Innovatieve algoritmes
voor de planning en routering van multimodaal transport

Innovative Algorithms
for the Planning and Routing of Multimodal Transportation

Sofie Demeyer

Promotoren: prof. dr. ir. M. Pickavet, dr. P. Audenaert
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. D. De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2012 - 2013

UNIVERSITEIT
GENT

ISBN 978-90-8578-589-7
NUR 980
Wettelijk depot: D/2013/10.500/22

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur
Vakgroep Informatietechnologie

Promotoren: prof. dr. ir. Mario Pickavet
dr. Pieter Audenaert

Leden van de examencommissie:

prof. dr. ir. Didier Colle, Universiteit Gent
prof. dr. ir. Piet Demeester, Universiteit Gent (secretaris)
prof. dr. Veerle Fack, Universiteit Gent
dr. ir. Steven Logghe, Go-Mobile
dr. Tom Michoel, Roslin Institute, University of Edinburgh
prof. dr. ir. Hendrik Van Landeghem, Universiteit Gent (voorzitter)

Universiteit Gent
Faculteit Ingenieurswetenschappen en Architectuur

Vakgroep Informatietechnologie
Gaston Crommenlaan 8 bus 201, B-9050 Gent, België

Tel.: +32-9-331.49.00
Fax.: +32-9-331.48.99

Proefschrift tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen:
Computerwetenschappen
Academiejaar 2012-2013

- voor papa -

Dankwoord

Een doctoraat is een beetje zoals een film. In het begin heb je enkel een titel, en als je geluk hebt ook een korte trailer. De rest ontdek je pas later.

Meestal volgt het scenario de klassieke structuur. Eerst raak je vertrouwd met de wereld waarin zich alles afspeelt en wordt het probleem in kaart gebracht. Vervolgens wordt dit probleem zo goed of zo kwaad mogelijk opgelost, waarbij er zich af en toe een nieuw probleem stelt. De spanning wordt verder opgebouwd tot het verhaal eindigt in een spetterende apotheose, waar jullie op dit eigenste moment getuige van zijn.

Het is quasi onmogelijk om één genre op deze film te plakken. Het is zowel een avonturenfilm als een komedie, zowal een science fiction film als een drama.

Maar bij een film komt meer kijken dan enkel een mooi verhaal: de juiste beelden, de gepaste muziek, een mooi decor en karakteristieke personages. Het is zeker geen one-(wo)man-show en alle mensen die meewerkten worden bedankt in de aftiteling. Normaalgezien loop je nu de zaal uit of zap je snel weg, maar deze ene keer wil ik jullie vragen de aftiteling toch te lezen. Misschien vind je wel je naam erin terug, want jullie hielpen allemaal mee aan de film van mijn doctoraat, al was het maar een figurantenrol.

voorzitters

Bedankt om de vakgroep en onderzoeksgroep
met zoveel enthousiasme te leiden

Piet Demeester
Daniël Dezutter
Paul Lagasse

promotoren

Dank je wel voor jullie kritische kijk, hulp en steun
tijdens mijn doctoraatsonderzoek

Mario Pickavet
Pieter Audenaert

samenwerking
Bedankt voor alle vruchtbare discussies

de projectpartners van MultiTr@ns
de projectpartners van MobiRoute
Tom Michoel

administratie en ondersteuning
Achter de schermen bij IBCN gebeuren wonderbare dingen,
mede dankzij volgende mensen

Martine & Davinia
de financiële dienst
Sandra & Sabrina
the A(dmin)-team

bureaugenoten
Bij IBCN kon je mij eerst in bureau 3.17 vinden, daarna in bureau 3.13
Ik had dus de eer om met heel wat mensen de werkruimte te delen

Abhishek	Eli	Maarten D.	Sahel	Stein	Tom
Bart P.	Florian	Maarten H.*	Sam	Stijn	Ward
Bart S.	Frank	Maarten S.	Sofie	Thijs*	Willem
Dimitri	Fréderic	Sachin	Steven	Tim	Wouter

**de 'elite' van de DNA-groep: keep on the good work!*

collega's
In de wandelgangen kom je al eens iemand tegen
Bedankt voor de babbel tijdens de koffiepauze, in de gang,
tijdens de lunch, waar dan ook . . .

Andy	Eric	Jonathan	Peter	Tom
Bert	Femke DB	Koen	Pieter	Wannes
Dieter DW	Femke O	Nicolas	Thomas DM	Wei
Dieter V	Jelle	Olivier	Thomas D	. . .

apero
Het begon allemaal die vrijdagavond in Gent . . .

Sarah & Pieter
Bart & Elke
Stijn & Cedric
Jan & Ellen
Benoît & Lisa
Sofie & Stefan
Bruno & Delphine

fontein
In Ieper staat een fontein
Laat die nu voor altijd de onze zijn

Maarten & Marieke
Karen & Frédéric
Karlien & Bert
Isabelle & Geert

vrienden van de animatiefilm
De tussentekeningen in mijn onderzoek:
de les, de spelletjesavonden, Annecy, . . .

Ann
Anne-Lise
Bart
Brecht
Carl
Charlotte
Iza
Geert
Philippe
Olivier
Richard
. . .

Klup Artistik
Allemaal mensen met een hoekje af,
een leuk hoekje af

Carl
Evy
Maarten
Nathalie
Olivier
Stefan
. . .

buiten categorie
Een aantal mensen die mijn resterende vrije tijd opvrolijkten
een filmpje, een movieshoot, een etentje,
een drankje, een zombie, een babbel, ...

Annelies & Björn
Bert
Filip
Tim
Zilke
en vele anderen

familie
Mijn steun en toeverlaat

mama & papa	DANK JE WEL, voor alles!
Evelien	Een zus uit de duizend!
Steven & Liesbeth	Op jullie kan ik altijd rekenen.
Noor & Fran	Een eer om jullie 'tantie' te mogen zijn.
meme	Bedankt om mij in alles te steunen.

Alle rechten voorbehouden
Iedere gelijkenis met bestaande gebeurtenissen
en/of personages berust op louter toeval

Sofie Demeyer
Gent
maart 2013

Table of Contents

List of Figures

List of Tables

Samenvatting
– Summary in Dutch –

Vooraleer GPS navigatiesystemen en online routeringsapplicaties gemeengoed werden, waren papieren kaarten het enige hulpmiddel om de beste route tussen twee locaties te vinden. Dit kon een heel tijdrovende bezigheid zijn. Om digitaal in een heel korte tijd de beste route te bepalen, dienden een aantal snelle kortste pad algoritmen voor transportnetwerken ontwikkeld te worden. Deze algoritmen gaan op zoek naar de kortste, de snelste of de meest betrouwbare route, of naar een route die tegelijkertijd kort, snel en betrouwbaar is. Dit betekent dat er meerdere kostentypes zouden gemodelleerd moeten worden in het netwerk. Doordat sommige transportmodi hun maximale capaciteit quasi bereikt hebben, wordt multimodaal transport gezien als een waardevol alternatief om de bestemming te bereiken. Dit wordt gedefinieerd als die vorm van transport waarbij meer dan één vervoersmiddel gebruikt wordt tijdens eenzelfde trip.

Deze dissertatie stelt vernieuwend onderzoek voor over het zoeken naar kortste paden in multimodale netwerken (voor zowel personen- als vrachtvervoer). Er wordt hierbij op verschillende fronten gewerkt. Eerst wordt een flexibel multimodaal netwerkmodel ontwikkeld. Vervolgens kijken we naar de routeringsalgoritmen. Er wordt onderzocht hoe een aantal van deze algoritmen versneld kunnen worden. Daarnaast bestuderen we alternatieve kostenstructuren en worden er algoritmen ontwikkeld die hiermee overweg kunnen.

In het multimodale netwerkmodel van hoofdstuk 2, worden de netwerken van iedere transportmodus voorgesteld door een aparte laag. Deze lagen zijn dan onderling verbonden door overslagtakken. Het voordeel van deze aanpak is de eenvoud waarmee een transportmodus toegevoegd of verwijderd kan worden. Wij hebben ervoor geopteerd om in de graaf (d.i. het netwerk) enkel de locatieafhankelijke informatie te verwerken en alle informatie over de kosten te modelleren als kostobjecten die aan de takken toegevoegd worden, wat betekent dat het tijdsafhankelijke netwerk model gebruikt wordt om tijdsafhankelijke informatie te modeleren. Er worden meerdere kostenmodellen voorgesteld. Enkelvoudige kosten zijn statische kosten met betrekking tot een enkel objectief, zoals bijvoorbeeld de afstand. Sommige transportkosten zijn tijdsafhankelijk, zoals bijvoorbeeld de variërende reistijd op de snelweg. In dit onderzoek worden tijdsafhankelijke kosten voorgesteld door (stuksgewijze lineaire) functies, die voorgesteld kunnen worden door een eindig aantal waarden. Daarnaast hebben bepaalde objectieven in transportnetwerken een inherente hoeveelheid onzekerheid. Bijvoorbeeld, reis-

tijden kunnen variëren door toevallige externe factoren (zoals het rijgedrag van automobilisten en de weersomstandigheden). Deze stochastische kosten worden voorgesteld door distributies, die in dit onderzoek vereenvoudigd worden tot een aantal percentielwaarden. In sommige gevallen wil men niet optimaliseren naar een enkel objectief, maar moeten meerdere objectieven tegelijkertijd geoptimaliseerd worden. De kostobjecten bestaan dan uit meerdere enkelvoudige kosten.

Naast een multimodaal netwerkmodel waarin meerdere kostentypes gemodelleerd kunnen worden, ligt de nadruk van dit doctoraatsonderzoek op het ontwerp van kortste pad algoritmes die ofwel performant zijn (d.i. ze hebben een korte uitvoeringstijd), ofwel aangepast zijn aan specifieke kosten. In eerste instantie ligt de nadruk op de performantie van de kortste pad algoritmen, waarbij er verondersteld wordt dat alle kosten enkelvoudig zijn. Twee nieuwe bestemmingsgerichte algoritmen worden voorgesteld: het voorgangersalgoritme en het accountingalgoritme. Het voorgangersalgoritme onderzoekt enkel de knopen waarvan de geschatte afstand tot de bestemming kleiner of gelijk is aan de geschatte afstand van zijn voorganger (d.i. de vorige knoop op het pad) tot de bestemming. Op die manier worden enkel stappen in de richting van de bestemming toegestaan. Doordat kortste paden in transportnetwerken veelal een (kleine) omweg maken om de bestemming te bereiken, wordt een tolerantiefactor ingevoerd die ervoor zorgt dat er meer paden gevonden worden. Daarenboven wordt er een verbetering voorgesteld die de gebieden rondom de oorsprong en de bestemming volledig onderzoekt. Een andere optimalisatie vergelijkt een knoop met zijn k-de voorganger in plaats van met zijn directe voorganger. Dit zorgt ervoor dat er meer knopen onderzocht worden en verhoogt aldus de kans om een pad te vinden. In het accountingalgoritme wordt een geschiedenis van het pad bijgehouden waarin geregistreerd wordt hoeveel stappen dit pad reeds in de goede en/of de verkeerde richting zette. Op die manier kan een pad meerdere malen de verkeerde richting uitgaan. Vervolgens worden twee optimalisaties voorgesteld: queue-optimalisatie en de feedbacklus. Bij queue-optimalisatie worden de berekeningen versneld door telkens een knoop te vergelijken met de knoop die voordien onderzocht werd. Het feedbacklusmechanisme daarentegen garandeert dat er telkens een pad gevonden wordt door iteratief de tolerantiefactor te verhogen.

In plaats van één enkel objectief dienen in transportnetwerken veelal meerdere objectieven tegelijkertijd geoptimaliseerd te worden. Hiervoor ontwikkelde Martins een multi-objectiefalgoritme. Dit algoritme resulteert in een verzameling van Pareto-optimale paden. Dit zijn paden die niet verder verbeterd kunnen worden in één objectief zonder hierbij andere objectieven te benadelen. In deze dissertatie wordt dit multi-objectiefalgoritme versneld, en dit op twee manieren. Het algoritme van Martins berekent de Pareto-optimale paden van één enkele knoop naar alle andere knopen in het netwerk. Aangezien we meestal enkel geïnteresseerd zijn in de verzameling van Pareto-optimale paden tussen één enkele oorsprong en bestemming, zouden de berekeningen kunnen afgebroken worden van zodra deze verzameling gevonden is. In dit onderzoek wordt een nieuwe stopconditie voorgesteld, die zegt dat de zoektocht gestaakt kan worden wanneer de vector met de minimale waarden van de tijdelijke verzameling gedomineerd wordt door één

van de labels van de bestemmingsknoop. Er wordt theoretisch bewezen dat wanneer deze stopconditie geldt, de verzameling van Pareto-optimale paden gevonden is. Vervolgens stellen we een bidirectionele versie van het multi-objectiefalgoritme voor, waarbij de nadruk opnieuw ligt op het definiëren van een strikte stopconditie. Dit algoritme onderzoekt het netwerk simultaan vanuit de oorsprong en de bestemming en telkens wanneer de twee zoektochten elkaar tegenkomen wordt een nieuw kandidaatspad gevormd. De twee zoektochten kunnen afgebroken worden van zodra de vector met de som van de minima van de tijdelijke verzamelingen gedomineerd wordt door één van de resulterende paden. Er wordt theoretisch bewezen dat dit algoritme altijd de verzameling van Pareto-optimale paden vindt.

Terwijl het voorgaande onderzoek toegepast kan worden op zowel unimodaal als multimodaal transport met quasi alle mogelijke objectieven, focust het laatste deel van het doctoraatsonderzoek zich op het optimaliseren van de reistijd in multimodale transportnetwerken. Afhankelijk van de transportmodus kunnen deze reistijden tijdsafhankelijk en/of stochastisch zijn. Terwijl deze in wandelnetwerken meestal statisch en deterministisch zijn, zijn ze in een autonetwerk zowel tijdsafhankelijk als stochastisch. De reistijden in een spoornetwerk worden in het algemeen gemodelleerd als tijdsafhankelijke en deterministische kosten, terwijl die in een metronetwerk gezien worden als statisch en stochastisch. Om het kortste pad in een multimodaal netwerk te bepalen, moet een nieuw algoritme ontwikkeld worden dat in staat is om deze verschillende kosten te combineren. In deze dissertatie wordt een casestudy voorgesteld van een multimodaal routeringssysteem. Bovendien wordt er een algoritme voorgesteld dat omgaat met de verschillende kostentypes. Er wordt experimenteel aangetoond dat gebruik maken van tijdsafhankelijke en stochastische informatie inderdaad resulteert in snellere en meer betrouwbare routes. Daarenboven tonen we aan dat multimodaal reizen voordelen heeft ten opzichte van reizen met de wagen.

Naast routeringsalgoritmen voor multimodaal transport, werden tijdens dit doctoraatsonderzoek nog een aantal algoritmen en datastructuren ontwikkeld in andere toepassingsdomeinen. Deze onderwerpen worden voorgesteld in de bijgevoegde hoofdstukken. Verder onderzoek werd uitgevoerd op het ant-colony-optimalisatiegebaseerde algoritme dat ontwikkeld werd voor het routeren van opdrachten in optische gridnetwerken. Daarnaast werd onderzoek uitgevoerd in het gebied van de bioinformatica. Er wordt een subgraph matching algoritme voorgesteld dat motieven (d.z. patronen die vaker voorkomen dat verwacht) vindt in grote netwerken. Dit algoritme maakt gebruik van specifiek ontwikkelde datastructuren die de berekeningen versnellen.

Summary

Before GPS navigation systems and online routing applications became widely available, the only way to determine the best route between two locations was to seek for it manually on paper maps, which could be time-consuming. In order for the digital routing systems to determine the best route in a very short time, novel fast shortest path routing algorithms need to be designed for transportation networks. These algorithms should calculate the shortest route, the fastest route, the most reliable route, or route that is at the same time short, fast and reliable. This means that different types of costs should be modeled in the transportation network. Moreover, as some of the transport modes (e.g. the road) have become congested, a shift was observed towards multimodal transportation. This is defined as the form of transportation in which multiple transport modes are used during a single trip.

In this dissertation, innovative research is presented on shortest path routing in multimodal networks. Work was carried out on several fronts. Firstly, a flexible multimodal network model is developed. Subsequently, we looked at routing algorithms. It is investigated how some of the known algorithms can be accelerated. Moreover, alternative cost structures are studied and algorithms are developed that deal with these costs.

In the multimodal network model that is presented in chapter 2, each mode-specific network is represented by a layer and these layers are interconnected by trans-shipment links. The advantage of this approach is that adding or removing a transport mode is not a complex operation. We opted to only model the location specific information in the network graph and incorporate all cost information in cost objects that are assigned to the links. This means that, to model time-dependent information, the time-dependent network model was used. Different cost models are proposed. Single value costs are static costs for a single objective, like for example the distance. Some transportation costs are time-dependent, such as variable travel times on highways. In this research time-dependent costs are modeled as (piecewise linear) functions that are represented by a finite number of values. In transportation networks, objectives may have an inherent amount of uncertainty. For example, travel times may vary due to random external factors (e.g. other drivers behavior, weather conditions). Stochastic costs are represented by stochastic distributions which are in this research abstracted to an array of percentile values. Instead of optimizing the shortest path for one single objective, sometimes multiple objectives need to be optimized simultaneously. In this case, cost objects consist of multiple singular costs.

Next to a multimodal network model in which multiple types of costs can be modeled, the focus of this research was on the design of shortest path algorithms, which are either performant (w.r.t. the calculation times) or optimized for specific cost types. We will first focus on the performance of the shortest path algorithms, assuming that all costs are single values. Two novel goal-directed heuristics are presented: the predecessor and the accounting algorithm. The predecessor algorithm only investigates those nodes of which the estimated distance to the destination is less or equal than the estimated distance from its predecessor (i.e. the previous node on the path) to the destination. This way, only steps in the direction of the destination are allowed. As in transportation networks, shortest paths usually make some (small) detours in order to get to the destination, a tolerance factor was introduced that enables more paths to be found. An optimization is presented in which the areas around the origin and the destination are investigated completely. Furthermore, instead of comparing a node to its predecessor, it may be compared to its k-th predecessor. This also causes more nodes to be investigated, increasing the probability of finding the shortest path. In the accounting algorithm, for each path a history is stored that keeps track of the number of steps in the right and/or wrong direction. More than one step in the wrong direction is allowed, but still this number is limited. Next, two optimizations are presented: queue optimization and the feedback loop. Queue optimization speeds up the calculations by comparing nodes with the previously investigated nodes (instead of the previous nodes on the path). The feedback loop procedure is guaranteed to always find a path between the origin and the destination by iteratively increasing the tolerance factor.

Instead of one single objective, in transportation networks often multiple objectives need to be optimized simultaneously. For this, a multiple objective shortest path algorithm was developed by Martins. This algorithm results in a set of Pareto optimal paths, which are paths that cannot be optimized in one objective without deteriorating one or more of the other objectives. In this dissertation, we aimed at accelerating these multiple objective shortest path calculations. Two speedup measures are presented. As the algorithm of Martins calculates the multiple objective shortest paths from a single node to all other nodes of the network and we are often only interested in the set of Pareto optimal paths between a single origin and destination, the calculations should be aborted as soon as this set has been found. In this research, a novel stop condition is presented, which states that the search can be aborted as soon as the vector with the minimum values in the temporary set is dominated by one the destination labels. It is proven theoretically that, once this stop condition holds, the set of Pareto optimal paths has been found. Next, a bidirectional version of the multiple objective shortest path algorithm is presented. Again, the focus is on the definition of a strict stop condition. This algorithm searches the network from the origin and the destination simultaneously and every time the two searches meet, a candidate shortest path is found. The search processes can be aborted as soon as the vector with the sum of the minimum values of the temporary sets is dominated by one of the resulting costs. It is proven theoretically that this algorithm always finds the Pareto optimal set of solutions.

While the previous research may be applied to both multimodal and unimodal

networks with almost all possible objectives, the last part of this research focuses on optimizing the travel time in multimodal transportation networks. Depending on the mode of transportation, these travel time costs may be time-dependent and/or stochastic. While in a walking network they usually are static and deterministic, in a car network they are both time-dependent and stochastic. The travel times in a railroad network, on the other hand, are modeled as time-dependent and deterministic, while those in a subway network may be considered static and stochastic. To determine the shortest paths in these multimodal networks, a novel algorithm needs to be designed that is able to combine these different types of costs. In this dissertation, a case study is presented of a multimodal routing system. Moreover, an algorithm is proposed that copes with these different types of costs. It will be demonstrated experimentally that making use of the time-dependent and the stochastic travel time information indeed results in faster and more reliable routes. Moreover, it will be shown that traveling multimodally has its advantages over traveling by car only.

Next to the routing algorithms in multimodal transportation networks, during this PhD, a number of algorithms and data structures were designed for other application domains. These topics are covered in the appended chapters. Further research was performed on the ant colony optimization based algorithm designed for the routing of jobs in optical grid networks. Moreover, research was carried out in the field of bioinformatics. A subgraph matching algorithm is presented that finds motifs (i.e. patterns that occur more often than expected by chance) in large networks. This algorithm makes use of a number of custom designed data structures that speed up the calculations.

1 Introduction

"Logic will get you from A to B.
Imagination will take you everywhere."

–Albert Einstein (1879 - 1955)

This chapter situates the conducted research work, introduces the most widely known shortest path algorithms, summarizes the main contributions and outlines the structure of this dissertation. It also provides an overview of the publications that were authored during this research period.

1.1 Context

With the advent of high performing computing systems, routing manually with paper maps becomes a thing of the past. Nowadays, all map information can be stored digitally. Moreover, routing algorithms have been developed to find the shortest paths in these maps [1]. This enables us to find the best route from one location to another almost immediately on our PC, laptop, GPS, phone, etc. Consequentially, online routing engines, such as Google Maps, Mappy, ViaMichelin and Scotty, have become a very important player in the field of online applications.

During the past years the logistic world has observed a clear trend towards multimodal transportation. This is defined as the form of transportation that makes use of at least two transport modes (e.g. car, bike, train, ship, etc.). Figure 1.1 shows examples of two multimodal routes, one for personal transport and one for freight

transport. The key idea behind multimodal transportation is that by combining multiple transport modes, each with their own advantages and disadvantages, the disadvantages of certain modes can be avoided while benefiting from the advantages of other modes. One of the reasons of this shift to multimodal transportation is that some modes, such as the road transport, have become saturated, while there is still free capacity in other modes. For example, a commuter between Ghent and Brussels may prefer biking to the train station and taking the train over taking his car and getting stuck in traffic almost every day.

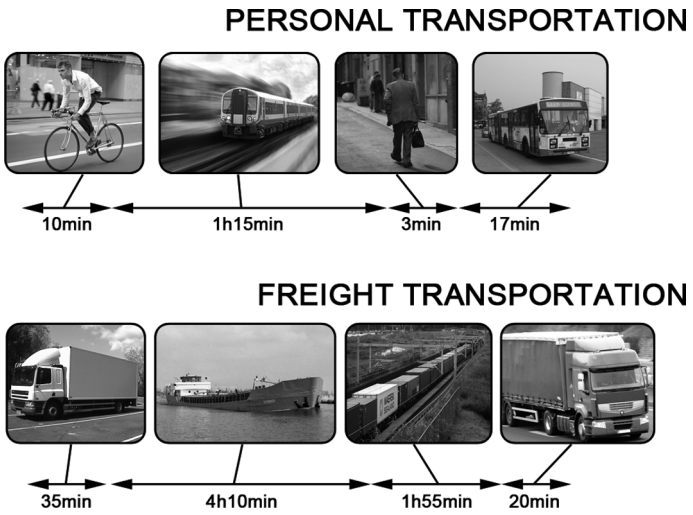

Figure 1.1: Examples of multimodal routes (i.e. routes that use multiple modes of transportation).

Multimodal transportation stems from the field of personal (public) transport. When traveling with public transport modes, switching between different means of transportation is nearly inevitable. People that are taking the train, often need a bus, tram or subway to get to their final destination. Computing systems encourage the use of public transportation and are able to find 'next generation' multimodal routes. For example, the best route may be the one taking the car to a station and continuing the route by train, as the roads to the destination may be blocked. Each mode of personal transportation has its own advantages and disadvantages. While driving a car, biking and walking gives access to almost all possible locations, public transport modes (trains, trams, buses, etc.) are restricted to their stations or stops. However, cars and bikes also have their limitations. Cars are not always the cheapest option (considering the gas prices) and bikes can only travel short distances. Personal multimodal route planners, such as Scotty, calculate a route that best matches the user's preferences, taking into account the characteristics of

network	$\|V\|$	$\|E\|$
Belgium (OpenStreetMap)	349 810	468 888
Benelux (iMinds ICON MobiRoute project)	425 480	519 915
Europe (OpenStreetMap)	17 681 668	22 927 991
USA (9th DIMACS implementation challenge)	23 947 347	58 333 344

Table 1.1: Number of nodes and links for several road networks. V represents the set of nodes (vertices) and E represents the set of links (edges). All information concerning shapes was omitted here.

each of the transport modes.

In the '80s, the term multimodal freight transport was coined for the first time, but only in recent years it is being more commonly employed. Possible freight transportation modes include trucks, trains, ships, planes, etc. Where in the early days the infrastructure for all transport modes was owned by the company that deployed multimodal transportation, nowadays this infrastructure may belong to different companies. Multimodal transportation differs from personal transport in the sense that it is not as flexible in the trans-shipment points. While persons may change from one transport mode to another independently, goods need additional infrastructure for their trans-shipments. Again, each mode of transportation has its own advantages and disadvantages. Trucks can reach almost all locations, while trains and ships are bound to stations and harbors. With respect to the energy cost, the European Commission [2] stated *'In terms of energy efficiency and the weight of goods that can be moved one kilometer by one liter of fuel, the figure for road haulage is 50 tonnes, for rail haulage 97 tonnes and for inland waterways 127 tonnes' (p. 41).* Routing multimodally comes down to finding a good balance between the characteristics of each of the transport modes, benefiting from the advantages, while trying to eliminate the disadvantages.

1.2 Challenges

With the advances in technology, more and more transportation network data becomes available. This results in larger networks. Table 1.1 shows the size of a number of road networks. As the calculation time of most shortest path algorithms is proportional to the size of the network (for example, the complexity of the algorithm of Dijkstra is $O(|E| + |V|log|V|)$, with V the set of nodes and E the set of links) and transportation networks are relatively large, the design of efficient data structures and algorithms is of key importance. This is one of the main concerns of the research presented here.

