University of Twente

Master's Thesis

Computing time dependent travel times in vehicle routing problems

Author:
Mathijs W.H. Waegemakers MSc.

Supervisors:
Dr. Ir. E.C. van Berkum
Dr. Ir. M.R.K. Mes
S.K. den Heijer MSc.

March 2017

To my father, the real map specialist.

Management summary

Motivation:
One of the products delivered by ORTEC is a software suite called *ORTEC Routing & Dispatch (ORD)*, which manages and optimizes the distribution process of delivering goods with a fleet of vehicles. The optimizer within ORD uses a set of heuristics to create an efficient distribution plan. By taking into account traffic congestion, by including the *time dependent travel times (TD-TTs)* into the distribution plan, ORD improves the feasibility of the distribution plans and the overall solution quality by avoiding congested areas during rush hour. Currently, all exact algorithms that are able to compute the TD-TTs are too slow to be used in optimization heuristics. To overcome this shortcoming, it is possible to approximate the TD-TTs in favour of fast computations. ORD has the ability to use such an approximation algorithm, which we call the *Travel Time Calculator (TTC)*. In this thesis, we first research the accuracy of this TTC. Second, we develop a new approach which we call the *Congestion Hierarchies Algorithm (CH-algorithm)*.

Method:
We measure the accuracy of the TTC using the BeNeLux road network, which contains the historical TD-TTs on the majority of the edges in the network. We take these historical travel times as the ground truth, and exclude any real-time information from this research. Since it is possible to compute TD-TTs exactly, we are able to measure the loss of accuracy between the approximation algorithm and exact TD-TTs. To research under which conditions the current approach becomes inaccurate, we create three test groups consisting of a total of 15 test sets of 2500 randomly selected *origin-destinations pairs (OD-pairs)*. All OD-pairs in a test set have characteristics like path length and geographical location.

The CH-algorithm we developed is a TD-SP algorithm that uses multiple overlay levels to store precomputed congestion factors. The congestion factor is the delay percentage between the TD-TT and the *free flow travel time (FF-TT)* at a certain departure time. During optimization, the CH-algorithm calculates the TD-TTs by computing the FF-TT and multiplying it with the corresponding congestion factor. The method is fast as it only relies on a quick retrieval of the FF-TT, together with a table look-up and a multiplication. A quick FF-TT retrieval is possible using an algorithm like Highway Node Routing or Contraction Hierarchies. However, due to memory restrictions, simply storing all congestion factors is not an option. Therefore, we do not compute the congestion factors between each pair of singular nodes, but between areas of nodes. To benefit from a fine grid of areas, while remaining memory efficient, we use multiple overlay layers that divide the road network into quadrants. Each layer is a quadratic subdivision of the layer above it. In the end, the lowest layer has fine grid of many small areas, while the highest and second highest layer consist of only one and four areas, respectively. Only for the areas that are considered important enough, the algorithm will compute the congestion factors. If the CH-algorithm wants to retrieve the congestion factors between an OD-pair, it will search the layers from bottom to top to find the layer in which both the origin and the destination node is in area, which have congestion factors between them precomputed.

Results:
The CH-algorithm outperforms the TTC in the majority of the experiments we ran. The results show that the CH-algorithm is on average 34% more accurate in congested areas, compared to the TTC. It also shows, that the CH-algorithm is on average 28% more accurate for trips with a length of at most 30 minutes, compared to the TTC. These values are weekday averages, during rush hour these values

increase to 38% and 33% respectively. However, we decided while designing the algorithm that some accuracy for longer trips would be sacrificed in favour of the shorter trips, resulting in an accuracy drop for trips longer than 30 minutes. The accuracy decreases on average with 85%, meaning that the average deviation increases from 1.7% to 2.7% and from 1.0% to 2.0% for trips of 2 hours and 4 hours respectively. To put this increase into perspective, the duration of a 4-hour trip (in free flow) on average has an additional deviation of 2.5 minutes.

Recommendations: In this research, we present a proof of concept of the newly developed CH-algorithm. We showed promising results that could improve the feasibility of the vehicle routing solutions. This is beneficial for the customer, as their distributions plans get more reliable. This means, less driver time violations, more on-time delivery, and ultimately less rescheduling and deployment of additional vehicles. This will eventually lead to a more reliable customer, which result in overall positive business benefits. Before this algorithm can be used, we recommend to further research the four preprocessing steps of the algorithm. We expect that there is still some potential accuracy to be gained.

Acknowledgements

A few weeks back I went to get something to eat with a former roommate and a former fellow student of mine. During dinner, we had a wild discussion about technology, particularly about computer science and programming. Advantages and disadvantages of different languages, structures, and approaches were discussed widely. At one point, my former roommate pointed out that it was just two years that I knocked on his door with the question "that I wanted to learn *something* about programming", and that now we are having big discussions about things I had no knowledge of until recently. It then hit me how much I have learned over the past two years, knowledge that is going to help me the rest of my life.

This thesis is the result of a process starting back in the beginning of 2015, after I finished my master in Industrial Engineering & Management, with a thesis also completed at ORTEC. Exactly one year later I am able to present the result of my research. Although I cannot mention everyone explicitly, I like to thank the following: ORTEC, for giving me yet another opportunity to graduate at a wonderful company. Leendert Kok, who trusted me to come up with a solution that both benefits ORTEC as it also functions as a great thesis topic. Bas den Heijer, who had the rewarding task to answer all my minor questions about almost everything when I just started my graduation assignment. Laurien Verheijen, for proofreading my thesis. Marloes van der Maas, also for proofreading my thesis, but even more for all the mental support I got from you over the last two years. From the university, Eric van Berkum and Martijn Mes, who were did an amazing job in reviewing my work, and taking the time to provide me with great feedback. Without all of you, this thesis would not have been a success.

Looking back on this period, I am satisfied with what I have accomplished. Looking at my thesis I can say that I produced some useful results on computing time dependent travel times. I am even more pleased with everything I learned during the last year, especially the C++ and C# programming skills I developed. This, together with everything I learned during the year before that, makes that I am currently comfortable with developing software. Looking forward, I have the opportunity to grow within ORTEC, something I am grateful about.

This page is intentionally left blank.

Contents

Chapter 1

Introduction

This research is conducted at the Product Development department of ORTEC within the area of Transport & Logistics. ORTEC is a company that delivers optimization solutions in the field of *Operations Research (OR)*, as well as consulting services in which OR techniques are applied. One of the solutions that ORTEC delivers is a software product called *ORTEC Routing & Dispatch (ORD)*. ORD allows companies to manage the distribution of goods with a fleet of vehicles, and optimize their transport planning. In literature, this process is commonly known as the *Vehicle Routing Problem (VRP)* [1].

Another well-known OR problem is the *Shortest Path Problem (SPP)*, which is the problem of finding the path with the least impedance between two points in a graph. In this research the graph represents a road network, therefore the costs of the edges represent travel times. Many of the existing approaches assume that the edge weights of the graph are constant, meaning they have a single value representing the travel time of the edge. However, in real life, the travel time of an edge can vary over time, especially in busy urban areas. So the edges do not have a single constant value for travel time, but a *time-dependent travel time (TD-TT)* depending on the time of day on that edge. Algorithms that solve the SPP on spatial graphs with time-dependent edges, are known as *Time Dependent Shortest Path (TD-SP)* algorithms.

The VRP has been studied extensively over the years [2], and lately there has been an increased interest in including real life constraints, like time windows. However, most of the proposed models assume constant travel times between the nodes, while research shows that using TD-TTs instead of constant TTs results in more feasible solutions. Kok et al. [3] calculate that 99% of late arrivals at customers can be eliminated if one accounts for traffic congestion during the off-line planning phase. Demiryurek et al. [4] demonstrate that including TD-TTs improves the travel time of a trip on average with 36%, when a SP is found using a TD-SP algorithm instead of the constant travel time variant. The TD-SP algorithm tries to avoid congested areas at the cost of a small detour, while the constant SP algorithm does not include congestion, neglecting getting stuck in traffic. This value rises to 68% and 43% during the morning and afternoon commute respectively.

The TD-SP algorithms that are currently known in literature, are either not fast enough or use a gigantic amount of the main-memory, to be even considered as a part of vehicle routing algorithms [4]. As computational speed is an important factor within solving the VRP, the current *shortest path (SP)* algorithm at ORTEC returns an estimate of the TD-TT, in favour of a faster response. Getting the TD-TT between two locations is possible using an exact approach, but it takes a significant amount

of time to calculate it. The current TD-SP algorithm in ORD works as required, but only when it is specially configured for a single customer. It is possible to configure the TD-SP such that it functions for all customers at once, but at ORTEC the idea prevails that this will return inaccurate travel times, and therefore creates infeasible and non-optimal transport plans. With this research, we focus on developing an accurate method to determine TD-TTs, which is fast enough to be used for solving VRPs for networks, while being customer independent.

In Section 1.1 we discuss the terminology used throughout this thesis. In Section 1.2, we provide the background of this research. Section 1.3 describes the problem we want to solve. Section 1.4 describes the goal of this research. In Section 1.5, we describe the scope of this research. In Section 1.6 we provide the reader with the research approach of our research, including the research questions. Finally, Section 1.7 describes the structure of this dissertation.

1.1 Terminology

This thesis contains a lot of technical terminology. Part of this terminology is related to the current techniques used in the software of ORTEC. Even though most of the terminology is related to vehicle routing and shortest path algorithms, researchers tend to have different explanations for equal words, making definitions ambiguous. We notice that ORTEC as well has its own definitions related to transportation. To overcome this ambiguity problem, and to improve the readability of this thesis, we define the terminology as follows:

Route: The sequence of pick ups and deliveries performed by a truck or truck combination, starting and finishing at a depot.
Trip/path: Used interchangeably. Refers to the travel between a single origin and destination node. Where a "travel" only refers to the concept of something moving between two locations, a trip or path refers to the actual travelled path, or sequence of streets taken. Often used as "shortest path", which is the path with the lowest sum of the weights of the traversed edges.
Call/response: When the vehicle routing solver wants to know the travel times within a sequence of orders with a given departure time, it sends a request to the *Travel Time Calculator (TTC)*. We define this request as the "call" to the TTC. When the TTC has the associated travel times, it provides a response to the vehicle routing solver. We define this answer as the "response" to vehicle routing solver.
Query: A query is a term frequently used within the field of computer science for some kind of information retrieval. Within this research we use the term solely for the retrieval of travel times. We distinguish three types of queries; one-to-one, one-to-many, and many-to-many. The difference is the number of origins and destinations send in the query. A many-to-many query is typically used during optimization, as the travel times between multiple origins and destination can be retrieved at once.

1.2 Context Analysis

In this section, we provide the background of this research. Section 1.2.1 starts with the background of the related work on the vehicle routing problem with TD-TTs. In Section 1.2.2, we describe the travel time data ORTEC uses, and what we use in this research as well.

1.2.1 Related Work

Networks are typically modelled as directed graphs $G = (V, E)$, with n nodes and m edges. Figure 1.1 gives a representation of a graph, which is generated from a map. Nodes represent junctions and edges represent road segments, even though the opposite is possible [5]. Road networks are typical sparse and near planar graphs with short edge distances. Every edge $e \in E$ has a non-negative travel cost of t_e, which represents the cost of traversing that edge. Typically, this is the travel time it takes a vehicle, but other costs like distance costs, toll costs, fuel consumption, etc. may be included. In this research, we focus solely on the travel time of the links. Paths in the graph consist of an origin node $o \in V$, a destination node $d \in V$, and a corresponding path of $< s \rightarrow ... \rightarrow t >$ [6]. An optimal path in graph G is a path with a minimal total travel time. In case of route planning with free flow travel times, a single value is assigned to every t_e of $e \in E$. However, in route planning with TD-TTs, a *Travel Time Function (TTF)* $t_e(\tau)$ of $e \in E$ is assigned, where $t_e(\tau)$ is the cost of traveling edge e when starting at time τ [6].

FIGURE 1.1: A (simple) graph representation of a map. In this example, the nodes represent cities and the edges represent the roads connecting them. The nodes in a graph used for distribution planning are on a much smaller scale. Those nodes represent road intersections and the arcs are the roads connecting them.

Kok et al. [3] studied the performance of four different congestion avoidance strategies in a real world setting. They focussed their research on the results of the strategies, not so much on the performance of the strategies themselves (meaning computational time was not of interest). To solve the TD-SPP and TD-VRP they used a TD-Dijkstra algorithm and a dynamic programming heuristic respectively. As mentioned in the introduction, their results showed that 99% of late arrivals at customers can be eliminated if one accounts for traffic congestion during the off-line planning phase. To measure the performance of different travel time strategies, they used the number of vehicle routes, total duty time, total travel distance, number of late arrivals, number of late return times, maximum late time, and total late time as indicators.

Ichoua et al. [7], Lecluyse et al. [8], and Van Woensel et al. [9] used tabu search to solve the TD-VRP. All three research teams used TD-TTs that are simplifications of the real world, specifically because no full road graph was used. Instead, a path between two nodes is simplified as a single edge with a single distance and a single TTF. The used datasets are theoretical datasets, commonly used to compare performances of algorithms. The TTFs were based on aggregated data, provided by the Belgian and UK government. They only used two or three different road categories, hence only two or three different

TTFs were considered. The edges consisted of randomly assigned road categories, meaning the TTFs were also randomly distributed over the node pairs. Retrieving the TD-TT is no more than a simple table loop up, in a n-by-n matrix. All three research teams concluded that TD-TTs show significant improvements over constant travel times, indicating the usefulness of time-dependent information.

Dabia et al. [10] and Kritzinger et al. [11] both have a similar approach as Ichoua et al. [7], only they used a *branch-and-price (B&P)* and *variable-neighbourhood-search (VNS)* algorithm respectively to solve the TD-VRP. Both used the Solomon instances as their dataset for the TD-VRP, therefore all paths between the locations consist of a single edge. Dabia et al. [10] used three different TTFs that they randomly assigned over the edges. Each TTF consist of 5 time zones, each having a single value representing a moment during the day (night, morning commute, afternoon, evening commute, and evening). Kritzinger et al. [11] only used a single TTF, which is a function of an average day on the Vienne Highways. Both showed that including TD-TTs into the VRP with time constraints provides substantial improvements in the total travel time of the routes.

Donati et al. [12] and Hashimoto et al. [13] used a similar approach as Ichoua et al. [7] with TD-TTs represented by simple table look ups. Donati et al. [12] presented the idea to calculate the TD-TT on the fly, but chose to store the required set of paths beforehand, due to increased computational effort of repeatedly calculating the shortest path. The initial road network consisted of a set of 1522 geo-referenced nodes and 2579 arcs. Within the 1522 nodes, a set of 60 customers with given demands existed. They used an *Ant Colony System (ACS)* algorithm to solve the TD-VRP, and show an average improvement in the travel time gap of 8%. The travel time gap is the percentage between the total travel time found with the TD-SP algorithm and total travel time found with a free flow SP algorithm [14]. The latter total travel time is calculated by taking the free flow shortest path, but taking the travel time of the time dependent graph. Hashimoto et al. [13] used the Solomon's benchmark instances and Gehring and Homberger's benchmark instances to test their iterated local search algorithm. They incorporated three different road categories, each with a different speed for the morning, daytime and evening period. They show that their algorithm is highly efficient.

Li et al. [14] and Mancini [15] used a local search algorithm and a *Greedy Randomized Adaptive Search Procedure (GRASP)* respectively to solve the TD-VRP. To calculate the TD-TTs during optimization, Li et al. [14] used a TD-A* algorithm to calculate the TD-SP and corresponding TD-TT. The TD-A* algorithm is used on the Los Angeles(LA) road network dataset, which contains 111,532 vertices and 183,945 edges. They solve the TD-VRP with 1000 delivery location within 20 minutes, while achieving accuracy similar to the state-of-the-art approach. Mancini [15] created six degree polynomial functions, fitting the data points of the TD-TT data. His algorithm, that consisted of a construction and local search phase and that is similar to that of ORTEC (See Chapter 3), performs 12.5% better than when using constant travel times.

To conclude, all papers on TD-VRPs we found came to the same conclusion that the inclusion of TD-TT leads to a better result than the constant TT variants. However, most research has been done on extremely simplified time dependent models, which is a nice proof of concept, but it is questionable if it is useful in practice. The only paper that included a TD-SP algorithm, other than the slow TD-Dijkstra, still had a relatively small map (LA), compared to the larger maps typically used in practice, e.g.,BeNeLux (>6 million edges), Western Europe (>42 million edges), and USA (>57 million edges)[16]. It is a step in the right direction, but we search for an algorithm that is fast and applicable to larger scale maps.

1.2.2 Travel time data

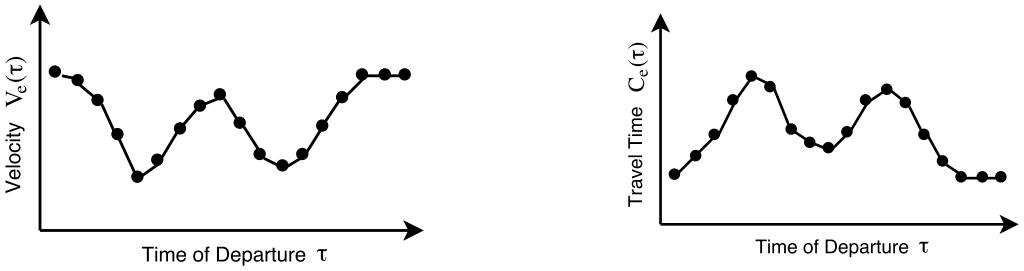
Gendreau et al. [17] propose a classification method to assess the quality and evolution of the travel time data (Table 1.1). Information evolution describes if the data changes over time. If the data is static, the travel times originates from historical *Travel Time Functions (TTFs)*. If the data is dynamic, the travel times come from a live feed connected to the current traffic situation. The information quality is whether or not the travel time is deterministic, or if it is based on travel time probability distribution functions that represent the road network's edges. ORTEC procures travel time information from a third party company that is specialized in collecting and processing data to make travel time predictions. Those travel time predictions are both historical and deterministic, leading to the travel time data that is static and deterministic.

	Information quality	
Information evolution	**Deterministic input**	**Stochastic input**
Input known beforehand	Static and deterministic	Static and stochastic
Input changes over time	Dynamic and deterministic	Dynamic and stochastic

TABLE 1.1: Taxonomy of the the quality and evolution of travel time data [17].

Travel time information is available on an individual edge level and on a 15 minute timespan, given the time of the day and day of the week. It is therefore possible to get the travel time of an individual edge at $4 * 24 * 7 = 672$ different moments of the week. The third party uses floating car data (GPS data points) to compute historical speed profiles, which is the average speed on an edge at a given time. Although in practice every edge has a unique speed profile, the third party aggregated the data to only 15,000 different speed profiles. Each edge in the graph is linked to one of these speed profiles, meaning multiple edges share the same speed profile. The speed profiles predict the travel time at a certain moment in time on any of the graph's edges. Although it is just a prediction, we consider these profiles as the "real" speeds in the road network.

ORTEC converts these speed profiles to TTFs, by dividing the edge distances by the speeds in the speed profile. A TTF consists of continuous piecewise linear functions, that are formed when connecting the travel time values. Figure 1.2 shows an example of a speed profile and the converted TTF. The TTFs all fulfill the *First In First Out (FIFO)* property, which means that no driver arrives earlier if they would depart later. This means none of the linear functions within a TTF, can have a slope smaller than -1. This property is important, as the problem of finding a time dependent shortest path in a FIFO network is polynomial solvable, while the problem gets NP-hard in non-FIFO networks [18].



(a) Speed profile

(b) Travel time profile

FIGURE 1.2: (a) Possible speed profile over a day of a single edge e in graph G. ORTEC acquires the speed profiles from a third party. (b) The travel time profiles results from the speed profile.

1.2.3 Current approach

So far we described two OR problems, the VRP and the SPP. The software program ORD is able to solve these two problems, giving customers the ability to plan their distribution processes efficiently. ORD uses construction, destruction, and local search heuristics to automatically solve the VRP, and thereby creating a near optimal set of routes. In essence, these heuristics generate an enormous amount of possible plans, and at the end select the plan with the best set of routes. Transport planning at companies almost always includes time constraints, e.g., time windows at locations and drivers legislation. Therefore, to solve the VRP and make sure no constraints are violated, heuristics need (time-dependent) travel times of shortest paths between all pairs of consecutive locations within the routes.

The ORD software is divided into multiple components. We focus on the components Vehicle Routing Algorithm and Shortest Path Algorithm. We call these components *COMTEC Vehicle Routing System (CVRS)* and the *Travel Time Calculator (TTC)* respectively. The latter contains more functionality than solely calculating (TD)-TTs, but in this research we solely focus on the TD-TTs. We describe the CVRS and the TTC in Chapter 3.

The algorithm in the TTC returns an estimate of the TD-TT, using a tailor made method called the representative approach. This estimate favours a fast response, as speed is an important factor within solving the VRP. In theory, it is possible to get the exact TD-TT for each node pair in the graph using the representative approach, but this increase of the accuracy in the method quickly results in the need for an unusable amount of main memory. Currently, it is decided that the main memory usage of the TTC cannot exceed the 1000 MB. A detailed description of the representative approach is given in Section 3.2.

Getting the TD-TT between two locations is possible using an exact approach, but it takes a significant amount of time to calculate it. The idea prevails that the current TD-SP algorithm does not return accurate travel times, and therefore creating infeasible and sub-optimal transport plans.

1.3 Problem Description

Our problem is strongly related to the *Time Dependent Shortest Path Problem (TD-SPP)*. Major difference is that it has to be applicable for vehicle routing algorithms. This means the algorithm

for solving the TD-SPP needs to be fast, and the algorithm only has to return the TD-TT and not necessarily the shortest path itself. For a more formal problem description of the TD-SPP, we refer the reader back to first part of Section 1.2.1.

The representatives method proves to be a fast algorithm for TD-TT queries. The major drawback of this approach is that it is only an estimation of the TD-TT based on the static and deterministic TTFs. Estimating the TD-TT results in a difference between what the vehicle routing algorithm (CVRS) uses and what is actually true based on the historical TTF. A significant discrepancy in the travel time causes the route planning to be unrealistic and/or non-optimal. When the estimated travel times are too short, it causes drivers to be late and miss the agreed delivery times. When the estimated travel times are too long, we create unnecessary slack within the schedule.

We do not know what the effects are of using the representatives method. Therefore, we do not know if *i*) the travel times are indeed sufficiently inaccurate, and *ii*) what the effects are on the functioning of CVRS. This means, that it is possible that the travel times are actually quite accurate and it has no effect or only a small effect on the quality of the transport plans of CVRS. It is also possible that the estimation of the TD-TTs is poor, but that it has no effect on the quality of the transport plans.

1.4 Research Goal

The research goal of this study is to develop an improved algorithm that calculates the travel time between two points on a map, for any departure time. The algorithm has to be more accurate than the current implementation, without increasing the computation times too much and preferably with less memory use than the current method.

1.5 Research Scope

In this section, we specify some boundaries of our research. This research focusses on TD-TTs based on static and deterministic TTFs. Realized travel times are outside the scope of this research, we are not interested in improving the provided TTFs. This data is provided by a third-party company and we see it as the ground truth. It is their responsibility to provide as reliable data as possible. Also, dynamic travel times are outside the scope, meaning we solely focus on TTFs that are based on historical data. We are already able to extract the exact TD-TTs from the TTFs, using time-dependent Dijkstra's (see Section 2.1). This comes at the cost of large computation times, something that is not favourable for customers, but it is good enough for research.

We focus on finding a fast method for extracting the TD-TTs when CVRS requests them. That means that we are searching for a fast TD-SP algorithm, which is as accurate as possible. Although routing is closely related to our research, we do not focus on changing the current behaviour of CVRS. Different approaches and strategies of the vehicle routing algorithm are outside the scope of this research. The TTC functions as required for individual customers, but we want to develop an algorithm that functions customer independent. This means that we cannot use any information and/or data that is specific to a single customer, to configure our algorithm and improve the accuracy.

1.6 Research Approach

To come to an appropriate answer for the problem and to reach the goal of this research, we formulate a number of research questions. We present our research questions, each with a small introduction to motivate its importance. Finally, we give an overview of our research approach.

First, we want to research what is currently known in literature about our research problem.

