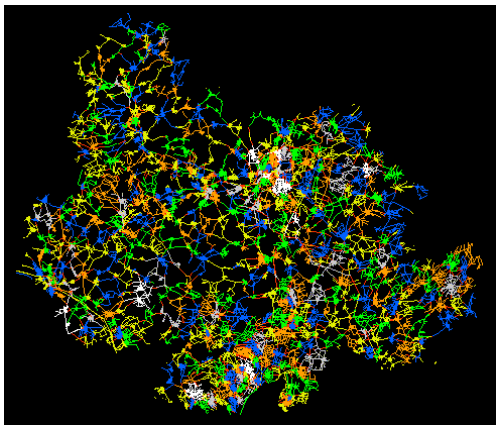


Figure C.21. *Partition of Giessen/Wetzlar with the Splitting-Algorithm, for* r_i^P.

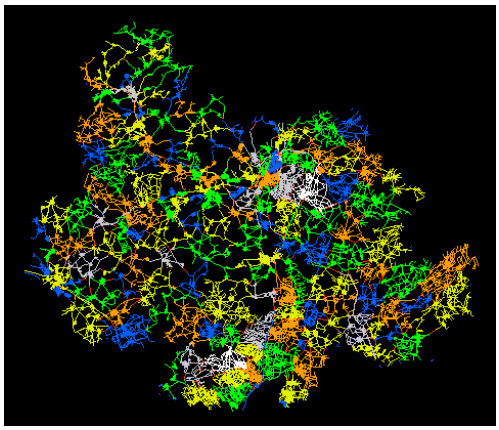


Figure C.22. *Partition of Giessen/Wetzlar with the Merging-Algorithm, for* r_i^C.

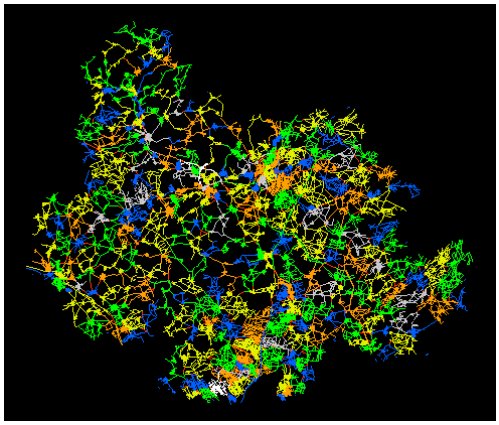


Figure C.23. *Partition of Giessen/Wetzlar with the Merging-Algorithm, for* r_i^S.

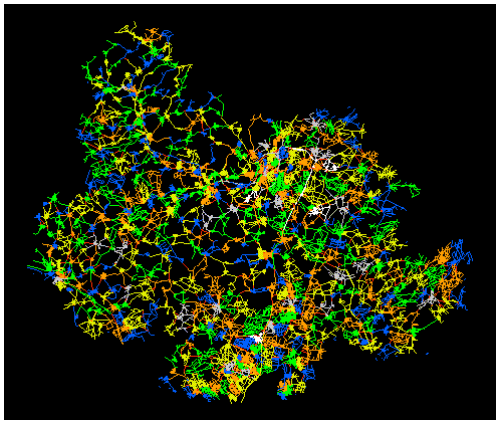


Figure C.24. *Partition of Giessen/Wetzlar with the Merging-Algorithm, for* r_i^P.

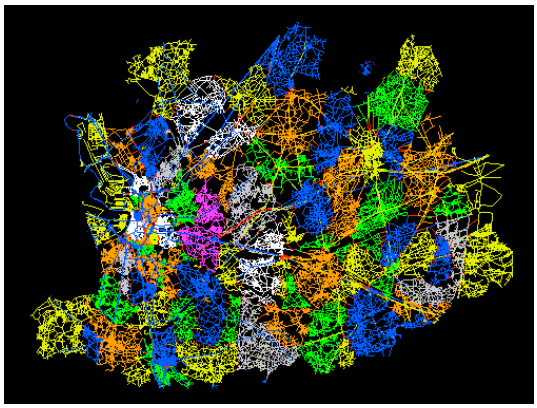


Figure C.25. *Partition of Antwerp with the Splitting-Algorithm, for* r_i^C.

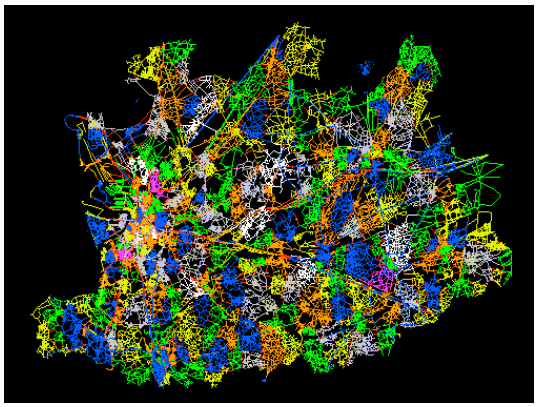


Figure C.26. *Partition of Antwerp with the Splitting-Algorithm, for* r_i^S.

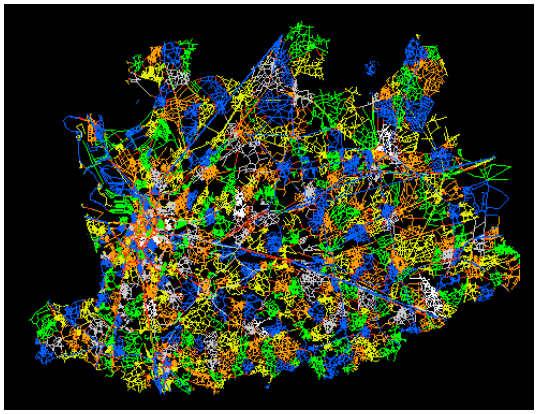


Figure C.27. *Partition of Antwerp with the Splitting-Algorithm, for* r_i^P.

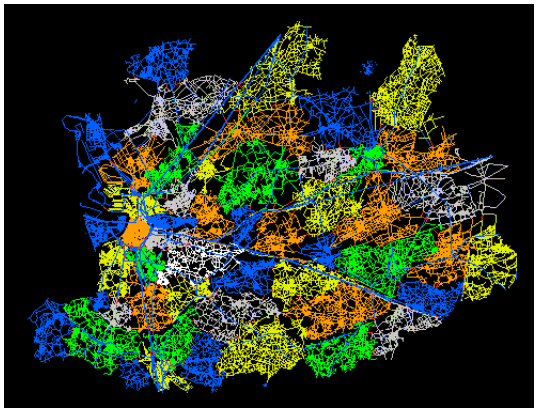


Figure C.28. *Partition of Antwerp with the Merging-Algorithm, for* r_i^C.

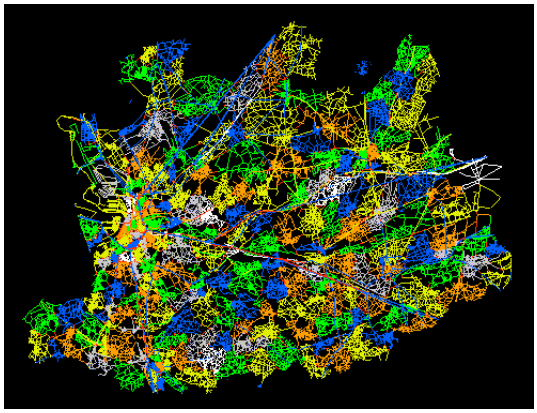


Figure C.29. *Partition of Antwerp with the Merging-Algorithm, for* r_i^S.

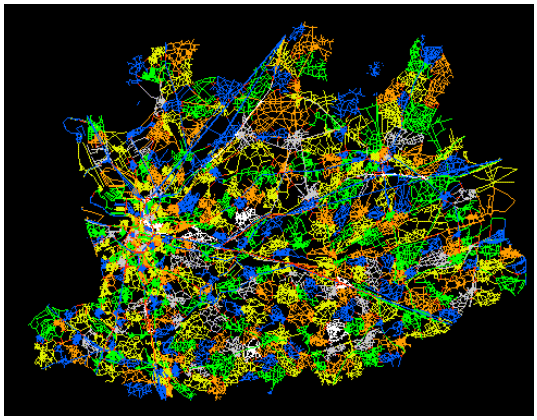


Figure C.30. *Partition of Antwerp with the Merging-Algorithm, for* r_i^P.

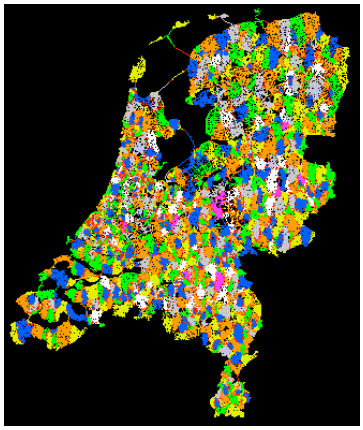


Figure C.31. *Partition of the Netherlands with the Splitting-Algorithm, for* r_i^C.

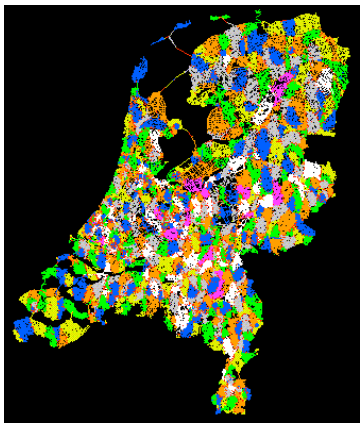


Figure C.32. *Partition of the Netherlands with the Splitting-Algorithm, for* r_i^S.