Multimodal transportation networks contain all network information of each of

the transport modes. As each mode specific network may be extensive, multimodal networks are even larger. Furthermore, multiple modes of transportation, each with their own characteristics, need to be combined. This requires a uniform network structure that allows each transport mode to incorporate its own characteristics. Moreover, trans-shipments between the different modes need to be modeled. In conclusion, the challenge is to develop an efficient network data structure that incorporates all mode specific characteristics in a uniform way, together with the trans-shipment information. This is tackled in chapter 2.

Once the network model has been developed, network costs need to be defined to realize the objectives (for example, minimize the distance of the route between the origin and the destination). Aside from fixed costs, a number of costs, such as the travel time, may be time-dependent. Some current routing systems already use this information for routing, but most of them only take into account the current situation to calculate routes in the (near) future. This may result in non-optimal routes. Suppose for example someone leaving in his car during rush hour. The routing system will advise him to avoid the congested highways on his route. Nevertheless, it is possible that some of the traffic jams are already dissolved by the time the driver would arrive there. In this case, avoiding the highway may lead to a longer route. So, instead of only taking into account the current situation, routing engines should use predictions to calculate an optimal route in the (near) future. These problems are dealt with in chapter 2 and chapter 5.

Furthermore, most current routing engines assume all costs to be deterministic in nature. This means that taking the same route at the same time always leads to the same arrival time. In real life this is not the case. For example, a route engine may have calculated a route that would get you to the airport in time in order to catch a flight. However, due to external factors, like reckless driving or bad weather conditions, you still might miss your flight. Therefore, a more reliable route should be calculated. To model this, the costs should be assumed stochastic. This gives rise to a number of other questions, such as how to represent this stochastic information and how to combine the stochastic distributions. This is tackled in chapter 5.

The most important challenge for routing in multimodal transportation networks is the optimization of the shortest path algorithms. The user does not want to wait too long for his route to be calculated. Moreover, online routing engines have a large number of users and only a limited amount of resources. They require the routes to be calculated as fast as possible. In chapters 3, 4 and 5, a number of fast shortest path algorithms are presented.

1.3 Point-to-Point Shortest Path Algorithms

A large part of this dissertation deals with the optimization of shortest path routing algorithms for transportation networks. In this section we would like to give an overview of the most common point-to-point shortest path algorithms and speedup techniques. After giving a short description of the algorithm of Dijkstra, the fundamental algorithm for shortest path calculations, a number of speedup techniques and their corresponding algorithms are presented. It should be noted that some of these techniques are heuristic and thus are not guaranteed to produce the optimal solution.

1.3.1 Shortest Path Algorithm of Dijkstra

Route planning in transportation networks means finding the shortest path between an origin and a destination. The most well-known algorithm that realizes this is the algorithm of Dijkstra [3]. This label setting algorithm finds the shortest paths from a single origin to all other nodes of the graph (i.e. network) and thus produces a shortest path tree. It is also used to find the shortest path between a single origin and a single destination, as the search process can be aborted once this single shortest path has been found. It should be noted that the algorithm of Dijkstra requires all link costs to be non-negative, which is true for almost all transportation costs, such as distance, travel time, etc.

In this algorithm each node has assigned to it a label that indicates the best distance found so far between the origin node and the node in question. At the start of the algorithm all labels are set to infinity, except the one of the origin node which is set to 0. Moreover, two sets of nodes are kept: a temporary set and a permanent set. At the start of the algorithm the temporary set only contains the origin node, while the permanent set is empty.

In each step of the algorithm the temporary node with the smallest label is made permanent, i.e. removed from the temporary set and added to the permanent set. Subsequently, the labels of the neighbors of this node are updated. This means calculating for each neighbor a new label, which is the sum of the permanent label and the link cost. If this new label is smaller than the current label of the neighbor, the neighbor's label is changed and the neighbor is added to the temporary set. The algorithm terminates when the temporary set is empty or when the destination node has been made permanent. This means that not the complete network needs to be investigated, but only a part that is proportional to the distance between the origin and the destination.

Below, the pseudo code of the algorithm of Dijkstra is presented which finds the shortest path between the origin `o` and the destination `d`. The label of node `v` is represented by `l(v)` and `T` and `P` are the temporary and the permanent set respectively. The cost of the link between node `u` and node `v` is denoted by `c(u,v)`.

```
1    forall(v ∈ V)
2        l(v) := ∞;
3    l(o) := 0;
4    P := EMPTY_SET;
5    T := {o};

6    while(!(T is empty) && !(d ∈ P)){
7        u := T.removeMin();
8        P.add(u);
9        forall(n :  neighbor(u)){
10           l_new := l(n) + c(u,n);
11           if(l_new < l(n){
12                l(n) = l_new;
13                T.add(n);
14           }
15       }
16   }
```

In the simplest implementation the execution time of the algorithm of Dijkstra (i.e. calculating the shortest path tree) is of the order $O(|E|+|V|^2) = O(|V|^2)$ [1], with E the set of links (edges) and V the set of nodes (vertices). By making use of a well-designed data structure to implement the priority queue, which is used to retrieve the node with the smallest label from the temporary set, this execution time can be reduced. It is known that the use of a Fibonacci heap [4, 5] improves the calculation time to $O(|E| + |V|log|V|)$.

1.3.2 Speeding up shortest path calculations

There are multiple ways to speed up the shortest path queries. Three major approaches can be distinguished [6]: limiting the search area, decomposing the search problem or limiting the number of search links. Limiting the search area means only investigating that part of the network that has a high chance of containing the shortest path, mostly the area between the two end points. Two algorithms that exploit this are the branch pruning algorithm and the A* algorithm. Decomposing the search problem consists of dividing the search into a number of smaller searches. One method, the bidirectional search, searches from the origin and the destination simultaneously until the two searches meet in the middle. Another method, the subgoal method, uses predefined intermediate points to split up the problem in a number of shorter path searches. When limiting the number of links searched, the network is extended with a number of bypass links in order to speed

up the search process. This is often denoted with the term hierarchical routing. We will now discuss a number of these speedup techniques, which have proven to be advantageous for routing in transportation networks.

1.3.2.1 Goal-Directed Search

Goal-directed algorithms limit the search to the area around the origin and the destination by ignoring the nodes which have a low probability of being part of the shortest path. This is illustrated in figure 1.2. The two most popular goal-directed algorithm are the branch pruning heuristic and the A* algorithm.

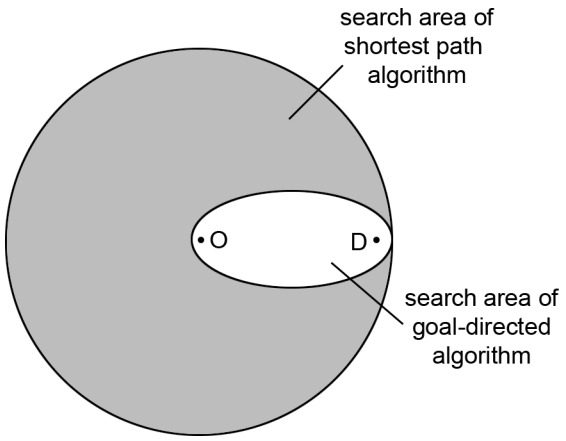

Figure 1.2: Basic principle of goal-directed algorithms.

The idea behind branch pruning [7, 8] is that all intermediate nodes that have a low possibility of being on the shortest path are pruned or eliminated. This requires a measure to indicate which nodes are favorable and which are not. In Fu's implementation [7] a node is only investigated if the sum of its label and an estimate of the distance between the node and the destination is smaller than an estimated upper bound of the distance between the origin and the destination. In transportation networks, often the Euclidian distance (i.e. the beeline distance), with or without an additional correction factor, is used to estimate the unknown distances. It is shown that, by making use of the appropriate estimates, branch pruning can save 40% to 60% on the calculation time [7, 9].

Branch pruning is a heuristic technique, as it is possible that nodes that are on the shortest path are pruned, based on the estimates. The A* algorithm [10–12] overcomes this problem by not eliminating the nodes that have a low probability of being on the shortest path. Instead they are given a lower priority to be investigated. This is accomplished by, instead of using the labels to determine the node to

be investigated next, using the sum of the label and an estimate of the distance between that node and the destination. It has been proven [13] that the A* algorithm can find the shortest path in $O(|V|)$, with V the set of nodes.

For a more detailed description of goal-directed algorithms we would like to refer to chapter 3 in which, next to branch pruning heuristic, two novel goal-directed algorithms are presented.

1.3.2.2 Bidirectional Search

Most point-to-point shortest path algorithms search the network starting from the origin node towards the destination node, i.e. unidirectionally. The bidirectional search algorithm [14, 15] speeds up the calculations by splitting up the search process into two separate search processes, that may be executed simultaneously. One search starts from the origin and the other from the destination. This is illustrated conceptually in figure 1.3. Two factors influence the efficiency of bidirectional search algorithms: the alternation between the two search processes and the criterion that decides when the algorithm can terminate.

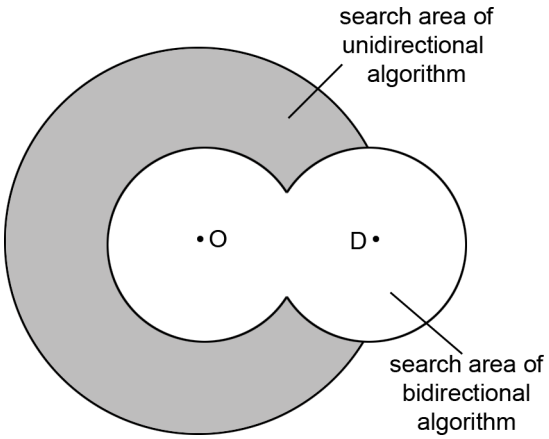

Figure 1.3: Basic principle of bidirectional algorithms.

Alternating equally between the two searches is the most straightforward way to divide the computation time between the two search processes, but has proven not to be the most efficient method for a number of network configurations. A more efficient method is to pick the search process that has the fewest nodes left in its temporary set.

Determining the stopping criterion is the most crucial factor of the two, as a badly defined criterion may lead to two complete network searches which can be twice as slow as the unidirectional counterpart. Nicholson [15] decided that

the search processes may be aborted as soon as the sum of the forward and the backward label in one network node is less or equal to the sum of the minimum labels of each of the temporary sets.

The strength of the bidirectional search method strongly depends upon the network configuration. Examples [16, 17] exist of networks in which searching bidirectionally is even slower than searching in only one direction.

Bidirectional methods are often used in hierarchical search procedures, as will be demonstrated in the following subsection.

1.3.2.3 Hierarchical Search

The basic idea behind searching hierarchically in a transportation network is that when drivers determine their route manually they will first determine which major roads or highways they will take and subsequently find access routes to these major roads [18]. Two approaches can be distinguished for hierarchical search. One is to divide all network links in a number of (hierarchical) categories [19–21] and make use of this category information when routing. The shortest path algorithms [22, 23] then start their search in the lowest category until they encounter an entry point to a higher category. Subsequently, the search is continued in this higher category until a new entry point is found, and so on. This is illustrated in figure 1.4.

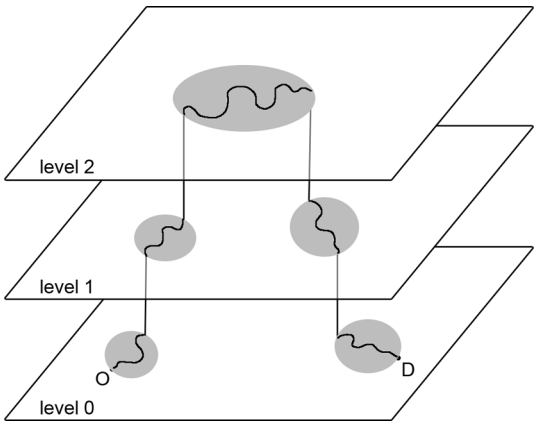

Figure 1.4: Searching the shortest path in a hierarchical network.

In the other approach, shortcut links are added to the network [24, 25]. The most well-known efficient hierarchical method for road networks is a contraction hierarchy [25]. Finding the shortest path in a contraction hierarchy comes down to searching bidirectionally while only visiting the nodes that are higher in the

hierarchy. It has been proven that this shortest path algorithm is guaranteed to produce the optimal solution.

Hierarchical search has proven to be very efficient, especially in transportation networks.

1.4 Outline & Research Contributions

The research presented in this dissertation is concerned with how to find the 'shortest' routes in multimodal transportation networks efficiently.

First, a network model is determined that comprises all characteristics of these mode specific networks and that is able to model the trans-shipments between the modes. We opted for a layered network model in which each layer represents one mode of transportation. Next, a generic cost model was developed to meet all possible objectives (distance, travel time, financial cost, etc.). Some of these costs may be time-dependent or stochastic and this may differ for the different transport modes, depending on the available information. The multimodal network model, together with the generic cost model, is presented in chapter 2.

The following three chapters focus on the optimization of point-to-point shortest path algorithms in transportation networks. In chapter 3 we investigate a number of goal-directed algorithms. Two novel algorithms are presented, viz the predecessor algorithm and the accounting algorithm. In this chapter it is assumed that the link costs are single values.

Chapter 4 deals with optimizing multiple objectives simultaneously. The links of the network now have multiple costs assigned to them. We present a bidirectional speedup of a well-known multiple objective shortest path algorithm [26]. This algorithm finds a set of Pareto-optimal paths with respect to the different objectives.

In chapter 5 we will look at a realistic multimodal transportation network in which some of the costs are time-dependent and/or stochastic, while others are not. A shortest path algorithm is presented that is able to cope with these different cost models.

Chapter 6 finishes this dissertation with a number of conclusions and an outlook on possible future research opportunities.

The two appendices present publications that present designs of efficient network algorithms and data structures for other application domains. In the first appendix, an ant colony optimization algorithm is presented that was optimized for the routing of jobs in grid networks. This algorithm is inspired by the natural phenomenon of ants foraging for food.

The second appendix is situated in the field of bioinformatics. Here, we present a subgraph matching algorithm that finds all motifs (i.e. patterns that occur more

often than expected by chance) in a network. The design of efficient data structures was of key importance for the performance of the algorithm.

1.5 Publications

The research results obtained during this PhD research have been published in scientific journals and presented at a series of international conferences. The following list provides an overview of the publications during my PhD research.

1.5.1 Publications in international journals (listed in the Science Citation Index [1])

1. **Sofie Demeyer**, Marc De Leenheer, Jurgen Baert, Mario Pickavet, and Piet Demeester. *Ant colony optimization for the routing of jobs in optical networks.* Published in the Journal of Optical Networking (JON), 7:160 - 172, 2008.

2. **Sofie Demeyer**, Jan Goedgebeur, Pieter Audenaert, Mario Pickavet, and Piet Demeester. *Speeding up Martins' algorithm for multiple objective shortest path problems*. Published online in A Quarterly Journal of Operations Research (4OR), Feb. 2013.

3. **Sofie Demeyer**, Pieter Audenaert, Mario Pickavet, and Piet Demeester. *Dynamic and Stochastic Routing for Multimodal Road-Rail Transportation Systems*. Accepted for publication in IET Intelligent Transport Systems (ITS), Jan. 2013.

4. **Sofie Demeyer**, Tom Michoel, Jan Fostier, Pieter Audenaert, Mario Pickavet, and Piet Demeester. *The Index-based Subgraph Matching Algorithm (ISMA): fast subgraph enumeration in large networks using optimized search trees.*. Accepted for publication in PLoS One, Mar. 2013.

5. Maarten Houbraken, **Sofie Demeyer**, Dimitri Staessens, Pieter Audenaert, Didier Colle, and Mario Pickavet. *Fault tolerant network design inspired by physarum polycephalum*. Published online in Natural Computing, Aug. 2012.

[1] The publications listed are recognized as 'A1 publications', according to the following definition used by Ghent University: A1 publications are articles listed in the Science Citation Index, the Social Science Citation Index or the Arts and Humanities Citation Index of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper.

6. **Sofie Demeyer**,Jan Goedgebeur, Pieter Audenaert, Didier Colle, Mario Pickavet and Piet Demeester. *The Predecessor and the Accounting Algorithm: Experimental Study of Two Novel Goal-Directed Algorithms*. Submitted to Algorithmica, Apr. 2013.

1.5.2 Publications in international conferences

1. **Sofie Demeyer**, Pieter Audenaert, Bart Slock, Mario Pickavet, and Piet Demeester. *Multimodal transport planning in a dynamic environment.* Published in Proceedings of Conference on Intelligent Public Transport Systems (IPTS), pages 155–167, Amsterdam, the Netherlands, Apr. 2008.

2. Mario Pickavet, Willem Vereecken, **Sofie Demeyer**, Pieter Audenaert, Didier Colle, Piet Demeester, and Bart Dhoedt. *Energy footprint of ICT: future outlook and challenges.* Published in Proceedings of International ICT Symposium, Brussels, Belgium, Oct. 2008.

3. Mario Pickavet, Ruth Van Caenegem, **Sofie Demeyer**, Pieter Audenaert, Brecht Vermeulen, Didier Colle, Bart Dhoedt, and Piet Demeester. *Energy efficiency of ICT.* Published in KEIO and Gent University G-COE Joint Workshop for future network, pages 26, Ghent, Belgium, Mar. 2008.

4. Mario Pickavet, Ruth Van Caenegem, **Sofie Demeyer**, Pieter Audenaert, Didier Colle, Piet Demeester, H. Foisel, M. Jaeger, R. Leppla, and A. Gladisch. *Energy footprint of ICT.* Published in Proceedings of Broadband Europe 2007, Antwerp, Belgium, Dec. 2007.

5. Mario Pickavet, Willem Vereecken, **Sofie Demeyer**, Pieter Audenaert, Brecht Vermeulen, Chris Develder, Didier Colle, Bart Dhoedt, and Piet Demeester. *Worldwide energy needs for ICT: the rise of power aware networking.* Published in Proceedings of 2nd International Symposium on Advanced Networks and Telecommunication, Bombay, India, Dec. 2008.

6. Mario Pickavet, Willem Vereecken, **Sofie Demeyer**, Pieter Audenaert, Didier Colle, Chris Develder, and Piet Demeester. *Contribution and role of network architectures in the footprint reduction of ICT.* Published in Proceedings of European Conference on Networks an Optical Communications, held in conjunction with the 4th Conference on Optical Cabling and Infrastructure, Valladolid, Spain, Jun. 2009.

7. **Sofie Demeyer**, Jan Goedgebeur, Pieter Audenaert, Mario Pickavet, and Piet Demeester. *The predecessor and the accounting algorithm speed up shortest path calculations in traffic routing applications.* Published in Proceedings of 13th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC), Funchal, Madeira Island, Sept. 2010.

8. **Sofie Demeyer**, Pieter Audenaert, and Mario Pickavet. *Time-dependent stochastic routing: A practical implementation.* Published in Proceedings of 21th International Symposium on Mathematical Programming (ISMP), Berlin, Germany, Aug. 2012.

9. **Sofie Demeyer**, Pieter Audenaert, Steven Logghe, Mario Pickavet, and Piet Demeester. *Practical Time-Dependent and Stochastic Routing with Historical Measurements of Travel Times.* Published in Proceedings of 15th International IEEE Annual Conference on Intelligent Transportation Systems (ITSC), Anchorage, Alaska USA, Sept. 2012.

1.5.3 Publications in national conferences

1. **Sofie Demeyer**, Pieter Audenaert, and Mario Pickavet. *Innovative ICT solutions for the routing and planning of multimodal transportation.* Published in UGent-FirW Phd Symposium, pages 162–163, Ghent, Belgium, Dec. 2008.

2. **Sofie Demeyer**, Pieter Audenaert, and Mario Pickavet. *Multple Objective Shortest Path Algoirthms for Transportation Problems.* Published in UGent-FirW Phd Symposium, pages 72–73, Ghent, Belgium, Dec. 2009.

3. **Sofie Demeyer**, Pieter Audenaert, and Mario Pickavet. *Multimodal Dynamic and Stochastic Routing.* Published in UGent-FirW Phd Symposium, pages 187, Ghent, Belgium, Dec. 2010.

4. **Sofie Demeyer**, Pieter Audenaert, Mario Pickavet, and Piet Demeester. *Practical Heuristic Algorithm for Routing Backwards in dynamic Transportation Networks.* Published in ORBEL 2011, pages 124–125, Ghent, Belgium, Feb. 2011.

5. **Sofie Demeyer**, Pieter Audenaert, and Mario Pickavet. *On Determining the Shortest Path through a Number of Intermediate Points.* Published in ORBEL 2012, pages 112–113, Brussels, Belgium, Feb. 2012.

6. **Sofie Demeyer**, Maarten Houbraken, Thijs Walcarius, Pieter Audenaert, Didier Colle, Mario Pickavet, and Piet Demeester. *Using Contraction Hierarchies to Find Dissimilar Paths in Transportation Networks.* Published online in ORBEL 2013, Kortrijk, Belgium, Feb. 2012.

7. Maarten Houbraken, Thijs Walcarius, **Sofie Demeyer**, Pieter Audenaert, Didier Colle, Mario Pickavet, and Piet Demeester. *An insertion-based heuristic for the constrainedpickup and delivery problem.* Published online in ORBEL 2013, Kortrijk, Belgium, Feb. 2012.

References

[1] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[2] E. Commission. *European transport policy for 2010: time to decide*. In White Paper. 2001.

[3] E. W. Dijkstra. *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1):269–271, 1959.

[4] M. L. Fredman and R. E. Tarjan. *Fibonacci heaps and their uses in improved network optimization algorithms*. Journal of the ACM, 34(3):596–615, 1987.

[5] G. S. Brodal, G. Lagogiannis, and R. E. Tarjan. *Strict fibonacci heaps*. In Proceedings of the 44th symposium on Theory of Computing, STOC '12, pages 1177–1184, New York, NY, USA, 2012. ACM.

[6] L. Fu, D. Sun, and L. Rilett. *Heuristic shortest path algorithms for transportation applications: State of the art*. Computers & Operations Research, 33(11):3324 – 3343, 2006.

[7] L. Fu. *Real-time Vehicle Routing and Scheduling in Dynamic and Stochastic Traffic Networks*. University of Alberta, Dept. of Civil and Environmental Engineering, 1996.

[8] H. A. Karimi. *Real-Time Optimal Route Computation: a Heuristic Approach*. Intelligent Transportation Systems (ITS) Journal, 3(2):111–127, 1996.

[9] J. Lysgaard. *A two-phase shortest path algorithm for networks with node coordinates*. European Journal of Operational Research, 87(2):368 – 374, 1995.

[10] P. Hart, N. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics, 4(2):100 –107, 1968.

[11] J. Nilsson. *Problem-solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[12] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Publishing Company, 1984.

[13] R. Sedgewick and J. Vitter. *Shortest paths in euclidean graphs*. Algorithmica, 1:31–48, 1986.

[14] G. Dantzig. *On the Shortest Path Through a Network*. Management Science, 6:188–190, 1960.

[15] T. A. J. Nicholson. *Finding the Shortest Route between Two Points in a Network*. The Computer Journal, 9(3):275–280, 1966.

[16] I. Pohl. *Bi-directional Search*. Machine Intelligence, 6:127–140, 1971.

[17] S. E. Dreyfus. *An Appraisal of Some Shortest-Path Algorithms*. Operations Research, 17(3):395–412, 1969.

[18] P. Bovy and E. Stern. *Route Choice: Wayfinding in Transport Networks*. Kluwer Academic Publishers, Dordrecht, 1990.

[19] S. Timpf, G. Volta, D. Pollock, and M. Egenhofer. *A conceptual model of wayfinding using multiple levels of abstraction*. In A. Frank, I. Campari, and U. Formentini, editors, Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, volume 639 of *Lecture Notes in Computer Science*, pages 348–367. Springer Berlin / Heidelberg, 1992.

[20] Y.-L. Chou, H. E. Romeijn, and R. L. Smith. *Approximating Shortest Paths in Large-Scale Networks with an Application to Intelligent Transportation Systems*. INFORMS Journal on Computing, 10(2):163–179, 1998.

[21] G. Jagadeesh, T. Srikanthan, and K. Quek. *Heuristic techniques for accelerating hierarchical routing on road networks*. Intelligent Transportation Systems, IEEE Transactions on, 3(4):301 – 309, 2002.

[22] A. Car and A. Frank. *General Principals of Hierarchical Spatial Reasoning - The Case of Wayfinding*. In Proceedings of the Sixth International Symposium on Spatial Data Handling, Edinburgh, pages 646–664. 1994.

[23] J. Shapiro, J. Waxman, and D. Nir. *Level graphs and approximate shortest path algorithms*. Networks, 22(7):691–717, 1992.

[24] S. Demeyer, P. Audenaert, B. Slock, M. Pickavet, and P. Demeester. *Multimodal transport planning in a dynamic environment*. In Proceedings of Conference on Intelligent Public Transport Systems (IPTS), Amsterdam, pages 155–167. 2008.

[25] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*. In C. McGeoch, editor, Experimental Algorithms, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin / Heidelberg, 2008.

[26] E. Q. V. Martins. *On a multicriteria shortest path problem*. European Journal of Operational Research, 16(2):236 – 245, 1984.

2

Modeling Multimodal Transportation Networks

This chapter deals with the modeling of multimodal transportation networks, with the focus on two aspects of this modeling. Firstly, a uniform data structure needs to be developed that captures the characteristics of each of the transportation modes, together with the characteristics of the trans-shipments between the different modes. Secondly, costs need to be assigned in this network. Transportation costs include distance, travel time, financial cost, etc. Some of these costs may be time-dependent and/or stochastic, introducing new challenges. It should be noted that these two aspects, network and cost modeling, cannot be treated independently as the cost model may influence the network model and vice versa.

2.1 Introduction

Network modeling or developing an efficient data structure to store all network information is of key importance in transportation applications. We opted to model the transportation network as a graph, a data structure developed to model relationships between objects or nodes. Graph theory finds its foundations in the problem 'Seven Bridges of Königsberg' addressed by Euler in 1736 [1]. Graph data structures (usually represented by $G = (V, E)$) consist of a set of nodes or vertices (V) that are connected by a set of links or edges (E). The way transportation networks are modeled as graphs may differ according to the mode of transportation and the

costs that need to be taken into account.

Road networks, which form the basis for cars, bikes and pedestrians, are typically modeled as follows. Each intersection is represented by a single node and two nodes are connected by a link if and only if there exists a feasible road between the two corresponding intersections. According to the mode's specific characteristics, links may be directed or undirected. When considering for example cars as the mode of transportation, the links should be directed in order to allow the modeling of one-way roads. In a walking network, on the other hand, the links are in most cases undirected, as walking is possible in both directions.