1. What is currently known in the literature about the use of TD-TTs in vehicle routing problems?
   (a) What kind of shortest path algorithms can be used to calculate TD-TTs from a weighted graph where the non-negative edge weights are time-varying?
   (b) What kind of algorithms are closely related to the TTC?

Second, the TTC has never been evaluated thoroughly. Before developing a new algorithm, we want to assess the accuracy of the TTC. We want to know if it is even necessary to develop a new one. In addition, this helps us to get a better understanding in which situations the TTC performs better than in others.

2. How accurate are the calculated TD-TTs from the TTC?
   (a) What is the difference between the exact TD-TT and the TD-TT calculated by the TTC?
   (b) How does the number of representatives affect the accuracy of the TTC?
   (c) How does the location of the origin and destination of a path affect the accuracy of the TTC?

Third, we design an alternative algorithm that quickly calculates the TD-TTs to be used in the VRP. It is important that it fits the current ORD framework as well.

3. What algorithm is suitable to calculate the TD-TT of the shortest path quickly within a weighted graph where the non-negative edge weights are time-varying?

Fourth, we want to know the accuracy and the performance of the developed algorithm. We evaluate our algorithm by comparing it to the TTC, both on the relative travel time gap as computational speed. We use the same datasets and evaluation criteria as used in research question 2.

4. What is the accuracy and performance of the developed algorithm?

1.7 Research Outline

The remainder of this thesis is organized as follows. In Chapter 2, we review the related work on both shortest path algorithms as vehicle routing algorithms that use time-dependent travel times. We use this literature review as basis for the development of the new algorithm. In Chapter 3, we describe the implementation of the TTC, to give an insight on the functioning of the software. We research the accuracy of the current algorithm in Chapter 4, by running a total of five experiments. The outcomes of these experiments also function as input for the development of the new algorithm. In Chapter 5, we present our developed algorithm, the *Congestion Hierarchies-algorithm (CH-algorithm)*. Chapter 6 shows the results of the experiments that were carried out to measure the performance of the CH-algorithm. Finally, Chapter 7 describes our conclusions, discussions, and an outlook for further research.

Chapter 2

Literature Review

This chapter describes the current state of literature related to our research. The following sections describe the path finding algorithms that are currently known within the literature. We compare these techniques using three variables, namely: speed-up, preprocessing time, and space overhead [6]. Speed-up is the factor to which the query time of a path find algorithm is faster than Dijkstra's algorithm. The preprocessing time is the time needed to pre-process the representation of the graph used by the specific technique. Space overhead is the memory usage needed to store the representation of the graph. Often, techniques can be tuned among these variables, resulting in a trade-off between the three.

Section 2.1 describes the basic techniques for path finding within a graph. In Section 2.2, we look at the path finding techniques that use hierarchies to speed-up the query time. In Section 2.3, we discuss labelling algorithms that store information on nodes, for the retrieval of shortest paths or to successfully prune edges during the path search. Finally, in Section 2.4 we focus on path finding techniques that include time dependencies within the graph.

2.1 Basic shortest path algorithms

In this section, we discuss several path finding algorithms. The core of every algorithm is the in 1959 developed Dijkstra's algorithm, which guarantees to find the shortest path in any graph. First we discuss Dijkstra's algorithm, and subsequently algorithms that add speed-up techniques, or add information to the graph to speed up the process as well.

Dijkstra: Already in 1956, Dijkstra developed an algorithm to determine the optimal path between two locations in a network [19]. The optimal path is the path with the least resistance between two vertices in a graph, which can be measured in, e.g., distance or time. Because the algorithm is relatively fast while giving optimal solutions, the algorithm is still used nowadays in many different type of routing problems.

The algorithm keeps a priority queue Q of all nodes in the graph, ordered by the total distance from starting point s. All node-to-node distances are initialized to infinity, except the distance from node s to node s, which is set to 0 and added to queue Q. During every iteration, the algorithm picks node u from the top of queue Q (node with least distance), and starts assessing all outgoing edges to all neighbour

nodes. For each edge, it determines the distance from node s, via node u, to the node v at the other end of the edge. If the distance s to u, plus the length of the edge, is shorter than the current value of node v, it updates the value of node v. Afterwards, the updated node is added to the priority queue Q. All visited nodes, until the target node t is reached, are referred to as the search space of the Dijkstra query of node s to node t. Dijkstra is applicable for all kind of graphs, as long as the edges have non-negative values. Also, no pre-processing is necessary, making it easy to update the graph. However, in gigantic graphs the computational time of Dijkstra for finding the shortest path between start s and target t becomes too high to be used conveniently.

Bi-directional Dijkstra: The search space used by Dijkstra can be reduced using *bi-directional Dijkstra* [20]. Instead of starting only at start-node s, the bi-directional search also does a backward Dijkstra search from target-node t. A backward search is similar to the normal Dijkstra search, but instead of looking at all outgoing edges, the algorithm considers all incoming edges. In an undirected graph, the forward and backward search are identical, due to the characteristics of the graph. For road networks, bi-directional Dijkstra reduces the search space roughly to half the size of the unidirectional approach, making the algorithm twice as fast. Bi-directional Dijkstra has the same advantages and disadvantages as the regular Dijkstra. However, time-dependent path finding is not possible, as during the backward search it is known yet what the time of arrival is going to be.

A* Search: Hart et al. [21] propose a goal-directed version of Dijkstra's Algorithm. The idea of *A* Search* is to traverse the edges that are in the general direction of the target-node as early as possible. Instead of picking the node u out of priority queue Q based on solely the distance from start node s to that node u, it adds the estimated distance from node u to target node t to that value. This way, the nodes that are closer to the target, are picked first. The estimating distance function can have different implementations. A possible implementation would be to calculate the euclidean distance based on the coordinates of the nodes. In practice, A* performs poorly compared to current modern techniques [22].

Geometric Containers: Schulz et al. [23] propose another goal-directed version of Dijkstra's Algorithm, called *Geometric Containers (GC)*. The algorithm pre-computes an edge label $L(e)$ that contains information on the target nodes that have edge(u,v) on their shortest path, given node u as the start node of that shortest path. During a query, all edges that do not have target node t in $L(e)$, can be safely pruned. Because it takes up too much memory space to save all nodes in every edge containers , the container contains geometric information on all nodes that have edge e on their shortest path. The geometric information can be angular, like at Schulz et al. [23], but can also be shaped like ellipses or a convex hulls [24]. A large disadvantage is that for every node u in graph G, a one-to-many Dijkstra search has to be completed during the pre-processing phase. This algorithm is commonly used within public transport networks, and not on road networks.

Arc Flags: The last goal-directed path finding algorithm we discuss, is *Arc Flags (AF)* [25, 26]. During a pre-processing phase, the algorithm subdivides the graph into K different cells. The areas are roughly balanced in the number of nodes it contains, and have a small number of boundary edges. Each edge contains a vector C_i of length K bits, in which bit i corresponds to cell i. If the edge belongs to a shortest path to cell i, i^th bit in C_i is set to 1. During the search, the algorithm prunes the edges that do not contain the cell in which target node t belongs. The big advantage is that it is a relatively easy query algorithm to implement. In addition, it has often optimal queries, meaning it only visited those edges that are on the shortest path of the query [27].

2.2 Hierarchical shortest path algorithms

This section focusses on path finding techniques where the algorithm modifies the graph in a preprocessing stage, to ensure faster queries. They are often called hierarchical techniques, as it transforms the original flat graph into a multi layered graph to exploit the inherent hierarchy of road networks [22]. The time-dependent variant of the contraction hierarchies method is currently being researched at ORTEC to replace the highway node routing method.

Highway Hierarchies: Sanders and Schultes [28] were the first to develop an algorithm that uses the hierarchical characteristics of the road network. *Highway Hierarchies (HH)* provides fast solutions, without losing the optimality Dijkstra has. This has to do with the typical characteristics of a road network, which bounds the need to have an algorithm that is applicable for all kinds of graphs. Highway Hierarchies starts with a preprocessing phase where the graph is modified to a graph with different hierarchical layers. These represent the same hierarchies we know in our road network, e.g., local access roads are lower in the hierarchy than highways. Note that the algorithm automatically finds the most important roads within the network, so the levels do not necessarily have to match the structure of the road designer. This pre-processing phase can take up several hours, but has to be calculated only once. The hierarchical graph allows for queries of trips of about 1 ms, which is considered very fast. The drawback of HH is that the travel time data is static, because no changes can be made to the graph without running the preprocessing phase again. Therefore, a typical HH-graph consists of only the free-flow travel times and all congestion is neglected. Also, even minor changes of the road network, result in completely reprocessing the HH-graph.

Highway Node Routing: Schultes and Sanders [29] developed a successor algorithm of HH, called *(Dynamic) Highway-Node Routing (HNR)*. HNR solves the problem of having to recalculate the complete graph, even if only one edge changes. HNR allows for fast updates of minor changes in the hierarchical graph, with a speed of $2 - 40$ ms (on the Western European map). Afterwards, this allows for fast queries of about 1 ms on average. A road network does not change that often, as it takes some time to construct new roads or upgrade them. Still, the graph contains only static travel times. Updating the graph after each query is not an option, as this results in updating all arcs within the graph. The query speed seems promising, but these speeds are only reached when just a few arcs are updated. Still, HNR is a predecessor of contraction hierarchies, that is in its core much simpler to implement.

Contraction Hierarchies: Geisberger et al. [16] developed the *Contraction Hierarchies (CH)* algorithm, which is a successor of HH and HNR. It is based on the idea of placing the more important nodes (the ones often on a shortest path) higher up the contraction graph than less important nodes. CH starts with a pre-processing phase, in which the nodes are one by one contracted from the original graph, in order of least important to most important. During the contraction of one node u from the graph, for each incoming and outgoing node pair, CH checks if path $< v, u, w >$ is the shortest path from node v to node w. If so, this shortest path is added as a short-cut in the remaining graph. Afterwards, a query for a (s, t) node pair is done with a forward upward Dijkstra search from node s, and a backward upward Dijkstra search from node t. One of the nodes where the two searches meet, is the node on which the shortest path is located. It is faster and simpler that it predecessors, HNR and HH.

2.3 Labelling shortest path algorithms

This section discusses the most recent developments in path finding techniques, called labelling methods. This technique precomputes label $L(v)$ for each node v, which contains information on the shortest paths within the graph. Using these labels results in successfully pruning edges that are not on a shortest path, or directly retrieving the distance of shortest path, without looking at the input graph.

Hub Labelling Algorithms: The *Hub Labelling (HL)* method pre-computes a label to every node $n \in V$ [30, 31]. These labels contain information on the shortest path to a set of nodes, so that the labels of both nodes $L(s)$ and $L(t)$, share at least one node. This way, a shortest path can be found for each node pair (u, v), just by assessing the labels of both nodes. For directed graphs, two different label sets for each node are computed; one for all outgoing edges, and one for all incoming edges.

Transit Node Routing: Bast et al. [32] based the development of the *Transit Node Routing (TNR)* algorithm on a few key observations. First, they observed for long distance travel, a particular start location has a few important traffic junctions, for which all paths will use one of those access nodes. Second, each access node is relevant for several nodes in its proximity. The union of access points T of all nodes in the graph V is small and is called the transit node set. The algorithm has a preprocessing phase, where the algorithm identifies the transit node set $T \subseteq V$ first. Second, the complete distance table between all access nodes in the transit node sets is calculated. Finally, for each node in graph V, its access nodes are determined. A query is a simple search of the smallest distance to each access node of both the start and target node and the distance between two access nodes. TNR seems to be a good starting point for our time-dependent path finding algorithm. However, as far as we know, it has never been tested on graphs with time-dependent edge weights.

Pruned Highway Labelling: Akiba et al. [33] developed the *Pruned Highway Labelling (PHL)* algorithm, which can be seen as a hybrid between a labelling algorithm and transit node routing. First, the algorithm preprocesses the graph into several different shortest paths. For each node s in graph V a label is created, so any shortest $s - t$ path can be expressed as $< s - u - w - t >$, where $< u - w >$ is a subpath of a path P that belongs to the labels s and t [22]. PHL is one of the fastest algorithms for querying shortest path. However, it has only been evaluated on undirected graphs.

2.4 Time dependent shortest path algorithms

Within this section we focus on time dependent path finding techniques. These techniques consider the variation in travel cost during a day.

Time dependent Dijkstra: Finding the shortest path with Dijkstra's algorithm over a time-dependent graph, is just as efficient as finding the shortest path over a graph with free flow travel times [34]. However, it is necessary that the FIFO property holds on the graph, meaning that no driver can arrive earlier at its destination by departing later. Then, the only difference is that the travel time has to be determined using the current arrival time of the top node in the priority queue. The same disadvantage holds as normal Dijkstra, that it is really slow for rather large graphs like road networks. Also, bidirectional queries are not possible, because backward searches from the target nodes cannot be done as the arrival time is not yet known.

Time-Dependent Contraction Hierarchies: Contraction hierarchies already proved to be extremely efficient for graphs with static travel times. *Time-dependent Contraction Hierarchies (TD-CH)* is a variant that includes time-dependent edge weights in the road network. It is the first hierarchical path finding technique for time-dependent paths that allow for bidirectional queries. Time-dependent contraction hierarchies is extremely useful even with larger graphs, and outperforms other TD-PF algorithms in the case of considerable time-dependence on the edges (map: Germany, weekdays). Unfortunately, the current implementation requires still too much memory during the preprocessing phase to be realistic to be implemented yet.

Customizable Route Planning: Delling and Wagner [27] developed Customizable Route Planning (CRP) with the idea to make an efficient real-world routing engine. It should incorporate turn restrictions, avoidance of U-turns, avoid left turns, avoid/prefer highways, and using different modes of transportation like biking and walking. This using as little memory as possible. Still, the algorithm should allow for fast graph updates and one-to-one queries. CRP has two preprocessing phases. First, a metric-independent phase only considers the topology of the graph, which is the data that changes very infrequently like edge distance, number of lanes, etc.. The second phase, metric customization, transforms the metric-independent data into a single metric. This single metric can change quite often, therefore the second phase takes a few seconds to complete. The preprocessing phase results in multilevel nested partitions. Within these partitions, or cells, shortcuts are inserted between the boundary nodes, so queries can skip the nodes within the cells in which the start or target node are not presented. The algorithm is able to get fast queries, but not as fast as the fastest existing methods currently known. However, the queries of CRP are robust and suitable for all above defined real-world requirements. Time dependent queries are possible due to real-time traffic updates of the graph. This feature is useful for real-time planning, but the updating phase takes to long to be useful during optimization, making it irrelevant for our purpose.

2.5 Conclusion

We gave an overview of the possible techniques available to solve a travel time query. The basic path finding techniques are not fast enough to handle large amounts of time-dependent travel time queries. However, these simple techniques are often used as a part of more sophisticated algorithms, like the ones we presented in Sections 2.2, 2.3, and 2.4. Table 2.1 presents the memory space usage per node, preprocessing time, and speed-up compared to Dijkstra of the discussed algorithms. The memory space and speed-up indicators are adjusted to be experiment setup independent, by using relative values. Memory space is measured in byte per node, in which the node is a node on the map. The speed-up is measured by comparing the algorithm's running time with the running time of Dijkstra's algorithm using the same setup, making the speed of the computer irrelevant. Keep in mind that the preprocessing time is the absolute value, so differences occur due to the use of a different setup. We were not able to find all values for the discussed algorithms, these values are therefore absent in Table 2.1.

Table 2.1 shows that the basic techniques do not perform well enough to be used in route optimization. During optimization TT queries should only cost a few microseconds, while a typical Dijkstra search on a European map takes a few seconds to be completed. The hierarchical path finding algorithms do have the performance we want, but miss the essential part of being time-dependent. Keep in mind that HNR is currently used for static travel times, with a locally developed algorithm to estimate the time dependent part. Within the labelling path finding techniques, HL and TNR both seem promising because of the

method	memory space [b/node]	preprocessing [min]	speedup [comp. to Dijkstra]	source
Dijk.	24	0	1	*(1)*
BDD	24	0	2	*(1)*
A*	-	-	-	-
GC	-	-	-	-
AF	36	20	$6.2 * 10^3$	*(1)*
HH	72	13	$10 * 10^3$	*(2)*
HNR	26	15	$7.1 * 10^3$	*(2)*
CH	-	5	$23 * 10^3$	*(1)*
HL	1121	37	$4.6 * 10^6$	*(1)*
TNR	149	20	$2.0 * 10^6$	*(1)*
PHL	828	50	$2.5 * 10^6$	*(3)*
TDD	24	24	1	*(1)*
TD-CH	523	285	$1.8 * 10^3$	*(2)*
CRP	54	60	$1.5 * 10^3$	*(1)*

TABLE 2.1: Overview of the path finding algorithms. For all algorithms, the Western European map from PTV AG was used. For A* and geometric containers no data was available. Source (1): Bast et al. [22]. Source (2): Batz [6]. Source (3): Akiba et al. [33].

fast query times. However, as to our knowledge, they have never been implemented on a time-dependent graph. In our opinion, both algorithms are a good starting point for research, but we keep in mind that memory space is going to be the main issue. PHL has as major drawback that it has only been tested on undirected graphs so far, making it unsuitable for our map. CRP would be the only possible candidate within the time-dependent path finding algorithms, but its design has been optimized to make fast updates possible, at the cost of slower queries. Within our research, we do not have the need to update the graph that often, but we are interested in fast time dependent queries. Therefore, CRP does not have our interest.

Chapter 3

Current Methods

In this chapter we discuss the current *Vehicle Routing Problem (VRP)* and *Time Dependent Shortest Path Problem (TD-SPP)* solving methods of ORTEC. In Section 3.1 we give a brief overview of the working of the *COMTEC Vehicle Routing System (CVRS)*. In Section 3.2 we describe in more detail the function of the shortest path algorithm and travel time calculation within the *Travel Time Calculator (TTC)*.

3.1 Vehicle Routing Algorithm (CVRS)

This part is confidential.

3.2 Time Dependent Shortest Path Algorithm (TTC)

This part is confidential.

Chapter 4

Benchmarking

This chapter describes the experiments we use to (i) research the effects of time dependent travel times on vehicle routes and to (ii) provide a benchmark of the TTC, to later compare with our algorithm. In Section 4.1 we describe the data we use in our experiments. Section 4.2 describes the evaluation criteria used in the experiments. In Section 4.3, we describe the different experiments we conduct. Section 4.4 describes the results of the experiments. Finally in Section 4.6, we draw our conclusions based on the results.

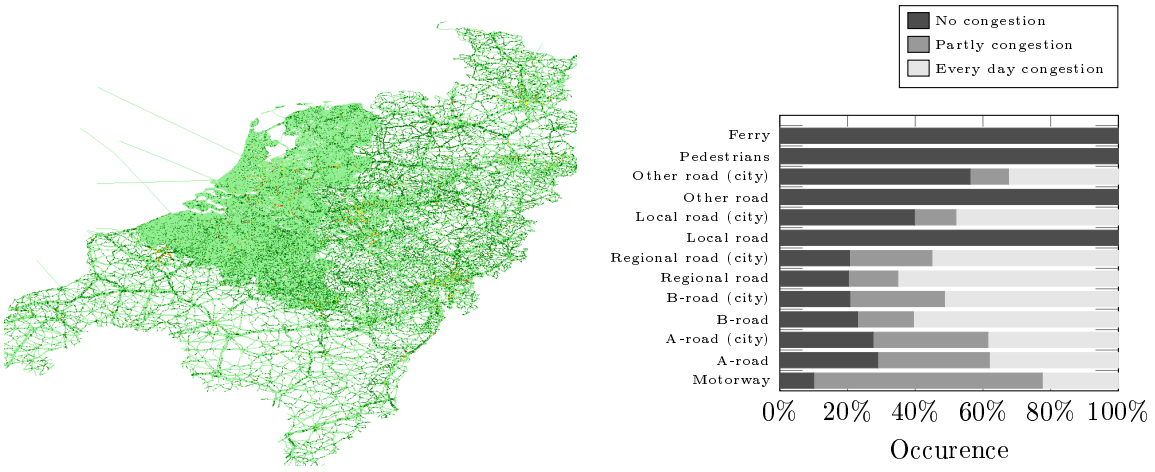
4.1 Data

This section discusses the three different types of data we need to carry out our experiments. The first subsection describes the map data, containing the graph and travel time profiles of the road network. In the second subsection we describe the test sets containing the origin and destination pairs to test the TTC and our algorithm. The last subsection describes the representatives that the TTC uses to approximate the time dependent travel times.

4.1.1 Map

The map consist of the complete road network of the BeNeLux, including the main roads of the regions of Northern France and West Germany. Figure 4.1a shows this map, including the congestion information of a Tuesday. The complete graph consist of 3,114,941 nodes and 6,636,596 edges. Each edge is connected to seven speed profiles, which correspond to the seven days in a week. In total 15,754 different speed profiles exist, meaning many edges share the same speed profile. To calculate the time dependent travel time, the speed at the time of departure over the edge is retrieved from the speed profile of that edge, and divided by the length of that edge. Within the road network, 3,727,986 edges have at least one day with a variating speed profile. This means 2,908,610 edges have no congestion, or no congestion was measured. 2,242,591 edges have congestion every day of the week. Figure 4.1b shows the percentages of the edges that have no, partly, or full congestion information. It shows that the majority of the road types have congestion data available. Figure 4.3 in Section 4.1.2 shows the zoomed road networks of four congested areas, where clearly congestion is visibly on the main road around the city. This means

that the provided map consist mainly of edges with non-constant speed profiles, making time dependent queries useful.



(a) Road network

(b) Distribution of speed profiles

FIGURE 4.1: (a) Overview of the congestion of the BeNeLux map on a Tuesday. The colors go from light green which mean no congestion via yellow, orange, red, purple, and black to the heavy congested areas. It shows that the BeNeLux part has a dense road network, while the parts in France and Germany only consist of the main roads. (b) The percentages of the edges that have no congestion, partly congestion, or have non-constant speed profiles for all 7 days. The percentages are categorized into the different road types of the graph.

4.1.2 Test sets

To effectively get a benchmark of the current algorithm (TTC) and to test the functioning of our algorithm, we define three different test groups. In total, the three test groups have 15 test sets of 2500 randomly selected Origin-Destinations pairs. We assume that a set of 2500 shortest paths is sufficient to draw conclusions based on the average values we calculate. We base all our test sets on a graph with edges containing truck speeds, as the majority of the customers of ORD use trucks as well. This means that all travel times are truck travel times.

Test group 1: Path lengths

We randomly generate seven test sets in test group 1, each consisting of 2500 different *Origin-Destination (O-D)* pairs. All O-D pairs in a single set have the same shortest free flow path length. As we explain later in this thesis, we use these test sets to study the relationship between the distance of a path and the accuracy of the TTC.

We select the O-D pairs in the test sets in the following way. First, we randomly select a node on the map which acts as the origin of the O-D pair. Second, we run the Dijkstra algorithm on the graph with free flow travel times, with the selected node as the initial start node. The Dijkstra algorithm continues until it exceeds the predetermined length of the test set. The first node after the predetermined length is reached, will be the destination node of the O-D pair. We repeat this procedure 2500 times for each test set. In the end, all seven test sets consist of different origin-destination pairs. The predetermined path lengths are 10 min, 20 min, 30 min, 60 min, 120 min, 180 min, 240 min. We select at maximum a predetermined path length of four hours, because of the size of the map prevent us from having longer trips. We name the test sets after the path lengths of the shortest path between the origin and destination

of the node pair. Hence, the test set that consists of shortest paths with a length of 10 minutes, is called "10-minute path length".

Test group 2: Vicinity of representatives
In test group 2, we randomly generate four test sets, each consisting of 2500 different *Origin-Destination (O-D)* pairs. All nodes in the test set, both origin and destination, have the same length from their representative. We use these test sets to study the relationship between the vicinity of the representatives and the accuracy of the TTC.

We use 220 representatives that are evenly distributed over the graph in a grid structure. We refer the reader to the next section for more information on the representatives. The test sets are created as follows. First, we randomly select two representatives out of the set of 220. Second, we run the Dijkstra algorithm twice, both have either one of the representatives as initial node. The Dijkstra algorithm continues until the distance between the source node and a target node exceeds the predetermined length of the test set. The two resulting target nodes of both runs, will be either the origin or destination node of the O-D pair. Afterwards, this process is repeated. In this way, all nodes of all O-D pairs within one test set have the same length towards the nearest representative. We name the test sets after the vicinity of the O-D to the representative. Hence, the test set that consist of O-D pairs that are at five minute distance from their representatives, is called "5-minute vicinity length ".