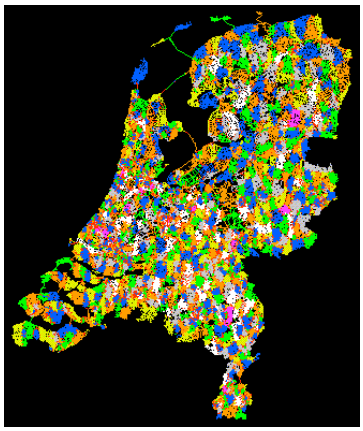


Figure C.33. *Partition of the Netherlands with the Splitting-Algorithm, for* r_i^P.

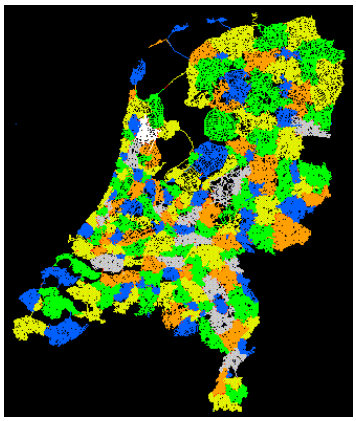


Figure C.34. *Partition of the Netherlands with the Merging-Algorithm, for* r_i^C.

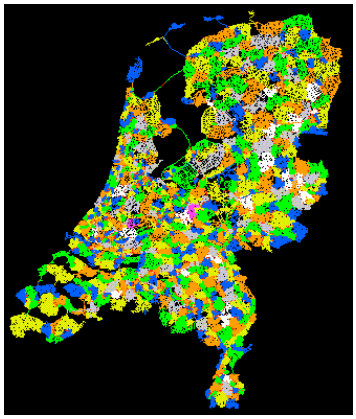


Figure C.35. *Partition of the Netherlands with the Merging-Algorithm, for* r_i^S.

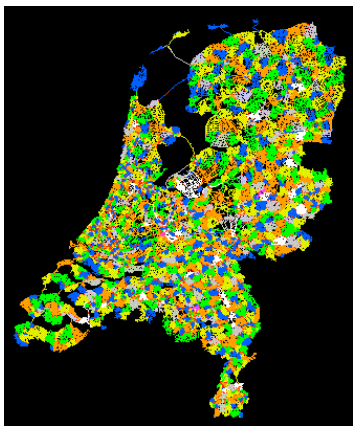


Figure C.36. *Partition of the Netherlands with the Merging-Algorithm, for* r_i^P.

D

Stochastic versus Deterministic Time-Dependent Routes

In this appendix we show the results of the comparison between stochastic time-dependent and deterministic time-dependent route planning for different parameter values of γ and ς with $\beta = 0$, see also Section 6.4.

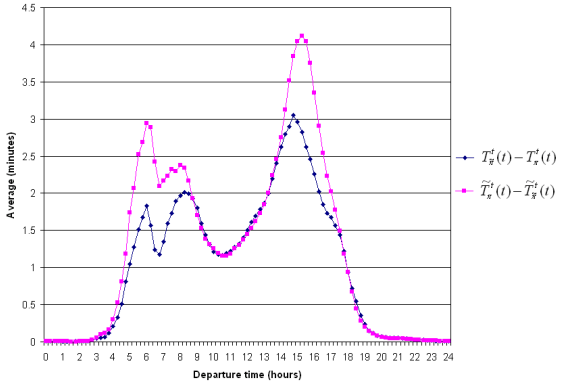


Figure D.1. *Average of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$.

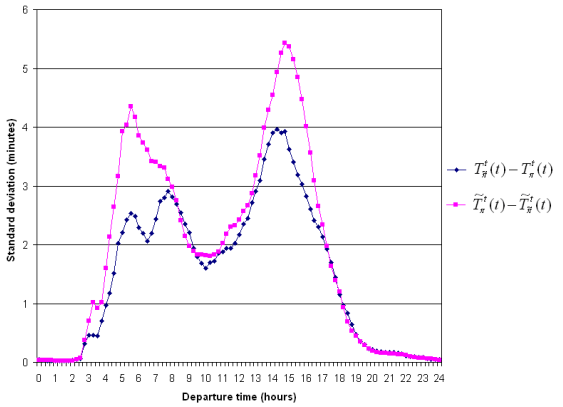


Figure D.2. *Standard deviation of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$.

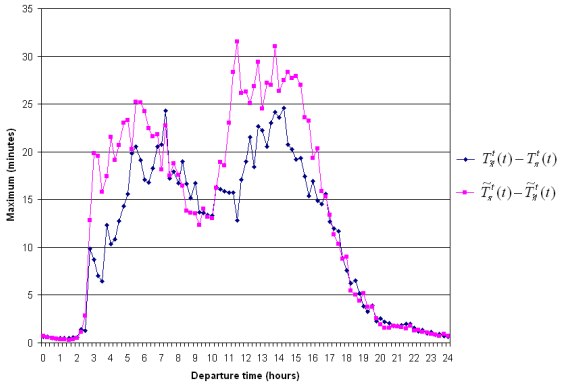


Figure D.3. *Maximum of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$.

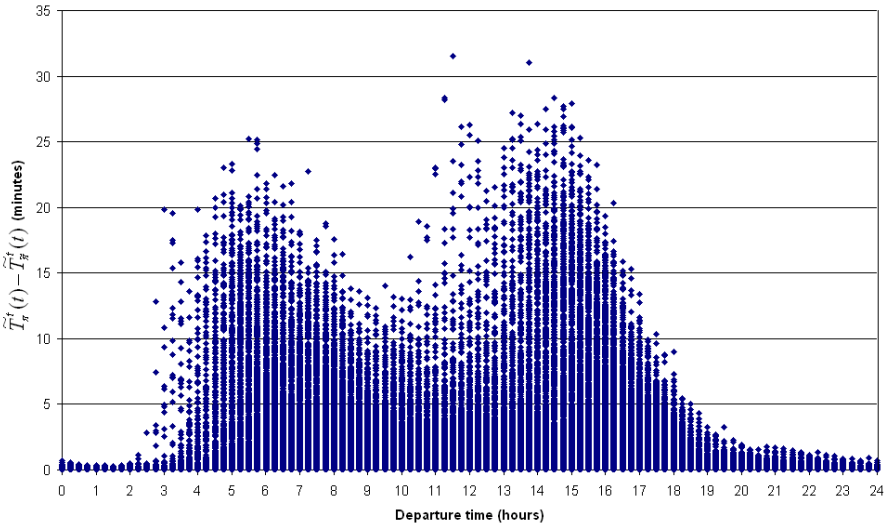


Figure D.4. *All realizations of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$.

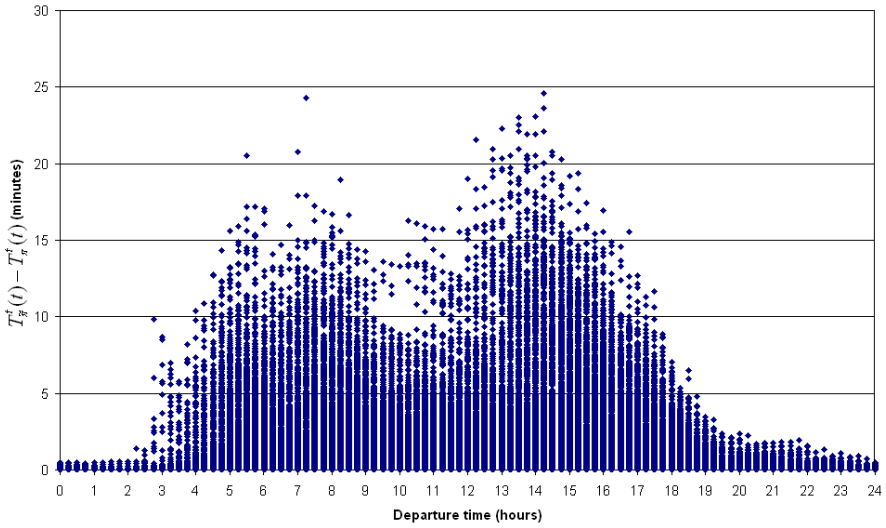


Figure D.5. *All realizations of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$.