Public transport networks are modeled differently. Here, a node typically represents a stop (i.e. a bus stop, a train station, an airport, etc.) and links represent the direct connections between these stops. Two major approaches can be distinguished with respect to the modeling of time-table information. In the first approach the network is strictly location based and all time-table information is modeled in the cost objects that are assigned to the links. In the second approach the time-table information determines the network structure. Nodes represent both locations and points of time and links are added for each trip (i.e. getting from one stop to another at a certain point of time) separately.

For multimodal transport networks additional challenges arise. First of all, a network model needs to be developed that incorporates all characteristics of each of the different transport modes more or less uniformly. Next, trans-shipments between the different modes of transportations need to be modeled. One approach is to model it as attributes of the (multimodal) nodes where trans-shipment is possible. Another approach is to add trans-shipment links between geographically co-located (unimodal) nodes of different transport modes.

This chapter presents the network (and cost) model that is used in this dissertation. Next to modeling the multimodal transportation network, the focus lies on the modeling of the different costs that are encountered in transportation networks. In the next section an elaborate overview is given of what can be found in literature on this topic, more specifically on multimodal network models, time-dependent cost models and stochastic cost models. The subsequent section then presents the multimodal network model that was used in our research. We opted for a layered model, as this conserves the mode-specific characteristics the best. Section 2.4 deals with the issue of modeling transportation costs. Different types of costs are identified and it is described how they are incorporated in our network model. The chapter finishes with a number of conclusions in section 2.5.

2.2 Literature Overview

In this section an overview is given of what can be found in literature on multimodal transportation models and their corresponding cost models. Depending

upon the objectives, different models have been proposed. In the first subsection multimodal models, for both personal and freight transportation are described elaborately. Subsequently we will focus on the cost modeling. As for time-independent deterministic costs, for both single and multiple objectives, this modeling is quite straightforward, we will focus on time-dependent and stochastic costs. Time-dependent costs may influence the network structure drastically, which will be shown in subsection 2.2.2. Next, it will be investigated how stochastic transportation costs have been modeled in literature. It should be noted that both the time-dependent and the stochastic cost models were mostly developed to deal with modeling of travel times.

2.2.1 Multimodal Transportation

There have been multiple multimodal network models proposed in literature. In general, two major approaches can be distinguished: the flat network model and the layered network model. This is illustrated in figure 2.1. While in the flat network model every geographic location is represented by a single node and the edges may have multiple transport modes assigned to them, in the layered network each transport mode has its own network and the different networks are interconnected by trans-shipment links between co-located nodes between which trans-shipment is possible. In the flat model, trans-shipments are modeled as attributes of the multimodal nodes. It should be noted that these two approaches are extreme cases and that some of the presented models are a combination of the two.

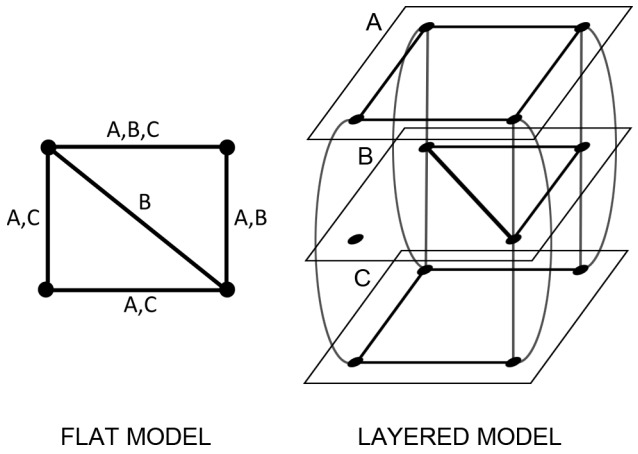

Figure 2.1: Generic approaches for multimodal transportation network models. This small example network has three modes of transportation: A,B and C. While in the flat model the trans-shipment costs are modeled in the multimodal nodes, in the layered network model these costs are assigned to the dedicated trans-shipment links.

Firstly, we will look at models that were developed especially for multimodal freight transportation. In these models transport means of the same mode (e.g. small and large ships) are handled separately and hence can be presumed as different modes of transportation. In [2] Jourquin and Beuthe present the NODUS software that was designed to build virtual networks for multimodal transportation. The concept of virtual networks was introduced earlier by Harker [3] and Crainic et al. [4]. They start from the flat network model and duplicate the nodes that are present in multiple transport modes. These duplicated nodes, called virtual nodes, are then connected with the corresponding mode-specific links. Moreover, virtual links are added to connect the co-located virtual nodes with each other and with their master node. In this way, two kinds of virtual links exists. While the links between the virtual nodes themselves represent actual trans-shipments, the virtual links between the master node and the virtual nodes model the loading and unloading of freight. Figure 2.2 shows the virtual network of the example network that was presented in figure 2.1. It can be seen that virtual networks more or less resembles the layered network model.

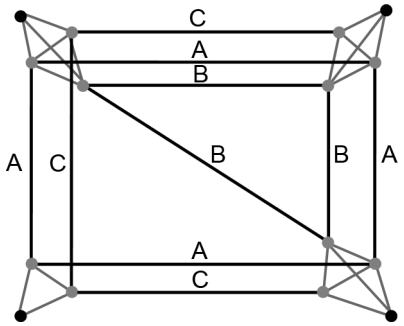

Figure 2.2: Example of a multimodal network modeled as a virtual network. The virtual nodes and links are shown in gray.

Southworth [5] presents a pure layered network model for multimodal freight transportation, in which each transport mode preserves its own network. Where trans-shipments are possible, terminals are added and both terminal access/egress links and within-terminal links are added. Dependent upon the characteristics of the trans-shipments, one geographic location may have one or more terminals that are used to interconnect the traffic from the different modes of transportation. This resembles the layered network model of figure 2.1 with additional terminal structures that are added between the layers. In this model all trans-shipments pass through terminals, which is realistic in freight transportation networks where goods often cannot be trans-shipped directly from one mode to another but have to pass through intermediate warehouses.

Next, we will look at the multimodal network models that were developed for personal transporation. Most of these models make use of the flat model approach, with some of them partly layered. Van Nes [6] proposes a network model in which each mode specific network has a hierarchical structure. In this way, he deals with the different scales in transportation networks, such as the urban scale (containing only the networks of single cities) and the national scale (with links that connect the different cities with each other). This hierarchical modeling is similar to the ones of Jung and Pramanik [7], Jing et al. [8] and Mainguenaud and Simatic [9]. In order to connect the different hierarchical networks, transfer nodes are identified, i.e. nodes where trans-shipment is possible. In his dissertation, Van Nes states that to model multimodal networks no significant restructuring of the transport networks is needed. The different unimodal networks, if properly modeled, can be reused and the only remaining concern is how to model the trans-shipments. As he makes use of transfer nodes, his model can be categorized as a flat model.

In [10] a similar model is applied, but now more attention is given to the transfer points, which are modeled as small networks themselves. This results in an overall network model that lies somewhere between the layered approach (in the transfer points, trans-shipment links are added) and the flat approach (on a higher level trans-shipments are still modeled in the transfer nodes).

In [11], Booth et al. present a personal transportation network model in which nodes represent unique geographic locations. Both the links and the nodes possess attributes that indicate their transport mode. Moreover, trans-shipments are modeled in the nodes themselves. This is a clear example of a flat network model.

Next to models that were designed specifically for either personal or freight transportation, a number of generic models were proposed that can be applied to both varieties of transportation. Bielli et al. [12], similar to Van Nes' approach, make use of the hierarchical structure of transportation networks to distinguish among the different scales in the network. The multimodal network is defined as a transit network which is divided in a number of disjunct subgraphs. The concept of a hypernetwork (containing hypernodes and hyperlinks) is introduced to model the different levels of abstraction in the hierarchy. This model can be categorized as layered in the lowest level of the hierarchy, while it is clearly flat in the highest level.

Pajor [13] builds a multimodal network out of a set of unimodal networks by applying two operations: merge and link. Merging comes down to building one large network out of the different mode-specific networks and labeling all nodes and links with their corresponding modes. In the linking operation the nodes of different transport modes are connected by means of trans-shipment links. This results in a layered network model.

In conclusion, two main approaches can be distinguished to model multimodal transportation networks: the flat and the layered network model. While

most freight transport network models lean towards the layered approach, personal transport network models have a tendency for the flat network model. Both approaches have been encountered in models that are independent of the nature of the transport. The key difference between the two approaches is the way how trans-shipments are modeled. So, determining the best multimodal network model comes down to determining how to model the trans-shipments.

From an algorithmic point of view both approaches have their advantages and disadvantages. The flat model usually results in smaller networks since links (e.g. roads) can be reused in different modes of transportation. On the other hand, calculating the cost of a path in a flat network is more complex than calculating it in its layered counterpart. While in the latter one links have assigned to them single cost objects, in the flat model multiple cost objects (of different transport modes) may be assigned to a single link. Moreover, node costs (i.e. the trans-shipment costs) need to be taken into account.

2.2.2 Time-Dependent Costs

Most research with respect to time-dependent costs originates from the modeling of time table information for public transportation. Nevertheless, some of the proposed models can also be used to model time-dependent information in private transportation.

Two major approaches can be distinguished: the time-dependent network model and the time-expanded network model [14]. Both approaches influence the network model considerably. In the time-dependent model [15–18] every node represents a single geographic location and all time-dependent information is modeled in the cost objects that are assigned to the links. In the time-expanded model [19–22] nodes represent both a location and a time stamp, which together form an event, and the links now model either waiting in one location or traveling from one location to another. To reduce the number of links in these time-expanded networks hypergraphs can be used [23]. In figure 2.3 both models are applied to a small example.

Choosing between the time-dependent and the time-expanded network model is a trade-off between the memory consumption and the performance of the routing algorithms. While in the time-dependent model most memory is consumed by the cost objects that are assigned to the links, in the time-expanded model the needed memory space is determined by the number of nodes and links in the network. In most cases, storing the travel time function in the cost object is less memory consuming than creating a node for every possible event. With respect to the performance, the routing algorithms applied in the time-dependent network are often slower than when applied in the time-expanded counterpart. The reason for this is that, despite of the fact that larger networks slow down the calculations, deter-

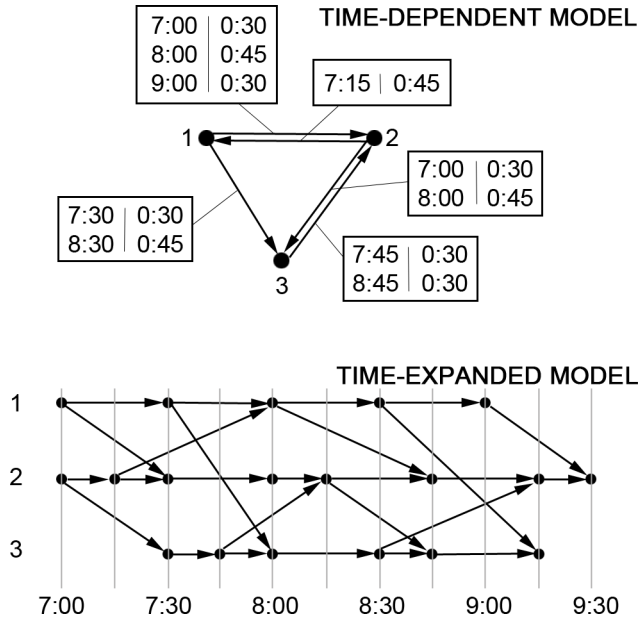

Figure 2.3: The time-dependent and the time-expanded model applied to a small example. In the time-dependent model a time table is assigned to every link with on the left the departure times and on the right the travel times. In the time-expanded model a node represents an event with both a location (here: 1, 2 or 3) and a time stamp. The travel time (or waiting) costs of the links can be deduced from the difference between the time stamps of the end points.

mining the exact travel time on a link is a relatively expensive operation, certainly when working with generic travel time functions. Suppose for example that the travel time function has q entries and the binary search algorithm is used to determine which entries are needed to calculate the exact travel time, then the lookup operation has a complexity of $O(log\ n)$. If, on the other hand, the data structure to represent time-dependent costs is optimized so that these lookup operations are possible in constant time, the time-dependent network model is the best option again, as then the network size is the determinant for the performance of the algorithm.

2.2.3 Stochastic Costs

Some costs, such as the travel time, may have an inherent amount of uncertainty. This has been recognized in the literature for a long time [24]. However, valuable practical research on how to model stochastic (travel time) costs and the operation of the corresponding algorithms only appeared in recent years.

A number of studies propose to fit stochastic travel time costs to one of the generic probability distributions. Both Wardop [24] and Herman and Lam [25] indicate that the travel times follow a skewed distribution. Moreover, the study of the latter one indicates that only the 60% lower values fit well to a (symmetrical) normal distribution, contradicting the assumption at that time that travel times are normally distributed. From Polus' empirical study [26] it is concluded that travel times best fit the gamma distribution. Dandy and McBean [27] suggest to use either a log-normal or a gamma distribution. Furthermore, there are many studies [28–32] which declare that the log-normal distribution is best suited to model stochastic travel times.

Most of the studies that were referred to here, and certainly the older ones, are only concerned with the type of probability distribution that best matches the stochastic travel times and do not consider how these distributions may be used in practical routing algorithms. Lomax et al. [33] assume the travel times to be normally distributed in order to simplify the routing algorithm. Furthermore, Kaparias [32] presents a routing algorithm that makes use of log-normal distributions.

The main disadvantage of using predefined distributions to model the stochastic information is that some of the important information, such as certain peaks, may be lost.

Nie and Fan [34] make use of generic stochastic distributions and present the SOTA (Stochastic On-Time Arrival) algorithm to route in stochastic transportation networks. Unfortunately, this algorithm only performs adequately in relatively small networks as calculations with continuous distributions require vast amounts of calculation time.

Samaranayake et al. [35] propose a novel algorithm to solve the stochastic on-time arrival problem in which they, similar to [34], assume that the stochastic distributions are continuous and may take any form. However, in the experiments only the first two moments (i.e. the mean and the variance) are used. Since the distributions are represented by only two values, valuable information might be lost.

It should be noted that modeling stochastic travel times with predefined distributions is only useful when the correlations between the different links are taken into account. However, calculating the correlations between all links and taking them into account when routing is remarkably time-consuming. Therefor, in practical stochastic routing systems assumptions are made about these distributions and their correlations (see chapter 5).

2.3 The Multimodal Network Model

In this section we present the multimodal network model that was developed during this PhD. The network model should be straightforward, so that it can be used

in practical routing systems. Moreover, the model should be able to incorporate the networks of different modes of transportation uniformly, which results in one large network in which the generic shortest path algorithms can be applied with only minor modifications. Furthermore, it should be possible to model the costs of each mode of transportation differently. The travel time of walking is time-independent and deterministic, while the travel times in a car network may be time-dependent and stochastic. As we aimed at uniformly incorporating the networks of each mode while restricting the memory usage, we opted for a layered time-dependent network model. This model is similar to the model that was presented in [36]. In this model each layer represents the network of one mode of transportation, in which the unimodal nodes represent geographic locations and all time-dependent information is modeled in the cost objects of the links. Trans-shipments between the different modes are modeled as inter-layer links between the geographically co-located nodes of the different modes. Furthermore, we will assume that all links are directed. This allows us to model direction-specific structures, such as one-way roads, while undirected links still can be represented by two anti-parallel links.

In this section we are only concerned with the network modeling. The modeling of the costs, that are assigned to the links in these networks, will be treated in the following section.

Mathematically, the network model can be described as follows. Let us consider a multimodal transport network with n transport modes. To each of these modes we assign a layer

$$L_i = (V_i, E_i), i = 1, \ldots, n \tag{2.1}$$

with V_i a set of nodes (vertices) and $E_i = \{(u, v) | u, v \in V_i\}$ a set of directed links (edges) in layer L_i.

Subsequently these transportation networks (layers) are interconnected by means of trans-shipment links, i.e. links between geographically co-located nodes of different transport modes. This can be represented by an equivalence relation R containing all pairs of vertices with the same geographic coordinates:

$$R = \{(u, v) | u \in V_i; v \in V_j; i \neq j; 1 \leq i, j \leq n; geo(u) = geo(v)\} \tag{2.2}$$

with $geo(v)$ the geographic coordinates of node v. It can be seen that the elements of R represent the trans-shipment links between the different transport modes. If no trans-shipment infrastructure between two modes is present at a specific location the cost of this link is set to infinity. Alternatively, these links could be removed from the networks, which would consume less memory.

The different transportation layers, together with the trans-shipment links, are a good representation of the multimodal transport network. However, in many

situations we want to determine the shortest route between two locations independently of the transport mode of the origin and the destination, which is not possible in the model as presented above. To enable this, an additional layer is introduced: the access layer. This layer contains a node for every possible geographic location in the multimodal network and no links. So, this layer can be defined as

$$L_0 = (V_0, E_0) \tag{2.3}$$

with

$$V_0 = (\bigcup_{i=1}^{n} V_i)/R \tag{2.4}$$

and

$$E_0 = \emptyset \tag{2.5}$$

V_0 represents a partitioning in the union of all mode-specific nodes using the equivalence relation R. In this way, each node in the access layer (L_0) in fact symbolizes a set of nodes which all have the same geographic coordinates. Furthermore, additional inter-layer edges are needed to connect each of the nodes of L_0 to the elements in its set. This can be represented by a relation R_0^S, which is the symmetric closure of

$$R_0 = \{(u, v) | u \in V_0, v \in V_i, v \in u, 1 \leq i \leq n\} \tag{2.6}$$

i.e.

$$R_0^S = R_0 \cup \{(u, v) | (v, u) \in R_0\} \tag{2.7}$$

We would like to remind that the nodes of the access layer ($u \in V_0$) are sets of nodes themselves, which makes the expression $v \in u$ possible for $v \in V_i$.

Combining the above, the complete or aggregated layered network can be defined as the graph

$$G_L = (V_L, E_L) \tag{2.8}$$

with

$$V_L = \bigcup_{i=0}^{n} V_i \tag{2.9}$$

and

$$E_L = (\bigcup_{i=0}^{n} E_i) \bigcup R \bigcup R_0^S \tag{2.10}$$

V_L consists of all nodes from all transport layers, including the warehouse layer. E_L contains all links from the different transport layers, together with the trans-shipment links and the access links (connecting the nodes of the access layer to their elements).

This multimodal network model is illustrated in figure 2.4 for the case of freight transport with three modes of transportation: truck, train and ship.

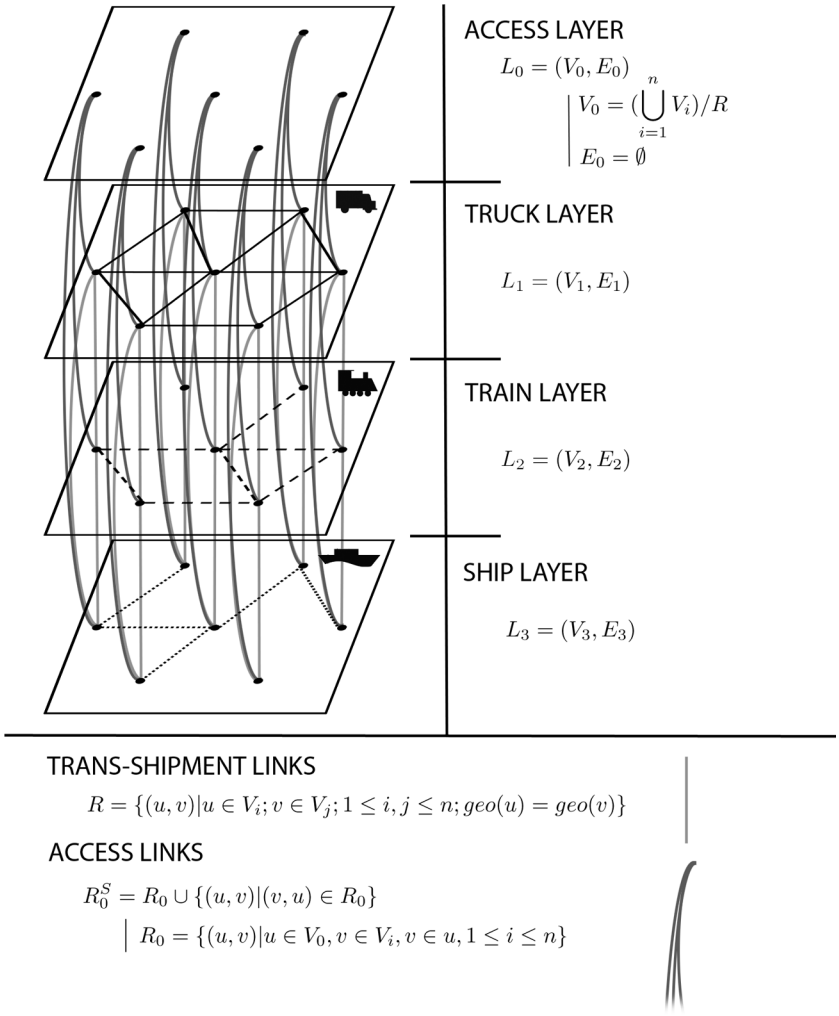

Figure 2.4: The layered multimodal network model of a freight transportation network with three transport modes: truck, train and ship. The trans-shipment links (light gray) connect the geographically co-located nodes of the different transport modes. The access links (dark gray) connect the access nodes with their co-located elements in the transport layers.

2.4 Cost Modeling

Once the network model has been defined, costs need to be assigned to the network links (and/or nodes). There are multiple objectives that may be optimized when searching for the shortest path in transportation networks and each objective requires different link costs. Some of these costs, such as the distance, the travel time, the number of transfers, etc., may me represented by single values. Here, only one objective is optimized when searching for the best route. Often, however, multiple objectives need to be optimized simultaneously, which requires the links to have cost objects that contain multiple values. Furthermore, some costs may vary in time. The travel time on a road, for example, is dependent upon the traffic situation and will thus be higher during rush hours. In this case, a time-dependent cost model should be applied. Moreover, not all costs are deterministic by nature. The time needed to get to a destination may be dependent upon a number of random external factors, such as individual drivers behavior, weather conditions, etc. To model this uncertainty, a stochastic cost model is used. In this section these different cost models are presented.

For detailed examples of each of these cost models, we would like to refer to section 5.2.2 of chapter 5, in which a case study is presented of a real-life multimodal routing system.

2.4.1 Single Value Costs

Most of the current data that has been made available for transportation networks consists of single values for each of the links of the network. There are costs that can indeed be represented as single values. The distance, for example, does not change as time passes or when some external factors change. Moreover, frequently more complex cost structures are simplified so that they can be represented by a single value. For example, despite of the fact that travel times may change according to the traffic situation, often the average travel times are used. The advantages of having single value costs are the lower memory consumption and the fact that most known shortest-path algorithms can be applied without any modifications. Mathematically single value costs are defined as

$$c(u, v) \tag{2.11}$$

where u and v indicate the link to which the cost is assigned ($(u, v) \in E$), and $c(u, v)$ represents a non-negative real number ($c(u, v) \in \mathbb{R}^+$). It should be noted that, in this dissertation, we assume all costs to be non-negative. Moreover, the network contains no loops with cost zero.

2.4.2 Multiple Objective Costs

Often, there is no unambiguous objective when determining the 'shortest' path in transportation networks. Some users are looking for the shortest path (distance), while others might want the fastest path (travel time). Moreover, when traveling with public transportation, one often also wants to minimize the number of transfers. When it is clear which objective needs to be minimized, single value costs can be used. In many cases however multiple objectives should be optimized simultaneously. In public transportation, for example, users want to get to their destination as fast as possible, but at the same time want to limit the number of transfers. This means that now multiple costs should be assigned to the network links, namely one for each objective. We opted to model this in the form of a vector. Mathematically, cost objects are now described as

$$c(u, v) = [c_1(u, v), c_2(u, v), \ldots, c_n(u, v)] \tag{2.12}$$

with n the number of objectives, u and v the end points of the link ($(u, v) \in E$) and $c_i(u, v), 1 \leq i \leq n$ all non-negative real values ($c_i(u, v) \in \mathbb{R}^+$).

2.4.3 Time-dependent Costs

Some costs may vary with time. For example, the travel time on a busy highway will be much higher during the rush hours than outside these hours. This means that a time-dependent cost model should be developed. In the generic case time-dependent costs are often modeled as piecewise linear functions (see figure 2.5). These functions can be described by a number of entries, each composed of a time stamp and a cost value. Intermediate values can then be calculated by linear interpolation of the neighboring entries.

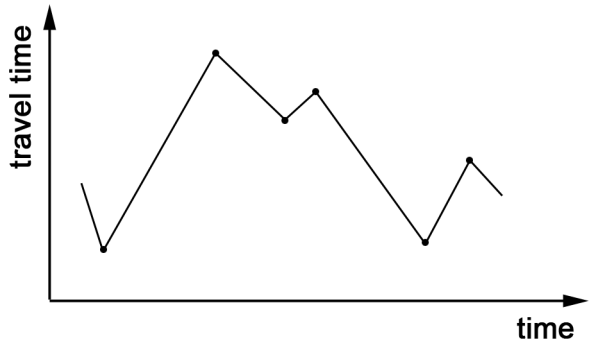

Figure 2.5: Example of a piecewise linear travel time function.

As the information on the transportation networks, such as the travel time on the links, is often measured regularly, we opted to divide the time window (for

example one day) into a number of equal time slots. For each time slot one single value is calculated, representing the cost value at a certain point in time inside this time slot. Interpolation is used to calculate the costs between the different points of time. The advantage of using equal time slots is the high performance of the lookup operations, as the translation of a point of time to its index is now possible in constant time. At the same time, a trade-off arises between the accuracy of the travel time function and the memory consumption. Smaller time slots (e.g. one minute) mean that more memory is needed to store the travel time function (e.g. 1440 values for one day). Larger time slots (e.g. one hour) consume less memory (e.g. 24 values for one day), but this comes at the cost of losing accuracy of the travel times. It is possible that during one hour the traffic may vary drastically. A travel time function with only one value per hour may not have captured this information.

Mathematically, the time-dependent cost model can be represented as

$$c(u,v) = [c_1(u,v), c_2(u,v), \ldots, c_k(u,v)] \tag{2.13}$$

with k the number of time slots, $(u,v) \in E$ and $c_i(u,v), 1 \leq i \leq k$ non-negative real numbers ($c_i(u,v) \in \mathbb{R}^+$). This is similar to the multiple objectives cost model, but here all costs are of the same quantity and have a time stamp (derived form the index) assigned to them.