The result of the Dijkstra algorithm with a representative as initial node, will always result in the same selected node after the test set length. The limited amount of representatives compared to the number of O-D pairs in the test set causes the O-D pairs to be limited to 220 different nodes. To overcome this problem of always selecting the same node from the Dijkstra queue for each representative, we randomize the lengths by a half percent. In that way, the Dijkstra algorithm quits after slightly different lengths, resulting in selecting a different node from the queue. The randomization is low enough, to not cause major variation in the test sets due to different lengths from the representatives. The vicinity lengths of test sets are 5 min, 10 min, 15 min, 20 min.

Test group 3: Congested areas
In test group 3, we randomly generate 4 test sets, each consisting of 2500 different O-D pairs. All O-D pairs in a single set, have origin and destinations within a certain predefined area that are congested. We expect that shorter trips within a congested area creates the largest inaccuracy. We conduct a preliminary experiment to indicate which parts of the BeNeLux map have a high level of congestion. Figure 4.2 depicts the top 5% of O-D couples with the highest level of delay, given the "Path length 10 min." test set.

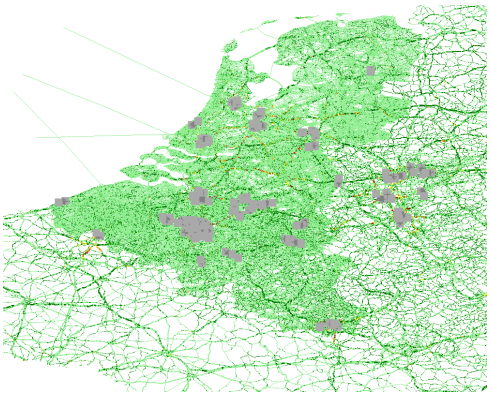
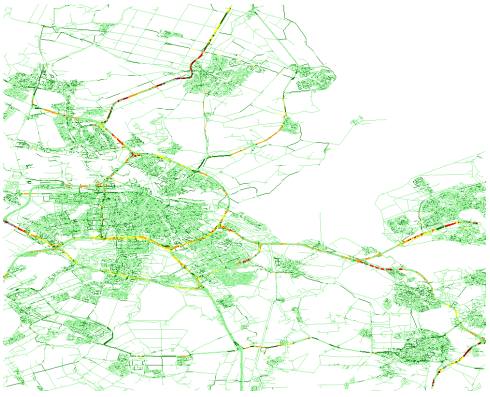
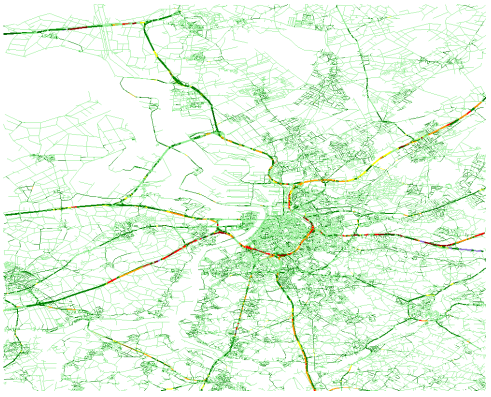


FIGURE 4.2: Displays the shortest paths of the 5% O-D couples within test set "Path length 10 min." with the highest level of delay. We observe major concentrations of paths around the major cities in The Netherlands, Belgium, Luxembourg, and the Rhine-Ruhr region.

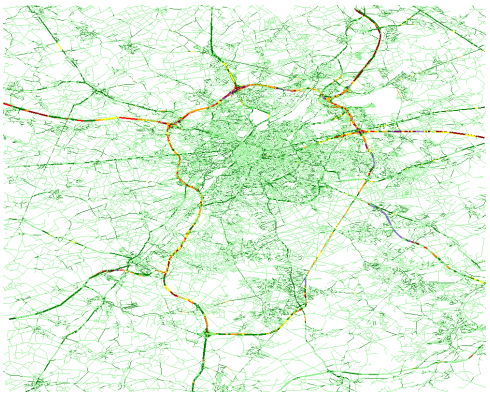
We observe major concentrations of paths around the major cities in The Netherlands, Belgium, Luxembourg, and the Rhine-Ruhr region. Luxembourg only show a few paths, and the Rhine-Ruhr region is the part of the map with a more sparse network. We make a selection of the cities in Belgium and The Netherlands, and we pick Amsterdam, Antwerp, Brussels, and Rotterdam to be the areas we select the test sets from. We use these test sets to study the effect on the accuracy of the TTC of having trips in a congested area. We name the test sets after the area they represent. Figure 4.3 shows the congestion on the road networks of the four areas.



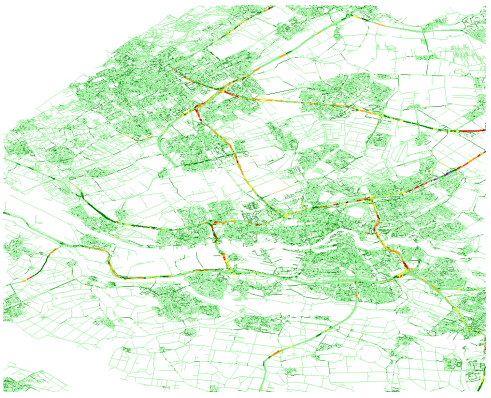
(a) Amsterdam



(b) Antwerp



(c) Brussels



(d) Rotterdam

FIGURE 4.3: Overview of the congestion of the different urbanized areas. The colors go from light green which mean no congestion via yellow, orange, red, purple, and black to the heavy congested areas. Note that it depends on the time of the day if congestion is an issue.

We select the O-D-pairs in the test sets by randomly selecting two nodes within the chosen areas. This results in O-D pairs that differ in the length of their shortest path, and the vicinity to their representative. All O-D pairs within one test set have in common that their origin and destination nodes are within a certain area.

Overview:
Table 4.1 shows an overview of all 15 test sets. Every test set consist of 2500 O-D pairs and are selected within the area of (49.316,2.581) and (53.473,7.483). Test group 3 consist of O-D pairs selected in even smaller areas, these areas are presented in column four of Table 4.1.

Testgroup 1: Path length	Testgroup 2: Vicinity length	Testgroup 3: Congested area	
10 min.	5 min.	Amsterdam	(52.18,4.74)(52.58,5.24)
20 min.	10 min.	Antwerp	(51.05,4.13)(51.45,4.63)
30 min.	15 min.	Brussels	(50.60,4.10)(51.00,4.60)
60 min.	20 min.	Rotterdam	(51.75,4.15)(52.15,4.65)
120 min.			
180 min.			
240 min.			

TABLE 4.1: Overview of the 3 test groups and 15 test sets. All test set consist of 2500 O-D pairs. Test group 1 and 2 consist of nodes selected within the area of (49.316,2.581) and (53.473,7.483). Test group 3 has different areas, the fourth column presents the coordinates of these areas.

4.1.3 Representatives

We use both representative (grid and address) strategies for our benchmarking experiments. The grid strategy is normally used for demo purposes only, while the address strategy is implemented at clients. However, we find it useful to use the grid strategy for our experiments as it provides an independent set of representatives that is not related to a particular set of addresses. Besides, it is unknown how the grid strategy performs, so it provides useful insights. The grid strategy selects representatives within the area of (49.316,2.581) and (53.473,7.483). These make roughly the outside border of the BeNeLux. This means that the edges outside of the BeNeLux are mainly mapped to the representatives at the border. The selected representatives are evenly distributed over the map, creating a grid-like structure.

In total, we select four sets of representatives using the grid strategy, this being, 57 (10 by 10), 220 (20 by 20), 519 (30 by 30), and 889 (40 by 40). The number of representatives is lower than the multiplication of the number of grids, which is caused by areas in the map where no roads are present. This mainly happens at the Dutch and Belgium part of the North sea that is within the selected area of the BeNeLux. We select two sets with 5, 10, 15, 20, and 30 representatives using the address strategy. The addresses for both sets come from the selected O-D pairs of the Antwerp and Brussels test sets.

4.2 Evaluation criteria

In this section, we define the criteria we use to evaluate the accuracy of the travel time algorithm. First, we define the notations that helps us explain the criteria. Please note, that we continue using the notations of Section 1.2.1.

Consider a node pair u, as a pair consisting of an origin node o and destination node d, where both nodes $o, d \in V$. Let U be the set of one or multiple node pairs u. Path p_{od} is a path between o and d, where path $< o \rightarrow ... \rightarrow d >$ consists of nodes that are within graph $G = (V, E)$. When $o, d = u$, then $p_{od} = p_u$. P_{od} is the set of all path p_{od} between o and d and P_U is the set of all paths of all od pairs in set U.

Path q_{od} is the shortest path between o and d, where all edges $e \in E$ of graph G have a constant travel time t_e. We define this constant travel time as the free flow travel time t_e^0 of edge e. We define Q_{od} as the set of shortest paths q_{od}, which in this case means $q_{od} = Q_{od}$ as there is only one shortest path. Q_U is the set of all shortest paths of all od pairs in set U.

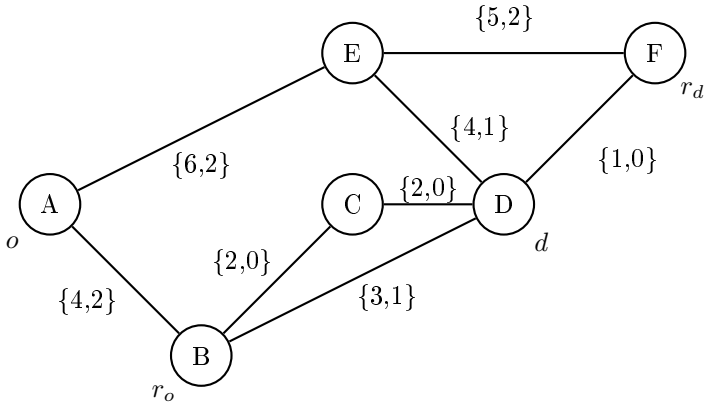
Path $r_{od\tau}$ is the shortest path between o and d at departure time τ, where all edges $e \in E$ of graph G have a *Travel Time Function (TTF)* $t_e(\tau)$. $t_e(\tau)$ is the cost of traveling edge e when starting at time τ. $R_{od\tau}$ is the set of all paths $R_{od\tau}$ between o and d at departure time τ. $R_{U\tau}$ is the set of all paths of all od pairs in set U at departure time τ.

Next, we define three different variants of the travel time between two nodes o and d. First, we define $T^0(p_{od})$ as the free flow travel time over path p_{od}. Second, we define $T(q_{od}, \tau)$ as the time dependent travel time over shortest path q_{od} at departure time τ. Note that q_{od} is the shortest path over edges with t_e^0, while the travel time is calculated with the $t_e(\tau)$ of the shortest path edges. Last, we define $T(r_{od\tau}, \tau)$ as the time dependent travel time over shortest path r_{od} at departure time τ.

We decide to use three evaluation criteria to evaluate the performance of the TTC. The criteria are the level of delay, the travel time gap, and shortest path share with shortest path of representatives. We discuss them in the following paragraphs. To clarify the criteria, we use an example graph, we present in Figure 4.4. This graph consist of six nodes and eight edges. Node o represents the origin and node d represents the destination. Node r_o and r_d represent the two representatives in the graph. Each edge in the graph has a constant travel time and a time dependent travel time, represented by $\{t_e^0, i \rightarrow t_e(\tau)\}$. We retrieve the $t_e(\tau)$ from the matrix $t_{\tau,i}$, by looking at the i value of the edge and departure time τ at which the edge is traversed.

Level of delay:
The level of delay is the difference in percentage between the time dependent travel time and the free flow travel time. It is an indicator of congestion, as a higher delay indicates more congestion. Consider $T(q_{od}\tau)$ as the time dependent travel time and $T^0(q_{od})$ as the free flow travel time. We define the level of delay between the nodes o and d at departure time τ as $LoD_{od}(\tau)$. Therefore, the equation is:

$$LoD_{od}(\tau) = \frac{T(q_{od}, \tau) - T^0(q_{od}, 0)}{T^0(q_{od}, 0)} \tag{4.1}$$



(a) Graph

$$t_{\tau,i} = \left| \begin{array}{ccc} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 3 & 7 & 5 \\ 3 & 7 & 5 \\ 2 & 6 & 5 \\ 2 & 6 & 5 \\ 1 & 5 & 4 \\ 1 & 5 & 4 \end{array} \right|$$

(b) Travel time functions

FIGURE 4.4: (a) A graph consisting of 6 nodes and 8 edges. Each edge consist of $\{t_e^0,\, i \to t_e(\tau)\}$. The first value is the free flow travel time, the second value refers to column in the matrix of (b). Thus, multiple edges share the same travel time function.

To calculate the average level of delay over a set of node pairs U, we use the following equation:

$$\overline{LoD_U}(\tau) = \frac{\sum_{od \in U} \Big(T(q_{od}, \tau) - T^0(q_{od}) \Big)}{\sum_{od \in U} \Big(T^0(q_{od}) \Big)} \tag{4.2}$$

To clarify, we provide an example using Figure 4.4. We calculate the level of delay between node A (origin o) and node D (destination d) at departure time 0. The shortest path q_{od} is $< A \to B \to D >$, which has a $T^0(q_{od})$ of $4+3=7$. The $T(q_{od}, 0)$ uses the same shortest path, but uses the time dependent travel times of the edges. Therefore, $T(q_{od}, 0) = 4+6 = 10$. The level of delay is therefore $\frac{10-7}{7} = 0.429$.

Travel time gap:

The travel time gap is the difference between the time dependent travel time and the travel time calculated by the TTC [14]. A smaller gap means a smaller difference between the calculated travel time and the actual travel time and thus yields higher accuracy. There are two different ways to calculate the travel time, using two different ways to compute the shortest path between o and d. Therefore, we have two different travel time gap criteria as well. We define the travel time gap between o and d over the free flow shortest path at departure time τ as:

$$TTG_{od}^{ff}(\tau) = \left| \frac{TTC(od, \tau) - T(q_{od}, \tau)}{T(q_{od}, \tau)} \right| \tag{4.3}$$

Therefore, we have two different travel time gap criteria as well. We define the travel time gap between o and d over the time dependent shortest path at departure time τ as:

$$TTG_{od}^{td}(\tau) = \left| \frac{TTC(od, \tau) - T(r_{od}, \tau)}{T(r_{od}, \tau)} \right| \tag{4.4}$$

To calculate the average travel time gaps over a set of node pairs U, we use the following equations:

$$\overline{TTG_U^{ff}}(\tau) = \frac{\sum_{od \in U} \Big(|TTC(od, \tau) - T(q_{od}, \tau)| \Big)}{\sum_{od \in U} \Big(T(q_{od}, \tau) \Big)} \tag{4.5}$$

$$\overline{TTG_U^{td}}(\tau) = \frac{\sum_{od \in U} \Big(|TTC(od, \tau) - T(r_{od}, \tau)| \Big)}{\sum_{od \in U} \Big(T(r_{od}, \tau) \Big)} \tag{4.6}$$

To clarify, we provide an example using Figure 4.4. We calculate the TTG^{ff} between node A and node D at departure time 0. The shortest path q_{od} is $< A \rightarrow B \rightarrow D >$, which has a $T^0(q_{od}, 0)$ of $4 + 3 = 7$. The $TTC(A, B, 0)$ returns a value of 8. Therefore, $TTG_{od}^{ff}(0)$ is $|\frac{8-7}{7}| = 0.142$.

We calculate the TTG^{td} using the same node pair and departure time. The shortest path r_{od} is $< A \rightarrow B \rightarrow C \rightarrow D >$, which has a $T(q_{od}, 0)$ of $4 + 6 = 10$. Because the TTC is a black box, let us in this example assume the $TTC(A, B, 0)$ returns a value of 8. Therefore, $TTG_{od}^{td}(0)$ is $|\frac{8-10}{10}| = 0.2$.

Calculating the $TTG_{od}^{td}(\tau)$ is a computational difficult task and this takes a tremendous amount of time. This makes extensive testing impossible, as it simply takes too much time. We argue that it is accurate enough to base our final results and conclusions solely on the $TTG_{od}^{ff}(\tau)$ instead of the $TTG_{od}^{td}(\tau)$. We conduct a preliminary experiment to research the loss in accuracy using the $TTG_{od}^{ff}(\tau)$ instead of the $TTG_{od}^{td}(\tau)$. For this experiment, we use test group 1 that is reduced from 2500 O-D pairs to 250 O-D pairs, as using 2500 would takes too much time to compute. We use two representing weekdays (Tuesday and Thursday) and two representing moments during morning and afternoon rush hour (8:00 and 17:00). The preliminary experiment shows that the implications are relatively small, and stay well beneath 2.5%. Figure 4.5 shows the average absolute difference between the time dependent travel time over the free flow shortest path and the time dependent travel time over the time dependent shortest path.

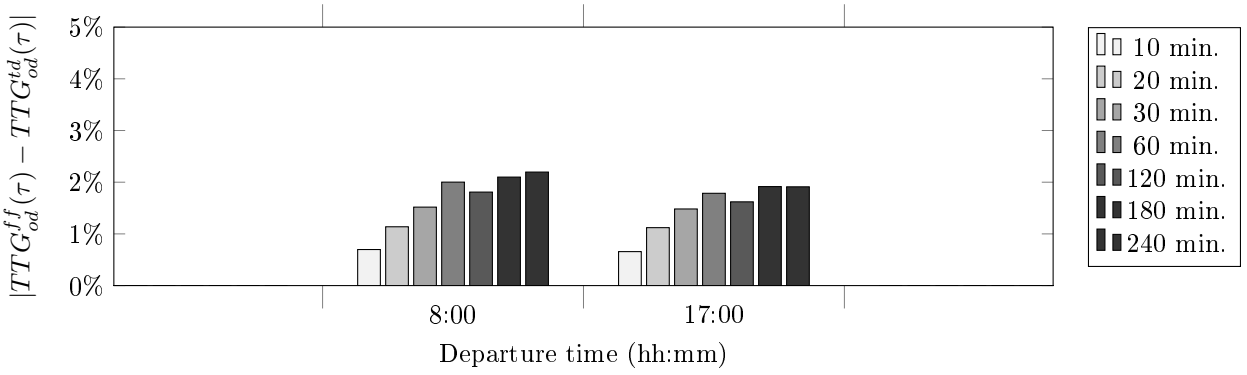


FIGURE 4.5: Percentage difference of the absolute difference between the $TTG_{od}^{ff}(\tau)$ and $TTG_{od}^{td}(\tau)$ of 250 O-D pairs per test set. The O-D's in the test sets differ in free flow path length between the origin and destination.

Shortest path share with shortest path of representatives:

This evaluation criterium indicates the percentage of shared shortest path between node o and d, and their representatives r_o and r_d. We calculate this by summing the edge lengths that are in both shortest

paths, and dividing it by the total length of all unique edges present in both shortest paths. We define the *Path Sharings Percentage (PSP)* as:

$$PSP_{od} = \frac{\sum_{e \in q_{od} \cap e \in q_{r_o r_d}} \left(t_e^0\right)}{T^0(q_{od}, 0) + T^0(q_{r_o r_d}, 0) - \sum_{e \in q_{od} \cap e \in q_{r_o r_d}} \left(t_e^0\right)} * 100\% \tag{4.7}$$

To clarify, we provide an example using Figure 4.4. The shortest free flow path between o and d is $< A \rightarrow B \rightarrow D >$. The shortest free flow path between r_o and r_d is $< B \rightarrow D \rightarrow F >$. The total free flow travel time between A and B equals $4 + 3 = 7$, while the total free flow travel time between r_o and r_d is $3 + 1 = 4$. Edge BD is present in both shortest free flow paths, with a length of 3. Therefore, the shared percentage is $\frac{3}{7+4-3} = 0.429$.

4.3 Setup of the experiments

In the following subsections, we describe the experiments in more detail.

4.3.1 Experiment 1: The effect of the path length on the time dependent travel time over different departure times during the day

Introduction: In the first experiment, we study the effect of different path lengths on the variability of the time-dependent travel time. We expect that an increase in path length leads to less variation of the travel time over the departure times, as well as a decrease in the travel time gap. We use the "Path lengths" test set for this experiment.

Calculation: We calculate for each O-D pair and each 15-minute interval, the time dependent travel time over the free flow shortest path. We compare the time dependent travel time with the free flow travel time to see the absolute difference (Travel Time) and the relevant difference (Level of Delay). We use this to research whether the paths follow a congestion pattern with a clear morning and afternoon peak. Afterwards, we retrieve the travel times calculated via the TTC. The difference between those travel times and the time dependent travel times is the so-called travel time gap. We expect that an increase of congestion leads to an increase of the travel time gap. This means, the travel time gap increases during the morning and afternoon rush hour. We take the average over two workdays (Tuesday and Thursday) and two weekend days (Saturday and Sunday) to see the effects of congestion over the week as well.

Experiment 1.1	
Test sets	Path lengths: 10 min, 20 min, 30 min, 60 min, 120 min, 180 min, 240 min
Representatives	20 by 20, grid strategy
f Map	BeNeLux
Criteria	Level of Delay, Travel Time Gap

TABLE 4.2: Overview of experiment 1.1

4.3.2 Experiment 2: The effect of the vicinity of the representative on the time dependent travel time over different departure times during the day

Introduction: In the second experiment, we study the effect of the vicinity of a representative on the travel time gap. We expect that paths that have their start and target nodes closer to their corresponding representatives have a lower travel time gap.

Calculation: We calculate for each O-D pair and each 15-minute interval, the time dependent travel time over the free flow shortest path. We compare the time dependent travel time with the free flow travel time to see the absolute difference. The time dependent travel times should follow a congestion pattern with a clear morning and afternoon peak. The time dependent travel times of all four vicinity test sets should be equal, as they are randomly selected. Afterwards, we retrieve the travel times calculated via the TTC. The differences between those travel times and the time dependent travel times is the so-called travel time gap. We take the average over two workdays (Tuesday and Thursday) and two weekend days (Saturday and Sunday) to see the effects of congestion over the week as well.

Experiment 2.1	
Test sets	Vicinity length: 5 min, 10 min, 15 min, 20min
Representatives	20 by 20, grid strategy
Map	BeNeLux
Criteria	Travel Time Gap

TABLE 4.3: Overview of experiment 2.1

4.3.3 Experiment 3: The effect of congestion on the time dependent travel time over different departure times during the day

Introduction: In the third experiment, we study the effect of congestion on the travel time gap. We expect that when congestion occurs on a path, that path has a higher travel time gap. Urbanized areas are generally more congested than rural areas, therefore we limit ourselves to known areas that are urbanized, and select our O-D pairs within these areas.

Calculation: We calculate for each O-D pair and each 15-minute interval, the time dependent travel time over the free flow shortest path. We compare the time dependent travel time with the free flow travel time to see the level of delay. We expect the level of delay over a day to have a clear morning and afternoon rush hour peak, which is a typical congestion pattern. Afterwards, we retrieve the travel times calculated via the TTC to calculate the travel time gap. We expect the travel time gap to be higher at the congested places and moments. We take the average over two workdays (Tuesday and Thursday) and two weekend days (Saturday and Sunday) to see the effects of congestion over the week as well.

Experiment 3.1	
Test sets	Amsterdam, Antwerp, Brussels, and Rotterdam.
Representatives	20 by 20, grid strategy
Map	BeNeLux (The areas of the test sets are a sub portion of the map)
Criteria	Level of Delay, Travel Time Gap

Table 4.4: Overview of experiment 3.1

4.3.4 Experiment 4: The effect of the number of representatives on the time dependent travel time over different departure times during the day

Introduction: In the fourth experiment, we study the effect of the number of the used representatives. We expect that a higher number of representatives results in a lower travel time gap.

Calculation: To study the effect of the number of representatives on the travel time gap, we take the test sets with the highest levels of delay because these are better for comparison. Therefore, we use the "Brussels area" and "10-minute path length" test set. We vary the number of representatives to research the effects. The four sets of representatives are all selected using the grid strategy and consist of 57 (10x10), 220 (20x20), 519 (30x30), and 889 (40x40) representatives. We refer the reader back to Section 4.1.3 for a more detailed explanation of the sets of representatives. We calculate for each O-D pair and each 15-minute interval, the travel time gap. We only accumulate two workdays (Tuesday and Thursday), because we are not interested in the effects of the different levels of congestion over the week.