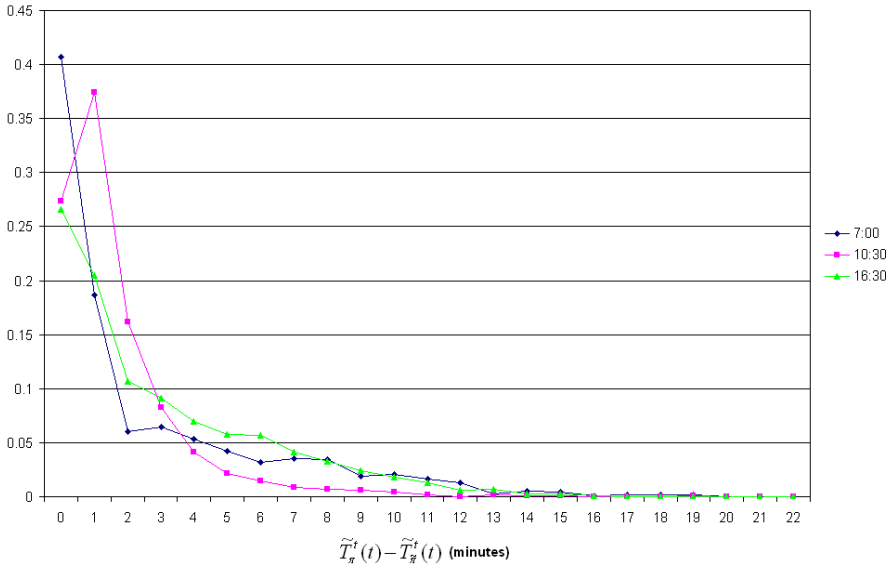


Figure D.6. *Distribution of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$ *at 7:00h, 10:30h, 16:30h.*

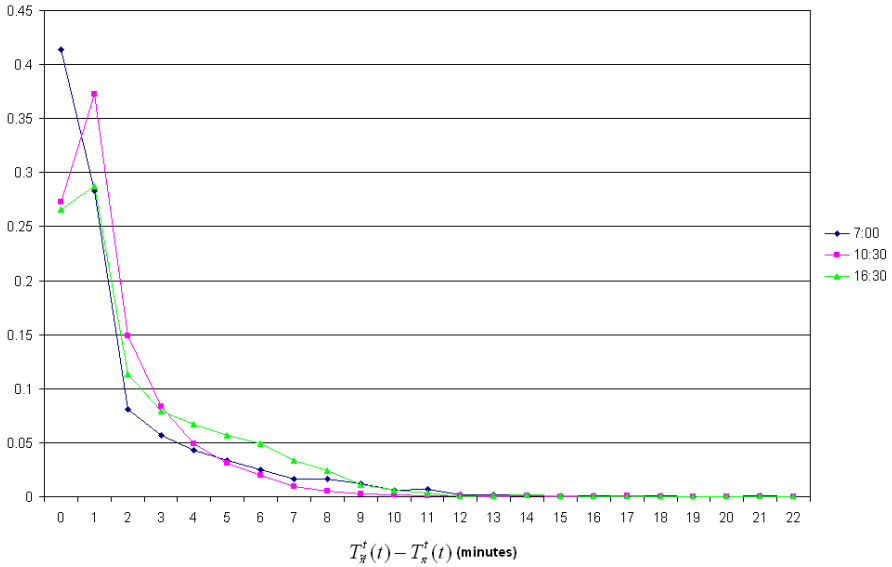


Figure D.7. *Distribution of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 1\frac{1}{2}$ *at 7:00h, 10:30h, 16:30h.*

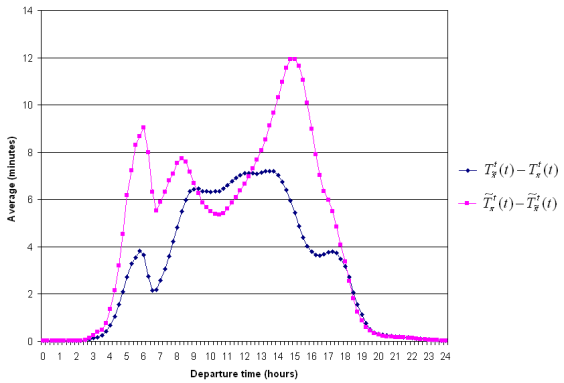


Figure D.8. *Average of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 2$, $\varsigma = 1\frac{1}{2}$.

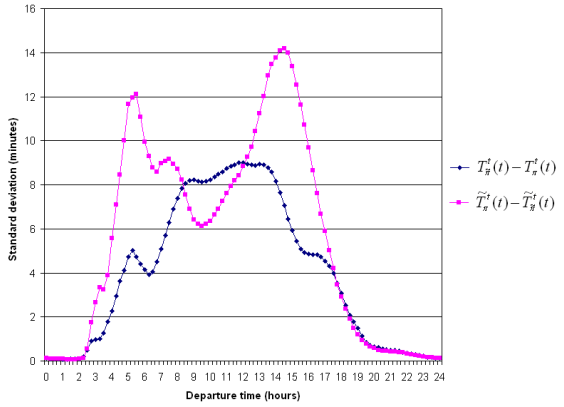


Figure D.9. *Standard deviation of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 2$, $\varsigma = 1\frac{1}{2}$.

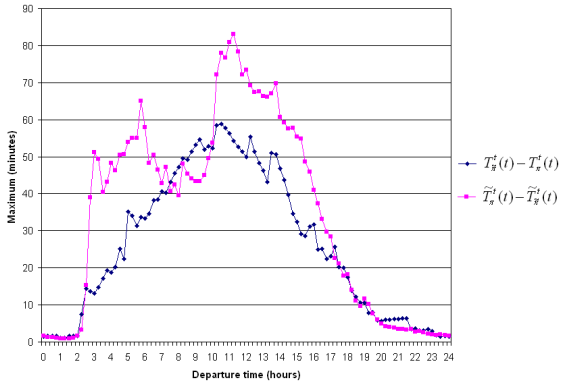


Figure D.10. *Maximum of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 2$, $\varsigma = 1\frac{1}{2}$.

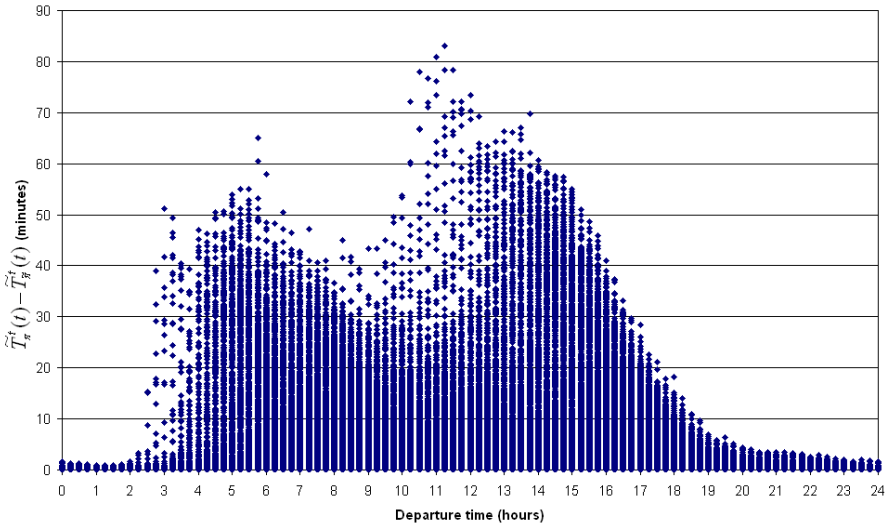


Figure D.11. *All realizations of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 2, \varsigma = 1\frac{1}{2}$.

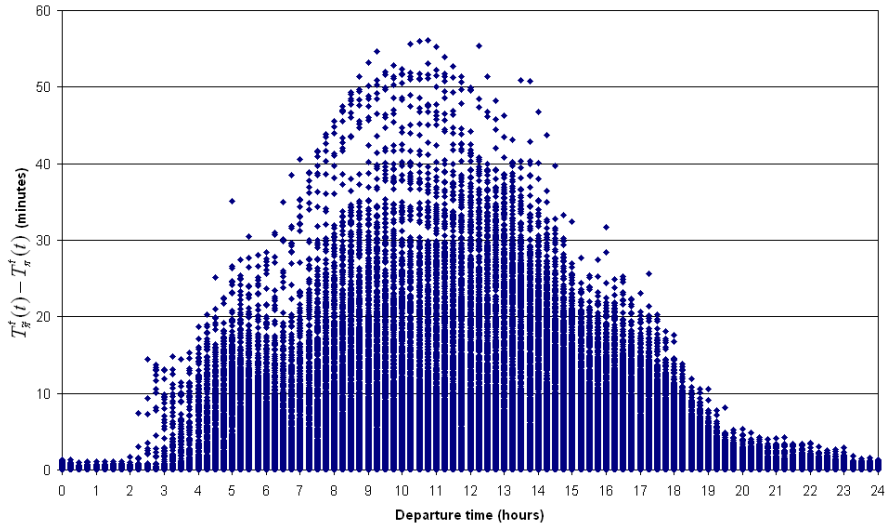


Figure D.12. *All realizations of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 2, \varsigma = 1\frac{1}{2}$.

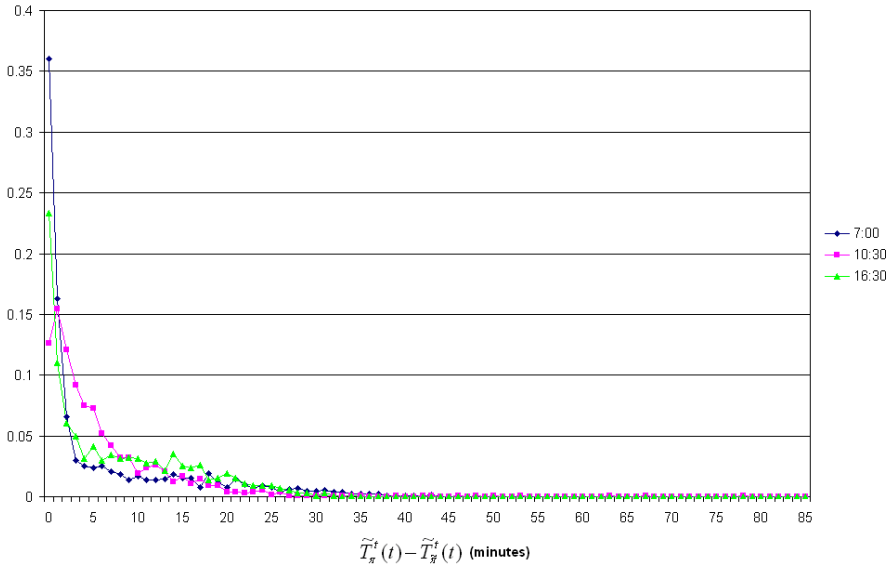


Figure D.13. *Distribution of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 2, \varsigma = 1\frac{1}{2}$ *at 7:00h, 10:30h, 16:30h.*