This cost model is also applicable when dealing with time table information in public transportation. Time slots then represent the smallest unit that is found in the time table. For the time slots that represent the points in time where a vehicle departs, the cost value represents the travel time to get to the next stop or station. For all other time slots, the values are a combination of the waiting time (in the station or at the stop) and the actual travel time of the vehicle. This is illustrated in figure 2.6 and explained in further detail in chapter 5 where a travel time cost model for the railroad network is presented.

The choice for this cost model for time dependent travel time information is based on the characteristics of the available travel time data. For the road networks we obtained travel time data that was measured for every quarter of an hour. With respect to public transportation, we only had the time table information of the railroad network at our disposition. As the railroad network is relatively small, applying this cost model had no real impact on the memory consumption. When other travel time data would be available and memory consumption becomes an issue, other data structures may be more favorable. For example, the time table information of a bus network could have a major impact on the memory consumption, since these networks are relatively large. Here, it would be advisable to represent a travel time function as a sorted list of entries, each with a departure and travel time. While this consumes less memory, more time is needed to determine the exact travel time (i.e. waiting and driving time). However, the data structure that

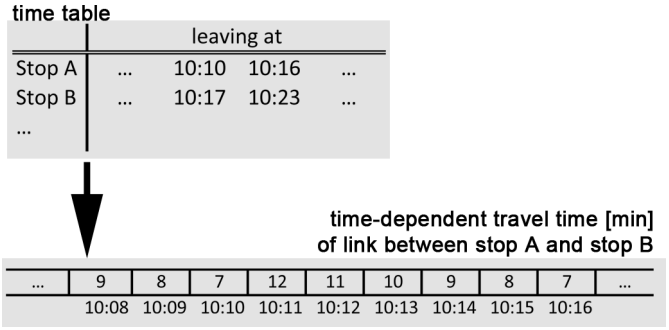

Figure 2.6: Determining the time-dependent travel time of the link between stop A and stop B making use of the time table information.

was presented in this section proves to perform well in practical routing systems in which memory consumption is no real constraint.

2.4.4 Stochastic Costs

A number of costs, like the travel time, have an inherent amount of uncertainty, as they may be influenced by random external factors. The travel time, for example, may vary due to individual drivers behavior, weather conditions, etc. These uncertainty characteristics can be incorporated in the cost model by assuming the costs are stochastic and representing them with a stochastic distribution. As mentioned earlier, these distributions may take a specific form, such as the log-normal distribution, or be independent of any known distribution. In this research it is assumed that the distributions may take any form. A new data structure was developed that represents a generic distribution. In order to minimize the memory consumption and simplify the calculations, we opted to define a stochastic distribution as a predefined number m of percentiles. The $x\%$ percentile is defined as the value that is higher than $x\%$ of all values of the distribution. The percentile values can be deduced from the cumulative distribution as shown in figure 2.7. Stochastic costs (i.e. distributions) can then be described mathematically as

$$c(u, v) = [c_{p_1\%}, c_{p_2\%}, \ldots, c_{p_m\%}] \tag{2.14}$$

with $c_{p_i\%}$ the value of the $p_i\%$ percentile and m the number of percentiles. This number m determines how detailed the information is. A large number of m provides a very detailed description of the stochastic distribution, at the cost of more memory consumption, while for lower values of m, which require less memory, important information of the distribution may be lost. So, determining a good m is a trade-off between memory consumption and accuracy of the information. For

a more complete description of stochastic costs, we would like to refer to chapter 5.

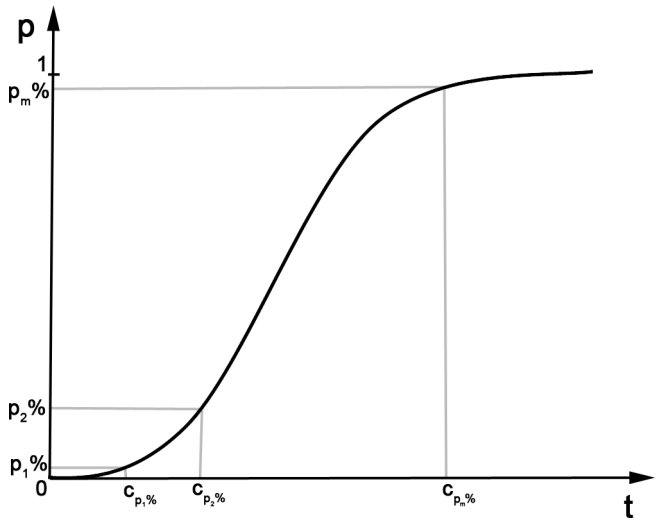

Figure 2.7: Determining the percentile values from the cumulative distribution. t = travel time, p = probability

2.4.5 Combining Different Types of Costs

The cost models that were presented in the previous subsections may be combined in order to form new cost models. The travel time of cars in a road network, for example, is in most cases both time-dependent and stochastic. The single values of each time slot in the time-dependent cost model should then be replaced by stochastic distributions. This results in a two-dimensional array to model the time-dependent and stochastic travel times.

Furthermore, in multimodal transportation networks each mode of transportation may have a different cost model to represent the costs. For example, the travel time cost for walking is a single value, while it is assumed time-dependent when taking the train. To determine the best route in a network, the costs of the different links need to be combined (mostly added). In order to combine costs with different cost models, the less detailed costs should be translated to fit into the most detailed cost model. Single values can easily be translated to time-dependent costs by assuming the same value for all time slots. Similarly, deterministic costs can be translated to stochastic ones by assuming the same value for all percentiles. In chapter 5 an algorithm is presented that is able to cope with these different types of costs.

2.5 Conclusion

In this chapter a generic multimodal transportation network model was presented that can be used for both personal and freight transportation. This layered network model preserves the networks of each of the transport modes, which are then interconnected by means of trans-shipment links. An access layer was added in order to make routing independent of the transport mode in the origin and the destination.

Furthermore, a number of cost models were presented. For time-independent (i.e. static) and deterministic costs, models were proposed for both single and multiple objective costs. They are represented by a single value and a vector of values respectively. In the time-dependent case, link costs are piecewise linear functions, that are represented by a list of values, where it was assumed that all intervals are of equal size. As some of the costs may be uncertain, a stochastic cost model was developed. We opted for a compact representation of stochastic distributions, viz by a predefined number of percentile values. Moreover, some insight was given on how to combine these different cost models.

The aim of this research was to determine a network model that can be used by different shortest path routing algorithms. The algorithms that are presented in the following chapters indeed may be applied to networks of this model. As no adequate multimodal data was available, the results in chapter 4 were gathered from unimodal networks. Nevertheless, making use of the network model presented here would make no difference. According to the objectives, different cost models are used. In chapter 3 we will work with single objective deterministic time-independent costs. Chapter 4 deals with multiple objective (deterministic time-dependent) costs. In the following chapter the time-dependent and stochastic character of the travel times will also be taken into account. Furthermore, in chapter 5 it will be proven that this network model indeed performs well in high-end transportation applications.

References

[1] L. Euler. *Solutio Problematis ad Geometriam Situs Pertinentis*. Commentarii Acedemiae Scientiarum Imperialis Petropolitanae, 8:128–140, 1736.

[2] B. Jourquin and M. Beuthe. *Transportation policy analysis with a geographic information system: The virtual network of freight transportation in Europe*. Transportation Research Part C: Emerging Technologies, 4(6):359 – 371, 1996.

[3] P. Harker. *Predicting Intercity Freight Flows*. VNU Science Press, Utrecht, 1987.

[4] T. Crainic, M. Florian, J. Guélat, and H. Spiess. *Strategic Planning of Freight Transportation: STAN, an interactive graphic system*. Transportation Research Record, 1283, 1990.

[5] F. Southworth and B. E. Peterson. *Intermodal and international freight network modeling*. Transportation Research Part C: Emerging Technologies, 8(1-6):147 – 166, 2000.

[6] R. Van Nes. *Design of multimodal transport networks, a hierarchical approach*. TRAIL Thesis Series. Delft University Press, The Netherlands, 2002.

[7] S. Jung and S. Pramanik. *An efficient path computation model for hierarchically structured topographical road maps*. Knowledge and Data Engineering, IEEE Transactions on, 14(5):1029 – 1046, 2002.

[8] N. Jing, Y.-W. Huang, and E. Rundensteiner. *Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation*. Knowledge and Data Engineering, IEEE Transactions on, 10(3):409 –432, 1998.

[9] M. Mainguenaud and X. Simatic. *A data model to deal with multi-scaled networks*. Computers, Environment and Urban Systems, 16(4):281 – 288, 1992.

[10] U.-J. Rüetschi and S. Timpf. *Modelling Wayfinding in Public Transport: Network Space and Scene Space*. In C. Freksa, M. Knauff, B. Krieg-Brckner, B. Nebel, and T. Barkowsky, editors, Spatial Cognition IV. Reasoning, Action, Interaction, volume 3343 of *Lecture Notes in Computer Science*, pages 24–41. Springer Berlin / Heidelberg, 2005.

[11] J. Booth, P. Sistla, O. Wolfson, and I. F. Cruz. *A data model for trip planning in multimodal transportation systems*. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in

Database Technology, EDBT '09, pages 994–1005, New York, NY, USA, 2009.

[12] M. Bielli, A. Boulmakoul, and H. Mouncif. *Object modeling and path computation for multimodal travel systems*. European Journal of Operational Research, 175(3):1705 – 1730, 2006.

[13] T. Pajor. *Multi-Modal Route Planning*. Univerity of Karlsruhe, Germany, 2009.

[14] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. *Efficient models for timetable information in public transportation systems*. Journal on Experimental Algorithmics, 12:1–39, 2008.

[15] G. S. Brodal and R. Jacob. *Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries*. Electronic Notes in Theoretical Computer Science, 92:3 – 15, 2004.

[16] K. Nachtigall. *Time depending shortest-path problems with applications to railway networks*. European Journal of Operational Research, 83(1):154 – 166, 1995.

[17] A. Orda and R. Rom. *Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length*. Journal of the ACM, 37(3):607–625, 1990.

[18] A. Orda and R. Rom. *Minimum weight paths in time-dependent networks*. Networks, 21(3):295–319, 1991.

[19] M. Muller-Hannemann, M. Schnee, and K. Weihe. *Getting Train Timetables into the Main Storage*. Electronic Notes in Theoretical Computer Science, 66(6):8 – 17, 2002.

[20] S. Pallottino and M. Scutellà. *Equilibrium and Advanced Transportation Modeling*. Kluwer Academic Publishing, Boston, 1998.

[21] F. Schulz, D. Wagner, and K. Weihe. *Dijkstra's algorithm on-line: an empirical case study from public railroad transport*. Journal on Experimental Algorithmics, 5, 2000.

[22] F. Schulz, D. Wagner, and C. Zaroliagis. *Using Multi-level Graphs for Timetable Information in Railway Systems*. In D. Mount and C. Stein, editors, Algorithm Engineering and Experiments, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer Berlin / Heidelberg, 2002.

[23] D. Pretolani. *A directed hypergraph model for random time dependent shortest paths*. European Journal of Operational Research, 123(2):315 – 324, 2000.

[24] J. Wardop. *Some theoretical aspects of road traffic research*. In Proceedings of the Institution of Civil Engineers, volume 1, pages 325–378, 1952.

[25] R. Herman and T. Lam. *Trip time characteristics of journeys to and from work*. In Proceedings of the 6th International Symposium on Transportation and Traffic Theory, pages 57–85, 1974.

[26] A. Polus. *A study of travel time and reliability on arterial routes*. Transportation, 8:141–151, 1979.

[27] G. Dandy and E. McBean. *Variability of individual travel time components*. ASCE Journal of Transportation Engineering, 110:340–357, 1984.

[28] M. Mogridge and S. Fry. *The variability of car journeys on a particular route in central London*. Traffic Engineering and Control, 25:510–511, 1984.

[29] F. O. Montgomery and A. D. May. *Factors affecting travel times on urban radial routes*. Traffic Engineering and Control, 28:452–458, 1987.

[30] H. Rakha, I. El-Shawarby, M. Arafeh, and F. Dion. *Estimating Path Travel-Time Reliability*. In Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE, pages 236 –241, 2006.

[31] Y. L. G. J. Chen, K. and H. Wen. *Characteristics analysis of road network reliability in Beijing based on the data logs from taxis*. In TRB 2007 Annual Meeting, Washington, DC, pages on CD–ROM, 2007.

[32] I. Kaparias. *Reliable dynamic in-vehicle navigation*. Imperial College London, UK, 2008.

[33] T. Lomax, D. Schrank, S. Turner, and R. Margiotta. *Selecting travel reliability measures*. Technical report, Texas Transportation Institute, 2003.

[34] Y. Nie and Y. Fan. *Arriving-on-Time Problem: Discrete Algorithm That Ensures Convergence*. Transportation Research Record: Journal of the Transportation Research Board, 1964:193–200, 2006.

[35] S. Samaranayake, S. Blandin, and A. Bayen. *A Tractable Class of Algorithms for Reliable Routing in Stochastic Networks*. Transportation Research Part C, 20:199 – 217, 2012.

[36] S. Demeyer, P. Audenaert, B. Slock, M. Pickavet, and P. Demeester. *Multimodal transport planning in a dynamic environment*. In Proceedings of IPTS2008, the Conference on Intelligent Public Transport Systems, pages 155–167, 2008.

3

The Predecessor and the Accounting Algorithm

Goal-directed algorithms guide the search towards the destination and neglect the less interesting areas of the network. By a limited search in the most promising area of the network, shortest path calculations can be sped up remarkably. In this chapter we will investigate how well goal-directed algorithms perform in multimodal transportation networks. Moreover, two novel goal-directed heuristics are presented: the predecessor and the accounting algorithm. While the predecessor algorithm makes use of local information to guide the search towards the destination, the accounting algorithm additionally uses the path's history. Moreover, these two heuristic algorithms can be further improved by a number of adaptations. Experiments were carried out to report on both the performance of the algorithms and the accuracy of the results in a multimodal transportation network.

3.1 Introduction

In route planning applications, minimal execution times of the shortest path algorithms are of uttermost importance. For one thing, these applications have a large amount of users, each with a number of requests. Moreover, current intelligent transportation systems usually react to occurring events on the route, like traffic jams and accidents. For this, new routes need to be calculated in real time to inform the drivers as soon as possible. This chapter focuses on goal-directed

techniques to speed up the shortest path calculations.

3.1.1 Related Work

In section 1.3.2 an overview (based on [1]) was given of the most commonly used speedup measures for shortest path calculations in transportation networks. These speedup methods limit the search area by either restricting it to a specific part of the network (e.g. the bidirectional search [2], the subgoal method [3] and the goal-directed search) or by selecting which links to investigate (e.g. the hierarchical search [4]).

This chapter focuses on goal-directed algorithms that limit the search area around the origin, the destination and the area between them. Nodes which have a low probability of being part of the shortest path between the origin and the destination, are ignored. Fine-tuning the parameters, which determine the area of interest, of these algorithms is a constant trade-off between the performance of the algorithm and the accuracy of the results.

There are two major goal-directed algorithms that can be applied in almost all networks and that do not require any preprocessing: the branch pruning and the A* algorithm. While the branch pruning algorithm eliminates the nodes that have a low probability of being on the shortest path, the A* algorithm gives them a lower priority to be investigated.

A straightforward implementation of the branch pruning algorithm can be found in [5]. In this implementation, nodes (n) for which the following inequality does not hold, are pruned from further examination:

$$l(n) + e(n, d) \leq E(o, d) \tag{3.1}$$

with $l(n)$ the label of node n, $e(n, d)$ the estimated cost from node n to the destination d, and $E(o, d)$ an estimated upper bound of the cost of the path between the origin and the destination node. For a complete and detailed description of this implementation, we would like to refer to section 3.2.

There are multiple ways to determine which part of the network should be investigated. In [6] a branch pruning algorithm is presented where the search is limited to a rectangular area, called a window. Lysgaard [7] presents a two-phase branch pruning algorithm where in the first phase an upper bound is determined heuristically and in the second phase this upper bound is used to determine whether or not to investigate a node.

The A* algorithm does not ignore the unfavorable parts of the network, but makes it less likely that they will be investigated, by lowering their priority. These priorities are calculated based on estimated 'distances'. In [8], these are indicated as potentials which are used together with the Dijkstra label to determine the priority of nodes to be processed. A higher value of the sum of the label and the

potential means that the node has a lower priority. Moreover, an implementation is presented in which the algorithm of Dijkstra may be applied directly. For this, the original cost $c(u, v)$ of a link (u, v) is replaced with

$$c'(u, v) = c(u, v) + (e_{od}(v) - e_{od}(u)) \tag{3.2}$$

where $e_{od}(v)$ indicates the potential of node v with respect to the path between the origin o and the destination d.

Different interpretations have been proposed to model potentials. The most straightforward one makes use of the Euclidean distance [9], but other interpretations have been investigated too, like for example distances from graph condensation [10]. Other implementations of the A* algorithm [11–13] do not replace the link costs but assign new labels to the nodes, taking into account the potentials. These are then used to determine the best node to be investigated next (i.e. the minimum of the temporary set).

Next to the branch pruning and the A* algorithm (also called geometric goal-directed search), a number of goal-directed algorithms were developed specifically for transportation networks [14]. In the edge labeling technique [8, 15], for each edge in the networks a superset of nodes is (pre)computed which are situated on the shortest paths that start with the edge in question. The most popular implementations [16–18] divide the network in k regions and assigns k flags to each link indicating whether nodes of these regions are visited by shortest paths starting from this link. The SHARC algorithm [19] is an extension of the edge flag approach that incorporates hierarchical concepts. In the landmark A* (ALT) approach [20, 21] the distances between the landmarks (i.e. nodes that are well distributed over the far ends of the network) are precomputed and used to determine the shortest paths. In [22], Maue et al. present the Precomputed Cluster Distances (PCD) algorithm. The network is partitioned in clusters and the shortest connections are calculated between all pairs of clusters. These yield lower and upper bounds for the distances that can be used to prune the search.

The major disadvantage of all these goal-directed algorithms is that they require a preprocessing step on the network. In this chapter, we avoid preprocessing in order to be able to cope with the dynamic character of transportation networks.

3.1.2 Setup of the Experiments

All algorithms that are addressed in this chapter were implemented in Java (version 1.6.0-18) on a machine with the following configuration: Intel ® Core ™ 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM.

In the experiments, a European multimodal freight network was used with three modes of transportation: ship, train and truck. To this network, the model, that was presented in chapter 2, was applied, resulting in an overall (directed) network with 178 077 nodes and 475 188 nodes. Table 3.1 gives a detailed overview

of the number of nodes and links in each of the layers of this network. It should be noted that both the access layer nodes and the access links contribute remarkably to the order and the size of the network. Therefore, in practical shortest path routing algorithms not all access links are actually added to the network, but only those of the origin and the destination node. This means that, in our example, the network would have only 6 actual access links, corresponding with the origin and the destination.

	# nodes	**# links**
access layer	88430	-
ship layer	1 937	3 830
train layer	37 107	80 736
truck layer	50 603	132 972
access links	-	179 294
trans-shipment ship-train	-	268
trans-shipment ship -truck	-	3 874
trans-shipment train-truck	-	74 214
total	178 077	475 188

Table 3.1: European multimodal freight network with three modes of transportation: ship, train and truck.

Each link of the transportation layers has a single static and deterministic cost assigned to it, representing the actual length of the link in kilometers. Furthermore, a fixed cost was assigned to all access and trans-shipment links.

The results presented in this chapter were gathered from 10 000 shortest path calculations between random origin and destination nodes.

This chapter focuses on two characteristics of the goal-directed algorithms: the performance (in terms of execution time) and the accuracy. In general, there is a trade-off between these two. Fast algorithms produce less accurate results and vice versa.

3.1.3 Outline

This chapter is organized as follows. In the next section, the branch pruning algorithm is resumed. In order to better understand the remainder of this chapter, an overview is given of our interpretation of this algorithm. Additionally, a comparison is made between our implementation of this goal-directed algorithm and the algorithm of Dijkstra with respect to the execution time and the optimality of the solution. The following section focuses on the two novel goal-directed heuristics, namely the predecessor algorithm and the accounting algorithm. While the prede-

cessor algorithm makes use of local information to guide the search towards the destination, the accounting algorithm additionally makes use of the history of the path found so far. Moreover, a number of adaptations/optimizations are presented that may be applied to either of these heuristics. After an elaborate theoretical explanation of these algorithms and the optimizations, experimental results with respect to the performance and accuracy are presented in the section 3.4. Finally, in the last section, a conclusion is formulated.

3.2 Branch Pruning Algorithm

In order to better understand the remainder of this chapter, in this section we will elaborate on one of the most common goal directed heuristics, the branch pruning algorithm, as this algorithm resembles most the algorithms that will be presented in the following section. It is based on the algorithm of Dijkstra [23] (see section 1.3.1 of chapter 1) with the difference that here only a fraction of the nodes is investigated. As, in this chapter, the shortest paths are calculated in a transportation network and the distance is used as objective function, we opted to use the Euclidean (i.e. beeline) distance as estimation function or potential. This can be easily calculated as nodes in transportation networks represent geographic locations.

The branch pruning heuristic limits the search to the area between the origin and the destination by only taking into account a part of the network and ignoring the remainder. Nodes which are unlikely to be on the shortest path between the origin and the destination are eliminated.

The basic idea behind branch pruning is similar to the one of the artificial intelligence technique IDA* (Iteratively Deepening A*) [24], a depth-first tree search algorithm in which a branch is cut off when its total cost exceeds a given threshold. In the branch pruning algorithm only the nodes that have a high possibility of being on the shortest path will be investigated. The links (or branches) that connect to nodes that are unlikely to be on the shortest path are pruned from the network. To determine whether a node has a high possibility to be situated on the shortest path, a condition needs to be determined. In our implementation (and most common implementations) a node n has a high probability of being on the shortest path between an origin o and a destination d if and only if

$$l(n) + \delta(n, d) \leq E(o, d) = \alpha.\delta(o, d) \tag{3.3}$$

Here $l(n)$ indicates the label of node n (i.e. the exact distance between o and n), while $\delta(u, v)$ denotes the estimated (Euclidean) distance between node u and node v. $E(o, d)$ represents an (upper bound) estimate of the length of the shortest path between the origin and the destination. In our implementation, this is the estimated distance between the origin and the destination multiplied by a correction factor

α. In the branch pruning algorithm a node n will only be investigated if condition 3.3 holds.

As mentioned earlier, this algorithm is a heuristic, which means that, while producing a result in a limited time, this algorithm does not guarantee to return the optimal solution. The accuracy of the solution depends upon the value of the right hand side of the inequality (in our case α). A higher value means that there is more chance that the optimal solution will be found. On the other hand, a lower value narrows the search space more which results in a faster algorithm, but less accurate solutions are produced. So a trade-off is made between efficiency and accuracy.

In order to always find the optimal solution, a number of restrictions should be taken into account when evaluating inequality 3.3. First of all, the right hand side should be an upper bound of the actual length of the path between the origin and the destination. Otherwise, the path shall never be found as the destination node will never be permanent, since its label $l(d)$ will always be higher than the right hand side of the inequality. As stated before, the lower this upper bound is, the faster the algorithm. If, on the other hand, this value is too high, too many nodes will be investigated.

Moreover, the estimated distance from the current node to the destination should be a lower bound of the actual distance. This can be demonstrated as follows. Suppose that the lowest possible bound (i.e. the actual length $l(d)$ of the shortest path) is used in the right hand side of the inequality and that we know that a certain node n is situated on the shortest path between the origin o and the destination d. To determine whether to investigate this node n during the execution of the algorithm, following inequality is evaluated:

$$l(n) + \delta(n, d) \leq l(d) \tag{3.4}$$

or

$$\delta(n, d) \leq l(d) - l(n) \tag{3.5}$$

Since node n is situated on the shortest path $l(d) - l(n)$ represents the length of the shortest path between the current node n and the destination d. This proves that the estimated distance should be less or equal than the length of the actual shortest path.

Unfortunately, it is impossible to optimally define the parameter and estimation functions in inequality 3.3. The upper bound in the right hand side should be as low as possible, in order to minimizes the number of nodes that are investigated and thus the execution time. However, the optimal bound is different for each origin-destination pair.

As stated previously, in our implementation, the Euclidean distance is used to estimate the distance between nodes of the network and the right hand side of the inequality 3.3 is defined as α times the estimated distance between the origin and

the destination. This correction factor α has a major influence on both the performance of the algorithm and the accuracy of the produced results. Experiments were carried out on the network that was presented in section 3.1.2. Figure 3.1 shows the execution time of the branch pruning algorithm with different correction factors in function of the length of the shortest path. The execution time of the algorithm of Dijkstra is also depicted for comparison. It can be seen that the branch pruning algorithm performs better than the algorithm of Dijkstra. Furthermore, as mentioned earlier, the execution time of the branch pruning algorithm is dependent upon the factor α: the higher this factor, the higher the execution time.

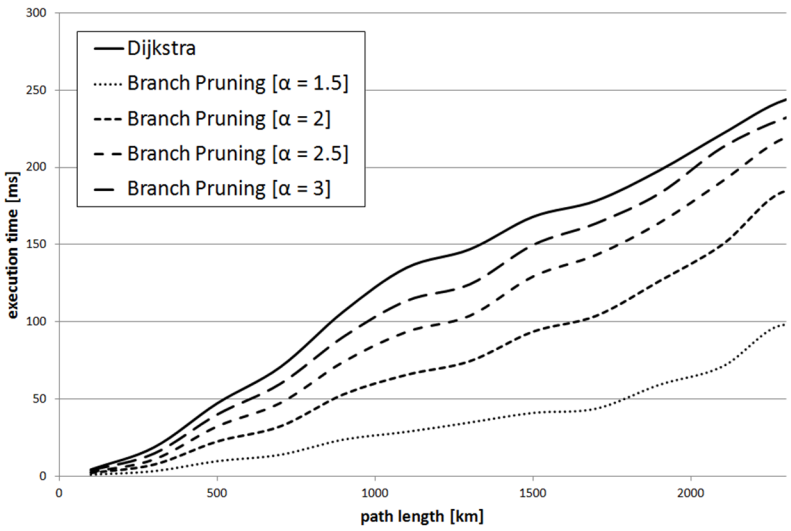

Figure 3.1: Execution time of the branch pruning algorithm with different values for α in function of the shortest path length.

Secondly, the accuracy of the produced results was investigated. It should be noted that, as this algorithm resembles closely the algorithm of Dijkstra, when a path has been found, this path is almost always optimal (in more than 99% of the cases). This means that only the amount of paths that were not found by the algorithm (i.e. the loss rate) should be reported. Table 3.2 shows these loss rates for the different values of α. It is clear that an increasing value of α means a decreasing loss rate. Unfortunately, but as expected, the branch pruning algorithms with the lowest loss rates also have execution times similar to those of the algorithm of Dijkstra.