Experiment 4.1	
Test sets	Brussels area, Path lengths: 10 min
Representatives	10 by 10, 20 by 20, 30 by 30, and 40 by 40 grid strategy
Map	BeNeLux (The areas of the test sets are a sub portion of the map)
Criteria	Travel time gap

Table 4.5: Overview of experiment 4.1

Calculation: In the second part of the experiment, we use the address strategy to compute representatives. We take two congested smaller areas as a relatively small number of representatives will have a higher impact on the travel time gap. We use the "Antwerp Area" and the "Brussels area" test set. We vary the number of representatives to research the effects of the address strategy of selecting representatives. We refer the reader back to Section 4.1.3 for a more detailed explanation of the sets of representatives. We calculate for each O-D pair and each 15-minute interval, the travel time gap. We only accumulate two workdays (Tuesday and Thursday), because we are not interested in the effects of the different levels of congestion over the week.

Experiment 4.2	
Test sets	Antwerp area, Brussels area
Representatives	5, 10, 20, 50, 100, 250 address strategy
Map	BeNeLux (The areas of the test sets are a sub portion of the map)
Criteria	Travel time gap

Table 4.6: Overview of experiment 4.2

4.3.5 Experiment 5: The effect of the percentage of the shortest path shared on the travel time gap

Introduction: In the fifth experiment, we study the effect between the free flow travel time gap and *Path Sharings Percentage (PSP)*. We refer the reader back to Section 4.2 for the explanation of both criteria. Our expectation are that a higher PSP results in a lower travel time gap, as the shortest path of the O-D is better represented.

Calculation: To study the effect of the percentage of the shortest path shared on the travel time gap, we take two test sets with smaller path lengths and two with longer path lengths. Longer paths have a higher probability of sharing a high percentage of the shortest path, because as the path reaches the highway network, it is more likely they use the same path from there. The shorter paths have lower shared percentages, as the time to reach the shortest path of the representative is on average a larger part of the total length of the path between two nodes. Also, the probability of having a complete different shortest path than between the representatives is higher. We use the 20 by 20 representative set and we only accumulate two workdays (Tuesday and Thursday), because we are not interested in the effects of the different levels of congestion over the week. We take the highest travel time gap found within these two days to compare to the shared percentage of the shortest path.

Experiment 5.1	
Test sets	Antwerp area, Brussels area, Vicinity length: 5 min, Path lengths: 120 min
Representatives	20 by 20 grid strategy
Map	BeNeLux (The areas of the test sets are a sub portion of the map)
Criteria	Shortest path share with shortest path of representatives, Travel time gap

Table 4.7: Overview of experiment 5.1

4.4 Results

In this section we present the results of our experiments. We present the results in the same order as we described the five experiments in Section 4.3.

4.4.1 Experiment 1: The effect of the path length on the time dependent travel time over different departure times during the day

In the first experiment we researched the effect of the length of the path on the time dependent travel time. In Figure 4.6 we present the average time dependent travel time over the 2500 O-D pairs in each of the seven different test sets of path lengths, for both two weekdays and two weekend days. The weekday data (Figure 4.6a) depicts the morning and afternoon rush hour as a peak in the average travel time at around 7:00 and 15:00. These peaks are earlier than we expected, but this is due to the graph showing the departures of the paths. For example, if a trip of 4 hours departs at 15:00, the majority of the travel would be inside the afternoon rush hour, that is generally between 15:30 and 18:30. The weekend day data (Figure 4.6b) presents an increase of travel time during the day, but there is an absence of the rush hour peaks.

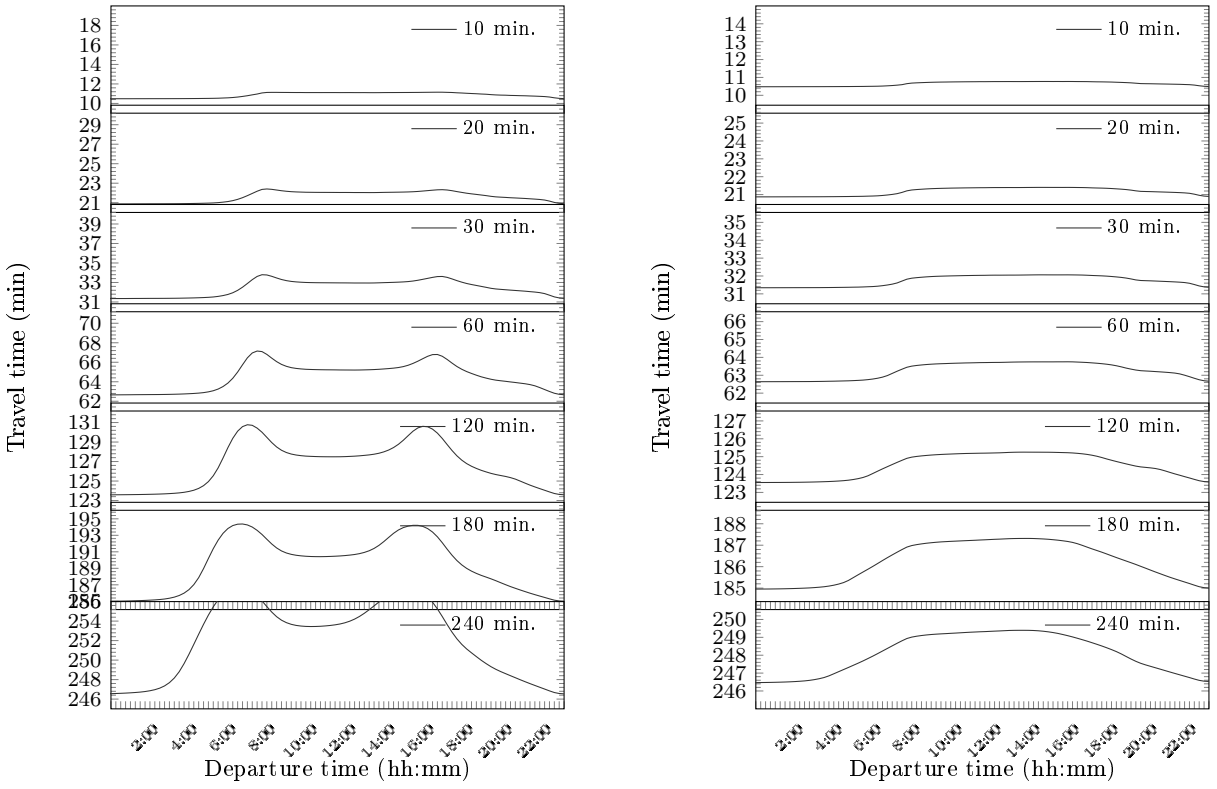


FIGURE 4.6: Average time dependent travel time of a truck, over paths with the same free flow travel time. (a) Is the average of a Tuesday and a Thursday. (b) Is the average of a Saturday and Sunday.

Figure 4.7 depicts the same test sets as in Figure 4.6, but as the percentage differs between the time dependent and free flow travel time. We observe the following;

First, we observe that the average travel time of random paths increases when congestion is applied. On average, we observe an increase of travel time between 6% and 12% due to congestion during the rush hour peaks. Second, both graphs in Figure 4.7 show the same patterns as the graphs in Figure 4.6. The weekday data show clearly the morning and afternoon rush hour peak, while during the weekend days we see only the effect of more traffic during daytime. Third, we see that as the path length decreases, the delay percentage decreases as well. This means that the congestion has relatively less influence on the travel time of longer paths, than that of shorter paths. Last, we observe a delay percentage during the night, while no traffic is expected and free flow condition should apply.

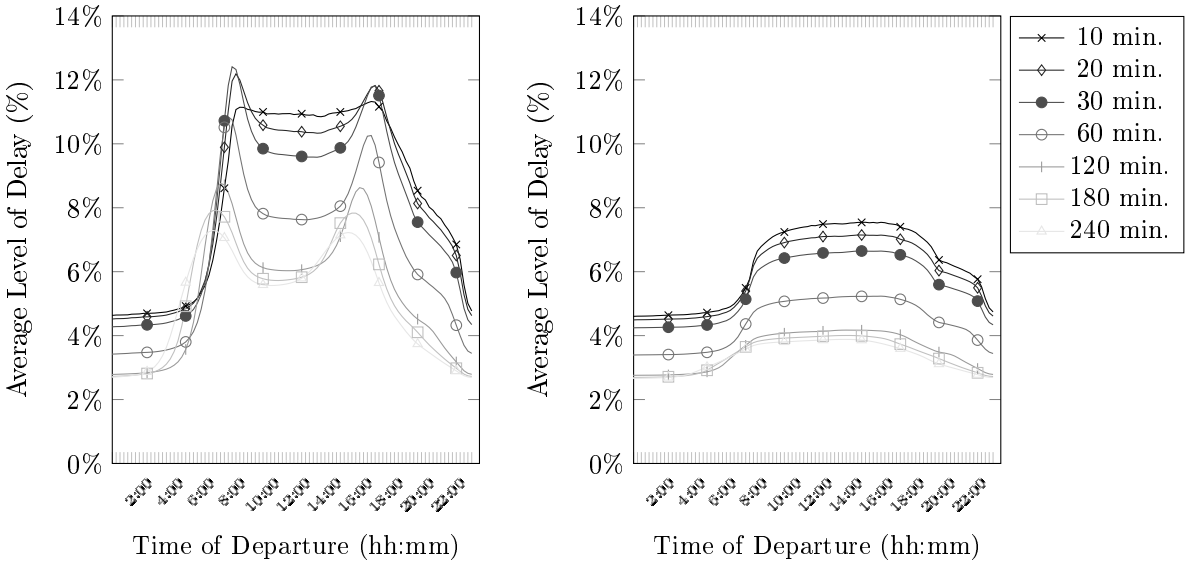


FIGURE 4.7: This figure shows the same average time dependent travel time as in Figure 4.6, but in percentage of the free flow travel time of each path. (a) Is the average of a Tuesday and a Thursday. (b) Is the average of a Saturday and Sunday.

Figure 4.8 presents the average free flow travel time gap between the time dependent travel time and the travel time calculated by the TTC. We observe that the travel time gap data shows the same pattern as the level of delay data. Another observation is that the travel time gap is on average two percent point lower than the level of delay. We conclude that the TTC using the 20 by 20 grid strategy results in 25% to 60% more accurate results. This is the relatively decrease between the average level of delay and average free flow travel time gap. Still, the time dependent travel times of the TTC have on average an error between the 2% and 9%. Note that up to 0.5% of the error is due to the rounding errors of the congestion factors. Because these numbers are on a scale between 0 to 255, with an average of 100, accuracy is lost.

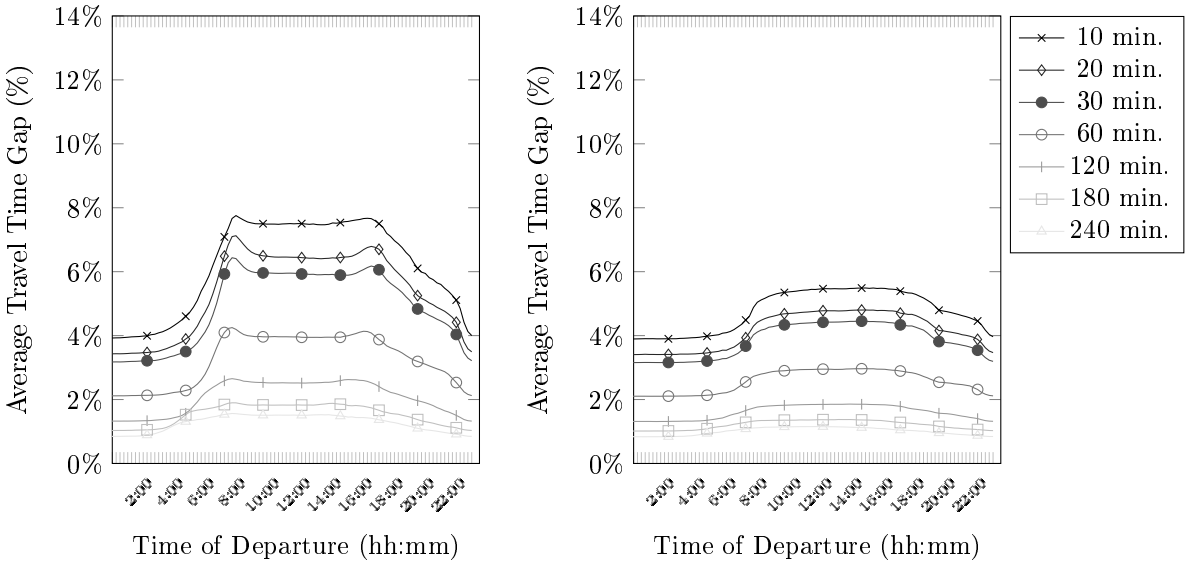


FIGURE 4.8: This figure depicts the travel time gap of the "Path length" test set. (a) Is the average of a Tuesday and a Thursday. (b) Is the average of a Saturday and Sunday.

In Figure 4.9, we present the level of delay and the free travel time gap of the top 10 percent paths with the highest level of delay in each of the seven "Path length" test sets. In other words, these graphs present the 250 O-D couples with the highest level of delays of each test set. We observe an average level of delay that is roughly twice as high for the test sets with the longer paths, while we observe an average level of delay that is roughly three or four times higher for the test sets with the shorter paths. The travel time gaps for the test sets with longer paths only increase by 25%, while the test sets with shorter paths increase with a factor two or three. This means the spread among the test sets with shorter paths is higher than the test sets with longer paths. The 10 percent most inaccurate time dependent travel times calculated with the TTC are still relatively accurate. On average, a trip of four hours would deviate around 5 minutes of the "real" travel time.

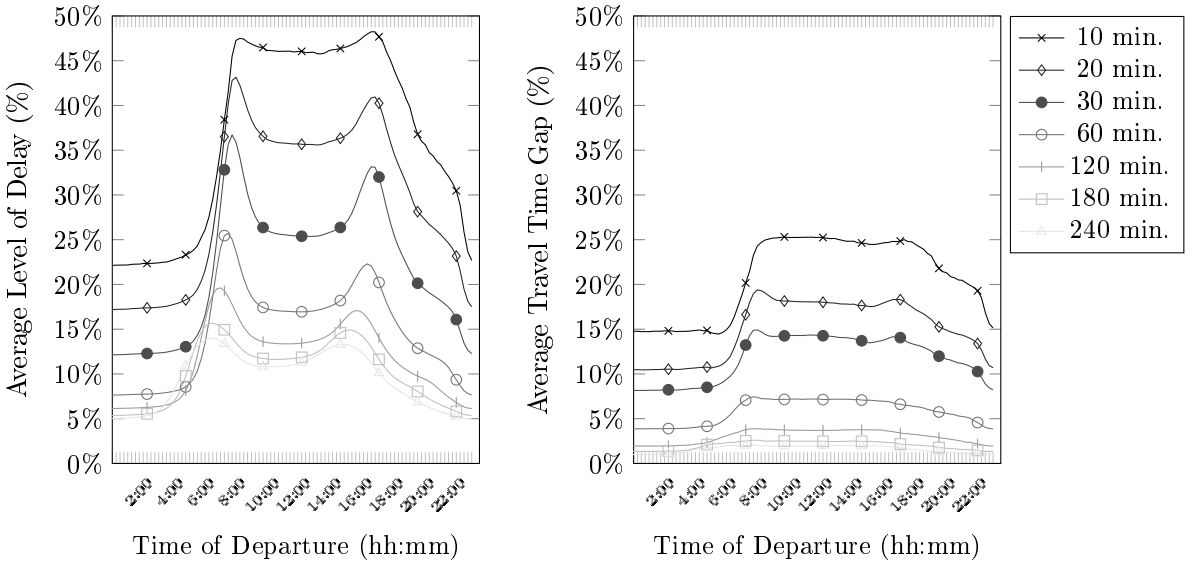


FIGURE 4.9: This figure depicts the level of delay (a) and the free flow travel time gap (b) of the 10 percent paths with the highest congestion within each of the seven "Path length" test sets. Both figures are the average of a Tuesday and a Thursday.

To conclude, we see travel time patterns as we expected them to be. We observe a clear morning and afternoon travel time peak, which also results in a higher travel time gap during these periods. For shorter paths, on average the percentage difference between time dependent and free flow travel time does not exceed 12%. During longer trips, the impact of having time dependent travel times for trucks is lower, namely, 8%. However, these are average values. Looking at solely the 10 % of trips with the highest level of delay, we observe that these levels of delay are much higher than taking the average. This means, that depending on where a trip originates and destines, there exist a difference in the impact of the congestion information.

We observe a travel time gap during the night period, something which is considered not in line with expectations as there typically does not exist any congestion during night. However, this makes sense as the FF-TT of an edge is based on the road type, while the TD-TT is based on the historical travel time functions. Therefore, it is possible that during the night the TD-TT is lower than the FF-TT, meaning that the measured travel time on an edge is higher than the assigned travel time to the edge's road type. This results in a congestion factor between two representative areas other than the value 100 . However, a lower TD-TT than FF-TT during the night is not consistent over all roads. This makes it possible to have the following situation between two representative areas: namely, shortest paths with no difference between the FF-TT and the TD-TT and shortest paths with a large difference between the FF-TT and the TD-TT. However, the FF-TT of each shortest path is divided by the same congestion factor, resulting in a travel time gap. This observation leads to the conclusion that the default travel times are sometimes overestimated compared to the values of the map provider.

4.4.2 Experiment 2: The effect of the vicinity of the representative on the time dependent travel time over different departure times during the day

In Experiment 2, we researched the effects of the vicinity of representatives to the origin and destination of a shortest path. Figure 4.10a displays the average time dependent travel time of the four test sets in test group 2. We expected that all four sets would have the same average travel time. The first three sets are indeed quite similar, with travel times ranging 208 and 214 during the night periods. The fourth set has a slightly higher average travel time, that we can explain by the use of the BeNeLux map. The map also covers a large part of Northern France and West Germany, but the representatives are only chosen within the BeNeLux area. When the vicinity increases, representatives in the center are not feasible anymore because after traversing 20 minutes from one representative, a different representative area is reached. The only feasible representatives are the ones on the South and East side of the BeNeLux, creating a set of paths that are not random. Sets with even higher vicinity distances gave even higher average travel times and were therefore omitted from this experiment.

Figure 4.10b displays the average free flow travel time gap of the four test sets. The average travel time gaps show the same pattern as the previous experiments, meaning there is an increase during the day. However, the average travel time gaps are low compared to the previous experiments, as it does not exceed the 2%. We do observe an increase in the average travel time gap, in the test sets with a larger vicinity. Thus as the vicinity increases of an O-D pair, the travel time gap increases as well. This pattern also includes the "20 min" vicinity set, although it is questionable if it should be included.

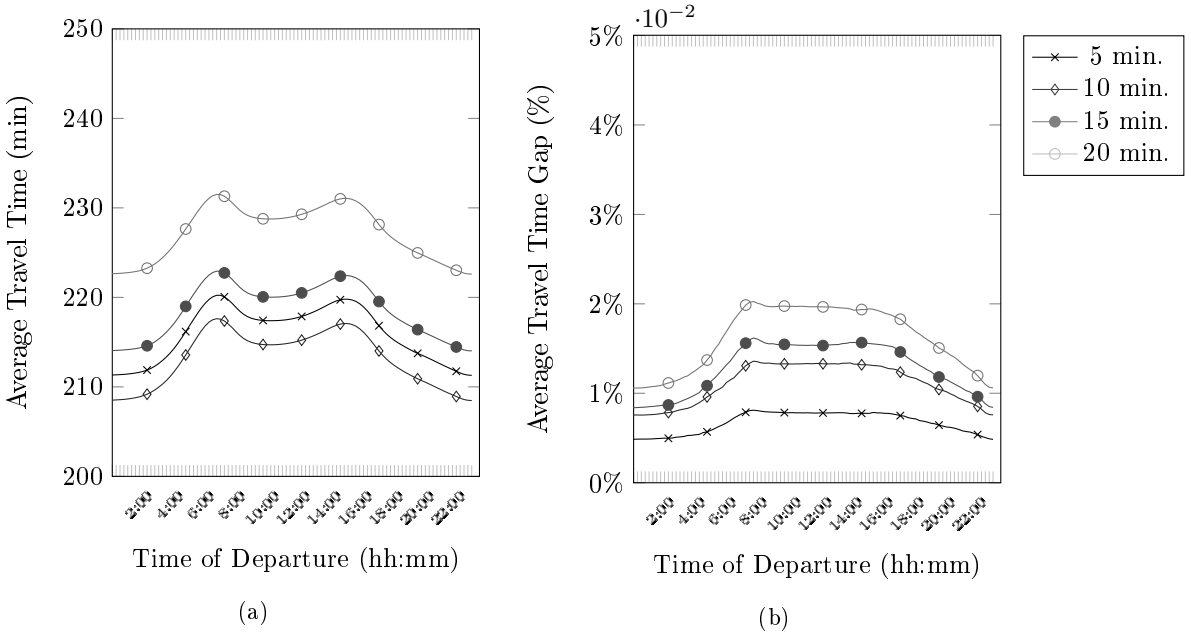


FIGURE 4.10: (a) Depicts the level of delay of the four "vicinity length" test sets. (b) Depicts the travel time gap of the four "vicinity length" test sets. Both graphs are the accumulated results of a Tuesday and Thursday.

4.4.3 Experiment 3: The effect of congestion on the time dependent travel time over different departure times during the day

In the third experiment, we researched the effect of the level of congestion on the time dependent travel time. To do this, we selected four areas we assume to be highly congested, namely: Amsterdam, Antwerp, Brussels, and Rotterdam. In Section 4.3, we describe how the nodes in these four test sets are selected. Figure 4.11 presents the delay percentage between the time dependent and free flow travel time. In Figure 4.11a, we observe that Antwerp and Brussels are indeed congested areas with delays of 30% and 40% during weekday rush hour. Amsterdam and Rotterdam are less affected by congestion with delay percentages between 15% and 20% during the morning and afternoon weekday rush hour. Figure 4.11b presents the same delay patterns as Figure 4.6b, showing only an increase in congestion during the daytime as a whole. The values of these test sets are higher than the "path length" test sets. Again, the Belgium cities show a higher delay percentage than the Dutch cities.

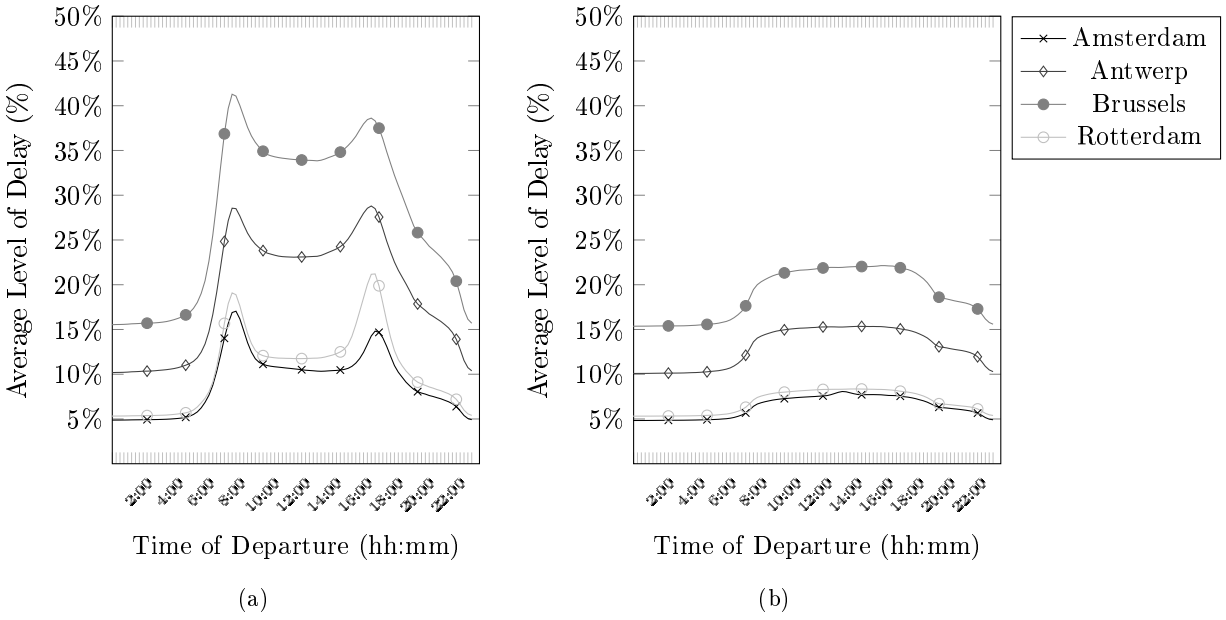


FIGURE 4.11: This figure shows the delay percentage over different departure times. This is the difference between the time dependent and free flow travel time. The plots are based on 2500 random paths within the areas. (a) Is the average of a Tuesday and a Thursday. (b) Is the average of a Saturday and Sunday.