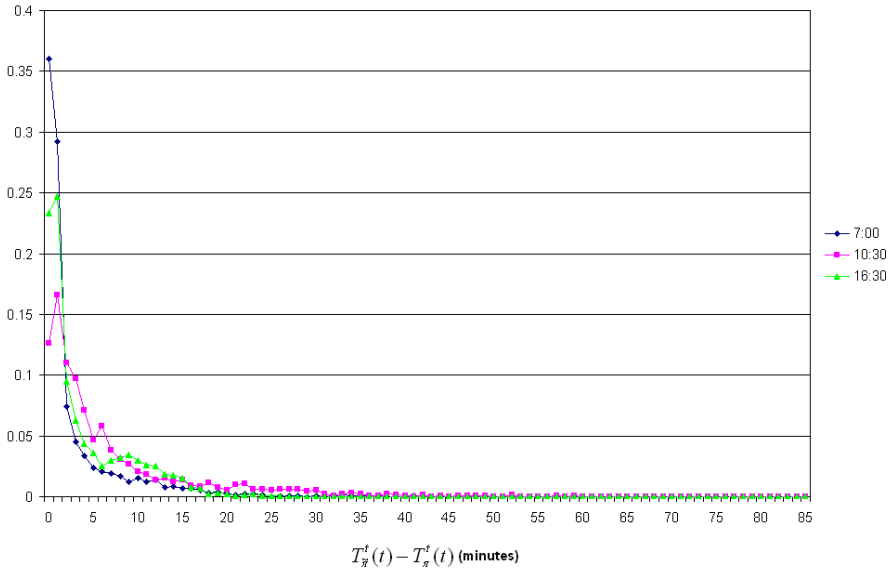


Figure D.14. *Distribution of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *for* $\gamma = 2, \varsigma = 1\frac{1}{2}$ *at 7:00h, 10:30h, 16:30h.*

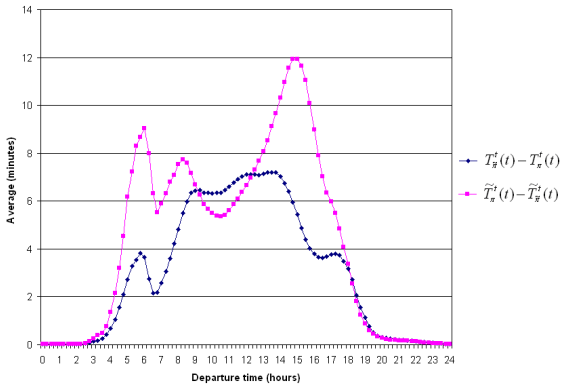


Figure D.15. *Average of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$.

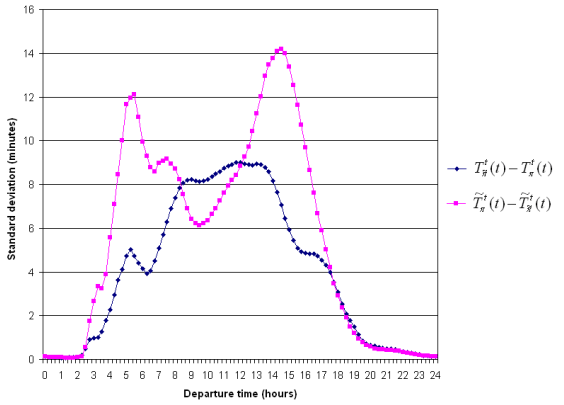


Figure D.16. *Standard deviation of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$.

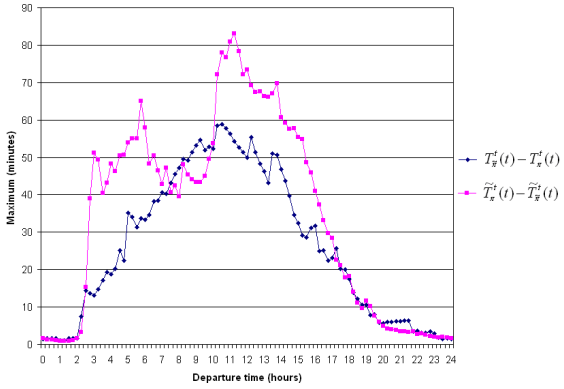


Figure D.17. *Maximum of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$.

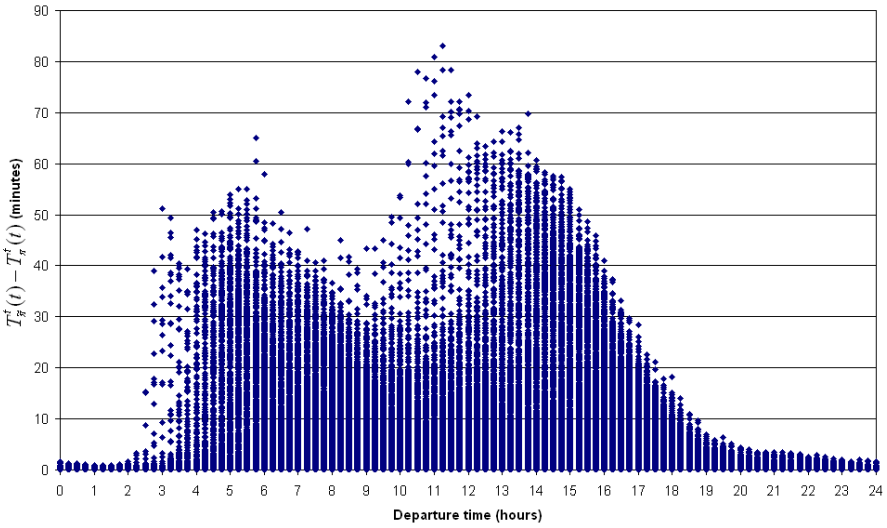


Figure D.18. *All realizations of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$.

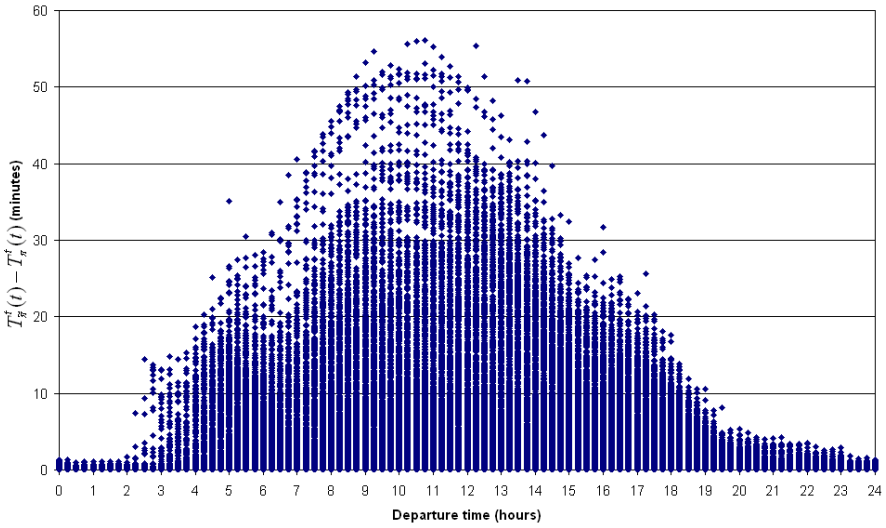


Figure D.19. *All realizations of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$.

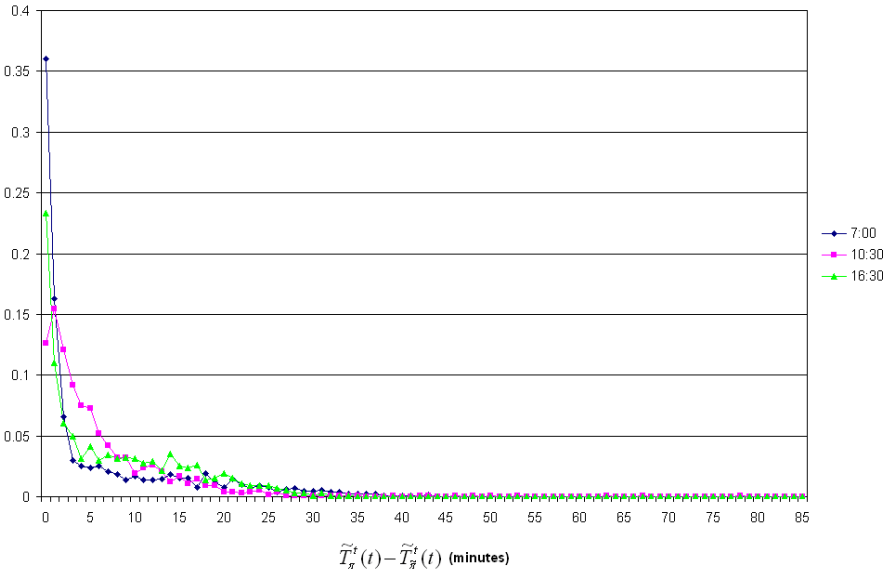


Figure D.20. *Distribution of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$ *at 7:00h, 10:30h, 16:30h.*