Next, we investigated the impact of the quality of the estimation function. Instead of the Euclidean distance, now the Euclidean distance divided by 2 was used. Moreover, the correction factors were multiplied by 2 in order to produce compa-

α	**loss rate**
1.5	80.3%
2	32.0%
2.5	5.1%
3	0.8%

Table 3.2: Loss rate of branch pruning algorithm in function of α

rable results. The execution times and loss rates are depicted in figure 3.2 and table 3.3 respectively. A worse estimation function also means higher calculation times and slightly less paths that are found. In conclusion, defining a good estimation function is of uttermost importance.

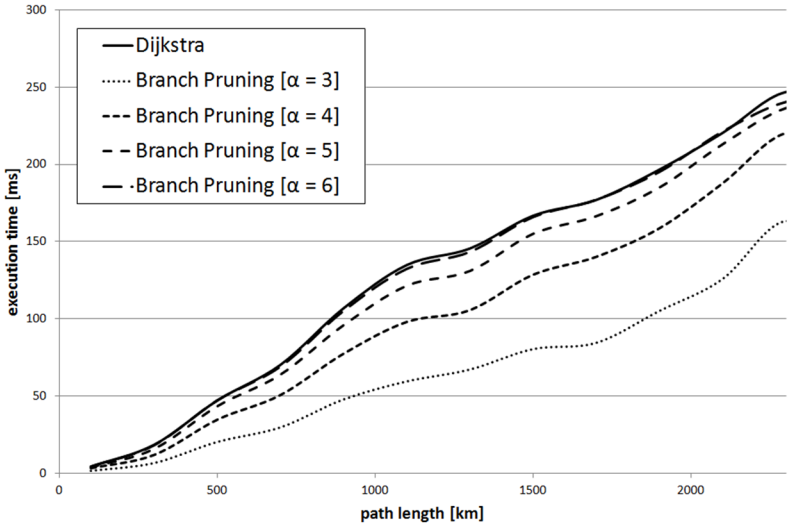

Figure 3.2: Execution time of the branch pruning algorithm with a lower (worse) estimation function in function of the shortest path length.

3.3 Novel Heuristics

The algorithm that was addressed in the previous section uses both estimated and exact (i.e. path lengths represented by labels) distances to determine whether or not a node is favorable to be on the shortest path. This implies that, in order for the algorithm to perform well, the estimates should be lower bounds of the

α	**loss rate**
3	82.8%
4	34.0%
5	6.2%
6	1.1%

Table 3.3: Loss rate of branch pruning algorithm with a lower (worse) estimation function for different values of α

actual shortest path lengths. The novel algorithms presented in this section avoid this requirement by only using estimates to determine whether a node should be investigated. This allows almost all estimation functions to be used, as long as they are more or less proportional to the actual path lengths.

Two novel heuristics are presented: the predecessor and the accounting algorithm. The predecessor algorithm makes use of local information to guide the search towards the destination. In the accounting algorithm a history of the path so far is taken into account to determine whether this path is worth investigating further. Moreover, a number of optimizations are presented. In subsection 3.3.1, two minor improvements are mentioned, namely full local search and the k-th predecessor. While the latter can be applied to both the predecessor and the accounting algorithm, the former is limited to the predecessor algorithm. Subsequently, in subsection 3.3.3, two major optimizations are introduced that can be applied to both heuristics: queue optimization and a feedback loop. A complete schematic overview of these algorithms and improvements is depicted in figure 3.3.

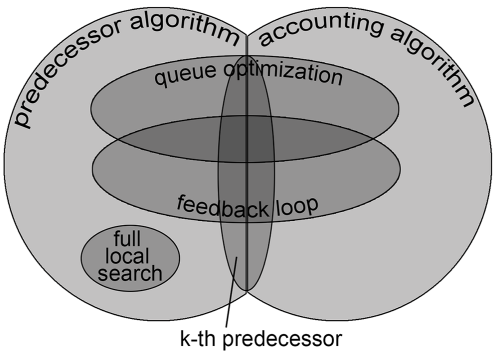

Figure 3.3: Schematic overview of the heuristics and optimizations.

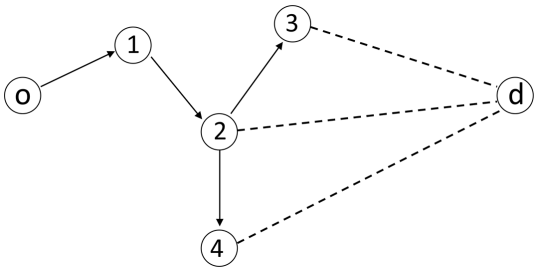

Figure 3.4: Basic concept of predecessor algorithm. Node 3 will be investigated as it is closer to the destination d than node 2. Node 4 will not be investigated since it is situated further away from the destination.

3.3.1 The Predecessor Algorithm

Similarly to the branch pruning algorithm, the predecessor heuristic will eliminate paths that do not evolve towards the destination. In the branch pruning algorithm the sum of the exact distance between the origin and the current node and the estimated distance from the current node to the destination is compared to the estimated distance between the origin and the destination. The predecessor algorithm avoids calculating this sum by comparing the estimated distance between a current node and the destination with the estimated distance between its predecessor node (on the path) and the destination. This way, it is determined whether adding the current node causes the path to grow in the direction of the destination.

In the predecessor algorithm, when calculating the shortest path between an origin o and a destination d, a node n is only investigated if and only if

$$\delta(n, d) \leq \delta(prev(n), d) \tag{3.6}$$

where $\delta(u, v)$ represents the estimated distance between node u and node v and $prev(n)$ denotes the previous node of n. This concept is illustrated in figure 3.4, in which node 3 will be investigated as it is closer to node d than node 2, while node 4 will not be investigated because the estimated distance between node 4 and node d is longer than the estimated distance between node 2 and the destination d.

This algorithm is similar to the algorithm of Dijkstra (see section 1.3.1), with the only difference that lines 9 to 15 (updating the neighbors of the current node u) are only executed if the inequality 3.6 holds for the node u that was just made permanent. This means that the following if-statement is added around the code from line 8 to line 15:

$$\texttt{if (}\delta(u, d) \leq \delta(prev(u), d)$$

Moreover, it should be noted that, to evaluate the condition 3.6, only one estimate

($\delta(n,d)$) needs to be determined as $\delta(prev(n),d)$ was already determined in a previous iteration.

Paths that are perceived as uninteresting by the algorithm (in this case paths which do not lead directly towards the destination) are eliminated. Unfortunately, a large number of paths (approximately 75%, see table 3.4) will not be found by the predecessor algorithm. The reason for this is that most shortest paths in transportation networks make a small detour to reach the destination. To overcome this disadvantage a tolerance factor is introduced, which allows paths to make these small detours. Now, to find the shortest path between an origin o and a destination d, a node n is only investigated if and only if

$$\delta(n,d) \leq (1+\gamma).\delta(prev(n),d) \tag{3.7}$$

where $(1+\gamma), \gamma \geq 0$ represents the tolerance factor. If $\gamma = 0$, this condition is equal to condition 3.6. When the tolerance factor $(1+\gamma)$ has a value greater than one, the predecessor algorithm allows the path to make (small) detours to get to the destination. The higher this tolerance factor is, the larger are the detours that are allowed.

In order to stimulate the algorithm to find more paths, two small optimizations are proposed: full local search and using the k-th predecessor. When adopting the full local search, the areas around the origin and the destination are investigated completely. This optimization stems from routing in transportation networks, where paths usually make detours close to the origin and the destination. These areas around the origin and the destination are often denoted as the first and the last mile respectively. We will assume that these first/last mile areas are circular with radius R. This radius may either be a constant or proportional to the estimated distance between the origin and the destination.

To conclude, in this improved version of the predecessor algorithm a node n is investigated if and only if one (or more) of the following conditions holds:

$$\begin{cases} \delta(o,n) & \leq R \\ \delta(n,d) & \leq R \\ \delta(n,d) & \leq (1+\gamma).\delta(prev(n),d) \end{cases} \tag{3.8}$$

with R the predefined radius of the area around the origin and the destination, and all other parameters as defined before.

In the predecessor algorithm a value is calculated for the current node (i.e. the estimated distance from this node to the destination), and this value is compared to the value of the previous node on the path. Instead of using the previous node, in the k-th predecessor algorithm the value of the current node is compared with the value of the k-th predecessor on the path. By doing so, larger detours are allowed, leading to more paths to be found. In contrast with a higher tolerance factor, this has not huge impact on the calculation time (see results in section 3.4.1). So, in the

k-th predecessor algorithm, to find the path between an origin o and a destination d, a node n is only investigated if and only if

$$\delta(n,d) \leq (1+\gamma).\delta(prev(n,k),d) \tag{3.9}$$

where $prev(n,k)$ denotes the k-th predecessor of node n. For $k=1$ this condition equals the condition 3.7. In the case the number of hops in the current path is smaller than k, the current node is compared to the origin node.

3.3.2 The Accounting Algorithm

While the predecessor algorithm makes use of local information, i.e. comparing estimated distances from the current node and its predecessor, in the accounting algorithm, additionally a memory is kept with information on the path found so far. Every path has an account with an amount of credits, which can be used for making detours. Each time a path is going in the wrong direction, a credit is taken away. To determine whether a path is going into the right direction, the same condition as in the predecessor algorithm is used. Thus, a credit is taken away from a path's account if and only if for the current node n on the path

$$\delta(n,d) > (1+\gamma).\delta(prev(n),d) \tag{3.10}$$

As we are routing from a single origin node (i.e. building the shortest path tree from this node), these accounts can be implemented similarly to labels. Each node n has assigned to it an additional value $A(n)$ that represents the amount of credits left in the account of the path on which the node is situated.

When searching for the shortest path, only the nodes on paths with positive accounts are investigated. This means that an `if`-statement should be added around line 8 to line 15 of the pseudo code of the Dijkstra algorithm. Moreover, if a new label is added to a node, the account of the corresponding path should be updated. The pseudo code of this algorithm is depicted below in which the lines that differ from the algorithm of Dijkstra are indicated by line numbers with additional characters.

```
1     forall(v ∈ V)
2         l(v) := ∞;
3     l(o) := 0;
3b    A(o) := SB;
4     P := EMPTY_SET;
5     T := {o};

6     while(!(T is empty) && !(d ∈ P)){
7         u := T.removeMin();
7a        if(A(u)>0){
8             P.add(u);
9             forall(n :  neighbor(u)){
10                l_new := l(n)+c(u,n);
11                if(l_new < l(n){
12                    l(n) = l_new;
13                    T.add(n);
13a                   A(n) := A(u);
13b                   if(δ(n,d) ≤ (1+γ).δ(prev(n),d))
13c                       A(n) := A(n)-1;
14                }
15            }
15b       }
16    }
```

It should be noted that at initialization only the origin node has an account value (starting budget SB), representing the initial credit for all paths.

Now the question arises which amount of initial credits should be given to these paths, as this starting budget determines the size of the detours. Determining a good starting budget is a trade-off between accuracy and performance. A higher amount of start credits causes the algorithm to visit more nodes, which slows down the calculations but at the same time increases the chances to find a result. It should be noted that the optimal starting budget is different for every network configuration.

This problem of determining a good starting budget can be diminished by, instead of only punishing paths by taking away credits when they go in the wrong direction, also rewarding paths by giving them extra credits when they go in the right direction. However, when the amount of credits that are taken away (when a step is taken in the wrong direction), is equal to the amount of credits granted (when a step is taken in the right direction), a path can take large detours, as is

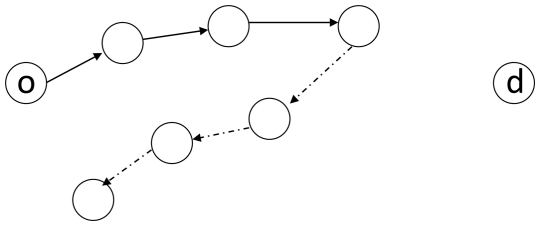

Figure 3.5: Shortcoming of accounting algorithm with awards and punishments (without upper bound).

illustrated in figure 3.5. This may lead to long zig-zagging paths, mostly on a local scale. To overcome this disadvantage, an upper bound needs to be set on the account, i.e. every path can only have a limited amount of credits. This upper bound (maximum budget MB) limits the size of the detours.

So, in this algorithm the level of the account of a path p at node n is

$$A(n) = \begin{cases} A(prev(n)) & , A(prev(n)) = MB \quad \& \\ & \quad \delta(n,d) \leq (1+\gamma)\delta(prev(n),d) \\ A(prev(n)) + 1 & , A(prev(n)) < MB \quad \& \\ & \quad \delta(n,d) \leq (1+\gamma)\delta(prev(n),d) \\ A(prev(n)) - 1 & , otherwise \end{cases} \tag{3.11}$$

where $A(n)$ denotes the account level at node n and the other parameters are defined as previously.

Similarly to the predecessor algorithm, this algorithm can be optimized by taking into account the k-th predecessor instead of the direct predecessor. This means that $prev(n)$ is replaced by $prev(n,k)$.

3.3.3 Further Optimizations

In the previous subsections goal-directed heuristics are presented that speed up the shortest path calculations at the cost of losing the optimality guarantee. These algorithms can be further improved by two other optimizations: queue optimization and a feedback loop. While the former uses a different criterion to determine whether the path is going into the right direction, the latter presents an iterative version of the algorithms presented above.

3.3.3.1 Queue Optimization

In the predecessor and the accounting algorithm, to determine whether a path is going into the right direction, the estimated distance from the current node was

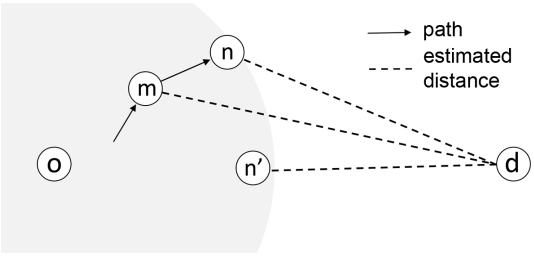

Figure 3.6: Queue Optimization. It should be noted that node n' was investigated just before node n. Node n will not be investigated here as it is situated further away from the destination d than node n', the previously investigated node. In the standard predecessor algorithm node n would have been investigated as it is closer to d than node m, the previous node on the path.

compared to the one from its previous node on the path. As will be shown in the results section, a speedup can be realized by comparing this estimated distance with the estimated distance from the previously investigated node. This is the node that was handled in the previous iteration. This technique is denoted with the term queue optimization (QO). If this is applied to the predecessor algorithm, to determine a path between an origin o and a destination d a node n is only investigated if and only if

$$\delta(n, d) \leq (1 + \gamma)\delta(prev_Q(n), d) \tag{3.12}$$

where $prev_Q(n)$ represents the node which is investigated before node n and the other variables are defined as previously. Queue optimization will cause more nodes to be skipped and thus speed up the calculations. Figure 3.6 shows an example of a node that will be skipped by the algorithm with queue optimization, but that would have been investigated in the algorithm without queue optimization.

3.3.3.2 Feedback Loop

While queue optimization speeds up the calculations, the feedback loop technique improves the accuracy of the produced results. To diminish the large amount of paths that were not found by the predecessor (and accounting) algorithm, a tolerance factor $(1 + \gamma)$ was introduced in section 3.3.1. This factor introduces a trade-off between the performance of the algorithm and the accuracy of the results. A higher value for γ means that more nodes will be investigated, resulting in higher execution times, but at the same time a higher chance of finding the optimal path, and vice versa.

The feedback loop technique aims at always finding a path between the origin and the destination by iteratively increasing the tolerance factor. Moreover, it lim-

its the execution times by keeping the tolerance factor as small as possible. The feedback loop technique starts with a small tolerance factor and in each iteration this factor is increased, as long as no path has been found. Here, it should be noted that all labels that were calculated during the previous iteration may be reused in the following iteration. This way, the execution time of one iteration is limited to only calculating the labels of the nodes that were ignored in the previous iteration.

There are multiple possibilities to model how the tolerance factor should be increased in each iteration. For example, a fixed amount can be added or it can be multiplied with a fixed factor. A small increase results in a large number of iterations until a path is found, while a large increase results in no additional time gain, as too much of the network is investigated early on.

The pseudo code of the feedback loop technique is given below, in which a path between origin node o and destination node d is calculated.

```
1   γ := start_value;
2   path := NULL;
3   while(path = NULL){
4       path := algorithm(γ).executeFurther(o,d);
5       γ := increase(γ);
6   }
```

In this code `algorithm(`γ`)` represents either the predecessor or the accounting algorithm in which the tolerance factor $(1 + \gamma)$ is used. In each iteration, the algorithm reuses the information that was calculated in the previous iteration.

3.4 Experimental Results

In the previous section two novel goal-directed heuristics were described, together with a number of improvements. This section presents experimental results that demonstrate the characteristics of these heuristics and the benefits of the improvements. For these experiments, the setup of section 3.1.2 was used.

3.4.1 The Predecessor Algorithm

The predecessor heuristic makes use of local information, i.e. the previous node on the path, to determine whether or not to handle the current node. A tolerance factor $(1 + \gamma)$ was introduced to allow more paths to be investigated. In this section we will demonstrate how this tolerance factor influences both the performance of the algorithm and the accuracy of the results. Moreover, this factor causes a trade-off between these two. Figure 3.7 shows the average execution time of the predecessor algorithm in function of the optimal shortest path length for different values of γ. For comparison the average execution time of the algorithm of Dijkstra is also

depicted. Similarly to the the branch pruning algorithm, an increasing tolerance factor means higher execution times. The highest speedup is perceived for the predecessor algorithm without a tolerance factor ($\gamma = 0$). This algorithm is on average 31.1 times faster than the algorithm of Dijkstra. If the tolerance factor is too high, the predecessor algorithm realizes no speedup at all and may even be slower than the algorithm of Dijkstra because of the additional estimation calculations.

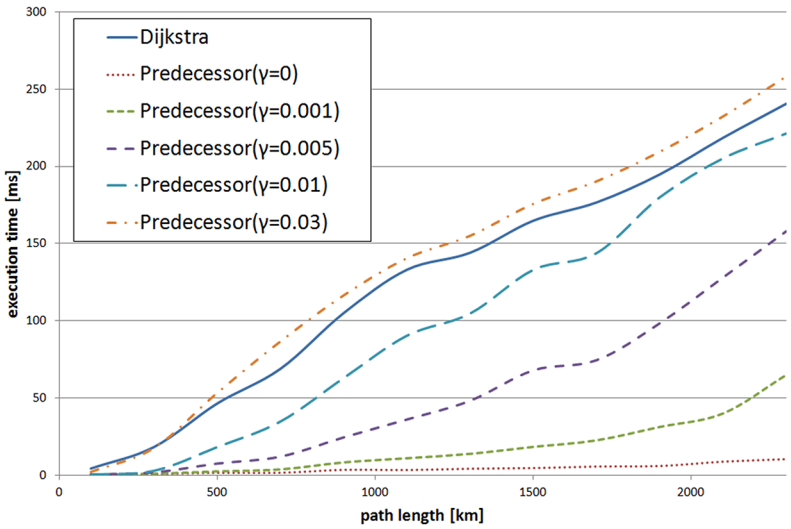

Figure 3.7: Execution time of the predecessor algorithm with different values of γ in function of the length of the optimal path.

While lower tolerance factors are more advantageous with respect to the performance of the algorithm, the reverse is true for the accuracy of the results. Table 3.4 shows for different values of γ both the loss rate (i.e. the percentage of paths that could not be found by the algorithm) and the average difference (Δ) in the case a path has been found. This latter one is defined as the percentage with which this found path differs from the optimal (Dijkstra) path:

$$\Delta = \frac{p_f - p_{opt}}{p_{opt}} \tag{3.13}$$

with p_f and p_{opt} the cost of the path found by the predecessor algorithm and the optimal path respectively. From the table, it can be seen that the loss rate and the average difference decrease as the tolerance factor increases. This means there is a trade-off between the performance and the accuracy. Higher execution times mean that more accurate results are produced, and vice versa.

As most of the 'detours' in transportation networks occur in the first and/or last

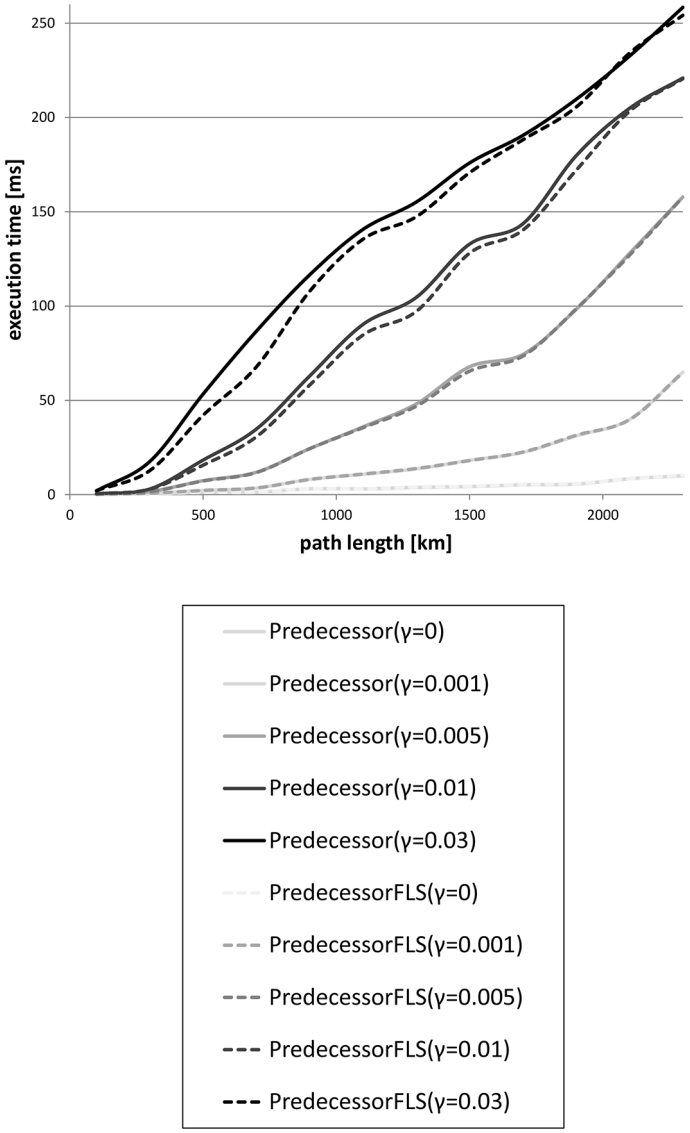

Figure 3.8: Execution time of the predecessor algorithm with and without Full Local Search (FLS) with different values of γ in function of the length of the optimal path.

	Predecessor Algorithm		**Predecessor Algorithm with FLS**	
γ	loss rate	avg. Δ	loss rate	avg. Δ
0	75.7%	7.01%	64.6%	6.68%
0,001	55.0%	4.02%	40.4%	3.40%
0,005	31.4%	2.84%	19.2%	1.47%
0,01	20.3%	2.81%	12.2%	1.28%
0,03	6.8%	1.96%	2.7%	0.62%

Table 3.4: Accuracy of predecessor algorithm and the predecessor algorithm with Full Local Search (FLS)

mile, an adaptation to the predecessor algorithm was proposed in which the areas around the origin and the destination are investigated completely. This is denoted with the term full local search. Experiments have shown that this measure indeed improves the predecessor algorithm. Figure 3.8 shows the execution times of the predecessor algorithm with and without Full Local Search (FLS) for the different values of γ. In these experiments, we opted to use an area radius R which is dependent on the estimated distance between the origin and the destination node: $R = \alpha.\delta(o, d)$ with $\alpha = 0.05$. From the figure, it can be seen that the predecessor algorithm with full local search performs as good as the one without full local search. Moreover, for the higher values of γ this algorithm is even faster, since no expensive comparisons are needed in the areas around the origin and the destination. Full local search not only performs well with respect to the execution time, but also produces more accurate results. On the right side of table 3.4, the loss rates and average differences Δ are given for the predecessor algorithm with full local search. Compared to the algorithm without full local search, this algorithm indeed finds more paths and these paths differ less from the optimal ones.

Next to full local search, another method was presented in section 3.3.1 to increase the amount of paths found by the algorithm. Instead of comparing an estimated value from the current node with the one from the previous node on the path, it is now compared with the one from the k-th previous node. This algorithm is called the k-th predecessor algorithm. We will demonstrate experimentally that this algorithm improves both the performance of the calculations and the accuracy of the results. In these experiments, we assumed $\gamma = 0.005$. Unless stated otherwise, this is the tolerance value that will be used in all following experiments. Figure 3.9 shows the execution time of this k-th predecessor algorithm for different values of k. It can be seen that an increasing value of k indeed speeds up the shortest path calculations of the longer paths. This time gain is the highest for $k = 3$ and less remarkable for higher values of k. However, higher values of k

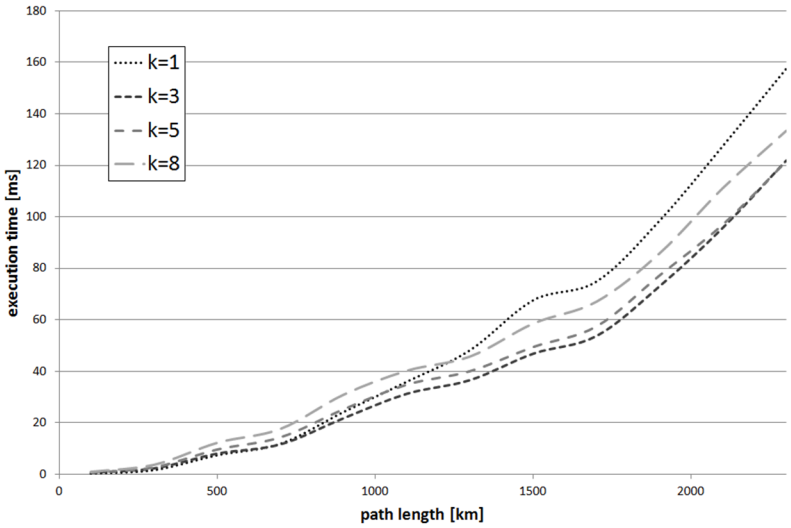

Figure 3.9: Execution times of the k-th predecessor algorithm in function of the optimal shortest path length for different values of k.

mean that more paths are found and that these paths differ less from the optimal ones. This is illustrated in table 3.5.

k	loss rate	avg. Δ
1	31,4%	2.84%
3	19.8%	1.69%
5	13.5%	1.25%
8	10.9%	0.94%

Table 3.5: Accuracy of k-th predecessor algorithm for different values of k.