Figure 4.12 depicts the travel time gap between the time dependent travel times and the TTC travel times. It shows the percentage difference between the travel time used by the vehicle routing algorithm and the time dependent travel time. The test set with random trips in Brussels caused the highest travel time gap of over 25% during daytime. This means the predicted travel times have on average an error of over 25%. During the weekend days, the errors are on average lower, with Brussels having the highest error of the four test sets.

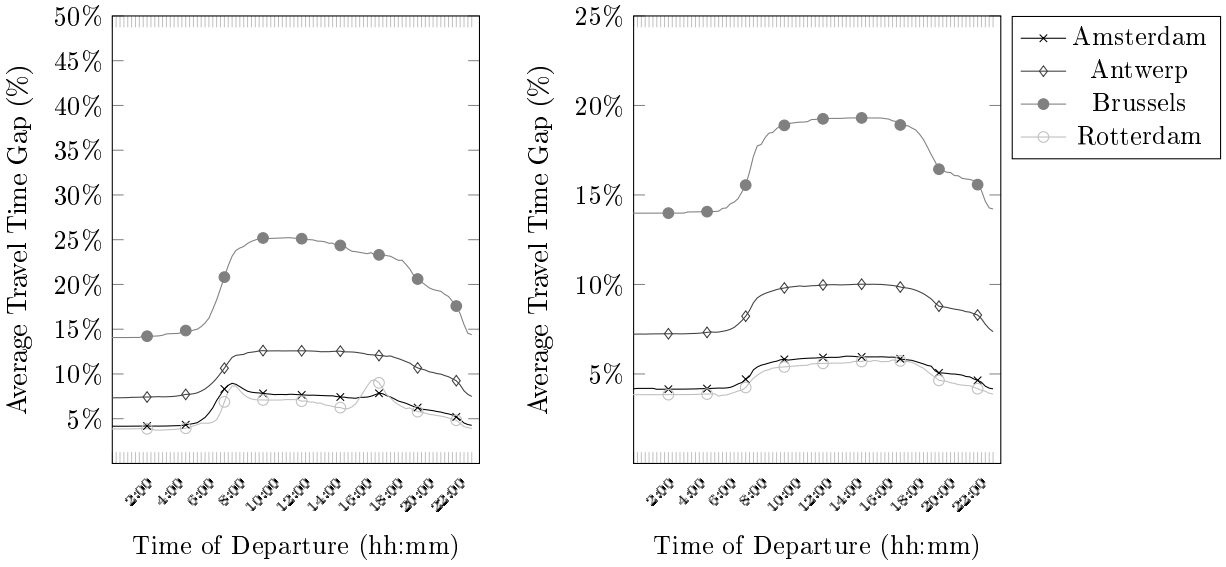


FIGURE 4.12: This figure shows the travel time gap over different departure times. The travel time gap is the percentage difference between the travel time used by the vehicle routing algorithm and the time dependent travel time. The plots are based on 2500 random paths within the areas. (a) Is the average of a Tuesday and a Thursday. (b) Is the average of a Saturday and Sunday.

To conclude, we observe that areas with higher congestion have indeed higher travel time gaps. Not all areas we thought to be congested, were as congested as we thought. The two Dutch areas only showed a bit more congestion as compared to the test set with path lengths.

4.4.4 Experiment 4: The effect of the number of representatives on the time dependent travel time over different departure times during the day

In the fourth experiment, we researched the effect of the number representatives on the time dependent travel time. We use both the grid and address strategy to compute the locations of the representatives. Figure 4.13 depicts the travel time gap for the "Brussels" and "10-minute path length" test set. Among the four plot lines in Figure 4.13a, we see a clear declining pattern. This means an increase of the number of representatives does indeed lead to a better approximation of the time dependent travel times using the TTC. In Figure 4.14b this pattern is absence. The increase of representatives does not result into a decrease of accuracy. All four test cases resulted in approximately the same results.

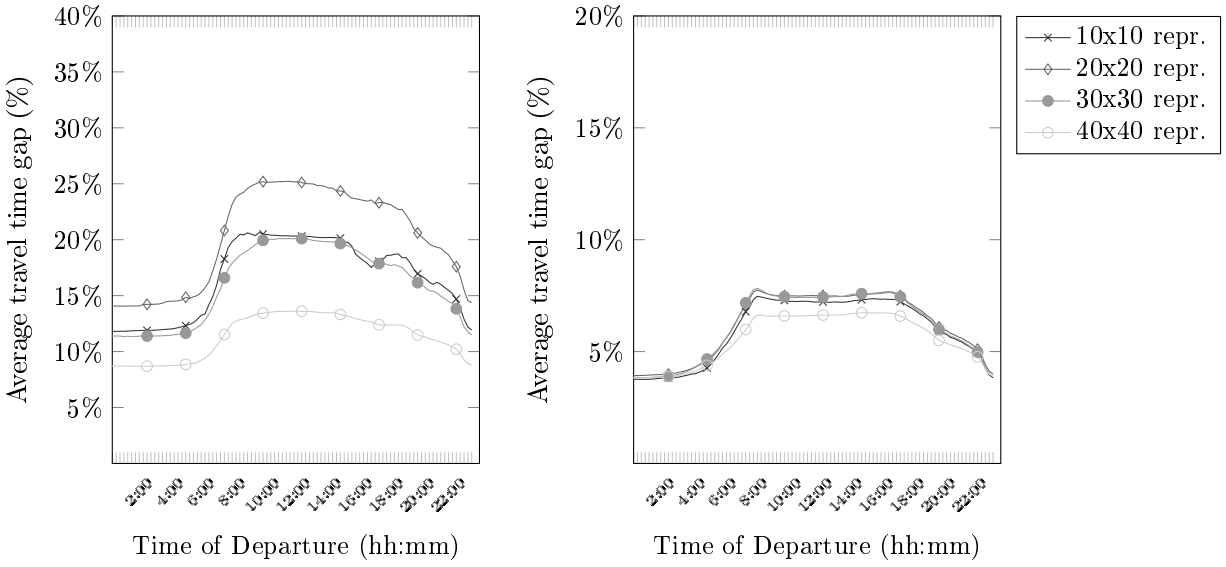


FIGURE 4.13: This figure shows the travel time gap over different departure times, using different sized sets of representatives. We generated the representative sets using the grid strategy. Both graphs are based on the average of a Tuesday and a Thursday. (a) Average of 2500 random paths within the Brussels area. (b) Average of 2500 random paths within the BeNeLux area, all with a free flow length of 10 minutes.

Figure 4.14 depicts the results of travel time gaps using the address strategy for different number of representatives. We see the same decreasing pattern as in Figure 4.13a in both Figures 4.14a and 4.14b. Especially the increase of representatives from five to fifteen made large improvements. Using twenty or thirty representatives within a single area resulted in a relatively small decrease of the travel time gap. Using five representatives in the Brussels areas, leads to the similar results compared to the 40x40 grid representatives of Figure 4.13a. Keep in mind that the five representatives used in these experiments were dedicated to the Brussels areas alone.

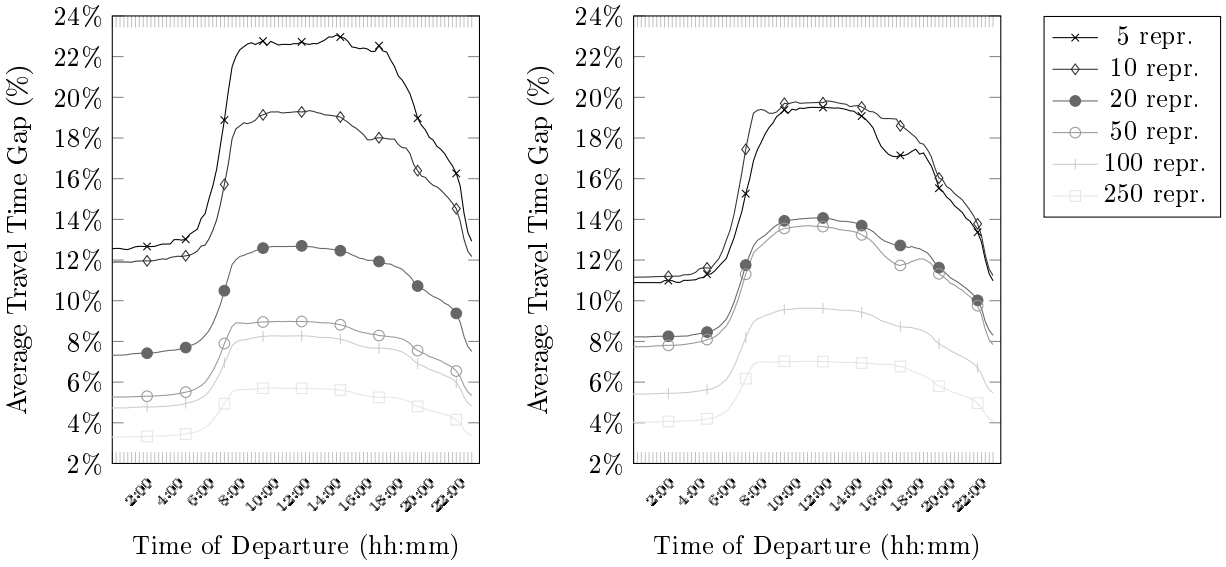


FIGURE 4.14: This figure shows the travel time gap over different departure times, using different amounts of representatives. We generated the representative sets using the address strategy. Both graphs are based on the average of a Tuesday and a Thursday. (a) Average of 2500 random paths within the Antwerp area. (b)Average of 2500 random paths within the Brussels area.

To conclude, we see travel time gap patterns as we expected them to be. We observe an overall decline when using more representatives, with as only exception the "10-minute path length" test set. Because shorter paths tend to stay within one representative area, an increase of the number of representatives does not change that fact. Furthermore we observe the travel time gap pattern increases during day time congestion, the same as we saw with the previous experiments.

4.4.5 Experiment 5: The effect of the percentage of the shortest path shared on the travel time gap

In the fifth experiment, we researched the effect between the travel time gap and the percentage of the shortest path between two nodes that is shared with the shortest path of the corresponding two representatives. Figure 4.15 shows the results of the experiment based on the Antwerp and Brussels test set. These graphs are based on 2008 and 1387 paths respectively, as the remaining paths of the 2500 paths of both test sets are within one representative area. These paths have therefore no shortest path to be compared with. Both graphs present a widely distributed declining pattern, something that is confirmed byt the two exponential declining trend lines of the results. However, the coefficient of determination of the Brussels test set has such a low value, it is only save to say there is a declining pattern. The coefficient of determination of the Antwerp test set is even lower, so we conclude no pattern is visible.

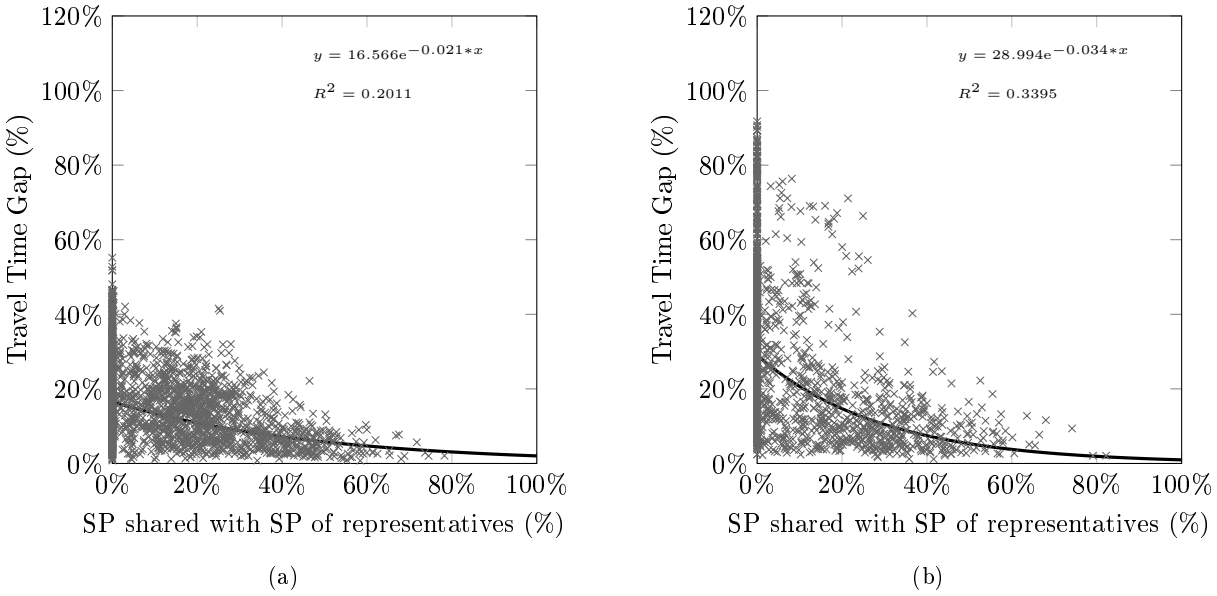


FIGURE 4.15: Display of the the travel time gaps against the shared percentage of the shortest path between nodes and the shortest path between the corresponding representatives. (a) Result of 2008 random paths within the Antwerp area. (b) Average of 1387 random paths within the Brussels area. The remaining paths were omitted as they were within one representative area.

Figure 4.16 shows the results of the same experiment, but based on the "2-hour path length" and "5-minute vicinity length" test set. We observe a similar widely distributed declining pattern in both graphs. Looking at the coefficients of determination of both the trend lines, we conclude that a higher percentage of a shared shortest path of a node pair and a representative pair results in a lower travel time gap. Because both test sets have on average longer paths than the Antwerp and Brussels test sets, a higher concentration of data points is visible near the 80% sharings percentage. Still, we observe a reasonable amount of data points located on the y-axis, meaning paths exist with no overlap with their corresponding representative paths. Furthermore, we observe data points with a small sharings percentage and a small travel time gap, which is in contradiction to our expectations. However, we did not correct for the level of delay found on the path, so the low travel time gap can also be explained by a lack of congestion.

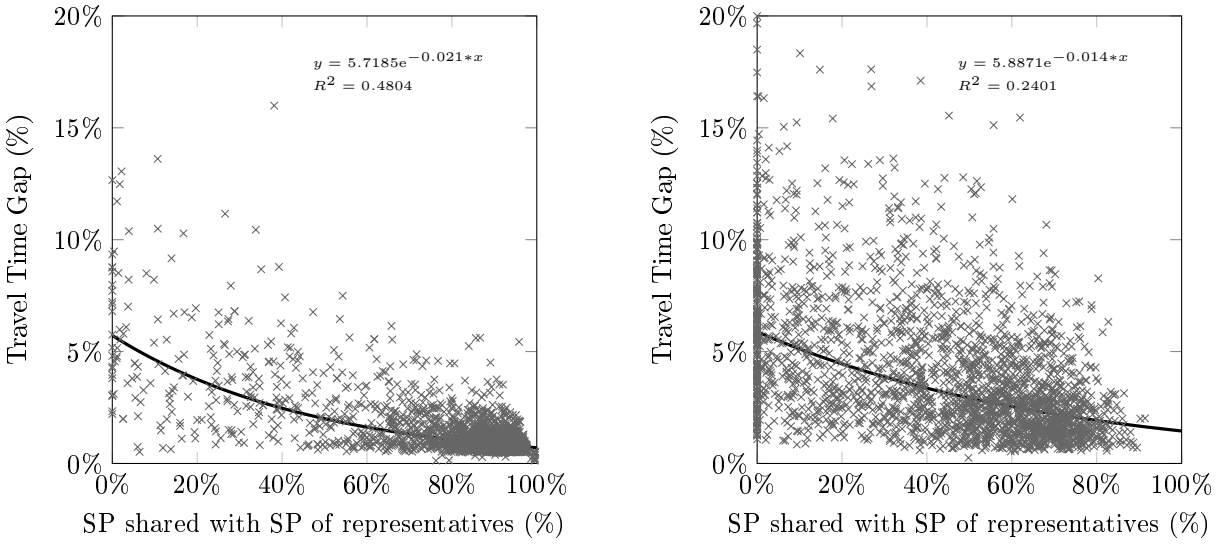


FIGURE 4.16: Display of the the travel time gaps against the shared percentage of the shortest path between nodes and the shortest path between the corresponding representatives. (a) Result of the 2500 random paths of the "5-minute vicinity length" test set. (b) Result of the 2500 random paths of the "120-minute path length" test set.

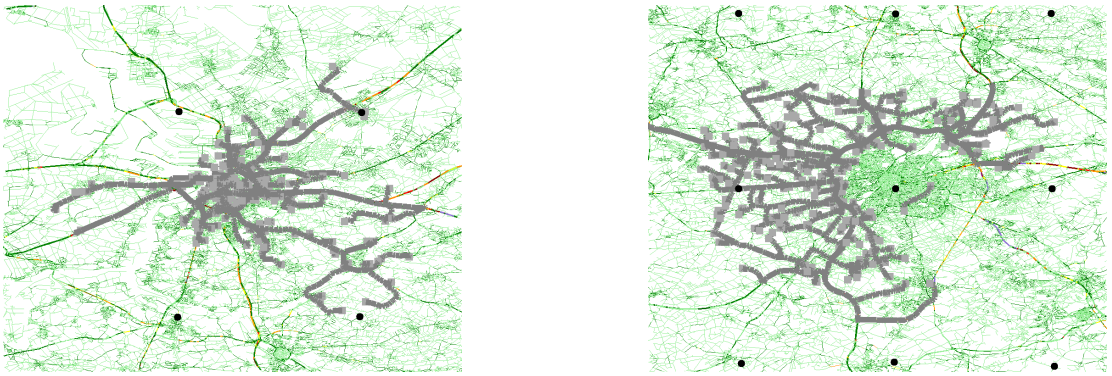
To conclude, we observe a declining pattern of the travel time gap when the representatives represent a large percentage of the shortest path of two nodes within two different representative areas. In the test sets of Antwerp and Brussels, this pattern is less visible than within the test sets with longer paths. Besides the average declining pattern, we observe that the overall distribution of the travel time gap over the y-axis decreases as the sharing percentage increases. This makes sense, as a higher sharing percentage means that the congestion factors represent a larger part of the shortest path of the OD-pair, resulting in a lower travel time gap. We observe a larger distribution on the left side of the graphs, as a lower sharing percentage results in higher travel time gaps. However, we also observe low travel time gaps with lower sharing percentages. This has two causes. First, if no congestion is present, no travel time gap can be measured. Second, the algorithm gets lucky and the congestion factors matches the congestion of the OD-pair. However, the second cause is less likely than the first one.

4.5 Zoom in

In the previous, section we researched the averages of different test sets. In this section, we take a closer look at the results, with the goal to distillate the situations which cause the high inaccuracies. In this way, we can design our algorithm to avoid these situations.

Figure 4.17 displays the shortest paths of the 5% of O-D couples within the test sets "Antwerp" and "Brussels" with the highest free flow travel time gap. In Figure 4.17a, we observe that the city center has a lot of the origins and/or destinations of these congested paths, meaning the TTC does a poor job at predicting the time dependent travel time in that area. When we look at Figure 4.17b, we see the paths with the highest free flow travel time gap are in the North-Western part of Brussels, while no paths cross the city center of Brussels. Both figures have in common that the origin and destination nodes tend to remain at distance from the representatives in the area. Because these nodes are near the border of

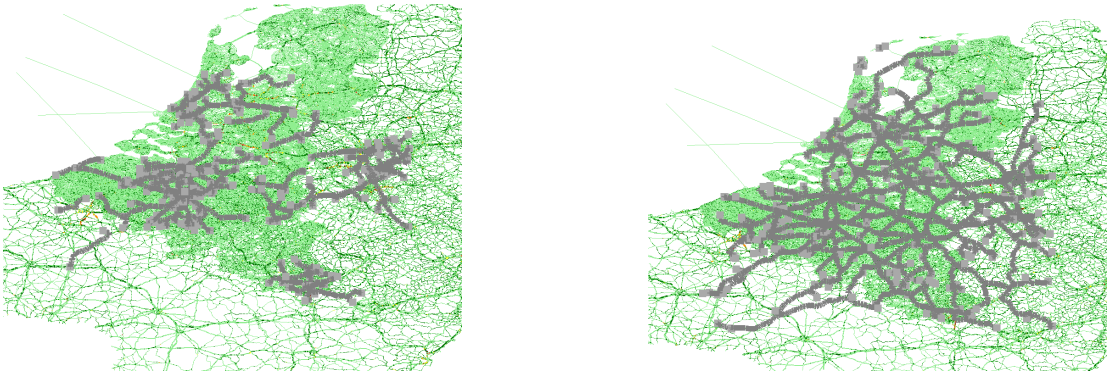
the representative areas, this causes the travel time gap to be higher, as the congestion factors of their representatives are based on a different trip than the O-D couple.



(a) (b)

FIGURE 4.17: Overview of the 5% of O-D couples within test set "Antwerp" (a) and "Brussels" (b) The gray lines depicts these shortest paths, the black dots are all representatives within the displayed area.

Figure 4.18 displays the shortest paths of the 5% of O-D couples within test set "Path length 60 min." (4.18a) and "Path length 240 min." (4.18b) with the highest free flow travel time gap. Figure 4.18a shows clusters of paths around the Randstad, Flemish Diamond, and the Rhine-Ruhr metropolitan regions. As these are the most crowded and congested areas, it is explainable that we observe the O-D couples with the highest free flow travel time gaps in these areas. Figure 4.18b shows a more evenly distributed paths over the BeNeLux map, compared to the Figure 4.18a. Only the North-Eastern and South-Western part of the map seem to be empty. This observation is consistent with the observation of the previous section that the average travel time gap is low for this test set, and the travel time dispersion is low compared to the other test sets. There are no clear areas where longer paths have on average a higher travel time gap, which corresponds with our conclusion that for longer trips the level of delay is lower and therefore has less effect.

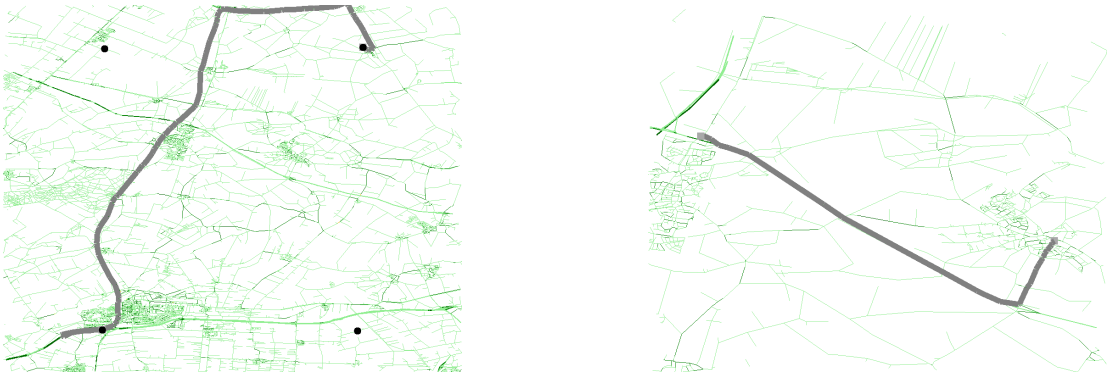


(a) (b)

FIGURE 4.18: Overview of the 5% of O-D couples within test set "Path length 60 min." (a) and "Path length 240 min." (b) The gray lines depicts these shortest paths, the black dots are all representatives within the displayed area.