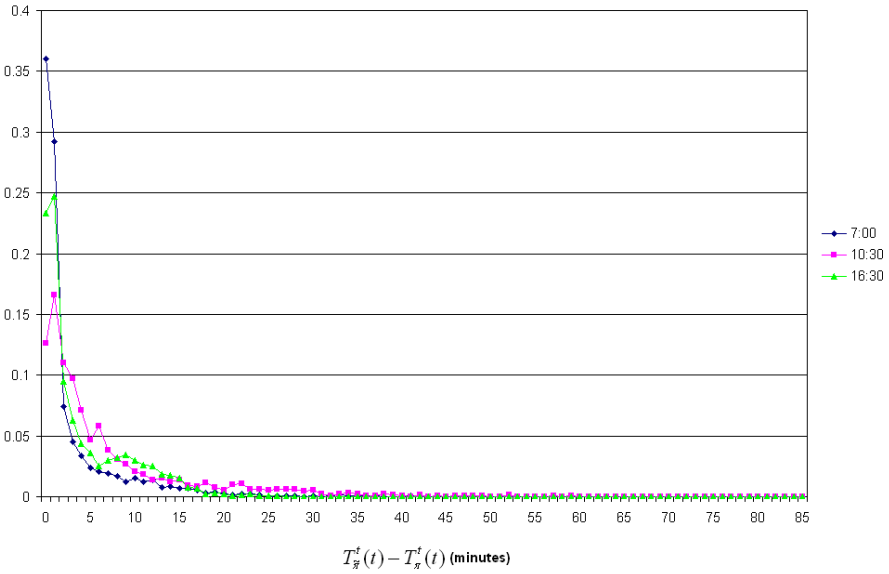


Figure D.21. *Distribution of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = \frac{3}{4}$ *at 7:00h, 10:30h, 16:30h.*

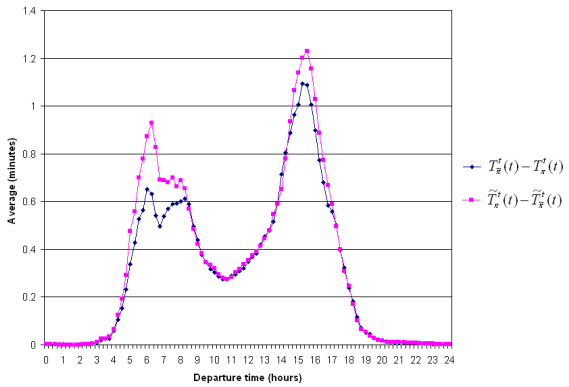


Figure D.22. *Average of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 3$.

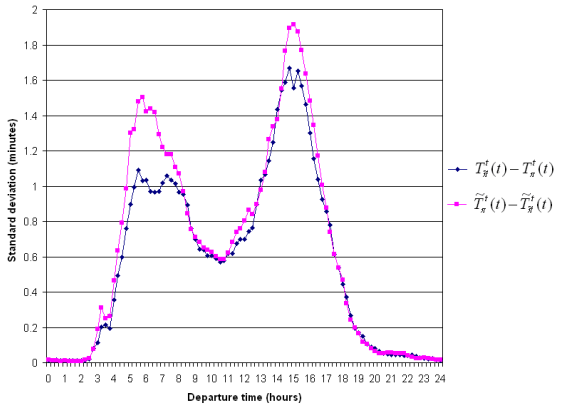


Figure D.23. *Standard deviation of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 3$.

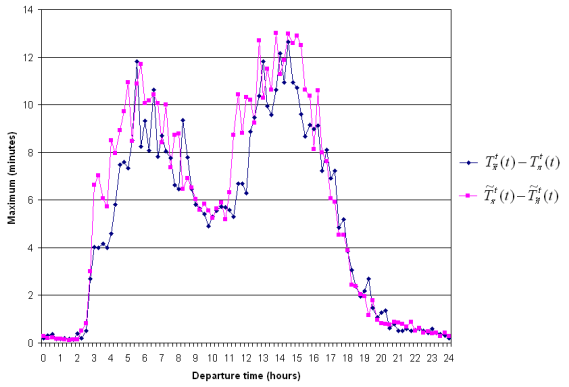


Figure D.24. *Maximum of* $T^t_{\tilde{\pi}}(t) - T^t_{\pi}(t)$ *and* $\tilde{T}^t_{\pi}(t) - \tilde{T}^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1, \varsigma = 3$.

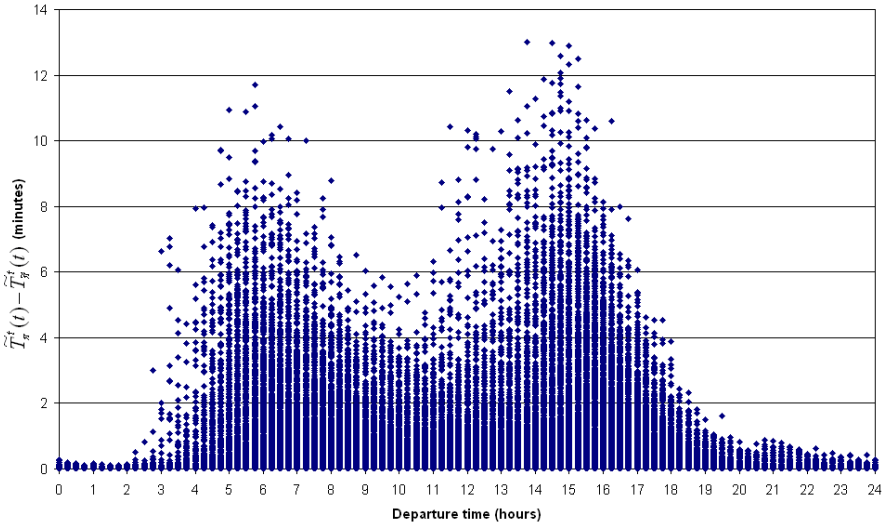


Figure D.25. *All realizations of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1$, $\varsigma = 3$.

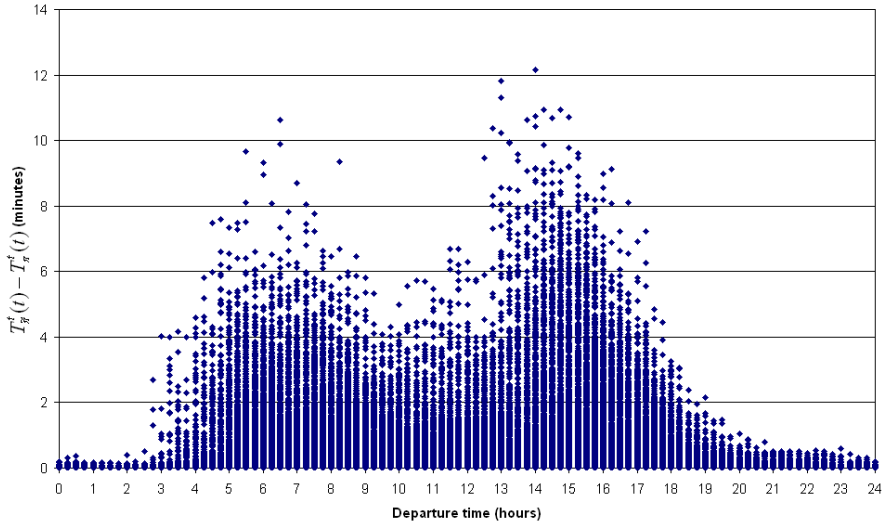


Figure D.26. *All realizations of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1$, $\varsigma = 3$.

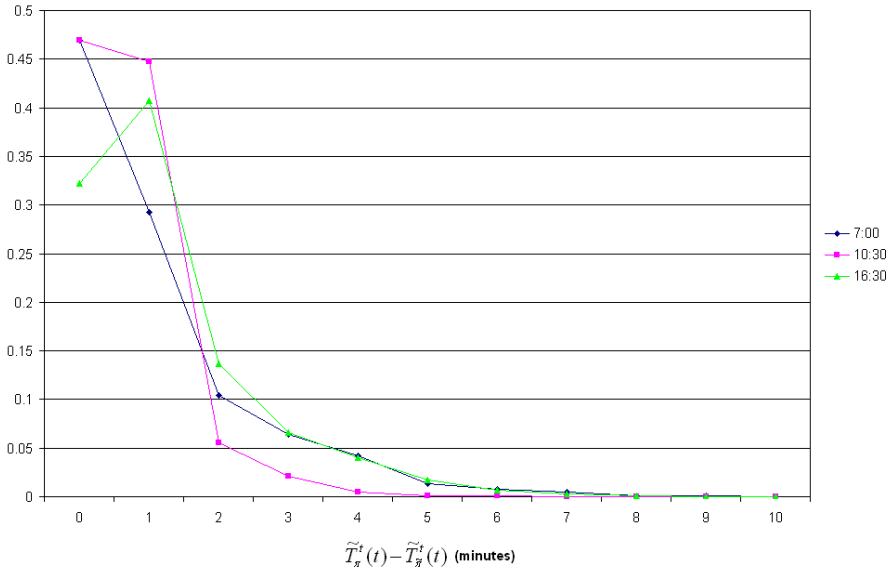


Figure D.27. *Distribution of* $\tilde{T}^t_{\tilde{\pi}}(t) - \tilde{T}^t_{\pi}(t)$ *for* $\gamma = 1$, $\varsigma = 3$ *at 7:00h, 10:30h, 16:30h.*