3.4.2 The Accounting Algorithm

The accounting algorithm is similar to the predecessor algorithm, but besides ignoring unfavorable nodes based on local information, a history is kept of the path to allow more than one step in the wrong direction. In the basic version of the algorithm an account is given to each path. At initialization the account of a path contains a fixed amount of credits, called the starting budget (SB). This starting budget determines the number of allowed steps in the wrong direction. Figure

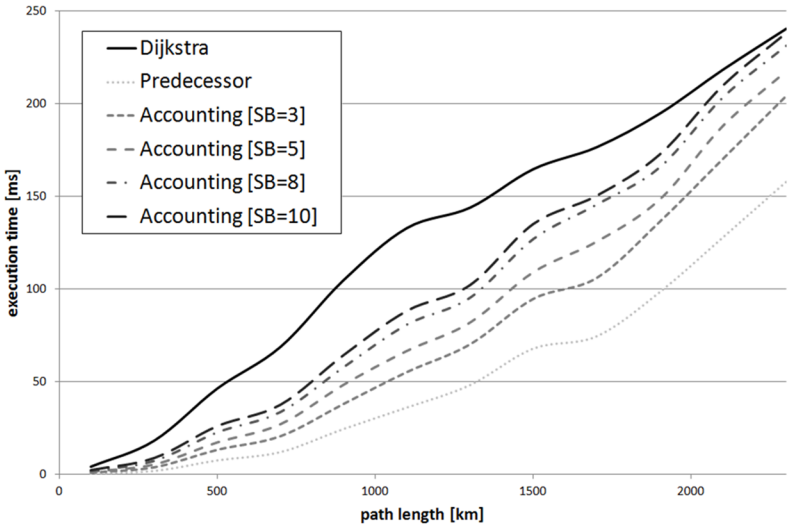

Figure 3.10: Execution times of the accounting algorithm in function of the optimal shortest path length for different starting budgets (SB).

3.10 shows the average execution time of the accounting algorithm with different starting budgets. Moreover, the execution times of the algorithm of Dijkstra and the predecessor algorithm are depicted for comparison. In the accounting and predecessor algorithm a tolerance factor with $\gamma = 0.005$ was used, as stated previously. It can be observed that the accounting heuristic performs worse than the predecessor heuristic, but nevertheless has lower execution times than the algorithm of Dijkstra. Moreover, the execution time increases with an increasing starting budget.

The main advantage of the accounting heuristic is the remarkable improvement of the accuracy of the results. This is shown in table 3.6, where loss rates are presented, together with the average difference (Δ) with the optimal path in the case that a path is found. It can be seen that the higher the starting budget is, the lower is the loss rate and the average difference.

In section 3.3.2, an improved version of the accounting algorithm was presented, in which steps in the wrong direction are punished, while steps in the right direction are rewarded. Furthermore, in order to avoid large detours a maximum budget was introduced that limits the number of steps in the wrong direction. Figure 3.11 shows the execution times of the accounting algorithm with a fixed starting budget of 5 (SB=5) and a variable maximum budget (MB). The accounting algorithm without a maximum budget is the algorithm that only punishes the wrong steps and does not reward right steps. These results show that the algorithm in

	loss rate	avg. Δ
Predecessor	31.4 %	2.84%
Accounting [SB = 3]	10.9%	2.38%
Accounting [SB = 5]	4.5%	1.42%
Accounting [SB = 8]	1.5%	0.61%
Accounting [SB = 10]	1.3%	0.16%

Table 3.6: Accuracy of accounting algorithm

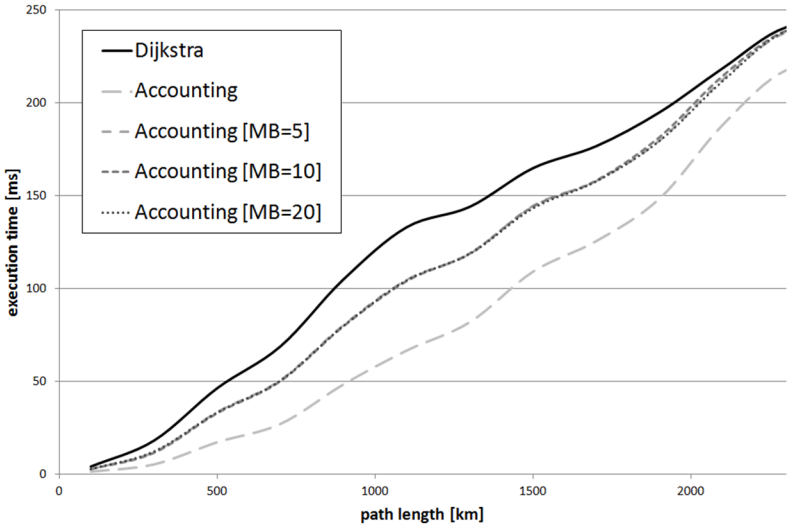

Figure 3.11: Execution times of the improved accounting algorithm (with SB=5) in function of the optimal shortest path length for different maximum budgets (MB).

which steps in the right direction are rewarded, is slower than the original accounting algorithm, but still performs better than the algorithm of Dijkstra. Moreover, it should be noted that increasing the maximum budget has no real impact on the execution times.

Again, the major advantage is the accuracy of the produced results. Table 3.7 shows both the loss rates and the the average differences of the found paths with the optimal ones for different starting and maximum budgets. It can be seen that both the loss rate and the average difference are close to zero. If the maximum budget is large enough this improved algorithm almost always finds the optimal path.

To determine whether a path is going in the right direction the estimated dis-

loss rate

	MB		
SB	5	10	20
3	0.7%	0.2%	0.1%
5	0.2%	0.2%	0.1%
8	0.2%	0.1%	0.1%
10	0.2%	0.1%	0.1%

average difference (Δ)

	MB		
SB	5	10	20
3	0.10%	0.05%	0.00%
5	0.06%	0.04%	0.00%
8	0.05%	0.02%	0.00%
10	0.03%	0.01%	0.00%

Table 3.7: Accuracy of the improved accounting algorithm

tance from the current mode may be compared to that from its k-th predecessor (in stead of the first predecessor). This results in an accounting algorithm with similar execution times as stated in this section. Moreover, an increasing k results in more paths to be found which also differ less from the optimal ones. As these results are similar to the results of the k-th predecessor algorithm, we will not discuss them here in detail.

3.4.3 Further Optimizations

Two optimizations, queue optimization and the feedback loop, were presented in section 3.3.3 that can be applied to both the predecessor and the accounting algorithm. In this section we will demonstrate the advantages of these optimizations applied to the predecessor algorithm. However, similar results are observed when applying them to the accounting algorithm.

3.4.3.1 Queue Optimization

Queue optimization (QO) is a technique in which the estimated distance from the current node (to the destination node) is compared to the estimated distance from the node that was investigated in the previous iteration. Figure 3.12 shows the average execution times for both the predecessor algorithm with and without queue optimization. Here the same tolerance factors were used as in figure 3.7. It is clear that queue optimization remarkably improves the performance of the algorithms with execution times of only a couple of milliseconds. Unfortunately, this speedup is accompanied with less accurate results. Table 3.8 shows that the predecessor algorithm with queue optimization only finds a small fraction of the paths and that most of these paths differ from the optimal ones. However, more accurate results are produced when a higher tolerance factor is used. Now the question arises whether using a (high) tolerance factor that produces accurate results, still has a reasonable execution time.

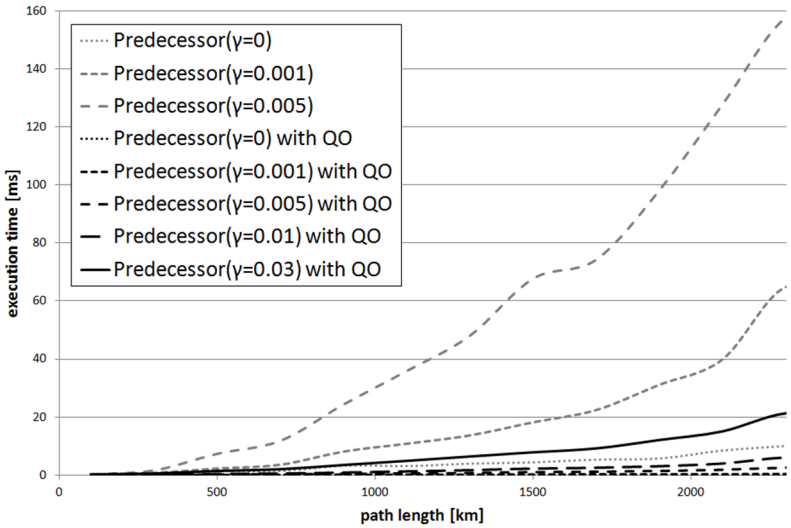

Figure 3.12: Execution time of the predecessor algorithm with and without queue optimization (QO) in function of the shortest path length.

γ	loss rate	avg. Δ
0	97.1%	15.69%
0.001	95.2%	12.66%
0.005	88.7%	11.27%
0.01	83.0%	7.00%
0.03	66.1%	6.03%

Table 3.8: Accuracy of the predecessor algorithm with queue optimization for different values of γ.

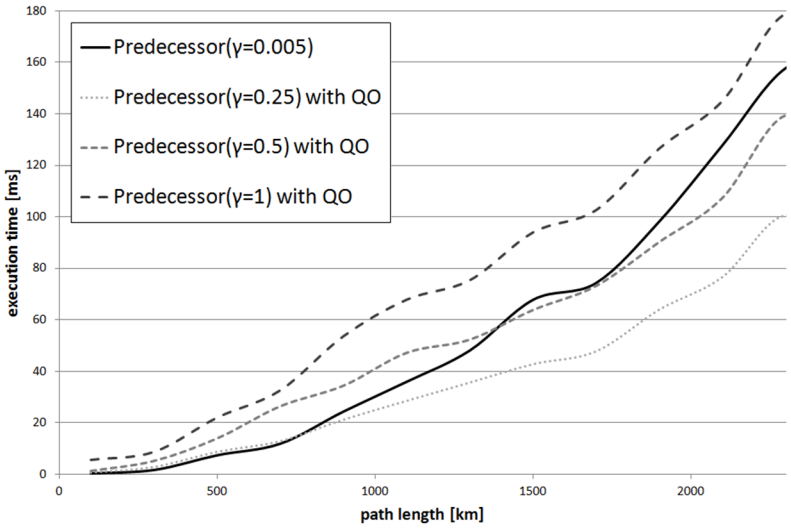

Figure 3.13: Execution time of the predecessor algorithm with queue optimization (QO) and higher tolerance factors in function of the shortest path length.

γ	loss rate	avg. Δ
0.25	15.4%	3.91%
0.50	4.3%	2.58%
1.00	1.1%	0.59%

Table 3.9: Accuracy of the predecessor algorithm with queue optimization for high values of γ.

Figure 3.13 shows the execution time of the predecessor algorithm with queue optimization in which the tolerance factors with $\gamma = 0.25$, $\gamma = 0.5$ and $\gamma = 1$ were used. The execution time of the predecessor algorithm without queue optimization and $\gamma = 0.005$ is also depicted for comparison. This demonstrates that high tolerance factors indeed still result in relatively low execution times. Moreover, more accurate results are produced. This is illustrated in table 3.9. For comparison, the predecessor algorithm without queue optimization and $\gamma = 0.005$ has a loss rate of 31.4% and the paths that are found differ averagely 2.84% from the optimal ones.

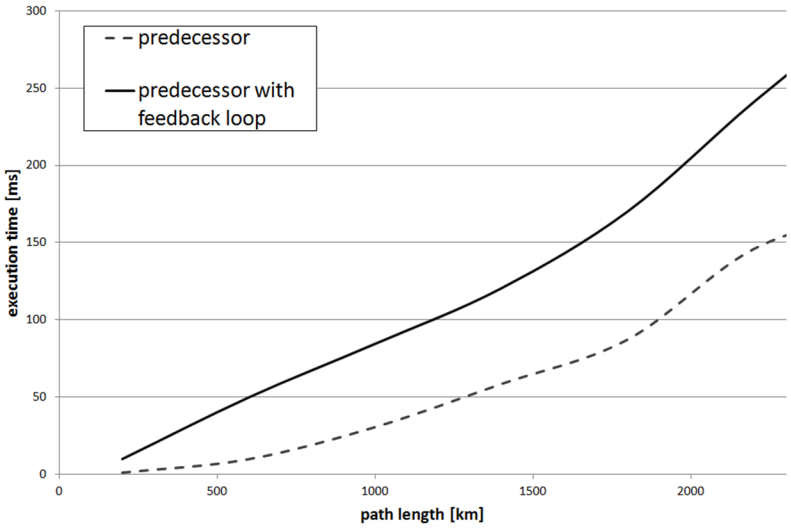

Figure 3.14: Execution time of the predecessor algorithm ($\gamma = 0.005$) with and without feedback loop in function of the shortest path length.

3.4.3.2 Feedback Loop

Next to queue optimization, the feedback loop technique was presented. In this optimization, the algorithm is repeated iteratively with an increasing tolerance factor, until a path between the origin and the destination has been found. Figure 3.14 shows the execution time of the predecessor algorithm with and without feedback loop in which a tolerance factor with $\gamma = 0.005$ was used. In these experiments, we opted to multiply γ with 2 when no path was found in the previous iteration. As expected, using the feedback loop technique results in slower algorithms. However, the execution time of the algorithm in which m iterations were executed, still is lower than m times the execution time of the algorithm without a feedback loop. The reason for this is that in each iteration a large amount of the data can be reused.

The main advantage of the feedback loop technique is that the loss rates are reduced to zero. It is guaranteed that the algorithm always finds a path between the origin and the destination. However, these paths may differ from the optimal ones. In our experiments, the paths found by the predecessor algorithm ($\gamma = 0.005$) with feedback loop are on average 5.47% longer than the optimal ones. This is higher than the percentage of the algorithm without feedback loop (see table 3.4) since this is an average over the paths that were not lost and these paths were found using the lowest possible tolerance factor. However, multiplying γ with a higher factor results in more accurate results. This is illustrated in table 3.10. It should be noted

that this has only a minor impact on the execution time, as this requires also less iterations.

multiplication factor	avg. Δ
2	5.47%
4	4.26%
8	3.45%
16	2.60%

Table 3.10: Accuracy of the predecessor algorithm ($\gamma = 0.005$) with feedback loop for different multiplication factors.

Next, the number of iterations needed in the feedback loop technique was investigated. As we applied a tolerance factor with $\gamma = 0.005$, in 31.4% of the cases more than one iteration is needed (see table 3.4). Only then applying the feedback loop is useful. We will thus limit our results to these cases. Table 3.11 shows for different multiplication factors the average number of iteration needed. As expected, a higher multiplication factor results in a lower amount of iterations, since with a higher tolerance factor the chance of finding a path is larger.

multiplication factor	avg. # iterations
2	3.37
4	2.48
8	2.22
16	2.12

Table 3.11: Average number of iterations needed by the feedback loop in the case no path was found in the first iteration.

3.5 Conclusions

In this chapter a number of novel goal-directed heuristics were presented, in order to speed up shortest path calculations in large transportation networks. While the predecessor algorithm uses local information to guide a path in the right direction, the accounting heuristic additionally keeps a history of this path. The main advantage of these algorithms, in comparison with the branch pruning algorithm is that they allow more freedom in the choice of the (distance) estimation function. The reason for this is that the actual path distances and the estimates are no longer combined and compared with each other.

In the experiments it is demonstrated that these novel heuristics, with the adjusted parameters, indeed perform better than the algorithm of Dijkstra and the branch pruning algorithm. Fine-tuning the parameters (e.g. the tolerance factor $(1 + \gamma)$, the value of k, the starting and maximum budget) is finding the correct balance between the performance of the algorithm and the accuracy of the produced results. Faster algorithms usually produce less accurate results, and vice versa.

A number of adaptations were presented in order to improve either the performance, the accuracy or both. When in the predecessor algorithm the areas around the origin and the destination nodes are investigated completely, the possible first and/or last mile detours in transportation networks are taken into account. This improves the accuracy of the produced results, while at the same time a slight improvement in the performance of the algorithm is observed.

Instead of comparing the estimated distance from the current node (to the destination) with the estimated distance from its predecessor, it may be compared to the estimated distance from its k-th predecessor. This improves the accuracy of the results, while maintaining the performance of the algorithm. This technique can be applied in both the predecessor and the accounting algorithm.

In the accounting algorithm, instead of only punishing steps in the wrong direction, also steps in the right direction may be rewarded. This results in slower algorithms, but nearly optimal results.

Furthermore, a technique called queue optimization was introduced. In this adaptation, nodes are not compared to their predecessors on the path, but to the nodes that were investigated in the previous iteration. Queue optimization improves the performance of the algorithms remarkably. Unfortunately, this comes at the cost of less accurate results. However, this may be overcome by using higher tolerance factors. This results in algorithms that produce more accurate results in a small amount of execution time.

Finally, the feedback loop was presented, which is guaranteed to always find a short path between the origin and the destination. By reusing the information of the previous iterations, the execution time of these algorithm may be kept relatively low. Both queue optimization and the feedback loop can be applied to the predecessor algorithm as well as to the accounting algorithm.

It should be noted that determining which algorithm and adaptations should be used, is dependent upon the desires of the users. If the user want to find a short path quickly, but it is not important that it is the optimal one, a predecessor algorithm with queue optimization should be applied. If the user, on the other hand wants to find the (nearly) optimal shortest path in a reasonable amount of time, then the best option is the accounting heuristic with a feedback loop.

To summarize this chapter, figure 3.15 gives a schematic overview of the strength of the heuristics and the adaptations/optimizations that were presented in this chap-

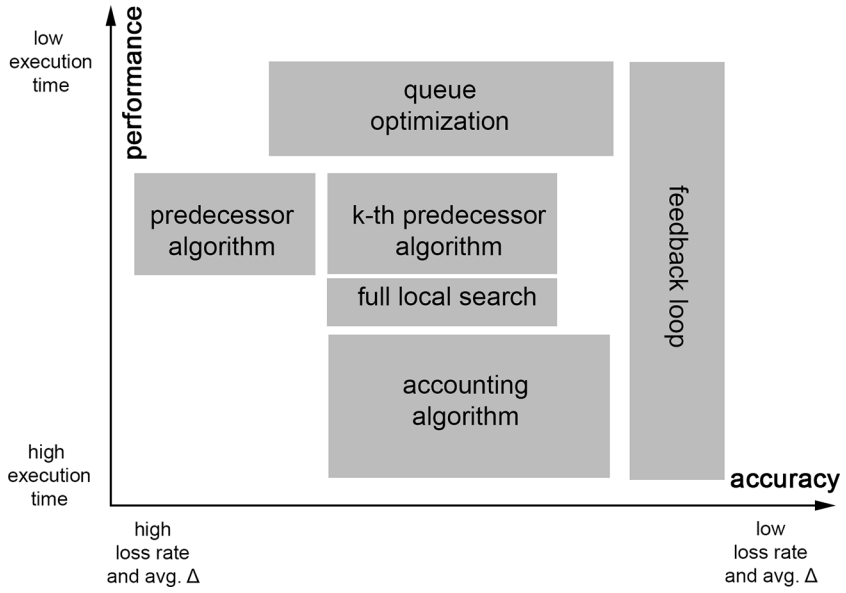

Figure 3.15: Schematic overview of the performance and accuracy of the presented algorithms and adaptations.

ter.

References

[1] L. Fu, D. Sun, and L. Rilett. *Heuristic shortest path algorithms for transportation applications: State of the art*. Computers & Operations Research, 33(11):3324 – 3343, 2006.

[2] I. Pohl. *Bi-directional search*. Machine Intelligence, 6:124–140, 1971.

[3] J. Bander and I. White, C.C. *A new route optimization algorithm for rapid decision support*. In Vehicle Navigation and Information Systems Conference, 1991, volume 2, pages 709 – 728, oct. 1991.

[4] P. Sanders and D. Schultes. *Engineering highway hierarchies*. In Proceedings of the 14th conference on Annual European Symposium - Volume 14, ESA'06, pages 804–816. Springer-Verlag, 2006.

[5] L. Fu, U. of Alberta. Dept. of Civil, and E. Engineering. *Real-time Vehicle Routing and Scheduling in Dynamic and Stochastic Traffic Networks*. University of Alberta, 1996.

[6] H. A. Karimi. *Real-Time Optimal Route Computation: a Heuristic Approach*. ITS Journal - Intelligent Transportation Systems Journal, 3(2):111–127, 1996.

[7] J. Lysgaard. *A two-phase shortest path algorithm for networks with node coordinates*. European Journal of Operational Research, 87(2):368 – 374, 1995.

[8] D. Wagner and T. Willhalm. *Speed-Up Techniques for Shortest-Path Computations*. In Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS '07), pages 23–36. Springer, 2007.

[9] R. Sedgewick and J. Vitter. *Shortest paths in euclidean graphs*. Algorithmica, 1:31–48, 1986.

[10] M. Müller-Hannemann and K. Weihe. *Pareto Shortest Paths is Often Feasible in Practice*. In Proceedings of the 5th International Workshop on Algorithm Engineering (WAE '01), pages 185–198. Springer-Verlag, 2001.

[11] P. Hart, N. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. Systems Science and Cybernetics, IEEE Transactions on, 4(2):100 –107, 1968.

[12] N. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[13] J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Publishing Company, 1984.

[14] P. Sanders and D. Schultes. *Engineering Fast Route Planning Algorithms*. In C. Demetrescu, editor, Experimental Algorithms, volume 4525 of *Lecture Notes in Computer Science*, pages 23–36. Springer Berlin Heidelberg, 2007.

[15] F. Schulz, D. Wagner, and K. Weihe. *Dijkstra's algorithm on-line: an empirical case study from public railroad transport*. Journal on Experimental Algorithmics, 5, 2000.

[16] E. Köhler, R. Möhring, and H. Schilling. *Acceleration of Shortest Path and Constrained Shortest Path Computation*. In S. Nikoletseas, editor, Experimental and Efficient Algorithms, volume 3503 of *Lecture Notes in Computer Science*, pages 126–138. Springer Berlin Heidelberg, 2005.

[17] U. Lauther. *An Extremely Fast, Exact Algorithm for Finding Shor test Paths in Static Networks with Geographical Background*. Geoinformation und Mobilität- von der Forschung zur praktischen Anwendung, 22:219–230, 2004.

[18] R. Möhring, H. Schilling, B. Schtz, D. Wagner, and T. Willhalm. *Partitioning Graphs to Speed Up Dijkstras Algorithm*. In S. Nikoletseas, editor, Experimental and Efficient Algorithms, volume 3503 of *Lecture Notes in Computer Science*, pages 189–202. Springer Berlin Heidelberg, 2005.

[19] R. Bauer and D. Delling. *SHARC: Fast and robust unidirectional routing*. Journal on Experimental Algorithmics, 14:4:2.4–4:2.29, 2010.

[20] A. V. Goldberg and C. Harrelson. *Computing the shortest path: A* search meets graph theory*. In Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '05), pages 156–165, 2005.

[21] A. Goldberg and W. R.F. *Computing Point-to-Point Shortest Paths from External Memory*. In Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX '05), pages 26,40, 2005.

[22] J. Maue, P. Sanders, and D. Matijevic. *Goal Directed Shortest Path Queries Using Precomputed Cluster Distances*. In C. lvarez and M. Serna, editors, Experimental Algorithms, volume 4007 of *Lecture Notes in Computer Science*, pages 316–327. Springer Berlin Heidelberg, 2006.

[23] E. W. Dijkstra. *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1):269–271, 1959.

[24] R. E. Korf. *Depth-first iterative-deepening: An optimal admissible tree search*. Artificial Intelligence, 27(1):97 – 109, 1985.

4

Speeding up Martins' algorithm for multiple objective shortest path problems

Users of routing applications often do not only want to find the shortest route, but also the fastest one and the one with the least transfers. These objectives may contradict each other and a route that is optimal for one objective (e.g. travel time) thus may not be optimal for another objective (e.g. number of transfers). This chapter deals with optimizing multiple objectives simultaneously, leading to a set of Pareto optimal solutions. The most widely known multiple objective shortest path algorithm is the one of Martins, which is based on the algorithm of Dijkstra. However, this algorithm tends to be too slow, especially in large networks such as transportation networks. In this chapter a number of speedup measures are investigated, resulting in new algorithms. It is shown that the calculation time can be reduced considerably. Moreover, it is mathematically proven that these algorithms still produce the Pareto optimal set of paths.

4.1 Introduction

This chapter presents two adaptations to a label setting algorithm, which was presented by Martins in 1984 [1], to solve the multiple objective shortest path problem. To date, this is still a reference work to solve the multiple objective shortest

path problem making use of graph algorithms. This section starts with a description of the problem tackled by this algorithm. Then, an overview is given of what can be found in literature about multiple objective shortest path problems and their solutions. Finally, the contributions and objectives of this research are explained in subsection 4.1.3.

4.1.1 Problem description

With the evolvement of GPS systems, routing in transportation networks gains more attention. One of the main characteristics of these transportation networks is that they tend to be large. For example the TIGER/Line network, covering the USA, consists of approximately 24 million nodes and 29 million links and the PTV Europe network contains approximately 19 million nodes and 23 million links [2]. In most applications, a preprocessing phase reduces the size of these networks, but even then, they are still extensive. On the other hand, these networks tend to be sparse with an average node degree between 2 and 3.

Moreover, in recent years, multimodal transportation (i.e. using multiple modes of transportation) becomes a valuable alternative to avoid road traffic jams. Multimodal networks are often represented by a layered network model in which each modus has its own transportation network and trans-shipments between the modes are modeled by trans-shipment links connecting nodes from different modes of transportation [3]. This results in even more extensive networks than the unimodal ones.

At the same time, logistic applications need to find the best route as soon as possible while taking into account multiple objectives. The transport needs to be both as fast, as cheap, as ecological, etc. as possible. The problem tackled in this chapter is how to calculate in a very short time the multiple objective shortest paths between an origin and a destination in a transportation network. As there are multiple objectives that need to be optimized, there is no single optimal path, but a set of paths that is called Pareto optimal. A path is called Pareto optimal if no objective can be ameliorated without deteriorating another objective (see section 2.1 for a more formal definition).

For this, we will further improve the MOSP (multiple objective shortest path) algorithm of Martins, which has proven to be an efficient way to find the complete Pareto optimal set of paths [4], especially for lower density networks such as transportation networks. Unfortunately, this algorithm, in its original form, has to examine the whole network to find the Pareto optimal set of routes. This is quite time consuming for large networks, such as (multimodal) transportation networks. Therefore, we will attempt to limit the search area of the algorithm, namely by formulating a stop condition to interrupt the search process and by searching the network bidirectionally (i.e. simultaneously from the origin and the destination).