In Figure 4.19, we take a closer look at single O-D pair, with a path sharings percentage of zero and a free flow travel time gap of only 0.68%. This means, that although the representatives shortest path does not represent any part of the shortest path between the O-D, it still results in an accurate time dependent

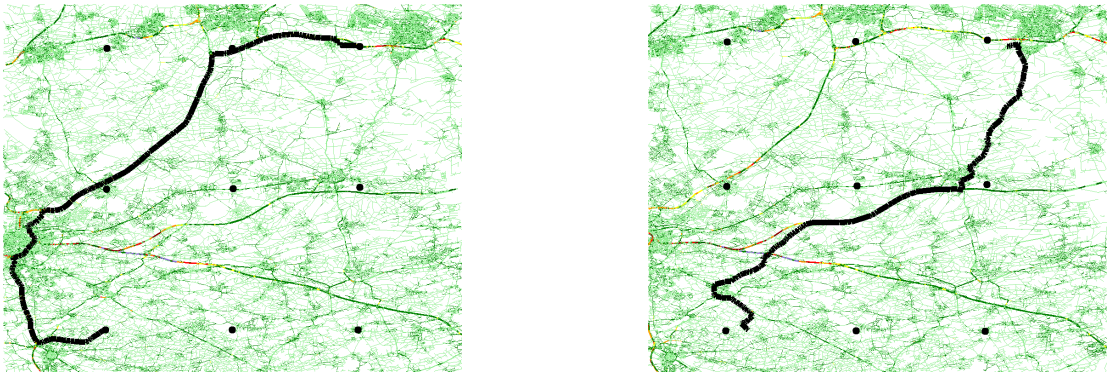
travel time prediction. Figure 4.19a shows the shortest path of the two representatives, and Figure 4.19b shows the shortest path of the O-D. The congestion factors of the two representatives have a factor of 100 during the night, and a factor of 99 during the day. This means that between the representatives no congestion is present, and therefore the travel time gap is also around zero.



(a) (b)

FIGURE 4.19: Overview of the shortest path between two nodes (a) and the shortest path between the two corresponding representatives (b), in the area South of Assen. The black lines depicts these shortest paths, the black dots are all representatives within the displayed area.

We take a closer look at some of the paths with a zero percent overlap of the "5-minute vicinity length" test set, as we expected that paths that originate and destinate close to a representative, would always share a large part of the shortest path. However, Figure 4.20b depicts that it is possible to have zero overlap even if the origin and destination are close to their corresponding representatives. Figure 4.20a shows that the shortest path between the two representatives runs via the left side of the map, using highways for the largest part. The shortest paths between the two closely located nodes runs via the central part of the map, using mainly regional roads. This results in a zero overlap between both paths.



(a) (b)

FIGURE 4.20: Overview of the shortest path between two nodes (a) and the shortest path between the two corresponding representatives (b), in the area West of Antwerp. The black line depicts these shortest paths, the black dots are all representatives within the displayed area.

4.6 Conclusion

In this chapter, we researched the accuracy of the TTC. We designed five different experiments to identify the situations in which the TTC results are inaccurate and when improvements are necessary. To conclude, the TTC performs poorly on relatively short paths, in congested areas and/or where the locations of the representatives are far away from the main roads of the road networks. Shorter paths showed a higher deviation between the travel times of the TTC and the time dependent travel times. This is a result of shorter paths being less represented by the representatives, as the shorter paths are more likely to have a completely different shortest path between two representative areas than the representatives themselves. See Figure 4.20 for an example of a different shortest path between two nodes than the shortest path between the two representatives. Congestion makes the need for correct approximations increasingly important, as poor approximations lead to large travel time deviations. These deviations are absent if the congestion is absent as well, therefore we need to focus mainly on the congested areas when designing our algorithm.

We assumed that at night the traffic intensity is so low that free flow conditions exist. However, we observed a travel time gap during the night, meaning there exists congestion during this period. The cause of this congestion is that the algorithm calculates the FF-TT using a default speed that is assigned to each of the road types. Each edge in the road network is linked to one of these road types. The default speed on especially city roads in urban areas is an overestimation of the actual measured speed on those types of roads during free flow conditions (at night). Because this overestimation is not consistent over all edges equally, the measured congestion is not equal between each OD pair. This results in a deviation between the travel times of the TTC and the time dependent travel times, hence, a travel time gap.

Chapter 5

Congestion Hierarchy Algorithm

In this chapter, we present our solution method. The literature review from Chapter 2, the current method from Chapter 3, and the findings from Chapter 4 form the basis of this method. In Section 5.1, we clarify the general solution of the *Congestion Hierarchy Algorithm (CH-algorithm)*. In Section 5.2, we discuss possible graph partitioning algorithms. In Section 5.3, we explain our travel time algorithm explicitly. We present the results and conclusions of these results in the next chapter.

5.1 General solution approach

The results of the benchmark experiments show us that the current algorithm performs well on average, especially in cases where the requested travel times are for relatively long paths (>2hr). However, the algorithm performs poorly in congested areas, for relative shorter paths (<1hr). This is caused by the fact that the difference between the free flow travel time and the time dependent travel time is higher, and the relative short distance between source and target node makes that the representatives are less likely to represent the actual delay between the two areas. Refining the grid leads to better results (i.e., increasing the number of representatives), but at the cost of more memory use. We dismiss the idea of just increasing the number of representatives, because one of the requirements of the algorithm is to work under the same conditions as the TTC. The memory usage is high because every representative couple has its own set of congestion factors. The good performance for longer paths, and the poor performance for the shorter paths, gives us the idea to create a new algorithm that looks similar to the current algorithm for longer paths, but is different for shorter paths.

We want to accomplish good TD-TT estimations for shorter and longer paths by generating more representatives in a smaller grid formation, but without the increase of main memory usage. Therefore, we argue that it is unnecessary to store all congestion factors between all representatives. If we do not have to store all congestion factors, we are able to refine the grid without the increase of main memory. We observed that the TTG of a longer path was lower, because it is more likely to share a large part of the shortest path of its representatives. For example, let's take five random origin nodes around Amsterdam and five random destination nodes around Luxembourg City. The twenty-five resulting trips going from Amsterdam to Luxembourg share around 95% of the same trip. It is therefore less necessary to store multiple representatives around Amsterdam all having congestion factors to multiple representatives

around Luxembourg City, while one for Amsterdam and one for Luxembourg City is sufficiently accurate at this distance.

In essence, we only store congestion factors between a representative and its surrounding representatives. In this way, a lot of memory can be saved and can be used to increase the number of representatives. However, only storing congestion factors of neighbouring representatives causes problems for two nodes that are not near each other. Therefore, we introduce layers of representatives, instead of just one single layer. Each layer divides the map into a number of areas, where each lower layer consist of areas that are subdivisions of the areas of the layer above it. The number of areas in a layer increases as the layer is lower in the hierarchy. Therefore, the lowest level consist of the most areas, and consequently these areas are the smallest. A small area ensures that all nodes in the area are relatively close to the representative in the center. Each node in the graph has a representative area in each of the layers.

A small representative area has the advantage that a single node has a representative in a small area for close distances, and representatives that are in areas that increase in size for paths that also increase in distances. The loss of accuracy for the longer distances is smaller than the increase of accuracy on shorter distances. To retrieve the delay between two nodes, we search from the bottom to the top layer, for the layer in which both nodes have a representative area that are adjacent to each other.

Until now we implied that each area should contain a representative. However, it is unnecessary to assign a representative to each of the representative areas, as certain areas are sparse and lack congestion. These areas will have similar congestion factors, which are likely to be constant patterns that show no congestion. Because these factors are so similar, it is unnecessary to calculate and store these congestion factors for each of the representative areas. If we omit these congestion factors, the nodes in those areas are still represented, but only by a representative at a higher level. The memory saved, can be used to increase the resolution around the dense areas, to further increase the accuracy.

In our new algorithm, we identify three parameters that adjust the performance of the algorithm. First, we can adjust the number of layers of representative areas. If we increase the number of layers, we increase the fineness of the grid of the lowest level, which increases the accuracy of the algorithm. This comes at the cost of more computation time during the preprocessing step and more memory usage of the algorithm itself. Second, we can adjust the number of neighbours around a representative. Increasing the coverage around a representative means that nodes that are not directly adjacent, still share congestion factors, which otherwise would share congestion factors at a higher level. Because lower level representatives are closer to their surrounding nodes, it increases the accuracy of the algorithm. Third, we can adjust the threshold to accept an area to get a representative. The lower the threshold, the easier an area gets a representative. This benefits the accuracy, but also increases the preprocessing time and the memory usage of the algorithm.

5.2 Partitioning graphs

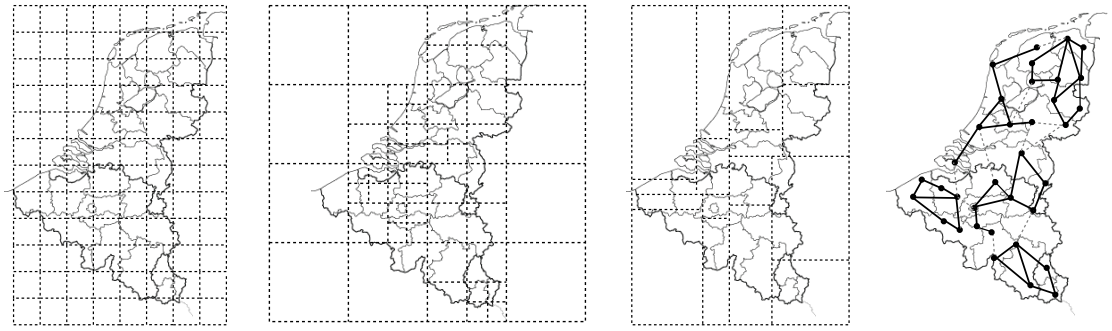
In this section, we look into the different ways to partition a graph. We explain four different algorithms. These algorithms are, grid, quadTree, kd-tree, and Kernighan–Lin [37].

5.2.1 Grid

The most simple way to partition a 2D graph is using a grid based structure. Each cell within a grid functions as one region of the graph. Cells can be rectangular or square shaped, and the number of columns and rows do not necessary have to be equal. The TTC uses this approach as one of two representative strategies.

5.2.2 Quadtree

A quadtree is a type of data structure, commonly used in data processing. The data structure consists of a tree, in which each node in the tree consists of exactly four child nodes, hence the name "quad". Finkel and Bentley [38] were the first to call this type of data structure a quadtree. The regions are typically square or rectangular in shape, even though other shaper are also possible. This is similar as to the grid partition, however, there is an equal number of rows and columns, as the division of a cell is always done by dividing into four smaller cells. There exist multiple forms of quadtrees, but they all share some basic characteristics. First, the goal of the algorithm is to split an area into adaptable cells. Second, each cell has a threshold capacity, when this capacity is exceeded, the cell is divided into four smaller cells. Third, the underlying tree structure follows the structure of the spatial decomposition. In essence, a quadtree adapts to the number of data points in its structure. However, when every node in the quadtree (except the leaf nodes) have four child nodes, we call this a tree-pyramid.



(a) (b) (c) (d)

FIGURE 5.1: The four different partition algorithms. Grid (a), quad-tree (b), kd-tree (c), and Kernighan–Lin (d).

5.2.3 Kd-Trees

Another type of data structure similar to the quadtree, is the so called kd-tree. It is an algorithm for organizing points within a k-dimensional space, hence the name kd-tree. It was developed by one the same authors of the quadtree (Bentley [39]). The quadtree algorithm divides the graph into areas based on the geographical properties, but it does not take into account the distribution of nodes over the different areas. The kd-algorithm starts dividing the graph into areas, by splitting them in half, such that both areas have an equal number of nodes. This way, the nodes are more equally divided over the areas, but the areas themselves can differ tremendously in size and shape. In the 2D-variant interesting for our research, this results in a number of different size rectangular areas.

5.2.4 METIS algorithm

Already in 1970, Kernighan and Lin [40] developed the *Kernighan-Lin(KL)*-algorithm that finds partitions in graphs, with the goal to have the partitions such that the sum of the edge weights connecting the partitions is minimized. The main advantages is that the algorithm does not need any geometric information of the graph, information on the connected edges and their weights is sufficient. The main disadvantages are that this algorithm was not designed to create multiple layers, the running time is relatively long, it requires an undirected graph as input. We can overcome the first disadvantage with some small adjustments to the algorithm. However, the computational time of the algorithm is $O(n^2 \log n)$, where n is the number of edges in the graph, and computational time is per division of a partition. This means that the computation time gets too large for larger road networks with edge numbers surplussing several million and several hierarchical levels. A way to avoid this, is to use the METIS implementation [41]. This algorithm first coarsens the graph, by clustering nodes into partitions, effectively reducing the number of nodes and edges of the graph. On the resulting coarsened graph, it runs the KL-algorithm to partition the graph. However, it still uses the KL-algorithm, which has an undirected graph as input.

5.2.5 Conclusion

Of the four algorithms presented, our main interest is on the quadtree algorithm. The TTC uses the classical grid algorithm, which performs in specific situations quite reasonable. As it is such a basic algorithm, we do not see any ways to enhance it to further improve the results. The kd-algorithm creates nicely distributed areas, but the major drawback are the irregular shapes. Because of these irregular and rectangular shapes, it is difficult to generate congestion factors that represent the complete area. The METIS algorithm has a high computational time for larger graphs, and uses the KL-algorithm, which relies on an undirected graph. The major advantages of the quadtree algorithm are its ability to scale based on the density of the graph, the resulting square areas, and its simplicity.

5.3 Our travel time algorithm: Congestion Hierarchy Algorithm (CH-algorithm)

In this section, we present the structure of our *Congestion Hierarchy Algorithm (CH-algorithm)*. First, we describe the preprocessing step of the CH-algorithm. The preprocessing step creates the data structures that are necessary for fast computations. Preprocessing is done once and is only redone when the map data is altered. Second, we describe the three data structures we use to store the calculated CH information. Third, we describe the process of calculating the time dependent travel times using the CH-algorithm.

5.3.1 Preprocessing

In this section, we describe the four preprocessing steps of the CH-algorithm. We distinguish four main preprocessing steps, which are, subdividing the map into representative areas, selecting the representatives, generating the congestion factors, and mapping the nodes to each area.

5.3.1.1 Subdividing the map

As mentioned in Section 5.1, we divide the map into smaller areas. This is similar as with the TTC, but we introduce a hierarchy among these areas. Due to the decision to not exceed a main memory usage of 1000 MiB, the current approach only allows for 1000 representative areas, so roughly a rectangular grid of $32 * 32$ representatives. We decide to use rectangular areas, that are roughly squares. This has the advantage that a grid is fast to compute. Still, it ensures that the graph is nicely distributed over the different areas and it has such a shape, that the centroid of the square makes a good representative.

The CH-algorithm starts at the highest level, which is a 1-by-1 grid and the top layer of the hierarchy. We start numbering from zero up, so this layer has number zero. We do this so we can easily find the number of rows and columns of the level, by calculating 2^{layer}. Afterwards, the CH-algorithm divides the map into a 2-by-2 grid, resulting in four equal areas. The algorithm takes the outer coordinates of the map as the outer coordinates of the grid, or these coordinates are given as input to the algorithm. If the latter is the case, not all nodes of the map are within the area of the grid and therefore it is not possible for these nodes to calculate the time dependent travel times. For clarity reasons, we assume for now the grid includes the complete road network. Each of the four areas is divided into four more areas, resulting in a 4-by-4 grid of 16 areas. This process continues until a predefined number of layers has been reached. This number is one of the three parameters we can adjust in this algorithm. In the end, each node of the road network is within a representative area at each of the layers. It is likely that a part of the areas have no nodes assigned to them, because they are in areas with no roads, like seas or lakes. To illustrate, we have a graph (road network) as in Figure 5.2a, and Figure 5.2b depicts the result when we omit the edges and is also the top layer of the hierarchy. Figures 5.2b to 5.2e show the first three layers of our hierarchal approach. Figure 5.2b is the top layer after which we divide the graph into four equal parts. Each of these four equal parts are further subdivided into four equal parts, resulting into Figure 5.2d. Applying the same subdivision to Figure 5.2d, results in Figure 5.2e.

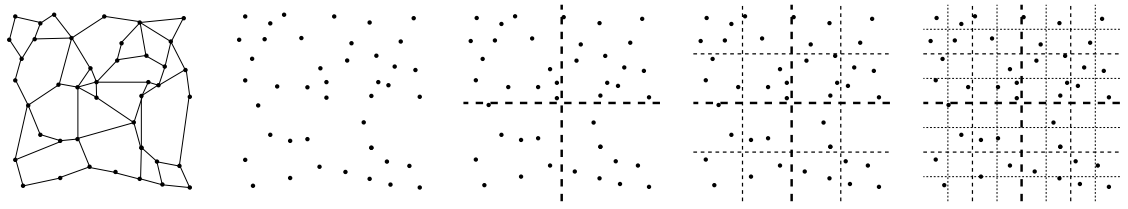


FIGURE 5.2: This figure depicts the first step of the CH-algorithm. We ignore the edges of the graph (a), and are only interested in the nodes (b). We geometrically divided the graph into four equal rectangular areas, which acts as the second layer of the CH (c). Each of the areas is subdivided even further (d,e), until the algorithm reaches a predefined number of layers.

5.3.1.2 Selecting representatives

After subdividing the map into multiple layers and multiple representative areas, we select the representatives for these areas. A representative will be a single node within the representative area, which is the node we base our congestion factors on. Because we introduced multiple layers, where the lowest layers have a high number of representative areas, we do not assign a representative to every single area. As noted in the previous section, multiple representative areas do not even contain nodes due to the sparseness of the graph. Therefore, we do not assign a representative to these areas. Also, we argue that it is unnecessary to have a representative in an area with only a few number of nodes and with no congestion. We do not assign a representative to an area when the number of nodes in that area is less than $\alpha * \bar{\rho}$, where $\alpha \geq 0$ and $\bar{\rho}$ is the average area density in number of nodes. We choose to use a linear expression for reasons of simplicity, but any type of relationship is possible. The value of α is one of the three parameters we can adjust.

Let's assume the situation in Figure 5.3a, where we have a graph with a total of 160 nodes divided over 2 layers (2x2 and 4x4) and where $\alpha = 0.6$. The second layer has four representative areas, so the average number of nodes in an area is $\rho = 160/4 = 40$. With an α of 0.6, an area needs to have at least 24 nodes to get a representative. Therefore, only area 0, 1, and 3 get a representative, which is also visible in Figure 5.3b. For the second layer the average number of nodes is $\rho = 160/16 = 10$, so an area should contain at least $10 * 0.6 = 6$ nodes to get a representative. Figure 5.3b depicts which of the 16 representative areas contains a representative and which one does not.

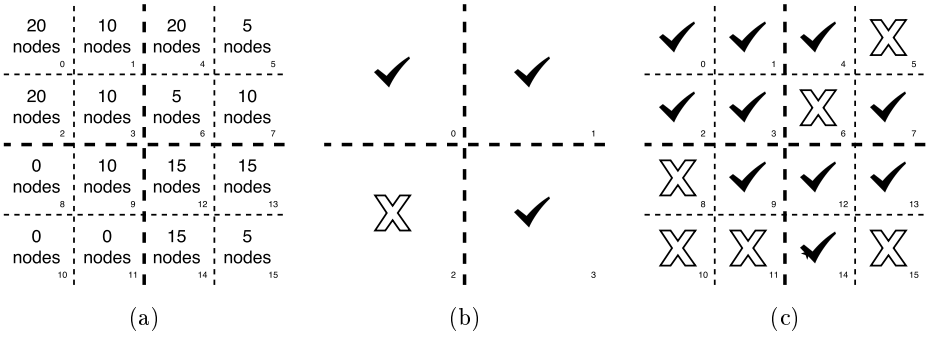


FIGURE 5.3: The three sub figures depict the process of determining which area gets a representative. Figure 5.3a shows the distribution of nodes over the areas. With an $\alpha = 0.6$, the minimum number of nodes in area has to be 24 in the first layer (b), and 6 in the second layer (c). A check mark (✓) shows the areas which get a representative, and a cross (X) shows the areas which do not get a representative.

After we determined which of the representative areas are suitable for an actual representative, we have to determine which of the nodes within an area should be the representative. We argue that selecting the most centered node within the area is a simple but effective way to do this, as this node is on average the closest to all other nodes in the area. We determine which node is the most centered by selecting the node with the smallest euclidean distance towards the center of the representative area.

5.3.1.3 Generating congestion factors

The next step is to generate the congestion factors between the selected representatives of the previous step. We use these congestion factors to be able to quickly compute the time dependent travel time. A

congestion factor is the time dependent travel time divided by the free flow travel, over the same free shortest path between an origin node o and a destination node d in the graph. The level of delay differs over the day, making the congestion factors also time dependent. Therefore, each congestion factor also depends on the departure time τ. Hence, the formula for a congestion factor is:

$$CF_{od}(\tau) = \frac{T_{(q_{od},\tau)}}{T^0_{q_{od}}}$$

Note that the calculation of the congestion factors is identical to the current algorithm, which we already described in Section 3.2. The CH-algorithm differs from the current algorithm, by not calculating congestion factors between each couple of representatives. The CH-algorithm only calculates the congestion factors to the neighbouring representative areas. The number of neighbours that are in the vicinity of a representative, is one of the three parameters we can adjust in this algorithm. Given the example in Figure 5.4, representative F5 (fig. 5.4a) has a vicinity of two neighbours (fig. 5.4b). However, in this example, only 9 of the 19 neighbours actually had a representative (fig. 5.4c), and so only 9 arrays of congestion factors are calculated and stored. Note that for each representative area with a representative, the internal congestion factors are calculated, in the same way as the current algorithm.

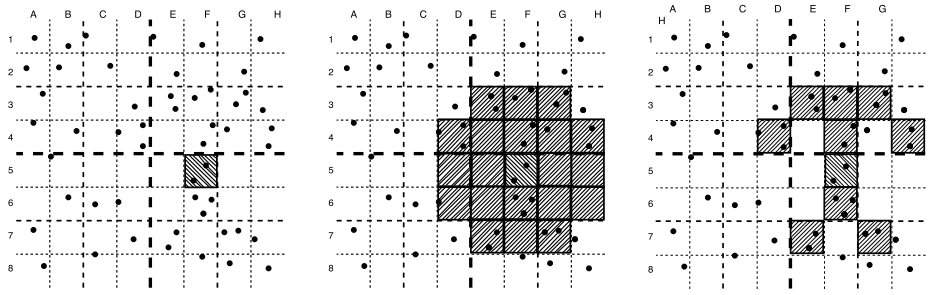
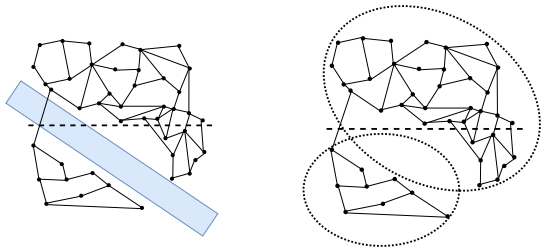


FIGURE 5.4: This figure shows the process of selecting the neighbouring areas around a representative area. In this example we are looking for the neighbours of area F5 (a), where we want to include the neighbouring areas that are within reach of 2 areas. (b) shows the covered area around area F5. Next, we remove the areas that do not have a representative as calculated in the previous section, which results in the final 9 representatives areas as shown in (c).

5.3.1.4 Mapping the nodes to each area

The last preprocessing step is mapping each node to each representative area. Until now, each node is mapped to a representative area based on its geographic location. However, mapping the nodes based on solely their geographic location, may lead to situations that a node could have been better represented by a different area as the concerning representative is much closer via the edges. Figure 5.5 shows such an example, where a canal divides a graph into two areas that are different than the current representative areas. It is therefore more logical to have the nodes on the right in the lower area, join the upper representative area.



(a) (b)

FIGURE 5.5: This figure shows an example with two (partial) representative areas. In Figure (a), we see a road network divided by a canal, it therefore makes more sense to group the nodes as shown in Figure (b).