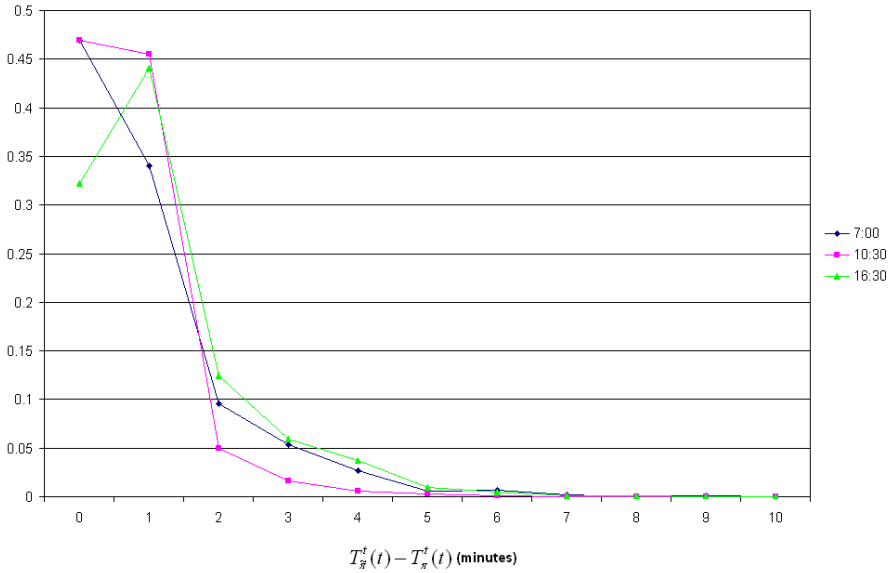


Figure D.28. *Distribution of* $T^t_{\pi}(t) - T^t_{\tilde{\pi}}(t)$ *for* $\gamma = 1$, $\varsigma = 3$ *at 7:00h, 10:30h, 16:30h.*

Bibliography

ALBER, JOCHEN, HENNING FERNAU, AND ROLF NIEDERMEIER [2003], Graph separators: a parameterized view, *Journal of Computer and System Sciences* **67**, 808–832.

ARONSON, JAY E. [1989], A survey of dynamic network flows, *Annals of Operations Research* **20**, 1–66.

AZARON, AMIR, AND FARHAD KIANFAR [2003], Dynamic shortest path in stochastic dynamic networks: Ship routing problem, *European Journal of Operational Research* **144**, 138–156.

BAÑOS, R., C. GIL, J. ORTEGA, AND F.G. MONTOYA [2004], A parallel multilevel metaheuristic for graph partitioning, *Journal of Heuristics* **10**, 315–336.

BELLMAN, RICHARD [1957], *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, United States.

BELLMAN, RICHARD [1958], On a routing problem, *Quarterly of Applied Mathematics* **16**, 87–90.

BERRY, JONATHAN W., AND MARK K. GOLDBERG [1999], Path optimization for graph partitioning problems, *Discrete Applied Mathematics* **90**, 27–50.

BERTSEKAS, DIMITRI P. [1998], *Network Optimization: Continuous and Discrete Models*, Athena Scientific, Nashua, New Hampshire, United States.

BOLLOBÁS, B., AND A.D. SCOTT [2004], Judicious partitions of bounded-degree graphs, *Journal of Graph Theory* **46**, 131–143.

BRIGHTWELL, GRAHAM, AND DOUGLAS B. WEST [2000], Chapter 11: Partially ordered sets, in: Kenneth H. Rosen (ed.), *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, London, United Kingdom.

CAR, ADRIJANA, HENNY MEHNER, AND GEORGE TAYLOR [1999], *Experimenting with Hierarchical Wayfinding*, Technical report 011999, University of Newcastle upon Tyne, Department of Geomatics, Newcastle upon Tyne, United Kingdom.

CHABINI, I. [2002], Algorithms for k-shortest paths and other routing problems in time-dependent networks, *Accepted for publication in Transportation Research Part B: Methodological*.

CHABINI, I., AND SHAN LAN [2002], Adaptations of the $A^\star$ algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks, *IEEE Transactions on Intelligent Transportation Systems* **3**, 60–74.

CHAN, EDWARD P.F., AND NING ZHANG [2001], Finding shortest paths in large network systems, in: Walid G. Aref (ed.), *Proceedings of the ninth ACM International Symposium on Advances in Geographic Information Systems*, Atlanta, Georgia, United States, ACM Press, 160–166.

CHEN, DANNY Z., OVIDIU DAESCU, XIAOBO (SHARON) HU, AND JINHUI XU [2003], Finding an optimal path without growing the tree, *Journal of Algorithms* **49**, 13–41.

CHEN, DANNY Z., AND JINHUI XU [2000], Shortest path queries in planar graphs, *Proceedings of the thirty-second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, United States, ACM Press, 469–478.

CORDONE, ROBERTO, AND FRANCESCO MAFFIOLI [2004], On the complexity of graph tree partition problems, *Discrete Applied Mathematics* **134**, 51–65.

CUNHA, CLAUDIO BARBIERI DA, AND JOFFRE SWAIT [2000], New dominance criteria for the generalized permanent labelling algorithm for the shortest path problem with time windows on dense graphs, *International Transactions in Operational Research* **7**, 139–157.

DEAN, BRIAN C. [2004], Algorithms for minimum-cost paths in time-dependent networks with waiting policies, *Networks* **44**, 41–46.

DIJKSTRA, E.W. [1959], A note on two problems in connexion with graphs, *Numerische Mathematik* **1**, 269–271.

ERTL, GERHARD [1998], Shortest path calculation in large road networks, *OR Spektrum* **20**, 15–20.

FALKNER, JULIE, FRANZ RENDL, AND HENRY WOLKOWICZ [1994], A computational study of graph partitioning, *Mathematical Programming* **66**, 211–239.

FERNÁNDEZ-MADRIGAL, JUAN-ANTONIO, AND JAVIER GONZÁLEZ [2002], Multihierarchical graph search, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, 103–113.

FIDUCCIA, C.M., AND R.M. MATTEYSES [1982], A linear-time heuristic for improving network partitions, *Proceedings of the nineteenth Design Automation Conference*, Las Vegas, Nevada, United States, IEEE Press, 175–181.

FLINSENBERG, INGRID [2003a], Graph partitioning for route planning in car navigation systems, *Proceedings of the 11'th IAIN World Congress, Smart Navigation - Systems and Services* -, Berlin, Germany.

FLINSENBERG, INGRID [2003b], Using turn restrictions for faster route planning with partitioned road networks, *10th Saint Petersburg International Conference on Integrated Navigation Systems*, Saint Petersburg, Russia, 198–206.

FLINSENBERG, INGRID, MARTIJN VAN DER HORST, JOHAN LUKKIEN, AND JACQUES VERRIET [2004], Creating graph partitions for fast optimum route planning, *WSEAS Transactions on Computers* **3**, 569–574.

FLOYD, ROBERT W. [1962], Algorithm 97 shortest path, *Communications of the*

ACM **5**, 345.

FORD JR., L.R., AND D.R. FULKERSON [1962], *Flows in Networks*, Princeton University Press, Princeton, New Jersey, United States.

FREDERICKSON, GREG N. [1985], Data structures for on-line updating of minimum spanning trees, with applications, *SIAM Journal on Computing* **14**, 781–798.

FRIGIONI, DANIELE [1998], Semidynamic algorithms for maintaining single-source shortest path trees, *Algorithmica* **22**, 250–274.

FU, LIPING [2001], An adaptive routing algorithm for in-vehicle route guidance systems with real-time information, *Transportation Research Part B: Methodological* **35**, 749–765.

GAO, SONG [2002], Routing problems in stochastic time-dependent networks with applications in dynamic traffic assignment, Master's thesis, Massachusetts Institute of Technology, Department of Civil and Environmental Engineering, Cambridge, Massachusetts, United States.

GAO, S., AND I. CHABINI [2002], The best routing policy problem in stochastic time-dependent networks, *Transportation Research Record* **1783**, 188–196.

GAREY, MICHAEL R., AND DAVID S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co, New York, New York, United States.

GELPERIN, DAVID [1977], On the optimality of A*, *Artificial Intelligence* **8**, 69–76.

GHIANI, GIANPAOLO, FRANCESCA GUERRIERO, GILBERT LAPORTE, AND ROBERTO MUSMANNO [2003], Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies, *European Journal of Operational Research* **151**, 1–11.

GLENN, ANDREW M. [2001], Algorithms for the shortest path problem with time windows and shortest path reoptimization in time-dependent networks, Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, Massachusetts, United States.

GOLSHANI, FOROUZAN, ENRIQUE CORTES-RELLO, AND THOMAS H. HOWELL [1996], Dynamic route planning with uncertain information, *Knowledge-Based Systems* **9**, 223–232.

GRANAT, JANUSZ, AND FRANCESCA GUERRIERO [2003], The interactive analysis of the multicriteria shortest path problem by the reference point method, *European Journal of Operational Research* **151**, 103–118.

HART, PETER E., NILS J. NILSSON, AND BERTRAM RAPHAEL [1968], A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions of Systems Science and Cybernetics* **4**, 100–107.

HE, RACHEL R., HENRY X. LIU, ALAIN L. KORNHAUSER, AND BIN RAN [2002], *Temporal and Spatial Variability of Travel Time*, Technical Report UCI-ITS-WP-02-10, Institute of Transportation Studies, University of California, Irvine,

California, United States.

HENZINGER, MONICA R., PHILIP KLEIN, SATISH RAO, AND SAIRAM SUBRAMANIAN [1997], Faster shortest-path algorithms for planar graphs, *Journal of Computer and System Sciences* **55**, 3–23.

HIRAISHI, HIRONORI, HAYATO OHWADA, AND FUMIO MIZOGUCHI [1999], Intercommunicating car navigation system with dynamic route finding, *Proceedings 1999 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, Tokyo, Japan, IEEE, 284–289.

HOFFMANN, ACHIM G. [1994], The dynamic locking heuristic - a new graph partitioning algorithm, *1994 IEEE International Symposium on Circuits and Systems*, London, United Kingdom, 173–176.