In this chapter we assume that all objectives need to be minimized and all concepts are defined according to this assumption. Moreover, it should be noted that the research presented here assumes that all costs are non-negative and that there are no cycles with cost 0.

4.1.2 Literature overview

Multiple objective shortest path problems, which are known to be NP-hard [5], started to gain attention in the 70s and the beginning of the 80s with the works of Vincke [6] and Hansen [7], in which they focused on the case with two objectives, the bi-objective shortest path problem. Since then, several exact methods, for both the general multi-objective and the bi-objective shortest path problem, have been developed to find all Pareto optimal paths between two nodes in a network.

The most straightforward (and easiest) way to deal with multiple objectives is by using single objective shortest path techniques with a weighted objective function. The shortest paths are optimized for a single objective that is the weighted sum of the different objectives ($\alpha_1 c_1 + \ldots + \alpha_2 c_2$). Unfortunately, this method only finds a subset of all Pareto-optimal paths. If we present all Pareto-optimal paths in a decision space, the paths found by this algorithm are situated on the convex hull. The other Pareto-optimal paths are situated in the so-called duality gaps. This is illustrated in figure 4.1. References [8] and [9] provide a more detailed discussion on this subject.

Exact methods to find the set of Pareto-optimal paths for the shortest path problem with two objectives have been thoroughly studied. As we focus in this article on the general multi-objective shortest path algorithm, we would just like to refer to the works of Skriver [10] and Raith and Ehrgott [9] for an overview of most of the bi-objective shortest path algorithms.

In [11], Clímaco and Pascoal give an overview to solve the multi-objective shortest path problems. According to the taxonomy presented in their article, the research presented in this paper can be categorized in the class of multicriteria path problems with additive metrics and a-posteriori aggregation of preferences method. In this class, two major categories can be distinguished: ranking techniques and labeling techniques.

Ranking techniques, like for example [12] and [13], use a ranking method for listing the paths in a decreasing order according to one of the objectives. Subsequently, from this list of paths a set of non-dominated paths is determined. Three groups of path ranking methods can be determined: deletion algorithms ([14], [15], [16]) in which at the start of each iteration the path found in the previous iteration is eliminated from the network, labeling algorithms ([17], [18], [19]) in which nodes may contain K labels (for the K paths) and that resemble the K shortest path algorithms, and deviation algorithms ([20], [21], [22]) in which a

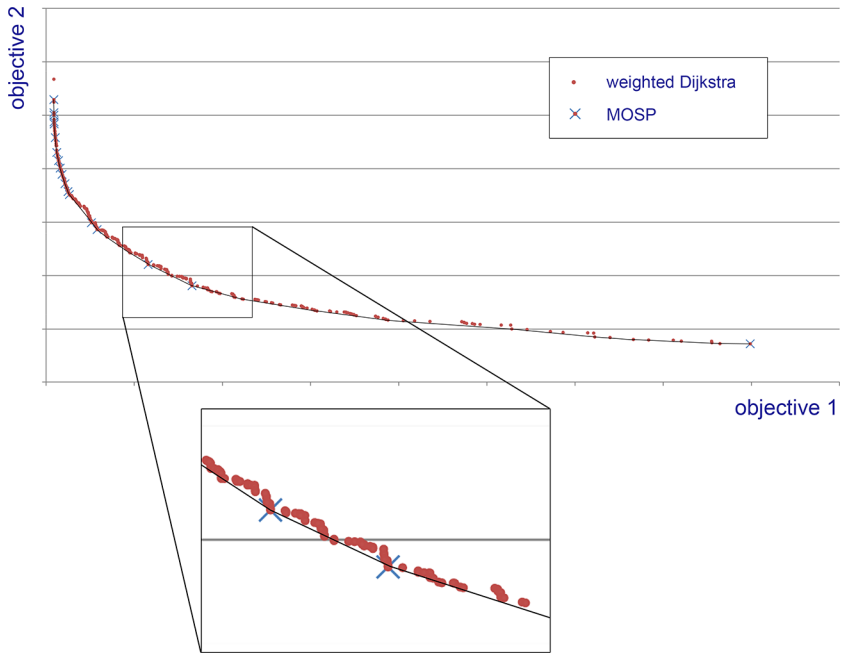

Figure 4.1: The bi-objective shortest path costs calculated by the weighted Dijkstra algorithm and by the multiple objective shortest path (MOSP) algorithm of Martins. This experiment was carried out on the Belgian road network (Transport(B) in the table 4.2) with random and completely uncorrelated objectives.
While the MOSP algorithm finds 363 unique paths, the weighted algorithm of Dijkstra only finds 18. Furthermore, these paths are situated on the convex hull (i.e. the black line) of the Pareto optimal paths.

new path is considered to be a deviation from the previous path.

Labeling techniques are well known to solve the single objective shortest path problem ([23], [24]). Similar to the single objective case, two categories can be distinguished: label-setting algorithms and label-correcting algorithms. An exact label-setting algorithm for the bi-objective shortest path problem was proposed by Hansen [7]. This work was then generalized by Martins for the multiple objective shortest path problem [1]. To this date, this article is still a reference work for the multi-objective shortest path algorithms.Next to label-setting algorithms, a number of label-correcting algorithms were presented. Similarly, this research started with bicriteria shortest path problems ([6], [25], [26]) and was then generalized for multiple criteria [27]. Guerriero and Musmanno studied the impact of label selection versus node selection in label-correcting algorithms [28]. For a detailed theoretical study of the properties of label correcting and label setting algorithm to solve the multi-objective shortest path problem, we would like to refer to [29].

Most of the research described above assumes that all objectives are additive, but also other objective types can be used. One of the most popular is a combination of both additive objectives and objectives of the bottleneck type (i.e. minmax, maxmin). Gandibleux et al. [30] generalized the algorithm of Martins to the case in which one of the objectives is a bottleneck. In [31] and [32] an algorithm is presented to solve the tricriteria shortest path problem in which at least two objectives are of the bottleneck type. A twofold extension to this latter research has been proposed by Bornstein et al. [33]. As we will focus on additive objectives in this research, we will not elaborate further on this.

Next to these exact methods, a number of well known heuristics have been applied to the multi-objective shortest path problem. In [34], for example, evolutionary algorithms are used to find a subset of the Pareto optimal paths.

A complete overview of multiple objective shortest path algorithms, both for bicriteria and multicriteria problems can be found in [35]. In this chapter we will focus on the exact label setting algorithm of Martins [1] with additive objectives, and propose a number of adaptations to speed up the calculation in the case where we want to find all non-dominated paths between a single origin node and a single destination node. In the following subsection more details are given on the objectives and the contributions that are made in this research.

4.1.3 Contributions and objectives

Our research focuses on exact and fast algorithms to solve multiple objective shortest path problems with n objectives, which are all sum problems. This problem can be described mathematically as

$$min_{p \in P_{o,d}}(z_1(p), z_2(p), ..., z_n(p)) \tag{4.1}$$

in which $P_{o,d}$ represents the set of all paths between an origin node o and a destination node d, and $z_i(p)$ is the cost of path p with respect to objective i. While some of the algorithms presented in literature are custom made for the bicriterion problem ([10], [9]), we opted not to limit the number of objectives (like for example in [28] and [27]). Moreover, we want to speed up the calculations without losing the guarantee of finding all Pareto optimal paths, in contrast with for example the heuristic search algorithm of [34].

This chapter focuses on two exact speedup techniques, namely a well-defined stop condition and searching the network bidirectionally. It is shown (both theoretically and experimentally) that the two proposed improvements achieve a considerable speedup compared to other exact graph algorithms, while at the same time maintaining the optimality of the solution.

In the first improvement a speedup is achieved by stopping the search as soon as all Pareto optimal paths between the origin and the destination are found, instead of searching the whole network. The second one searches the network from both the origin and the destination simultaneously, resulting in a speedup factor of more than 2 (where the bidirectional algorithm of Dijkstra achieves a speedup factor of around 2). This is clarified experimentally in section 4.5.

It should be noted that the algorithm and the proposed improvements require that all costs are non-negative and that the networks contain no zero cost cycles.

4.2 Unidirectional multiple objective shortest path algorithm

In order to better understand the remainder of the chapter, the unidirectional algorithm from which we started, is presented briefly in this section, together with a well-defined stop condition that finishes the search process as soon as all Pareto optimal paths are found. The first subsection discusses some basic (mathematical) concepts that are used by the algorithm. Subsequently, in the second subsection, the multiple objective shortest path algorithm is presented. In order to avoid investigating the whole network, as the basic version of this algorithm does, a speedup can be achieved by making use of an efficient stop condition which is presented in subsection 4.2.3, together with a proof of the correctness of this stop condition.

4.2.1 Basic concepts

A network is defined as a graph $G(V, E)$ with V the set of nodes and E the set of directed links. Each link (u, v), connecting node u and node v, has assigned to it a number of values representing the costs for the different objectives, which is depicted by the vector $c(u, v) = [c_1(u, v), c_2(u, v), \ldots, c_n(u, v)]$ (see section

2.4.2 of chapter 2). In this chapter we will assume that no parallel links are allowed. Nevertheless, parallel links would have no impact on the operation of the algorithms presented here.

A path or a route in a graph is defined as a vector with the costs of the path for each objective together with an ordered list of nodes, for which holds that each pair of consecutive nodes is connected by a link. It can be represented as $[z(p), p]$ with $z(p) = [z_1(p), z_2(p), \ldots, z_n(p)]$ the cost vector and $p = [v_1, \ldots, v_k]$ the ordered list of nodes with $\forall i \in \{1, \ldots, k-1\} : (v_i, v_{i+1}) \in E$. Here, the path contains k nodes and thus $k-1$ links.

Similarly to the algorithm of Dijkstra [23], the algorithms presented in this chapter make use of labels to indicate the 'distance' to a certain node. A label contains a cost value for every objective and a reference to a previous label. This way, a label can be represented as $l(v) = [c(v), P(l(v))]$ with v the node to which this label is assigned, $c(v) = [c_1(v), c_2(v), \ldots, c_n(v)]$ the value vector and $P(l(v))$ the reference to the previous label. The reference to the previous label enables reconstructing the path afterwards by a backtracking mechanism. It should be noticed that, this way, from every label a path can be constructed. While in the algorithm of Dijkstra paths are reconstructed by backtracking nodes, here this happens at label level.

In order to compare labels with each other, two relationships need to be defined on the cost vectors: dominance and ordering.

We will first define the dominance relationship between vectors in order to come to a definition of Pareto optimality, as the result of the MOSP algorithm is a Pareto optimal set.

Definition 1 (dominance)
The vector $[a_1, \ldots, a_n]$ dominates the vector $[b_1, \ldots, b_n]$ if and only if

$$(\forall i \in \{1, \ldots, n\} : a_i \leq b_i) \wedge (\exists i \in \{1, \ldots, n\} : a_i < b_i) \tag{4.2}$$

A path P_1 dominates another path P_2 if and only if the cost vector of P_1 dominates the one of P_2.
A label L_1 dominates another label L_2 if and only if the vector of L_1 dominates the one of L_2.

Definition 2 (Pareto optimality)
A path is Pareto optimal if and only if there exists no feasible path which is better in one of the objectives and not worse in all the other objectives.
A set of paths S is called Pareto optimal if and only if none of the elements of S is dominated by another feasible path.

So, for example, if we have the paths A, B and C with cost vectors $\bar{a} =$

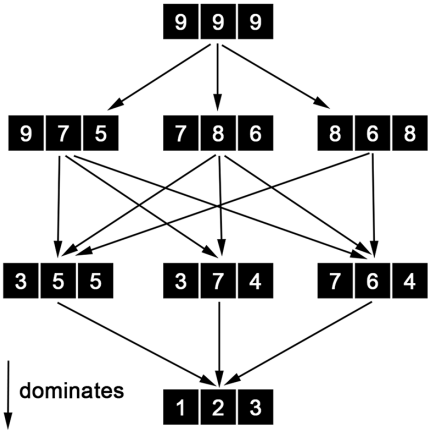

Figure 4.2: Example of a lattice, in which all objectives are maximized. The vectors at the same level form a Pareto optimal set. For example, $[9, 7, 5]$*,* $[7, 8, 6]$ *and* $[8, 6, 8]$ *form a Pareto optimal set. Moreover, the joins and the meets can be deduced from this lattice. For example, a possible meet of* $[3, 5, 5]$ *and* $[7, 6, 4]$ *is the vector* $[7, 8, 6]$*, while* $[1, 2, 3]$ *is a join.*

$[5, 9, 3]$, $\bar{b} = [2, 1, 3]$ and $\bar{c} = [5, 3, 2]$ respectively, then A is dominated by both B and C. The paths B and C, on the other hand, form a set of Pareto optimal paths within the set $\{A, B, C\}$.

This relationship (i.e. vector x dominates vector y) can be represented graphically as a lattice where every element is dominated by all elements higher in the lattice and dominates all elements lower. Elements at the same level in the lattice form a Pareto optimal set. For every two elements in this tree there is a least upper bound, called the join, and a least lower bound, called a meet, which means that this relationship is a partial ordering. Furthermore, the dominance relationship is transitive, but not reflexive nor symmetric or anti-symmetric. An example of a lattice is given in figure 4.2. Not all dominance arrows are depicted, but they follow from the transitive character of this relationship.

At certain points in the algorithm, the smallest/largest label needs to be determined, so an ordering needs to be defined. We opted for a lexicographical ordering according to following definition.

Definition 3 (lexicographic ordering)
The vector $[a_1, \ldots, a_n]$ is lexicographically smaller than (denoted by $<_l$) the vector $[b_1, \ldots, b_n]$ if and only if

$$\exists k \in \{1, \ldots, n\} : (\forall i < k : a_i = b_i) \wedge (a_k < b_k) \tag{4.3}$$

A label L_1 is lexicographically smaller than a label L_2 if and only if the vector of L_1 is lexicographically smaller than the one of L_2

Now, if we want to order the vectors a, b and c, as presented previously, from small to large then we get the sequence $b <_l c <_l a$.

This relationship is a total ordering relation, as it is reflexive, anti-symmetric and transitive and two elements can always be compared to each other.

The lexicographic ordering is not the best ordering for multi-objective shortest path algorithms. In [36] it is shown that a weighted sum aggregate ordering results in a remarkable reduction of the calculation time. Nevertheless, we opted for a lexicographic ordering as it is easy to incorporate and the aggregate ordering requires a normalization of the objectives (and according fine tuning of the parameters) which may differ from network to network.

4.2.2 The basic algorithm

The basic algorithm is based on the multiple objective shortest path algorithm of Martins [1], which is, in turn, based on the algorithm of Dijkstra [23]. Instead of only one label, nodes will now contain a set of labels. Moreover, this set only contains non-dominated labels. So, when adding a new label to a node, this label is only added when it is not dominated by one of the existing labels. Moreover, labels that are dominated by the new label are removed from the set. Below the pseudo code of the algorithm is given, which calculates for every node of the network the set of Pareto optimal paths from a specific origin o. An explanation of the expressions used in the pseudo code is given afterwards.

Algorithm 1

```
1    forall(v ∈ V)
2        L(v) := EMPTY_SET;
3    l(o) := [[0, ..., 0], NULL];
4    L(o).add(l(o));
5    T := {l(o)};
6    while(!(T is empty)){
7        l(u) := T.removeMin_l();
8        forall(m:  neighbor(u)){
```

```
9             l_new := [[c_i(u)+c_i(u,m)],l(u)];
10            if(∄ l(m) ∈ L(m):  l(m) dominates l_new){
11               m.addLabel(l_new);
12               T.add(l_new);
13            }
14        }
15    }
```

Explanation of the pseudo code

- **T**: the set of temporary labels
- **l(v) = [[**$c_1(v)$**,...,**$c_n(v)$**],<label>]**:
  the label of node v represented by a vector of values and the previous label
- **L(v)**: the set of labels of node v
- **T.removeMin**$_l$**()**:
  removes the minimum label from T according to the lexicographic ordering
- **getLinkBetween(<node1>,<node2>)**: returns the link from node1 to node2
- **[**a_i**+**b_i**]**: the pointwise sum of a and b.
  This is a short notation for $[a_1 + b_1, \ldots, a_n + b_n]$

Two phases can be distinguished: the initialization and the search phase. At initialization all nodes have empty label sets, except for the origin node which contains the null label, i.e. a label containing the null vector and a null reference. This label is added to the temporary set T, which now contains labels instead of nodes. The algorithm runs until this temporary set is empty. In every iteration the minimum label, according to the lexicographic ordering relationship, is determined and new labels are constructed for the neighbors of its owner. Here, $[c_i(u) + c_i(u, m)]$ denotes a vector which represents the pointwise sum of the vector in the label $l(u)$ and the vector representing the cost of the link (u, m). If not dominated by any other, these new labels are added to both the corresponding neighbor and the temporary set. Adding a new label to a node means adding it to its set of labels and removing from this set the labels that are dominated by this new label. The pseudo code of this function (applied to a node v) is depicted below.

addLabel(l_new)

```
1    forall(l(v) ∈ L(v)){
2        if(l_new dominates l(v))
3            L(v).remove(l(v));
4    L(v).add(l_new);
```

The main difference of this algorithm with the algorithm of Dijkstra is that this algorithm works at label level instead of node level. The temporary set contains labels and instead of changing the label of a node, the Pareto optimal set of a node is altered by adding the new label and removing the dominated ones. Moreover, while in the algorithm of Dijkstra, to reconstruct the paths, labels have pointers to the predecessor node on the path, in the multiple objective algorithm these pointers point to other labels, as nodes may have more than one label.

4.2.3 Stop condition

The algorithm presented in section 4.2.2 investigates the whole network in order to find a set of Pareto optimal paths from one node to all others. In this research, we are interested in all Pareto optimal paths between a single origin and a single destination node. Investigating the whole network then is superfluous as the Pareto optimal set is often already found before the algorithm terminated. In this section we present a stop condition, which is similar to the one in [37], [38] and [39], also known as 'dominance by early results' or multi-objective A*. The basic idea is that labels that are dominated by one of the labels of the destination node can never lead to a Pareto optimal path, and thus are not worth investigating further. This follows from the fact that all costs are non-negative and the objectives are of the additive type.

In contrast with the multi-objective A* algorithm, we opted not to check this condition for every label when removed from the temporary set, but to keep track of the minimum values for each objective of all labels present in the temporary set and check whether this vector is dominated by one of the destination labels. As the vector with the minimum values only differs little in each iteration, the number of comparison operations needed to check the dominance can be limited. This means that labels which are dominated by one of the result labels (i.e. the labels of the destination node) may still be investigated, but this does not outweigh the smaller number of comparison operations.

If we assume that $min(T) = [min_i(T)]$ represents the pointwise minimum of every objective value of all labels in the temporary set T, this results in the vector $[min_1(T), min_2(T), \ldots, min_n(T)]$. It can be shown that once this minimum vector is dominated by any of the destination labels, the search can be finished.

Stop Condition 1 (unidirectional s-t algorithm)
The search process can be stopped as soon as $[min_i(T)]$ is dominated by at least one of the destination labels $l(d) \in L(destination)$.

This can be proven (by contradiction) as follows.

Proof. Assume that, once the stop condition holds, there still exists a label $l(v)$ with vector $[c_i(v)]$ which is an element of the temporary set T and which is not dominated by any of the destination labels. As this label is part of T:

$$c_i(v) \geq min_i(T), \forall i \tag{4.4}$$

The stop condition says that $[min_1(T), \ldots, min_n(T)]$ is dominated by at least one of the destination labels $l(d) = [[c_i(d)], P(l(d))] \in L(destination)$, which means that

$$min_i(T) \geq c_i(d), \forall i \tag{4.5}$$

with at least one j for which

$$min_j(T) > c_j(d) \tag{4.6}$$

This results in

$$c_i(v) \geq c_i(d), \forall i \tag{4.7}$$

with at least one j for which

$$c_j(v) > c_j(d) \tag{4.8}$$

According to the definition of dominance, this means that $l(v)$ is dominated by $l(d)$, which means that our initial assumption is false and that there exist no non-dominated labels in T. □

Applying this stop condition to the basic algorithm can easily be done by substituting the line '`while(!(T is empty))`' with '`while(!(T is empty) & (∄ l(d) ∈ L(destination): l(d) dominates min(T)))`'

4.3 Bidirectional multiple objective shortest path algorithm

As searching the network bidirectionally has been proven useful to speed up algorithms (see section 1.3.2.2 of chapter 1), in this section a bidirectional multiple objective shortest path algorithm is presented, together with the proof that this algorithm always returns the Pareto optimal set of paths.

4.3.1 The algorithm

Multiple possibilities exist to speed up shortest path queries [40], like goal-directed search [41], bidirectional search [42], hierarchical search [43], etc. Some of them have already been applied to the multi-objective shortest path problem, like for example the goal-directed (A*) algorithm of [37]. We opted to investigate the

bidirectional speedup technique, as, with a well defined stop condition, this technique is able to guarantee still finding the optimal path for single objective shortest path problems. The basic idea of a bidirectional shortest path algorithm is that the search procedure is divided into two separate procedures: a forward search starting from the origin node and a backward search starting from the destination node.

There are two factors that influence the efficiency of bidirectional shortest path algorithms, namely the alternation between the forward and the backward search and the stop condition, of which the former is less critical than the latter. One can decide to alternate equally between the two searches or to pick the search direction with the smallest label. In this research we opted to alternate equally, but this can easily be changed.

The stop condition determines when it is guaranteed that the optimal path has been found, and thus when both searches can be aborted. For the single objective shortest path search, it has been proven [42] that the shortest path is found once the cost of the shortest path found so far (i.e. the minimum sum of the forward and the backward label of the same node) is smaller than or equal to the sum of the minimum labels of the forward and the backward temporary set. It should be noted that the performance of the bidirectional shortest path algorithm (with the stop condition as mentioned before) is dependent on the network configuration. Examples exist of networks in which the two searches proceed in different directions, resulting in two nearly completely unidirectional search trees. Nevertheless, for most planar and evenly distributed networks (i.e. networks in which the link costs are of the same magnitude), like the networks we are using in our research, a speedup of around 2 can be achieved.

The bidirectional multiple objective shortest path algorithm starts two search processes simultaneously: one from the origin and one from the destination. These search processes are similar to the search process of the unidirectional algorithm, in which new labels are constructed for the neighbors of the owner of the investigated label. These new labels are then added to the (forward or backward) label set of the specific nodes, if not dominated by any of the existing labels. Moreover, labels, which are dominated by this new label, are removed from this set. It should be noted that a node v now contains two (Pareto optimal) label sets: one for the forward search $L_F(v)$ and one for the backward search $L_B(v)$. When a new label is added to one of the label sets of a node and the other label set is not empty (i.e. this node has been visited in the other direction), this new label is combined with all labels of the other label set in order to form paths between the origin and the destination. These paths are then added to the temporary Pareto optimal solution set $L(result)$. It should be noted that a path is represented by a cost vector, together with a description of the path itself (i.e. a list of nodes/links). However, in order to save memory space, in the implementation only the cost vector together with the forward and the backward label, that were combined, are stored in the

solution set. As each label contains a reference to its predecessor label, the path can then be constructed from these two stored labels.

When all Pareto optimal paths are found both searches should be aborted. Therefore, a stop condition needs to be defined which guarantees that all Pareto optimal paths have been found. For this, we use the pointwise minima (defined as in section 4.2.3) $[min_i(T_F)]$ and $[min_i(T_B)]$ of the forward and the backward temporary set respectively. A new label will be constructed which contains the pointwise sum of these minima $[min_i(T_F) + min_i(T_B)]$ and this label will be compared to the paths of the solution set ($p \in L(result)$). In the next section, it will be proven that the following stop condition guarantees that all feasible Pareto optimal paths have been found.

Stop Condition 2 (bidirectional s-t algorithm)
The forward and the backward search can be aborted when
$[min_i(T_F) + min_i(T_B)]$ is dominated by any of the result paths $p \in L(result)$

Below, pseudo code of the bidirectional multiple objective algorithm is given. Expressions which have not been used in algorithm 1 are explained afterwards.

Algorithm 2

```
1    forall(v ∈ V){
2        L_F(v) := EMPTY_SET;
3        L_B(v) := EMPTY_SET;
4    }
5    L(result) := EMPTY_SET;

6    l^F(o) := [[0, ..., 0], NULL];
7    L_F(o).add(l^F(o));
8    T_F := {l^F(o)};

6    l^B(d) := [[0, ..., 0], NULL];
7    L_B(d).add(l^B(d));
8    T_B := {l^B(d)};

9    while(∄ p ∈ L(result):
                p dominates [min_i(T_F) + min_i(T_B)]){
10       d := getDirection();
11       l(u) := T_d.removeMin();
12       forall(m:  neighbor(u)){
13           l_new := l_new := [[c_i(u)+c_i(u,m)],l(u)];
14           if(∄ l^d(m) ∈ L_d(m):  l^d(m) dominates l_new){
```

```
15              m.addLabel(l_new);
16              T_d.add(l_new);
17              if(!(L_d'(m) is empty)){
18                   results := combine(l_new, L_d'(m));
19                   L(result).addResults(results);
20              }
21          }
22      }
23  }
```

Explanation of the pseudo code

- T_d: the set of temporary labels in direction `d`
- L_d**`(v)`**: the set of labels of node `v` in direction `d`
- **`L(result)`**: the set of Pareto optimal (result) paths
- **`getDirection()`**: determines in which direction the iteration of the algorithm will take place
- **`d'`**: the opposite direction of `d`
- **`combine(<label>, <set of labels>)`**: combines the label with each of the elements of the set to form a set of paths

First, the forward and backward label sets of all nodes as well as the result set `L(result)` are initialized with an empty set. These are all Pareto optimal sets, which means that labels/paths can be added only if they are not dominated by any of the existing labels/paths and that existing labels/paths which are dominated by the newly added label/path have to be removed from the set. Subsequently, a null label, i.e. a label containing the null vector and a null reference, is assigned to both the origin and the destination node. The origin label is added to the forward temporary set T_F, while the destination label is added to the backward set T_B. The temporary sets are ordered according to the lexicographic ordering relationship, similar to the unidirectional algorithm.