To map each node to the right representative area, we create a Voronoi diagram around each representative, using the free flow travel times of the edges as the distance function. We generate these Voronoi diagrams by running Dijkstra's algorithm from each of the representative nodes. However, we do not run the algorithm iteratively, but we start Dijkstra's algorithm at each node simultaneously. Therefore, instead of a single source, we enter all representatives of a single layer as the sources into Dijkstra's algorithm. When the algorithm finds a node that already has been visited, it knows that it reached the border of another Voronoi diagram. Additionally, we mark all nodes that are currently not part of a representative area in that layer as visited, omitting them from possibly being included in any of the Voronoi diagrams. The algorithm ends when the Dijkstra queue is empty, and we retrieve which nodes belong to which representative nodes. We repeat this process for each of the layers.

5.3.2 Data overview

After the fourth pre-processing step, the CH-algorithm stores the preprocessed data into an efficient data format. In this section, we give an overview of the used data structures. In total, we use three different data structures to store the information. The first structure is a mapping from each node in the graph to their representative areas. The second structure is a mapping from the representative areas to the congestion factors. The third data structure contains the actual congestion factors. The reason to add the second data structure, is that many of the congestion factors are identical. Because the congestion factors consist of 96 bytes per day, we greatly reduce the memory usage by storing them only once, and have multiple representative areas with the same congestion factors share them.

5.3.2.1 Mapping nodes to representative areas

The first data structure contains the mapping between each node in the graph and its respective representative areas. During the first preprocessing step the algorithm already maps the nodes to a representative area, but this mapping is based on the geographical location of the node. This leads to strange situations at the borders of a representative area, as nodes may be closer to the centre of a neighbouring area over the road. Therefore, during the last step, the CH-algorithm re-evaluates the representative areas of each node, by calculating the closest representative over the graph using the free flow travel time.

Node ID:	Area IDs:
:	:
1.986.215	[1,6,25,97]
1.986.216	[1,6,25,97]
1.986.217	[1,6,25,97]
1.986.218	[1,6,25,97]
1.986.219	[1,6,25,98]
1.986.220	[1,6,25,99]
1.986.221	[1,6,25,99]
1.986.222	[1,6,26,100]
1.986.223	[1,6,26,100]
1.986.224	[1,6,26,101]
1.986.225	[1,6,26,101]
:	:

FIGURE 5.6: A small sub selection of a possible mapping between the nodes and their respective representative areas.

The result is a list of all nodes with their corresponding representative area IDs (short: area IDs), as shown in Figure 5.6. The number of area IDs for each node corresponds with the number of layers in the congestion hierarchies, as each node has an area ID in each layer of the congestion hierarchies. So let us look at the example in Figure 5.6: we see that node 1.986.217 is within representative area 1 in the first layer, area 6 in the second layer, area 25 in the third layer, and area 97 at the lowest layer. This does not mean that the areas where the nodes are present actually have a representative, the next data structure stores this information.

If we would use the initial node mapping based on the geographical locations of the nodes, we are able to reduce it to a single area ID. Performing simple bitshift operations on the area ID, we are then able to retrieve the other layers as well. We choose to use the free flow mapping strategy as described above, as it make more sense to group nodes based on their distance to their representative than on their geographic location.

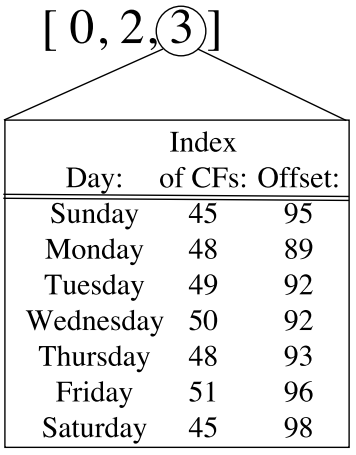
5.3.2.2 Mapping representative areas to congestion factors

The second data structure contains the mapping between each representative area ID and its surrounding areas with congestion factors. We choose to divide this data structure into two different figures (5.7a and 5.7a) for clarity reasons. Because we start numbering the representative areas from zero up to each of the layers, this leads to conflicts between areas with the same ID, but which are in different layers. Therefore, we also sort the area IDs on their layer level to resolve this conflict. Figure 5.7a shows how the area IDs (column 2) are sorted on layer (column 1) and which areas are the neighbouring areas of the area ID (column 3). Column 3 only contains the area IDs of the areas that are both in the vicinity of the area and also contain a representative node.

The TTC stores all seven day congestion factor arrays as a single array at once. During preliminary research we observed that the congestion day patterns are quite similar. Therefore, it makes sense to split the single week array into seven arrays of one day. This increases the likelihood of two congestion factor arrays to be similar, as there are seven times more arrays, and the array itself is seven times as short. To even further increase the likelihood of two arrays of congestion factors to be the same, we normalize all arrays to have the lowest value in the array to be zero, and store the offset of this normalization to later retrieve the real congestion factors. Each area ID in column three of Figure 5.7a actually contains a data structure with the mapping to the congestion factors. Figure 5.7b shows this data structure, and in this example it is the mapping to the congestion factors of area ID 1 to area ID 3 in the first layer. For each neighbouring area and each day of the week, the CH-algorithm stores an index number to the list of congestion factors and an offset number. The index number corresponds with the index to the congestion factor in the data structure in Section 5.3.2.3. The offset value is the value to which the stored congestion factors in the data structure have to be incremented.

Layer:	Area ID:	Neighbouring areas:
1 (2x2)	0	[1,2,3]
	1	[0,2,3]
:	:	:
2 (4x4)	0	[1,2,3]
:	:	:
	6	[1,3,4,5,7,13]
	7	[4,5,6,13,16,18,24]
:	:	:
3 (8x8)	0	[0,1,2]
:	:	:
	25	[19,22,24,26,27,28]
	26	[13,15,24,25,27,37,48,49]
:	:	:

(a)



(b)

FIGURE 5.7: This figure presents the mapping between the representative areas and the congestion factor data structure. (a) stores for each Area ID (column 2) which areas are within the vicinity of the area and contain a representative (column 3). Because multiple areas in different layers share the same area ID, the area IDs are sorted on layer (column 1). (b) presents the mapping structure that is behind each of the area IDs in column 3 of (a). The indices point to the indexes of the data structure of Section 5.3.2.3.

5.3.2.3 Storing the congestion factors

Index of CFs:	Congestion factors:
:	:
45	[4,4,4,4 ... 3,2,2,2 ... 4,4,4]
46	[7,6,6,6 ... 2,1,1,2 ... 8,8,8]
47	[8,8,7,7 ... 1,1,1,2 ... 8,8,8]
48	[9,9,9,8 ... 4,4,5,5 ... 9,9,9]
49	[4,4,4,4 ... 3,2,2,2 ... 4,4,4]
50	[8,8,7,7 ... 3,3,3,3 ... 7,7,7]
51	[4,3,3,3 ... 2,2,2,2 ... 4,4,4]
52	[5,5,4,4 ... 3,2,2,2 ... 5,4,4]
53	[8,8,8,9 ... 6,6,6,6 ... 8,9,8]
:	:

FIGURE 5.8: This figure presents the data structure of a small sub selection of a possible set of congestion factors.

The last data structure consists of the actual congestion factors. In the previous section we described how the representatives areas are connected to the congestion factors in this data structure. By accessing the row index, the CH-algorithm is able to retrieve the array of congestion factors. These factors still need to be incremented with the offset number, which is stored in the previous data structure as well. The resulting array consists of the congestion factors of a single day. In our case this is an array of 96 bytes, where each byte represents a 15 minute interval. Using interpolation, the CH-algorithm calculates the exact congestion factor for each moment in the 24-hour period of that day. Figure 5.8 presents a small sub selection of a possible set of congestion factors.

5.3.3 Calculating the TD-TT using the CH-algorithm

In this section, we give an example how the CH-algorithm retrieves a time dependent travel time using the precalculated congestion factors. Say, the CH-algorithm needs to calculate the time dependent travel time from node 1.986.221 to node 1.986.222 at a Monday at 00:48. The free flow travel time is already known, and is in this case 60 minutes. First, the algorithm checks at the lowest level if area ID 99 and 100 have representatives and are neighbours (see Figure 5.6). We know that they are neighbours at the lowest level (area ID 99 and 100), but in this example we assume both areas do not have a representative on this level (Figure 5.7a does not show this level). Therefore, the algorithm moves up one layer and checks

the same for area ID 25 and 26. These two areas are neighbours as well, and both have a representative as well (see Figure 5.7a). Therefore, it is possible to retrieve the congestion factors.

Let us assume that the index and offset values of area ID 25 to area ID 26 are the same as area 1 to area 3, we use the same information as shown in Figure 5.7b. The Monday congestion factors are stored in row 48 of the congestion factor data structure. The congestion factors consist of 96 bytes of information, but only 12 bytes are shown in Figure 5.7. Because the departure time is at 00:48, we need the third and fourth congestion factor within the congestion factor array. In this example this is 9 and 8, and with the increasing of these factors by the offset, we get 98 and 97. Using interpolation, the final congestion factor at 00:48 is $98 + \frac{00:48-00:45}{00:15} * (97 - 98) = 97.8$. If in this example, the free flow travel time would be 60 minutes, the time dependent (estimated) travel time would be $60/97.8 * 100 = 61.35$ minutes.

Chapter 6

Experiments & Results

This chapter describes the experiments to test our algorithm and evaluate its results. In Chapter 4, we already described the benchmark experiments to test the current algorithm. In this chapter we use the same datasets and performance indicators, but we conduct different experiments. We describe the data and the evaluation criteria in respectively Section 6.1 an Section 6.2. Section 6.3 describes the experiments to test our solution method. In Section 6.4 we analyse and discuss the results of the experiments.

6.1 Data

In this section, we discuss the three different types of data we need to conduct the experiments. These three data types are similar to the types we discussed in Section 4.1. First, we have the map data for which we use the same BeNeLux map as during the benchmarking experiments. We refer the reader back to Section 4.1.1 for a detailed description of the map data. Second, we use test data that contain a large number of different random origin and destination pairs that we use to test the TTC and the CH-algorithm. From the three test groups described in Section 4.1.2, we use test group 1 (path lengths) and test group 3 (congested areas) in this chapter as well. Test group 2 contains O-D pairs with different lengths towards their representatives. The representatives of this test group are in a 20x20 grid formation. Because the CH-algorithm uses a different set of representatives, each of the four test sets in the test group does not hold the property anymore that all origin and destination nodes are located at the same distance from their representative. Therefore, this test group becomes useless in the following experiments.

Last, we use the congestion data that is created during the preprocessing stage of the CH-algorithm. We refer the reader back to Section 5.3.1 for the full explanation of this preprocessing stage. In that section, we explained that we have three parameters we can adjust during the preprocessing stage. These three parameters are the threshold to accept an area to get a representative (α), the number of neighbours around a representative (inclusion width), and the number of layers of representative areas. We split the latter parameter in two different parameters, namely the "first layer" and the "number of layers". Preliminary results show that several longer paths tend to only share congestion factors at the 2x2 and the 4x4 layers. However, these layers produce inaccurate results as the representatives represent a too large area to be accurate. Therefore, we decide to add a parameter known as the "first layer", which is

the first layer in the congestion hierarchies where the congestion factors for each representative to all other representatives are calculated. Essentially, in this layer we compute the congestion factors as in the current algorithm. In this way, we force the algorithm to always find the congestion factors in this "first layer". The preliminary results also show that calculating all congestion factors in the 16x16 layer, did not significantly improve the accuracy compared to doing the same for the 8x8 layer. As calculating all congestion factors in 8x8 layer uses less memory that in the 16x16 layer, we chose the 8x8 layer as the "first layer". The "number of layers" is the number of times we split an area into four equal areas. If we have 9 layers, the bottom layer consist of $2^9 = 512 => 512 * 512 = 262.144$ areas.

Table 6.1 shows three different datasets with preprocessed congestion data of the CH-algorithm. The four columns of the table show the values of the parameters used to compute that congestion dataset. The first two columns contain the parameters that are fixed, and cannot be altered. Preliminary experiments show that it is enough to fixate the "first layer" value to 8x8, and to only differentiate among the value of α. The 4x4 layer and 16x16 layer were respectively not accurate enough or equally accurate compared to the 8x8 layer. The last two columns contain the parameters, for which different values can be simulated. Although more layers and a larger inclusion is available, it is possible to restrict these numbers during the experiments. Therefore, the values in these columns can be considered as the maximum values of the dataset.

	α	First layer	Number of layers	Inclusion width
Set 1	0.5	3 (8x8)	9 (512x512)	5
Set 2	1	3 (8x8)	9 (512x512)	5
Set 3	2	3 (8x8)	9 (512x512)	5
Set 4	1	3 (8x8)	8 (256x256)	9

TABLE 6.1: Overview of the parameter values of the three congestion datasets we use for the experiments.

6.2 Evaluation criteria

In this section, we describe the performance indicators to evaluate the CH-algorithm. In Section 4.2, we already described three different indicators to evaluate the *Travel Time Calculator (TTC)*, of those three indicators we only use the *free flow Travel Time Gap (ff-TTG)* during the following experiments.

Due to the large number of different parameter settings and datasets, we are going to adjust the ff-TTG indicator slightly to make it easier to present our results. Instead of presenting the ff-TTG value for each departure time over the day, we take the maximum ff-TTG of that day. It is easier to present this single value together with different values in a graph. Hence, instead of:

$$TTG_{od}^{ff}(\tau) = \left| \frac{TTC(od,\tau) - T(q_{od},\tau)}{T(q_{od},\tau)} \right| \tag{6.1}$$

We use:

$$TTG^{ff} = \max_{\tau} \{ \frac{1}{OD} \sum_{od \in OD} TTG_{od}^{ff}(\tau) \} \tag{6.2}$$

where OD stands for all od-pairs within a test set. We refer the reader back to Section 4.2 for a full explanation on the ff-TTG formula.

6.3 Setup of experiments

In this section, we describe the experiments to test the accuracy of the CH-algorithm. We also provide the memory usage, preprocessing time, and query time of the algorithm. Unfortunately, both the preprocessing and querying run in an experimental environment, making it difficult to compare these results with the TTC.

We use test group 1 and 3 to test the CH-algorithm, making use of three sets of preprocessed congestion data. We calculate for each O-D pair of the two test groups and each 15-minute interval, the time dependent travel time using the TTC and the CH-algorithm. We take the average over two workdays (Tuesday and Thursday), as during these days the most congestion is present. We need the time dependent travel times of both algorithms to compute the free flow travel time gap. Analysis of the travel time patterns was already conducted in Chapter 4, and therefore it is not included in this chapter.

We set-up two experiments: first a sensitivity analysis of the first three datasets and second an analysis of the accuracy of dataset 4. The previous section explains that there are four parameters that influence the accuracy of the algorithm. The first parameter "first layer" is fixed, while the other three are adjustable. These adjustable parameters are the "number of layers", the "inclusion width", and the α-factor. Table 6.2 shows the possible values of these parameters. We alter the parameters of the three congestion datasets to research which parameter set shows the best result, i.e., best accuracy compared to usage of resources. In total, we have $3 * 3 * 3 = 27$ different sets of parameters.

Parameter	Values
Number of layers	7 (128x128), 8 (256x256), 9 (512x512)
Inclusion width	1, 3, 5
Alpha	0.5, 1, 2

TABLE 6.2: Overview of the values of the parameters that are used for the sensitivity analysis.

The outcomes of the first experiment were used to construct the fourth dataset. The fourth dataset has less number of layers, but the inclusion width is greatly increased. For this dataset we compute the average ff-TTGs for a typical weekday (Tuesday and Thursday) on test group 1 and 3.

6.4 Results

In this section, we present our results of running the CH-algorithm. First, we show the performance of the CH-algorithm in terms of preprocessing time, query time, and memory usage. Second, we show a sensitivity analysis of the first three datasets under different parameter settings. In this way, we research how different parameter settings affect the performance of the CH-algorithm performs. Third,

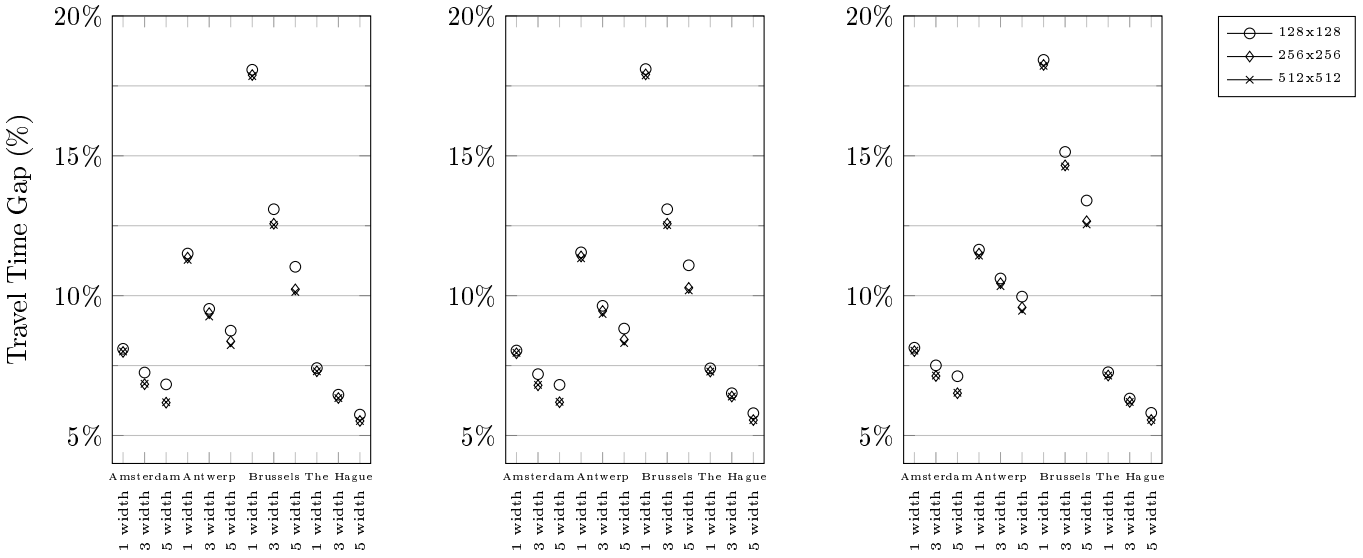
we present the results of dataset 4, where we use the insights of the sensitive analysis to adequately adapt the preprocessing step, to generate a dataset that results in higher accurate time dependent travel time.

Table 6.3 shows the values of the preprocessing time, memory usage, and query time of the different datasets. We observe several things. First, we observe high preprocessing times for all four of the dataset, as well as large differences among these computational times. The decreasing preprocessing time in dataset 1 to dataset 3 makes sense, as the increasing value for α among these sets lead to less congestion factors that have to be computed. Dataset 4 has an even lower preprocessing time, as the lowest level in this dataset is 8 (256x256), while the other three sets continue to 9 (512x512). Second, we observe a declining pattern in overall memory usage in dataset 1 to dataset 3, which has the same explanation as for the preprocessing times. For dataset 4, the memory usage is higher, as the parameter "inclusion width" is set to a higher value. Finally, we observe similar average query times among the dataset, which are between 0.02 and 0.03 ms. The parameter setting of the dataset does not influence the query time of the algorithm.

	Preprocessing	Query	Memory usage		
	time	time	Mapping nodes	Mapping areas	Factors
Set 1	29.9h	0.03ms	106mb	199mb	1387mb
Set 2	13.5h	0.02ms	106mb	104mb	731mb
Set 3	9.8h	0.02ms	106mb	38mb	331mb
Set 4	4.5h	0.03ms	95mb	97mb	809mb

TABLE 6.3: Overview of the preprocessing time, query time, and memory usage of the different preprocessed datasets.

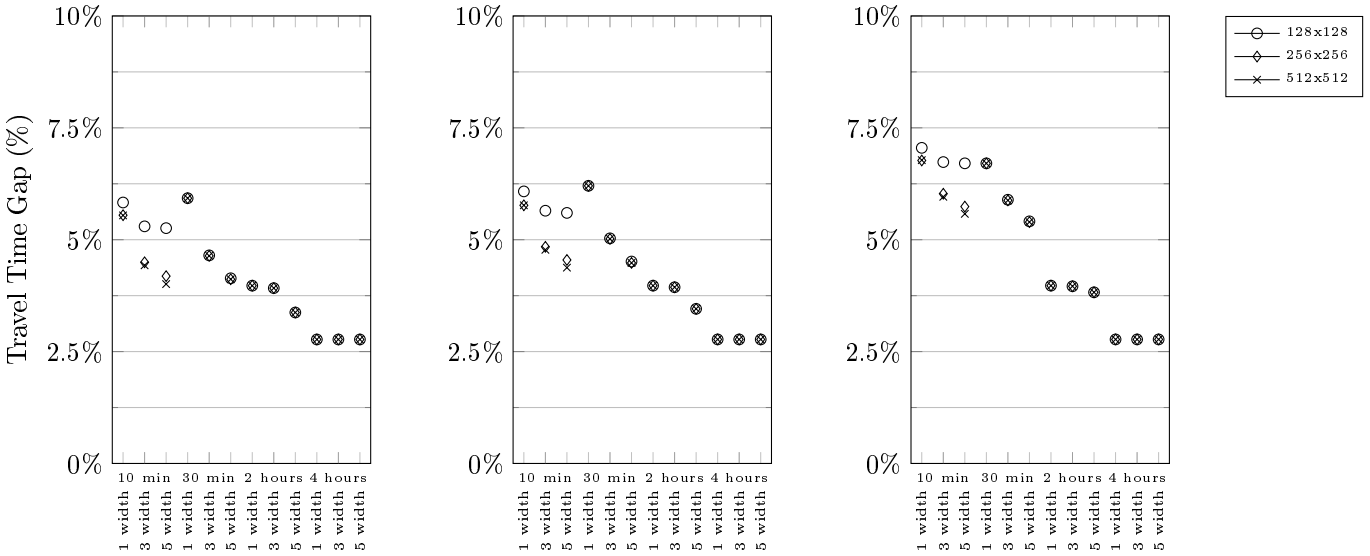
Figure 6.1 contains the sensitivity data of the first experiment for the "Areas" test group and Figure 6.2 contains the sensitivity data of the "Lengths" test group. To differentiate between the different values of the parameters "number of layers", "inclusion width", and the value of "α", we present the values as follows. The figures depict the average maximum free flow Travel Time Gap of a single set of parameters and one test set as a single data point. The three subfigures each have a different value for the parameter α. Over the x-axis we separate the four test sets of each of the test group, where each test set is further subdivided into three different inclusion widths. To depict the different values for the different "number of layers", we use different icons as shown in the legend of Figures 6.1 and 6.2.



(a) $\alpha = 0.5$ (b) $\alpha = 1$ (c) $\alpha = 2$

FIGURE 6.1: These figures depict the average maximum free flow Travel Time Gap of each run on each testset within the "Areas" testgroup. The x-axis differentiate the four different testsets (Amsterdam, Antwerp, Brussels, The Hague) together with the three different inclusion widths. Per tick on the x-axis, the figure shows the results of the tree different number of layers (128x128, 256x256, 512x512), which are visible in the legend as well.

In Figure 6.1 we observe that an increase of the inclusion width, an increase of the number of layers, and a decrease of α value all result in an increase of the accuracy. We also observe the same pattern among the different test sets as during the benchmarking experiments, meaning that the Brussels test set also has the highest TTG using the CH-algorithm. The increase of the accuracy for different alpha values is difficult to observe, we mainly see differences between $\alpha = 1$ and $\alpha = 2$, for the Antwerp and Brussels test set. We explain this rather small increase of accuracy as the congested areas of this test group are within areas with a large number of nodes. Therefore, the chance of assigning a representative to an area in these regions is large. This results in almost no difference between $\alpha = 0.5$ and $\alpha = 1$ graph. The increase of accuracy for different "number of layers" values is better observable. Especially for increasing values of the inclusion width, the accuracy increases when a dataset is used with more layers. This makes sense as less "number of layers" results in larger representative areas. Therefore, the chance of a destination node being in the same area or in the direct connected area at the lowest level is relatively low. Whereas, if the inclusion width is larger, this chance is higher and the CH-algorithm actually benefits from smaller representative areas. Lastly, we observe a clear increase of the accuracy when the inclusion width is increased. Especially in the Brussels area, we observe an increase from 18% at an inclusion width of 1, to 10-11% with an inclusion width of 5.