HOLZER, MARTIN [2003], Hierarchical speed-up techniques for shortest-path algorithms, Diplomarbeit, Universität Konstanz, Fachbereich Informatik und Informationswissenschaft, Konstanz, Germany.

HOLZER, MARTIN, FRANK SCHULZ, AND THOMAS WILLHALM [2004], Combining speed-up techniques for shortest-path computations, in: Celso C. Ribeiro and Simone L. Martins (eds.), *Experimental and Efficient Algorithms: Third International Workshop, WEA 2004*, Lecture Notes in Computer Science 3059, Angra dos Reis, Brazil, Springer, 269–284.

HORN, MARK E. T. [2000], Efficient modeling of travel in networks with time-varying link speeds, *Networks* **36**, 80–90.

HORST, M.G. VAN DER [2003], Optimal route planning for car navigation systems, Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computing Science, Eindhoven, The Netherlands.

HUANG, YUN-WU, NING JING, AND ELKE A. RUNDENSTEINER [1995], Hierarchical path views: A model based on fragmentation and transportation road types, *Proceedings of the 3rd ACM International Workshop on Advances in Geographic Information Systems*, Baltimore, Maryland, United States, ACM Press, 93–100.

HUANG, YUN-WU, NING JING, AND ELKE A. RUNDENSTEINER [1996], Path queries for transportation networks: dynamic reordering and sliding window paging techniques, *Proceedings of the fourth ACM Workshop on Advances in Geographic Information Systems*, Rockville, Maryland, United States, ACM Press, 9–16.

HUANG, YUN-WU, NING JING, AND ELKE A. RUNDENSTEINER [1997], A hierarchical path view model for path finding in intelligent transportation systems, *GeoInformatica* **1**, 125–159.

HUANG, YUN-WU, NING JING, AND ELKE A. RUNDENSTEINER [2000], Optimizing path query performance: graph clustering strategies, *Transportation Research Part C: Emerging Technologies* **8**, 381–408.

JAHN, OLAF, ROLF H. MÖHRING, AND ANDREAS S. SCHULZ [2000], Optimal routing of traffic flows with length restrictions in networks with congestion, *Operations Research Proceedings 1999*, Magdeburg, Germany, Springer.

JING, NING, YUN-WU HUANG, AND ELKE A. RUNDENSTEINER [1996], Hierarchical optimization of optimal path finding for transportation applications, *Proceedings of the fifth International Conference on Information and Knowledge Management*, Rockville, Maryland, United States, ACM Press, 261–268.

JING, NING, YUN-WU HUANG, AND ELKE A. RUNDENSTEINER [1998], Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation, *IEEE Transactions on Knowledge and Data Engineering* **10**, 409–432.

JOHNSON, ELLIS L. [1972], On shortest paths and sorting, *Proceedings of the 25th ACM Annual Conference*, Boston, Massachusetts, United States, 510–517.

JUNG, SUNGWON, AND SAKTI PRAMANIK [2002], An efficient path computation model for hierarchically structured topographical road maps, *IEEE Transactions on Knowledge and Data Engineering* **14**, 1029–1046.

KAINDL, HERMANN, AND GERHARD KAINZ [1997], Bidirectional heuristic search reconsidered, *Journal of Artificial Intelligence Research* **7**, 283–317.

KARGER, DAVID R., AND CLIFFORD STEIN [1996], A new approach to the minimum cut problem, *Journal of the ACM* **43**, 601–640.

KARYPIS, GEORGE, AND VIPIN KUMAR [1998], A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing* **20**, 359–392.

KAUFMAN, DAVID E., AND ROBERT L. SMITH [1993], Fastest paths in time-dependent networks for intelligent vehicle-highway systems application, *IVHS Journal* **1**, 1–11.

KERNIGHAN, B. W., AND S. LIN [1970], An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal* **49**, 291–307.

KIM, KIHONG, SEUNGWON YOO, AND SANG K. CHA [1998], A partitioning scheme for hierarchical path finding robust to link cost update, *Proceedings of the 5th World Congress on ITS*, Seoul, Korea.

KIM, YONG-HYUK, AND BYUNG-RO MOON [2004a], Investigation of the fitness landscapes in graph bipartitioning: An empirical study, *Journal of Heuristics* **10**, 111–133.

KIM, YONG-HYUK, AND BYUNG-RO MOON [2004b], Lock-gain based graph partitioning, *Journal of Heuristics* **10**, 37–57.

KLEIN, P. N., AND S. SUBRAMANIAN [1998], A fully dynamic approximation scheme for shortest paths in planar graphs, *Algorithmica* **22**, 235–249.

KRISHNAN, RAJESH, RAM RAMANATHAN, AND MARTHA STEENSTRUP [1999], Optimization algorithms for large self-structuring networks, *Proceedings of*

the Conference on Computer Communications (IEEE Infocom), New York, New York, United States, 71–78.

LIPTON, RICHARD J., AND ROBERT ENDRE TARJAN [1979], A separator theorem for planar graphs, *SIAM Journal on Applied Mathematics* **36**, 177–189.

LIU, HENRY X., WILL RECKER, AND ANTHONY CHEN [2004], Uncovering the contribution of travel time reliability to dynamic route choice using real-time loop data, *Transportation Research Part A: Policy and Practice* **38**, 435–453.

MANDOW, L., AND J.L. PÉREZ DE LA CRUZ [2003], Multicriteria heuristic search, *European Journal of Operational Research* **150**, 253–280.

MEYER, ULRICH [2001], Single-source shortest-paths on arbitrary directed graphs in linear average-case time, *Proceedings of the twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington, District of Columbia, United States, Society for Industrial and Applied Mathematics, 797–806.

MEYER, ULRICH [2002], Buckets strike back: Improved parallel shortest-paths, *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IEEE Computer Society, Fort Lauderdale, Florida, United States.

MEYER, U., AND P. SANDERS [2003], Δ-stepping: a parallelizable shortest path algorithm, *Journal of Algorithms* **49**, 114–152.

MOHAJER, KEYVAN, ALMIR MUTAPCIC, AND MAJID EMAMI [2004], Estimation-pruning (EP) algorithm for point-to-point travel cost minimization in a non-fifo dynamic network, *Stanford Electrical Engineering and Computer Science Research Journal*, 4–8, Spring.

MONIEN, BURKHARD, AND RALF DIEKMANN [1997], A local graph partitioning heuristic meeting bisection bounds, *Proceedings of the eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, Minnesota, United States, SIAM.

MONIEN, BURKHARD, ROBERT PREIS, AND RALF DIEKMANN [2000], Quality matching and local improvement for multilevel graph-partitioning, *Parallel Computing* **26**, 1609–1634.

MONTEMANNI, R., AND L.M. GAMBARDELLA [2004], An exact algorithm for the robust shortest path problem with interval data, *Computers & Operations Research* **31**, 1667–1680.

MONTEMANNI, R., L.M. GAMBARDELLA, AND A.V. DONATI [2004], A branch and bound algorithm for the robust shortest path problem with interval data, *Operations Research Letters* **32**, 225–232.

MONTGOMERY, DOUGLAS C., ELIZABETH A. PECK, AND G. GEOFFREY VINING [2001], *Introduction to Linear Regression Analysis, 3rd Edition*, John Wiley & Sons, New York, New York, United States.

NARVÁEZ, PAOLO, KAI-YEUNG SIU, AND HONG-YI TZENG [2001], New dynamic SPT algorithm based on a ball-and-string model, *IEEE/ACM Transac-*

tions on Networking **9**, 706–718.

NGUYEN, SANG, STEFANO PALLOTTINO, AND MARIA GRAZIA SCUTELLÀ [2002], A new dual algorithm for shortest path reoptimization, in: Michel Gendreau and Patrice Marcotte (eds.), *Transportation and Network Analysis: Current Trends*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 221–235.

ORDA, ARIEL, AND RAPHAEL ROM [1990], Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length, *Journal of the ACM* **37**, 607–625.

ORDA, ARIEL, AND RAPHAEL ROM [1991], Minimum weight paths in time-dependent networks, *Networks* **21**, 295–319.

ORDA, ARIEL, AND RAPHAEL ROM [1996], Distributed shortest-path protocols for time-dependent networks, *Distributed Computing* **10**, 49–62.

ORLIN, J.B. [1984], Minimum convex cost dynamic network flows, *Mathematics of Operations Research* **9**, 190–207.

OSOGAMI, TAKAYUKI, AND MOR HARCHOL-BALTER [2003], A closed-form solution for mapping general distributions to minimal PH distributions, *Computer Performance*, Lecture Notes in Computer Science 2794, Springer-Verlag Heidelberg, Heidelberg, Germany, 200–217.

PALLOTTINO, STEFANO, AND MARIA GRAZIA SCUTELLÀ [2003], A new algorithm for reoptimizing shortest paths when the arc costs change, *Operations Research Letters* **31**, 149–160.

PAPE, U. [1974], Implementation and efficiency of Moore-algorithms for the shortest route problem, *Mathematical Programming* **7**, 212–222.

PEARL, JUDEA [1984], *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, Massachusetts, United States.

PETTIE, SETH [2002], A faster all-pairs shortest path algorithm for real-weighted sparse graphs, *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 2380, 85–97.

POTHEN, ALEX [1997], Graph partitioning algorithms with applications to scientific computing, in: David E. Keyes, Ahmed Sameh, and V. Venkatakrishnan (eds.), *Parallel Numerical Algorithms*, Kluwer Academic Publishers, Dordrecht, Netherlands, 323–368.