(a) $\alpha = 0.5$ (b) $\alpha = 1$ (c) $\alpha = 2$

FIGURE 6.2: These figures depict the average maximum free flow Travel Time Gap of each run on four of the seven test sets within the "length" test set. The x-axis differentiate the four different testsets (10 min, 30 min, 2 hours, 4 hours) together with the three different inclusion widths. Per tick on the x-axis, the figure shows the results of the tree different number of layers (128x128, 256x256, 512x512), which are visible in the legend as well

In Figure 6.2 we see similar behaviour in the 10-minute test set, but the test sets of 30-minute and longer show different patterns. Because longer paths share congestion factors at a higher layer, there is no difference between having a maximum depth of 7 (128x128), 8 (256x256), or 9 (512x512). The inclusion width does influence the accuracy, but for the longest path of 4 hours this does not have an effect as the origin and destination of a path most likely only share congestion factors at the highest level of 3 (8x8). In contrast to the "areas" test set, this dataset is more sensitive to the value α. This has to do with the fact that the origin and destinations are selected from the complete map, instead of just congested areas.

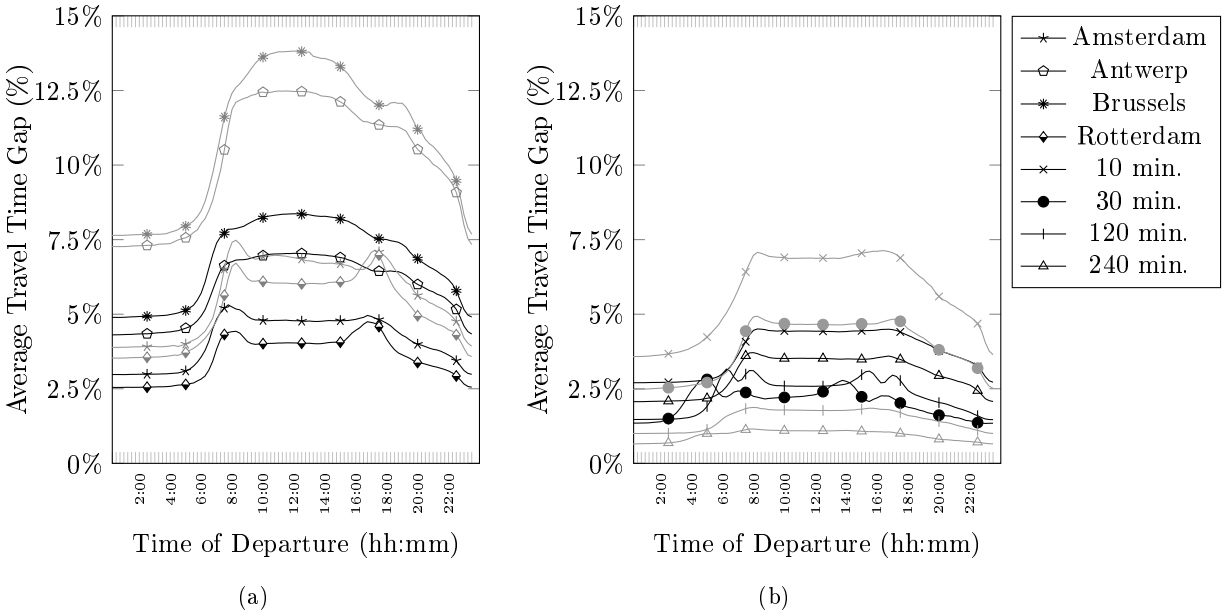


FIGURE 6.3: This figure depicts the travel time gap of the "Areas" (a) and "Path length" (b) test sets. Both graphs present the average ff-TTG of a Tuesday and a Thursday. The black lines represent the values resulting from using the CH-algorithm with dataset 4, while the gray lines depict the results of using a similar size congestion matrix and the TTC.

Figure 6.3 presents the ff-TTG over different departure times of a typical weekday (Tuesday and Thursday), for the four test sets of the "Areas" test group and the four test sets of the "Lengths" test group. The gray line depicts the average ff-TTG of the TTC algorithm using a 42x42 grid matrix, while the black line depicts the average ff-TTG of the CH-algorithm using the parameters, $\alpha = 1$, inclusion width = 9, and first layer is 8 (256x256). The line consist of 96 measurements, which are the departure times at a 15-minute interval during a single day.

We observe large improvements on all the four test sets within the "Area" test group. The ff-TTG in the test sets Amsterdam, Antwerp, Brussels, and The Hague dropped on average 32%, 44%, 40%, and 35% respectively during the rush hour period of a weekday. In the four test sets within the "Lengths" test group, we observe both large decreases as increases of the ff-TTG. For the test sets of "10-min." and "30-min.", the ff-TTG dropped on average 39% and 27% respectively during the rush hour period of a weekday. However, for the test sets of "2-hr." and "4-hr.", we observe an increase of respectively 60% and 100%. These are large percentage increases, but the absolute differences between the ff-TTGs are relatively low. The travel time gap increases respectively from 1.7% to 2.7% and from 1.0% to 2.0%. This means that the duration of a 4-hour trip (in free flow) on average has an additional deviation of 2.5 minutes.

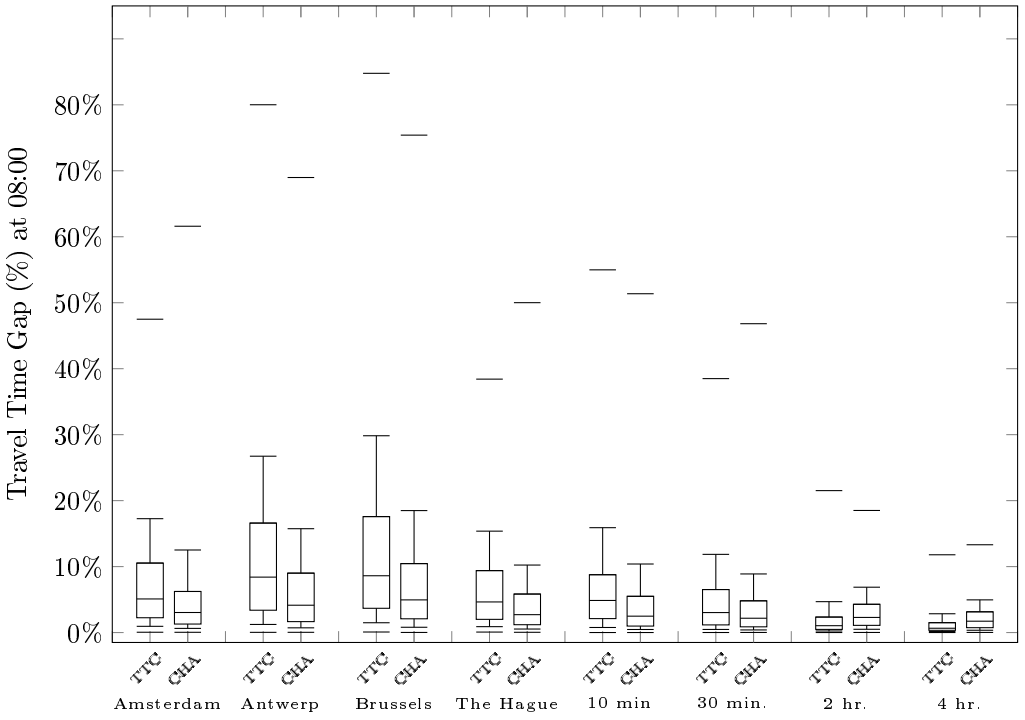


FIGURE 6.4: This figure depicts the variation in the average travel time gap. It is presented as a box plot, using the "seven-figure summary"[42]. The horizontal lines in each single box plot represent the following (top to bottom): sample maximum, 90^{th} percentile, 75^{th} percentile, 50^{th} percentile, 25^{th} percentile, 10^{th} percentile, and sample minimum.

For every test set that is presented in Figure 6.3, Figure 6.4 presents the box plots of the variation of the TTG. For each of the test set, we ran both the TTC and the CH-algorithm. The both box plots of the variation are presented in Figure 6.4. We observe smaller interquartile ranges (IQR) for all "Areas" test sets and for the "10-min." and "30-min." test sets. The reduced size of the IQR's corresponds to the same downward trend in the average TTG. Again, the test sets of "2-hour" and "4-hour" show an opposite pattern, meaning the IQR values have increased.

6.5 Conclusion

In this chapter, we assessed the performance of the CH-algorithm. We created four different datasets, which we generated during the preprocessing phase of the CH-algorithm. Three of the four datasets were used to test 27 different parameter sets, to analyze the performance of the different parameters. The observations of the results of the different parameter sets, led to the fourth dataset. We used this fourth dataset for our final analysis of the performance of the CH-algorithm.

First, we observed high preprocessing times for all four of the datasets. These high computational times were a result of using the Dijkstra's algorithm, which is known to be easy to implement, but slow in large scale road networks. Therefore, the measured times are not a representation of the actual speed of the algorithm. A different implementation using a faster algorithm like Highway Node Routing or Contraction Hierarchies, would have led to different results. We consider the query times of the algorithm, being between 0.02 and 0.03 ms as extremely fast. This is a result of a compact data structure and fast

table and list look ups. The memory usage is large, but the fourth dataset is equal in size as the grid dataset of the TTC we compare it to.

Second, we observed a large improvement in accuracy using eight layers over seven layers for shorter paths, while this increase is not visible using nine layers over eight layers. In the "area" test group we observe similar outcomes. Using $\alpha = 1$ instead of $\alpha = 2$, leads to an increase of the accuracy of an additional 1% point. Between $\alpha = 0.5$ and $\alpha = 1$, this increase is only $\pm 0.3\%$ point. In the "area" test group we observe improvements for the Antwerp and Brussels test set, but for the Amsterdam and The Hague test set the value of α has no effect. For all shortest path test sets (all areas and 10 min.) we see an improvement of the accuracy using larger inclusion widths.

Finally, we used the fourth dataset to test the performance of the CH-algorithm. We observe on average an increase of accuracy of 34% in congested urban areas during a weekday, and on average an increase of accuracy of 28% for trips with a length of at most 30 minutes. These values are weekday averages, during rush hour these values increase to 38% and 33% respectively. However, longer trips have on average a decrease of accuracy of 85%. This decrease is something we expected and accepted beforehand, as it heavily benefits the accuracy for shorter trips. The decrease of accuracy looks much larger than the increase we manage to make, but this decrease has less effect as the travel time gap for longer trips still is lower than for shorter trips. While the travel time gap increases respectively from 1.7% to 2.7% and from 1.0% to 2.0% for 2 hour and 4 hour trips, the travel time gap decreases from 5.5% to 3.4% and from 4.8% to 3.5% for 10 minute and 30 minute trips. Therefore, the CH-algorithm causes a 4-hour trip (in free flow conditions) on average to have an additional deviation of 2.5 minutes. We conclude that the CH-algorithm outperforms the TTC both in congested areas and in the case of shorter trips.

Chapter 7

Conclusions and Recommendations

In Chapter 1, we stated four research questions to which the answers contribute to our research goal. The chapters that followed answered these research questions. Chapter 2 discussed the literature related to shortest path algorithms that focus on computing fast (time dependent) travel times. In Chapter 3 and 4, we described the current algorithm and analyzed its performance. In Chapter 5, we designed the CH-algorithm and in Chapter 6 we assessed its performance. In this chapter, we discuss the conclusions (7.1) of this research and provide a discussion (7.2), and finish with our recommendations for further research (7.3).

7.1 Conclusions

We developed a travel time algorithm that calculates the time dependent travel time between an origin node o and a destination node d in a directed graph with time varying weights at departure time τ, fast enough so it can be used in VRP solving algorithms. The CH-algorithm is able to work customer independent, meaning we do not use any customer specific data or information for configuration purposes. In this section, we discuss the conclusions of this research based on our findings.

First, we determined the accuracy of the TTC. The TTC performs as desired for longer trips (>2h) and in areas with a low level of congestion. However, it performs poorly on relatively short paths, in congested areas and/or where the locations of the representatives are distanced from the main roads of the road networks. We use these findings to design our new algorithm, by using the characteristics of the current algorithm for the retrieval of travel times for longer paths and designing a different approach for shorter paths.

Second, we developed the *Congestion Hierarchies Algorithm (CH-algorithm)* that calculates the travel time between two points on a map, for any departure time at any day. The results of the test ran on the CH-algorithm shows that the CH-algorithm is generally more accurate than the TTC. The results show that the CH-algorithm is on average 34% more accurate in congested areas, compared to the TTC. It also shows, that the CH-algorithm is on average 28% more accurate for trips with a length of at most 30 minutes, compared to the TTC. However, for trips longer than 30 minutes, the CH-algorithm is on average 85% less accurate for trip longer than 30 minutes, compared to the TTC. These three observations

makes sense, as we decided beforehand that some accuracy would be sacrificed for longer trips for the benefit of the shorter trips, and this is a design decision of the algorithm. We measured the performance of the algorithm based on the preprocessing time, query time, and memory usage. The preprocessing time of our algorithm was with 4,5 hr relatively large, but the cause was the use of Dijkstra's algorithm instead of a faster SPP-algorithm. If a different algorithm was used, the preprocessing time would have been several times faster. The query time of the CH-algorithm is with 0.03 millisecond considered as extremely fast, but was measured outside of the ORTEC software framework. Therefore it is not possible to make a comparison with the TTC, but we think that as the query time is extremely fast, it will not have a negative impact on the overall performance of ORD. The memory usage of the CH-algorithm is 1001 MB, which is similar to the memory usage of the TTC. All together, we state that the performance of the CH-algorithm will neither outperform or worsen the performance of the TTC.

To conclude, the CH-algorithm improves the accuracy of TD-TT estimations and is fast enough to be used in vehicle routing algorithms. Literature shows that improving the TD-TT accuracy is beneficial for the feasibility of the vehicle routing solution, although we were not able to proof this ourselves. Clients using ORD benefit from this new algorithm, as the constructed vehicle routing solutions are more reliable, meaning the deliveries are on time more often. Especially clients with deliveries clustered together in a relatively small area will benefit from the CH-algorithm, for instance, inner city parcel deliveries.

7.2 Discussion

In the previous section, we provided the conclusions of our research. In this section, we critically assess these conclusions, and put them in a broader context.

In this paragraph, we discuss the loss of accuracy in the data and in the CH-algorithm we developed. First, we state that we are able to compute the exact time dependent travel times, but the exactness is questionable. We use the time dependent travel times over the free flow shortest path, instead of using the time dependent shortest path. We found a maximum average deviation of at most 2% during rush hour periods, and assumed this deviation was low enough to use the free flow shortest path. Second, we assumed that the provided data was the exact representation of the travel time. However, the travel time data of the third party is also just an approximation of the actual travel times. The data looses precision as it is rounded to whole numbers and data is only available for a single week and only on a 15-minute basis. Leaving aside the fact that the data describes historical patterns, and therefore neglects actual traffic conditions and real time distortions. Third, we measured a travel time gap during the night period on all test sets we ran experiments on. A travel time gap indicates that congestion is present, something that we would not expect during the night as presumably no traffic will be present. The explanation is that within the graph, certain edges have a higher FF-TT than the TD-TT during the night. As the FF-TT is assigned to a road type, this effectively means that on certain edges this is an overestimation of the actual travel time. However, these differences are not consistent for all edges, meaning the calculated congestion factors will not fully represent the actual present travel time difference. Finally, the congestion factors are stored as a byte, meaning they have a maximum of 256 different values. Of these 256 values, only a limited range is used, namely between 60 and 100. So even if both the origin and the destination node are a representative, there is a rounding error within the time dependent travel time, with a maximum of 0.5%. We recommend to research the potential of scaling the congestion factors to the complete 256 spectrum of the byte.

In this paragraph, we question the need of high accuracy of time dependent travel time calculation. We know from literature that including time dependency improves the feasibility of the solution of the VRP. However, we were not able to prove this observation in our research as well. We imagine that including time dependent travel times improves the accuracy, but we question whether or not it also improves the feasibility of the solution. This means that the time windows of the addresses are still met, even though the arrivals at the addresses differ due to inaccurate travel time predictions. Especially when the time windows at the addresses are wide enough, and the non-absolute average travel time gap is around zero, we wonder if an even higher accuracy has an impact on the feasibility of the VRP solutions. We recommend to research the effect on VRP solutions using real customer data, to see what the effects on the feasibility of the solutions are.

In this final paragraph, we put our solution method in perspective with possible other methods that produce exact time dependent travel times. A solution method would be to recalculate the congestion factors for a specific set of addresses, each time the algorithm calls for optimization. As an optimization call can take up to several hours to complete, it does not matter if congestion factors have to be computed at the beginning of the call, as long as it only takes a fraction of the total optimization call. This is a solution when the number of addresses is limited (say $<$100), as computation time increases quadratically with the number of addresses. Also, the addresses should be located around the same region, so the mutual distances are limited as well. This approach is therefore limited to smaller customers with a regional importance.

7.3 Further research

In this section, we present our recommendations for further research. The following ideas are the result of the findings of the CH-algorithm, or ideas that we did not pursue due to time restrictions of this research. First, we have some general remarks for further research, then we will follow the four preprocessing steps of the algorithm to explain our recommendations.

General

The experiments of this research were conducted on the BeNeLux map, a map significantly smaller than the Western European or USA map, used in experiments on algorithms found in literature. We think the experiments within this research should also be ran on a larger map to check if the outcomes are similar to the results we found in this research. Also, we think more research should be performed in finding more efficient ways of compressing the data, for instance, the congestion factors. We already pointed out that the stored data did not use the full range within a byte. But also reducing the number of time intervals could be a possibility, for instance, reducing the night period to a single point in time.

Partitioning

Currently, the CH-algorithm uses the quad-tree algorithm to partition the road network into smaller areas. The quad-tree algorithm was extremely suitable due to its easy implementation and multilevel characteristics. The resulting partition is based on square-like areas, that do not take into account the characteristics of the road network itself. In Section 5.2, we dismissed the idea of using the METIS algorithm, due to its computational time and the necessity to use undirected graphs. However, Delling et al. [43] use the METIS algorithm (together with a similar algorithm called PUNCH) for partitioning their road networks, which they use as input for their shortest path algorithm. This indicates that

the METIS algorithm is applicable on large road networks, which makes us rethink the dismissal of the algorithm. The METIS algorithm produces areas based on the graph characteristics, rather than the geographical locations of the nodes. This makes us believe it creates better partitions than our combination of square areas and the Dijkstra algorithm. We recommend further research in the use of different partitioning algorithms that include the characteristics and the so-called natural cuts of a road network.

Selecting representatives

During the process of selecting the representatives, we perform two consecutive steps. First, we decide which areas receive a representative node, based on the number of nodes within the area. Second, we decide which node of that area is going to be the representative, which in our case is the node closest to the centroid of the area.

We recommend to further research the effects of different methods that decide to allocate a node to an area. Currently, this is a linear relation between the average number of nodes in the area at a certain level and the actual number of nodes within an area. We think it is possible to use a non-linear relationship, in which some levels have a high probability to assign a representative to an area and lower levels have a lower probability. In this way, we might be able to remove the necessity to have a layer where all congestion factors are computed. An even better possibility is to allocate representatives based on the congestion in the area. In that way, congested areas are preferred over well accessible areas. We recommend to research proper ways to indicate the level of historical congestion of the edges and to use this as an indicator to allocate representatives.

In the second step, we determine the most central node within an area. However, this does not necessary has to be the best node to represent the area, as it could be a junction in the middle of a rural area. We recommend to look for fast methods to determine the most important node of an area. An idea would be to pick the node that is on most shortest paths between each of the border nodes of the area.

Generating congestion factors

The CH-algorithm currently generates the congestion factors based on a single representative within the area. We do this because this a simple method to generate the congestion factors. However, it is questionable if congestion factors based on a single node are a good representation of a complete area. We recommend to research the possibility to remove the idea of the representative (node) completely, and instead use some kind of average of the area. So, instead of a single node, we use an average of all or a subset of nodes and/or edges of the graph.

Mapping the nodes

We have no recommendations for the last preprocessing step, as this was an addition to the CH-algorithm to overcome a shortcoming of the quad-tree algorithm. If we use a different partition algorithm that includes the natural characteristics of a road network, this step becomes obsolete as the nodes in the partitions are directly mapped to the right areas.

Bibliography

[1] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1): 80–91, 1959.

[2] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*, volume 18. Siam, 2014.

[3] Adrianus Leendert Kok, EW Hans, and JMJ Schutten. Vehicle routing under time-dependent travel times: the impact of congestion avoidance. *Computers & Operations Research*, 39(5):910–918, 2012.

[4] Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 474–477. ACM, 2010.

[5] Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

[6] Gernot Veit Eberhard Batz. *Time-Dependent Route Planning with Contraction Hierarchies*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2014, 2014.

[7] Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle dispatching with time-dependent travel times. *European journal of operational research*, 144(2):379–396, 2003.

[8] Christophe Lecluyse, Tom Van Woensel, and Herbert Peremans. Vehicle routing with stochastic time-dependent travel times. *4OR*, 7(4):363–377, 2009.

[9] Tom Van Woensel, Laoucine Kerbache, Herbert Peremans, and Nico Vandaele. Vehicle routing with dynamic travel times: A queueing approach. *European journal of operational research*, 186(3): 990–1007, 2008.

[10] Said Dabia, Stefan Ropke, Tom Van Woensel, and Ton De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396, 2013.

[11] Stefanie Kritzinger, Fabien Tricoire, Karl F Doerner, and Richard F Hartl. Variable neighborhood search for the time-dependent vehicle routing problem with soft time windows. In *Learning and Intelligent Optimization*, pages 61–75. Springer, 2011.

[12] Alberto V Donati, Roberto Montemanni, Norman Casagrande, Andrea E Rizzoli, and Luca M Gambardella. Time dependent vehicle routing problem with a multi ant colony system. *European journal of operational research*, 185(3):1174–1191, 2008.

[13] Hideki Hashimoto, Mutsunori Yagiura, and Toshihide Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, 5(2): 434–456, 2008.

[14] Yaguang Li, Dingxiong Deng, Ugur Demiryurek, Cyrus Shahabi, and Siva Ravada. Towards fast and accurate solutions to vehicle routing in a large-scale and dynamic environment. In *Advances in Spatial and Temporal Databases*, pages 119–136. Springer, 2015.

[15] Simona Mancini. Time dependent travel speed vehicle routing and scheduling on a real road network: The case of torino. *Transportation Research Procedia*, 3:433–441, 2014.

[16] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, 2008.

[17] Michel Gendreau, Gianpaolo Ghiani, and Emanuela Guerriero. Time-dependent routing problems: A review. *Computers & Operations Research*, 64:189–197, 2015.

[18] Brian C Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. *Rapport technique, Massachusetts Institute of Technology*, 2004.

[19] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1 (1):269–271, 1959.

[20] George B Dantzig. Linear programming and its extensions, 1963.

[21] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[22] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *arXiv preprint arXiv:1504.05140*, 2015.

[23] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: an empirical case study from public railroad transport. *Journal of Experimental Algorithmics (JEA)*, 5:12, 2000.

[24] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. Geometric containers for efficient shortest-path computation. *Journal of Experimental Algorithmics (JEA)*, 10:1–3, 2005.

[25] Ulrich Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. *Geoinformation und Mobilität-von der Forschung zur praktischen Anwendung*, 22:219–230, 2004.

[26] Ekkehard Köhler, Rolf H Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. *9th Dimacs implementation challenge*, 2006.

[27] Daniel Delling and Dorothea Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, pages 207–230. Springer, 2009.

[28] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *Algorithms–Esa 2005*, pages 568–579. Springer, 2005.

[29] Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *Experimental Algorithms*, pages 66–79. Springer, 2007.

[30] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 210–219. Society for Industrial and Applied Mathematics, 2001.

[31] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[32] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566–566, 2007.

[33] Takuya Akiba, Yoichi Iwata, Ken-ichi Kawarabayashi, and Yuki Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *ALENEX*, pages 147–154. SIAM, 2014.

[34] Stuart E Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[35] Koen Demkes. Automated tuning of an algorithm for the vehicle routing problem. 2014.

[36] Tim van Dijk. Tuning the parameters of a loading algorithm. 2014.

[37] Rolf H Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning graphs to speedup dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 11:2–8, 2007.

[38] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[39] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[40] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.

[41] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[42] Arthur Lyon Bowley. *An elementary manual of statistics*. PS King & son, Limited, 1915.

[43] Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Customizable route planning in road networks. *Transportation Science*, 2015.

[44] Irene Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25(12): 905–910, 1982.