PREIS, ROBERT [1999], Analytical methods for multilevel graph-partitioning, in: Kurt Beiersdörfer, Gregor Engels, and Wilhelm Schäfer (eds.), *Informatik '99: Informatik überwindet Grenzen*, Paderborn, Germany, Springer, 223–230.

ROWE, N.C., AND R.S. ALEXANDER [2000], Finding optimal-path maps for path planning across weighted regions, *The International Journal of Robotics Research* **19**, 83–95.

SCHLOEGEL, KIRK, GEORGE KARYPIS, AND VIPIN KUMAR [2000], Graph partitioning for high performance scientific simulations, in: J. Dongarra, I. Foster, G. Fox, K. Kennedy, and A. White (eds.), *CRPC Parallel Computing Handbook*, Morgan Kaufmann, San Francisco, California, United States.

SCHLOTT, STEFAN [1997], *Vehicle navigation : route planning, positioning and route guidance*, verlag moderne industrie, Landsberg/Lech, Germany.

SCHMID, WOLFGANG [2000], *Berechnung Kürzester Wege in Straßennetzen mit Wegeverboten*, Dissertation, Universität Stuttgart, Stuttgart, Germany.

SCHULZ, FRANK, DOROTHEA WAGNER, AND CHRISTOS ZAROLIAGIS [2002], Using multi-level graphs for timetable information in railway systems, *Algorithm Engineering and Experiments: 4th International Workshop, ALENEX 2002*, Lecture Notes in Computer Science 2409, San Francisco, California, United States, Springer-Verlag, 43–59.

SEONG, MYEONGKI, KISEOK SUNG, AND SOONDAL PARK [1998], Finding a route in a hierarchical urban transportation network, *Proceedings of 5th World Congress on Intelligent Transport Systems*, Seoul, Korea.

SHEKHAR, SHASHI, ANDREW FETTERER, AND BRAJESH GOYAL [1997], Materialization trade-offs in hierarchical shortest path algorithms, *Proceedings of the 5th International Symposium on Large Spatial Databases*, Lecture Notes in Computer Science 1262, Berlin, Germany, Springer-Verlag, 94–111.

SHEKHAR, SHASHI, AND BABAK HAMIDZADEH [1993], Self-adjusting real-time search: A summary of results, *Proceedings of the Fifth International Conference on Tools with Artificial Intelligence (ICTAI '93)*, Boston, Massachusetts, United States, IEEE Computer Society, 224–231.

SUBRAMANIAN, SHIVARAM [1997], Routing algorithms for dynamic, intelligent transportation networks, Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, United States.

SZEIDER, STEFAN [2003], Finding paths in graphs avoiding forbidden transitions, *Discrete Applied Mathematics* **126**, 261–273.

THORUP, MIKKEL [1997], Undirected single source shortest paths in linear time, *38th Annual Symposium on Foundations of Computer Science*, Miami Beach, Florida, United States, IEEE, 12–21.

VAN WOENSEL, TOM [2003], *Models for uninterrupted traffic flows: A queueing approach*, Proefschrift, Universiteit Antwerpen, Faculteit Toegepaste Economische Wetenschappen, Antwerp, Belgium.

WANG, JINCHANG, AND THOMAS KAEMPKE [2004], Shortest route computation in distributed systems, *Computers & Operations Research* **31**, 1621–1633.

WELLMAN, MICHAEL, MATTHEW FORD, AND KENNETH LARSON [1995], Path planning under time-dependent uncertainty, *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, Montreal, Que-

bec, Canada, Morgan Kaufmann Publishers, 532–539.

WINTER, STEPHAN [2002], Modeling costs of turns in route planning, *GeoInformatica* **6**, 345–361.

ZHAO, YILIN [1997], *Vehicle Location and Navigation Systems*, Artech House Inc., Norwood, Massachusetts, United States.

ZHU, S.Q. [2004], Stochastic time-dependent route planning in car navigation systems, Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computing Science, Eindhoven, The Netherlands.

ZWILLINGER, DANIEL, AND STEPHEN KOKOSKA (eds.) [1999], *CRC Standard Probability and Statistics Tables and Formulae*, CRC Press, London, United Kingdom.

Notational Index

The numbers refer to the pages of the first occurrence.

Roadgraph

Stochastic

Time-dependent

Author Index

Subject Index

Curriculum Vitae

Ingrid Flinsenberg was born on 21 July 1977 in Helmond. After receiving her diploma in 1995 from the athenaeum of the Macropedius College in Gemert, she started to study econometrics at Tilburg University. In 2000 she graduated with honors in the field of operations research after completing her Master's thesis on the topic of queuing theory at KPN Research. Subsequently, she started as a PhD-student at the Eindhoven Embedded Systems Institute on the topic of route planning algorithms for car navigation. The research was carried out at Siemens VDO Automotive in Eindhoven under supervision of Prof. dr. Emile Aarts and dr. Jacques Verriet. It resulted in several patent applications, presentations and publications at international conferences and this thesis.

Samenvatting

Autofabrikanten bieden steeds vaker standaard een autonavigatiesysteem aan in hun nieuwe auto's. Deze autonavigatiesystemen geven de bestuurder gesproken en vaak ook visuele adviezen hoe zijn bestemming bijvoorbeeld zo snel mogelijk te bereiken is. Nadat de bestuurder zijn bestemming heeft ingegeven, en om begeleiding gevraagd heeft, moeten deze adviezen snel bepaald worden omdat de bestuurder meteen wil vertrekken en niet wil wachten tot het systeem zijn route bepaald heeft. Bovendien verwacht de bestuurder niet alleen snel advies maar ook de beste route naar zijn bestemming. Gezien het feit dat de kaarten die gebruikt worden om deze routes te bepalen steeds groter worden, het wegennetwerk van heel Europa is tegenwoordig beschikbaar op CD of DVD, is het zeer snel kunnen bepalen van optimale routes op zeer grote wegennetwerken nog steeds een uitdaging.

Dit proefschrift beschrijft algoritmen die gebruikt kunnen worden om optimale routes op zeer grote wegennetwerken snel te kunnen bepalen. Daarbij wordt niet alleen rekening gehouden met de maximale snelheid, maar ook met dagelijkse congestie op snelwegen en eventuele onzekerheid in de reistijden als gevolg van deze congestie. Ook worden de resultaten van de gepresenteerde algoritmen vergeleken met de algoritmen die gebruikt worden in commerciële autonavigatiesystemen. We concluderen dat the gepresenteerde aanpak resulteert in betere routes die sneller bepaald worden dan in het commerciële VDO Dayton autonavigatiesysteem van Siemens VDO Automotive dat ter vergelijking gebruikt is.

Dit resultaat is bereikt door allereerst het wegennetwerk zo goed mogelijk op te delen in regio's, zo genoemde cellen. Om dit zo goed mogelijk te doen is een partitioneringscriterium bepaald dat een goede maat is voor de snelheid van het route planningsalgoritme dat met behulp van deze partitionering de beste route moet bepalen. Om de beste partitionering volgens dit criterium te bepalen zijn twee benaderingsalgoritmen ontwikkeld, geïmplementeerd en getest op verschillende echte wegennetwerken. Ook is deze aanpak nog uitgebreid door ook cellen nogmaals op te delen waardoor de route planning nog verder versneld wordt. Ook hiervoor zijn weer twee algoritmen ontwikkeld waarvan de meest belovende is geïmplementeerd en getest. De resultaten laten zien dat met behulp van deze aanpak optimale routes op zeer grote wegennetwerken snel bepaald kunnen worden.

Vervolgens is er een model opgesteld voor de dagelijkse congestie op het wegennetwerk, en is, met behulp van data betreffende dagelijkse congestie op de snelweg

A1/E19 nabij Antwerpen in 1995 en op de belangrijkste snelwegen in Nederland in 2002, bepaald wat het effect is van deze congestie op de aankomsttijd van de bestuurder en op de beste route gegeven deze congestie. Gebleken is dat vooral in de spitsuren door rekening te houden met deze congestie tijdens het bepalen van de beste route, een behoorlijke reductie in de verwachte aankomsttijd te behalen is. Vervolgens is een aanpak voorgesteld om, ook wanneer gebruik wordt gemaakt van een partitie, routes te kunnen bepalen van goede kwaliteit zonder al te veel data te moeten opslaan.

Daarnaast is ook het effect van onzekerheid in de reistijden op de aankomsttijd van de bestuurder en de beste route bestudeerd. Hierbij hebben we de beste route zo gedefiniëerd dat een bestuurder kan aangeven hoeveel belang hij hecht aan het niet overschrijden van de verwachte aankomsttijd en aan de onzekerheid in deze aankomsttijd. Ook hierbij is gebruik gemaakt van data betreffende onzekerheid in de reistijden op Nederlandse snelwegen in 2002. Uit de resultaten is gebleken dat vooral wanneer de bestuurder erg veel waarde hecht aan zekerheid omtrent zijn (laatst mogelijke) aankomsttijd, het meenemen van informatie betreffende onzekerheid in reistijden zinvol is. Om ook dit te kunnen combineren met het gebruik van een partitie is een aanpak voorgesteld waarmee wederom goede kwaliteit routes kunnen worden bepaald zonder dat te veel data moet worden opgeslagen om dit te bereiken.

Tot slot zijn de resultaten van de voorgestelde algoritmen vergeleken met die van algoritmen die gebruikt worden in commerciële autonavigatiesystemen. Het is gebleken dat de gepresenteerde algoritmen leiden tot snellere route planning en hogere kwaliteit routes. Tevens zijn enige aanbevelingen gedaan over hoe de overige functionaliteit van autonavigatiesystemen kan worden gehandhaafd wanneer gebruik wordt gemaakt van de voorgestelde aanpak.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ.* Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12