After initialization, the following steps are repeated until the stop condition (i.e. stop condition 2) is met. First, a direction is determined. Multiple possibilities exist to do this, like for example choosing the direction with the largest/smallest temporary set or alternating between the forward and the backward search. We opted for the latter one in order not to favor one of the directions. Then, the smallest label (according to a lexicographic ordering) is taken from the temporary set of the specific direction to be investigated. This means that all neighbors of the owner of this label are determined and new labels are constructed from the investigated

label and the cost vectors of the links between this owner and its neighbors. Every new label is then added to the specific (forward/backward) label set of the specific neighbor, if it is not dominated by any of the existing labels of this set. When added successfully, all existing labels which are dominated by the new label are removed from the set and the new label is added to the temporary set of the specific direction. Similarly to the unidirectional algorithm this happens in the `addLabel` function of which the pseudo code is depicted below. Here a label `l_new` is added to node `v` in the direction `d`.

addLabel(l_new,d)

```
1    forall(l^d(v) ∈ L_d(v)){
2        if(l_new dominates l^d(v))
3            L_d(v).remove(l^d(v));
4    L_d(v).add(l_new);
```

Moreover, if this neighbor contains labels in the other direction d' (i.e. $L_{d'}(m)$), this new label is combined with each of the labels of the label set of the other direction in order to form new paths. These new paths are then added to the solution set `L(result)`. It should be noted that this solution set is Pareto optimal, meaning that if the new paths are dominated by one of the existing paths, they will not be added, and that, if they are added, all existing paths dominated by them will be eliminated. The pseudo code of the `addPaths` function, applied the result set `L(result)` is given below.

L(result).addPaths(paths)

```
1    forall(p ∈ paths){
2        if(∄ q ∈ L(result):  q dominates p){
3            forall(r ∈ L(result)){
4                if(P dominates r)
5                    L(result).remove(r);
6            }
7        }
8    }
```

This is repeated until the stop condition is met, which means that the label containing the pointwise sum of the pointwise minima of the temporary sets is dominated by at least one of the resulting paths. These minima of the temporary sets are updated every time a new label is added to the set and every time a label is removed from it (when this label is investigated or when a temporary label is removed from a label set of a node).

This algorithm is guaranteed to find all Pareto optimal paths between the origin and the destination, as will be proven in the next section.

4.3.2 Proof of the optimality

In this section we will prove the correctness of following theorem.

Theorem 1 (optimality bidirectional MOSP algorithm)
All Pareto optimal paths are found once the stop condition holds for the bidirectional multiple objective shortest path algorithm.

Proof. The bidirectional algorithm searches the network simultaneously from the origin and the destination, similar to the search process of the unidirectional algorithm. When a node contains both forward and backward labels, these labels are combined to form paths, which are then added to the solution set, if Pareto optimal. Once the stop condition holds, all forward labels $l^F(v) \in L_F(v)$ for which

$$\exists i : c_i^F(v) < min_i(T_F) \tag{4.9}$$

and all backward labels $l^B(v) \in L_B(v)$ for which

$$\exists i : c_i^B(v) < min_i(T_B) \tag{4.10}$$

have been investigated, i.e. they are permanent labels.
This means that for all forward labels $l^F(u)$ which have not been investigated yet holds that

$$\forall i : c_i^F(u) \geq min_i(T_F) \tag{4.11}$$

and for all uninvestigated backward labels $l^B(u)$

$$\forall i : c_i^B(u) \geq min_i(T_B) \tag{4.12}$$

The stop condition says that the pointwise sum of the pointwise minima of the temporary sets is dominated by at least one path $p = [[p_i], P_p]$ of the Pareto optimal solution set $L(result)$, or

$$(\forall i : min_i(T_F) + min_i(T_B) \geq p_i) \wedge (\exists i : min_i(T_F) + min_i(T_B) > p_i) \tag{4.13}$$

Let us now assume that there still exists a path $q = [[q_i], P_q]$, which has not been found yet, but should be an element of the solution set. This means that this path is not dominated by any of the result paths $p = [[p_i], P_p], p \in L(result)$. According to the dominance definition, this implies that

$$\exists i : q_i < p_i \tag{4.14}$$

From the operation of bidirectional algorithm we know that this path q is constructed by combining two labels of the same node w, namely $q^F(w)$ and $q^B(w)$, thus

$$\forall i : q_i = q_i^F(w) + q_i^B(w) \tag{4.15}$$

Combining equations (4.13), (4.14) and (4.15) results in

$$\exists i : q_i^F(w) + q_i^B(w) < p_i \leq min_i(T_F) + min_i(T_B) \tag{4.16}$$

or

$$\exists i : q_i^F(w) + q_i^B(w) < min_i(T_F) + min_i(T_B) \tag{4.17}$$

We will now demonstrate that this path q does not exist or that all paths for which condition (4.17) holds, have already been found by the algorithm. As path q has not been found yet, this means that either $\forall i : q_i^F(w) \geq min_i(T_F)$ or $\forall i : q_i^B(w) \geq min_i(T_B)$ or both. The latter case is in contradiction with comparison (4.17) and thus can never lead to a non-dominated path of the Pareto optimal solution set. Two possible situations remain, namely $\forall i : c_i^F \geq min_i(T_F) \wedge (\exists i : c_i^B < min_i(T_B) \vee \forall i : c_i^B = min_i(T_B))$ and $(\exists i : c_i^F < min_i(T_F) \vee \forall i : c_i^F = min_i(T_F)) \wedge \forall i : c_i^B \geq min_i(T_B)$. Due to the symmetric character of the bidirectional algorithm we will only discuss one of these cases. The other one then can be deduced analogously. Let us assume that

$$(\exists i : c_i^F < min_i(T_F) \vee \forall i : c_i^F = min_i(T_F)) \wedge \forall i : c_i^B \geq min_i(T_B) \tag{4.18}$$

This means that the backward label has not been investigated yet, while the forward label is already made permanent. The investigated forward label implies that all neighboring labels are already present in the network (some of them permanent, others not). If the path q has not been found yet by the algorithm, this means that the label $q^B(w)$ is not yet present in the network.

We will now look at the predecessor label $q^B(w')$ of this backward label $q^B(w)$. Let w' be the owner of this label $q^B(w')$, while w denotes the owner of the labels $q^F(w)$ and $q^B(w)$. Since $q^B(w)$ and $q^B(w')$ are neighboring labels, the nodes w and w' are connected. Moreover, $q^F(w)$ is a permanent label, which implies that its neighboring labels are already present in the network. From this, we can deduce that w' contains a forward label $q^F(w')$ which has $q^F(w)$ as predecessor label. All 'labels' can be divided into three subsets (areas) for both the forward and the backward search: labels which have been investigated (i.e. permanent labels), labels which are added to a node but have not been investigated yet (i.e. temporary labels) and labels which have not been found yet by the algorithm. We know that $q^F(w')$ is already present in the network, which means that it is either permanent or temporary. Since label $q^B(w)$ has not been found yet by the algorithm, label $q^B(w')$ has not been investigated yet, which means that is either a temporary label or a label which has not been found yet. This leads to four cases for the position of w' and thus $q^F(w')$ and $q^B(w')$ as illustrated in figure 4.3.

Case 1: $q^F(w')$ is a permanent label and $q^B(w')$ is a temporary label. This path has been found by the algorithm as both the forward and the backward label are

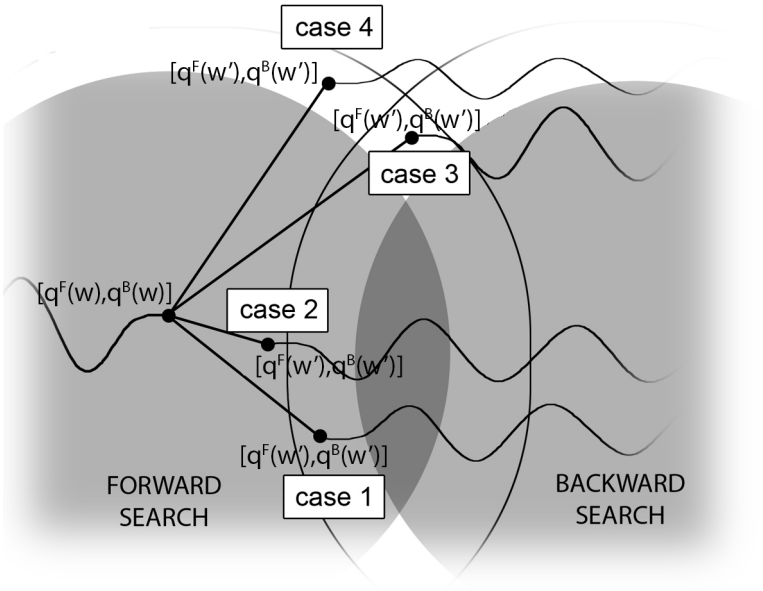

Figure 4.3: Part of the forward and the backward search area. The gray areas contain all permanent labels, while the rings around these areas contain the labels of the temporary sets. There are four possibilities for the position of the node w'.

present in the network.

Case 2: $q^F(w')$ is permanent and $q^B(w')$ is not an element of the backward search space (i.e has not been found yet). This means that

$$(\exists i : c_i'^F < min_i(T_F) \vee \forall i : c_i'^F = min_i(T_F)) \wedge \forall i : c_i'^B \geq min_i(T_B) \quad (4.19)$$

which can be reduced to the initial situation by translating w' to w.

Case 3: Both $q^F(w')$ and $q^B(w')$ are temporary labels. Similar to case 1, this path has been found since both the forward and the backward label are present in the network.

Case 4: $q^F(w')$ is an element of the forward temporary set and $q^B(w')$ has not been found yet. As the label $q^F(w')$ is an element of the forward temporary set, this means that $\forall i : q_i^F(w') \geq min_i(T_F)$. Moreover, the label $q^B(w')$ is situated outside the backward search area, which means that $\forall i : q_i^B(w') \geq min_i(T_B)$. Combining these labels will result in a path for which equation (4.17) does not hold and which thus cannot be an element of the Pareto optimal set.

From this we can conclude that there exists no path which has not been found yet, but which should be an element of the Pareto optimal solution. Once the stop condition holds all Pareto optimal paths are found. □

4.4 Theoretical analysis

In this section the complexity of the (bidirectional) multiple objective shortest path algorithm will be estimated. Experiments have shown that most of the calculation time goes to investigating the labels, i.e. removing the label from the temporary set and updating the labels of its neighbors. So, the number of calculations is more or less proportional to the number of handled (or investigated) nodes. This number is dependent upon the size of the network, the density of the network and the number of objectives. In this section an intuitive notion is given of the operation of the multiple objective shortest path algorithms by means of experiments and calculations.

First, a rough estimate of the upper bound of the number of handled labels is made by maximizing the density, i.e. working with complete graphs. In the worst case (and without the stop condition), when executing the algorithm, all possible paths between the origin and all other nodes of the network are calculated. These paths thus consist of the origin node followed by an arrangement of i nodes (with i ranging from 0 to $(N-1)$) from the remaining set of $(N-1)$ nodes. This follows from the fact that we are considering complete (full mesh) graphs. The number

of possible paths then is the sum over i (from 0 to $(N-1)$) of the variations of i elements out of a set of $(N-1)$ elements. Mathemathically, the number of paths (i.e. labels) in a complete graph with N nodes can be described as

$$\#labels = \sum_{i=0}^{N-1} \frac{(N-1)!}{(N-1-i)!} \tag{4.20}$$

It should be noted that for large values of N, this sum may be rounded to

$$\#labels = e.(N-1)! \tag{4.21}$$

Table 4.1 shows these number for different values of N. It can be seen that the number of labels increases tremendously as the size of the network increases.

N	# labels	N	# labels
1	1	**6**	326
2	2	**7**	1957
3	5	**8**	13700
4	16	**9**	109601
5	65	**10**	986210

Table 4.1: Number of labels investigated in complete network of size n.

Next, we investigated the growth of the search areas. As depicted in figure 4.4, two areas can be distinguished: the permanent and the temporary area. The permanent search area contains all nodes for which the stop condition 1 holds, which means that for every node at least one label dominates the minima of the temporary set $[min_i(T)]$. The temporary search area contains all nodes with labels which are not permanent yet.

As mentioned earlier, the execution time of the multiple objective shortest path algorithm depends mostly upon the number of investigated labels. This number was compared to the number of permanent nodes, i.e. the size of the permanent search area, during the execution of the unidirectional algorithm. As can be seen in figure 4.5, this number of investigated labels grows polynomially in function of the number of permanent nodes. Here, the objectives were chosen to be independent of each other. The functions in the figures were determined making use of the method of least squares, where it should be noted that a function of the form $y = \alpha.x^{\beta}$ fits best. Furthermore, the exponent β is dependent on the number of objectives n: the more objectives, the larger this β is. For the (single objective) algorithm of Dijkstra this function is more or less linear, which means that this exponent is equal to 1. This implies that β is greater than 1 for all $n > 1$.

From the previous reasoning we can induce following formula:

$$\#labels_{uni} = \alpha.x^{\beta} \tag{4.22}$$

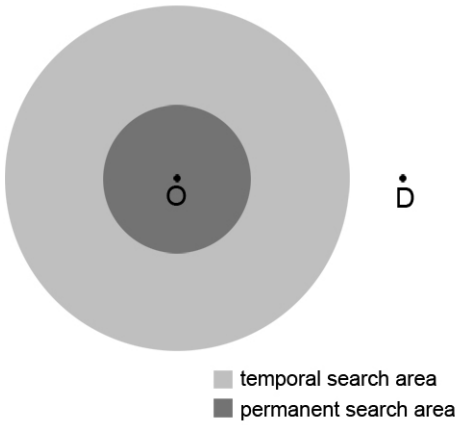

Figure 4.4: The permanent and the temporal search area (of the single objective shortest path algorithm).

with x the number of permanent nodes and β the exponent which increases as the number of objectives increases. The factor α is a constant scaling factor. The number of permanent nodes x is an indication of the size of the permanent area. For simplification, we will assume that this area is more or less circular. Furthermore, if all objectives are completely uncorrelated, the search area of the MOSP algorithm with n objectives may be assumed $(n+1)$-dimensional. While for the algorithm of Dijkstra it is 2-dimensional (see figure 4.4), for the MOSP algorithm with 2 objectives it is 3-dimensional. This means that the area of permanent labels can be represented by $x \approx \pi.r^{(n+1)}$ with r the radius. So for a certain number of objectives n the number of investigated labels in the unidirectional algorithm approximates

$$\#labels_{uni} \approx \alpha.(\pi.r^{(n+1)})^{\beta} \tag{4.23}$$

We will now determine theoretically how much faster the bidirectional algorithm is in comparison to the unidirectional algorithm. The bidirectional multiple objective shortest path algorithm investigates the network simultaneously from the origin and the destination until both searches meet in the middle. We will now assume, in consensus with the algorithm of Dijkstra, that the radii of both searches are equal to $\frac{r}{2}$, with r the radius of the permanent area of the unidirectional algorithm, which is an acceptable approximation. The number of nodes investigated by the

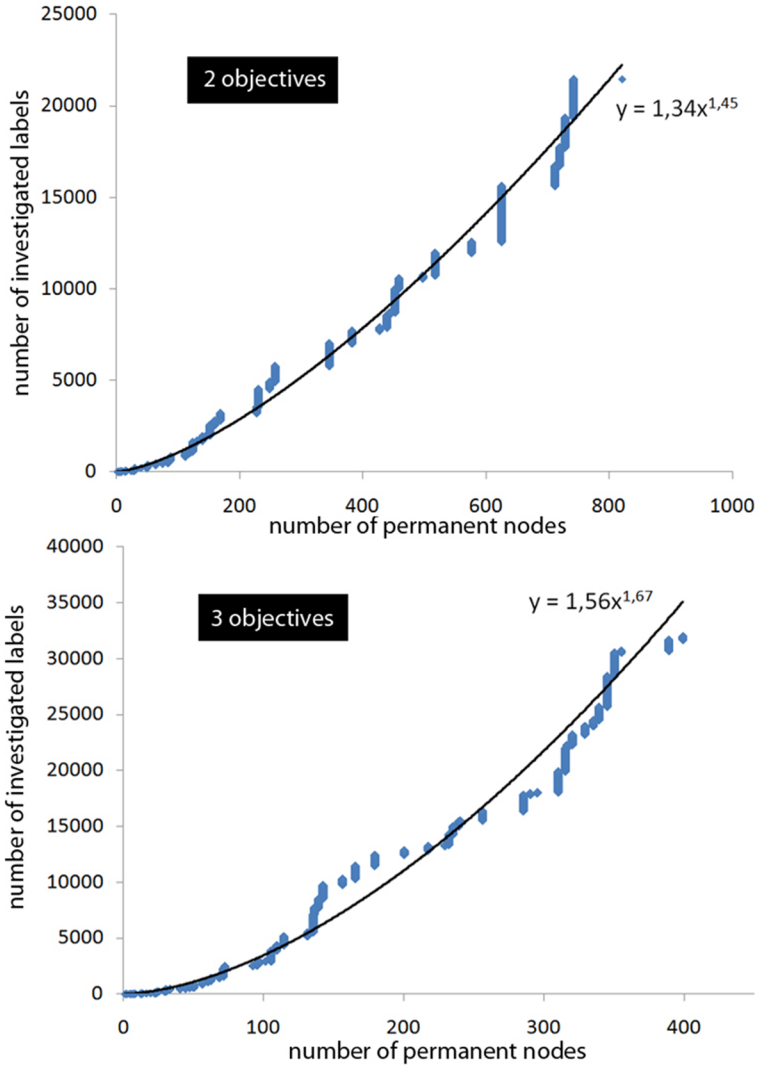

Figure 4.5: The number of investigated labels in function of the size of the permanent search area for two (top) and three (bottom) objectives.

bidirectional algorithm then equals

$$\#labels_{bi} \approx 2.\alpha.(\pi.(\frac{r}{2})^{(n+1)})^{\beta} \tag{4.24}$$

or

$$\#labels_{bi} \approx \frac{\alpha.(\pi.r^{(n+1)})^{\beta}}{2^{\beta(n+1)-1}} \tag{4.25}$$

or

$$\#labels_{bi} \approx \frac{\#labels_{uni}}{2^{\beta(n+1)-1}} \tag{4.26}$$

As $\beta > 1$ and $n > 1$ (and thus $\beta(n+1) - 1 > 1$), this means that a time gain of more than 2 can be achieved, which will also be verified in the experiments in the next subsection.

4.5 Experimental results

In this section the experimental results are presented. First, an overview is given of the setup of the experiments, together with a description of the networks in which the algorithms were applied. Subsequently, in the next subsection, it is shown for which networks applying the stop condition of section 4.2.3 in the unidirectional $s - t$ algorithm is really advantageous. Moreover, an analysis is made of the execution time in one specific network and it is investigated how the number of objectives influences the speedup factor. In the last subsection, we will look at the bidirectional algorithm, more specifically at the speedup factors in different network configurations, the execution time in one specific network and the impact of an increasing number of objectives. Finally, the cardinality of the Pareto optimal solution set is discussed briefly.

4.5.1 Setup of the experiments

All algorithms presented in this paper were implemented in Java (version 1.6.0-18). We opted to represent the temporary set by a (lexicographically ordered) priority queue. Moreover, the label sets of the nodes were implemented as hashsets. All experiments were executed on a machine with the following configuration: Intel (R) Core(TM) 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM.

All graphs that are used in this research are assumed to be directed. Undirected links were replaced by two inverse directed links. There are 4 network configurations that are used: transportation, square grid, random and complete networks.

A number of transportation networks (Transport X) were provided by a Belgian company who is a major industrial player in the field of traffic information. In this research, we made use of a network representing the Belgian road network (B). Moreover, the three transportation networks that were used in [9] and [39]

were retrieved. These networks are road networks of the US, more specifically those of Washington DC (DC), Rhode Island (RI) and New Jersey (NJ). Next to the length of the links (i.e. streets, or part of streets), travel time information was made available. For the experiments, the length (in meters) was utilized as first objective and the travel time (in milliseconds) as second objective. Since we do not have more data at hand, for the other objectives random values between 0 and 1000 were used.

In a square grid network (Square Grid N) nodes are placed on a square grid ($N \cdot N$) and all the direct (horizontal and vertical) neighbors are connected to each other. This means that each node has at most 4 neighbors. This type of grid network is often denoted with the term 'Manhattan network'. The cost vectors assigned to the links consist of n (assumed that there are n objectives) completely uncorrelated random values between 0 and 1000.

A random network (Random $N(d)$) contains N nodes and $(d \cdot N)$ links. These nodes are randomly connected to each other by the specified number of links, where we made sure that the networks are connected. This means that the number of links needs to be at least as big as the number of nodes (or $d \geq 1$). Again, (completely uncorrelated) random values between 0 and 1000 were used to represent the different costs of the links. Random networks have the advantage that the number of nodes and the number of links can be changed easily. In this research we will use the random networks to determine trends in the speedup factors in function of the number of nodes in the networks and the average node degree in the networks.

Finally, a complete (full dense or full mesh) network (Complete N) is a network with N nodes in which each node is connected to all other nodes. We again used n completely uncorrelated random values (between 0 and 1000) to represent the cost vectors of the links.

An overview of all networks that were used in the experiments is given in table 4.2.

All experiments were carried out for 1000 randomly selected origin-destination pairs and the presented results are averages over these 1000 multiple objective shortest path calculations. The execution times of the algorithms (without preprocessing and postprocessing) were measured with a timer that has a precision of nanoseconds and an accuracy of microseconds. For both presented speedup techniques, speedup factors are reported. The speedup factor (SUF) of algorithm A over algorithm B is defined as:

$$SUF(A/B) = \frac{avg.\ execution\ time\ algorithm\ B}{avg.\ execution\ time\ algorithm\ A} \tag{4.27}$$

A speedup factor higher than 1 means that algorithm A is faster than algorithm B. For clarity issues, we will denote the unidirectional MOSP algorithm without the

network configuration	# nodes	# links	average outdegree
Transport (DC)	9 559	29 765	3.1
Transport (B)	23 080	50 364	2.2
Transport (RI)	53 658	138 083	2.6
Transport (NJ)	330 386	869 471	2.6
Complete 100	100	9 900	99
Complete 300	300	89 700	299
Square Grid 30	900	3 480	3.86
Square Grid 100	10 000	39 600	3.96
Random 10000(3)	10 000	30 000	3
Random 20000(3)	20 000	60 000	3
Random 100000(3)	100 000	300 000	3
Random 10000(4)	10 000	40 000	4
Random 10000(5)	10 000	50 000	5
Random 10000(6)	10 000	60 000	6

Table 4.2: Overview of all network configurations used for the experiments.

stop condition with U, the unidirectional MOSP algorithm with the stop condition with US and the bidirectional MOSP algorithm (with the stop condition) with B.

4.5.2 The unidirectional MOSP algorithm

In this section it will be demonstrated that using the stop condition (see section 4.2.3) indeed speeds up a number of unidirectional shortest path calculations in some network configurations. In the first part, a comparison will be made of the speedup factors for the different network configurations. Subsequently, we will look at one specific configuration and see how the calculation time varies with a varying 'distance' between the origin and the destination.

Table 4.3 shows the average execution time of the unidirectional MOSP algorithm without the stop condition and the speedup achieved by using stop condition 1 (SUF(US/U)) for a number of network configurations, and this for both 2 and 3 objectives. One can see that the speedup factors for the transportation networks are much higher than those for the other networks. The results for the three US network (DC, RI and NJ) are comparable to the results of [39]. For the square grid networks, on the other hand, slightly higher speedup factors are perceived. Applying the stop condition in a complete network does not result in smaller execution times. At all times, the whole network needs to be investigated and the algorithm with the stop condition is even slightly slower than the algorithm without the stop condition, due to the overhead of checking the stop condition. Looking at the random networks, two trends are perceived. When the number of nodes in the

	2 OBJECTIVES		3 OBJECTIVES	
network configuration	**CPU time (ms)**	**SUF(US/U)**	**CPU time (ms)**	**SUF(US/U)**
Transport (DC)	127	2.35	962	2.55
Transport (B)	443	3.35	47 657	3.41
Transport (RI)	1 914	2.39	156 494	2.86
Transport (NJ)	18 191	2.96	1 784 817	3.25
Complete 100	40	0.99	229	0.98
Complete 300	839	0.99	7 837	0.99
Square Grid 30	136	1.33	16 312	1.44
Square Grid 100	978	1.59	103 435	1.62
Random 10000(3)	96	1.25	102	1.41
Random 20000(3)	547	1.58	717	1.96
Random 100000(3)	2 155	2.00	6 124	2.69
Random 10000(4)	199	1.15	674	1.19
Random 10000(5)	314	1.02	1 225	1.01
Random 10000(6)	498	0.99	2 068	0.99

Table 4.3: The execution times of the unidirectional MOSP algorithm without the stop condition and the speedup factors of the unidirectional algorithm with the stop condition over the one without the stop condition (SUF(US/U)) in different network configurations.

network increases, the speedup factor increases as well. The larger the network is, the higher the chance that the origin and the destination node are 'closer' to each other and that only a smaller part of the network needs to be investigated. If the number of nodes in the network is kept constant and the average out degree of the nodes is increased, the speedup factor decreases. For the highest out degrees, this is similar to the results of the complete networks.

It is known that for the algorithm of Dijkstra interrupting the search process as soon as the label of the destination node is permanent is only advantageous (with respect to the calculation time) if the origin and the destination node are situated not too far from each other. The calculation time increases monotonously in function of the distance (i.e. shortest path length with respect to the diameter of the network) between the origin and the destination node. Now, we want to investigate the relationship between the calculation time of the unidirectional MOSP algorithm and the 'distance' between the origin and the destination node. Since, for multiple objective shortest path algorithms, there is no single value that indicates the 'distance' between two nodes, an estimation needs to be used. We will assume the minimum value of the first objective as an approximation of how far apart two nodes are from each other.

As the speedup factors for the transportation networks are the highest, and the concept of 'distance' has more significance in these networks, we opted to examine this for transportation networks only. Here, we made use of the Transport (DC) network. Figure 4.6 shows the execution times of the unidirectional algorithm with and without stop condition in function of the 'distance' (i.e. minimum value of the first objective) between the origin and the destination node, for both 2 and 3 objectives. Moreover, the minimum and the maximum values of the execution times are indicated. The execution times of the algorithm without the stop condition are constant as each time the whole network is investigated. As expected, when the stop condition is applied smaller execution times are perceived for the 'shorter' paths and for the 'longer' paths the execution times tilt to those of the algorithm without the stop condition. The bars in the figures show that there is more variability in the execution times for the paths with medium 'length'. The curve for 3 objectives reaches its upper limit sooner than the one for 2 objectives. When dealing with 3 objectives, more of the network needs to be investigated, before the search can be aborted.

In table 4.3 results are shown for both two and three objectives. The speedup factors of the unidirectional algorithm with the stop condition over the one without the stop condition are slightly higher for 3 objectives than for 2 objectives. We will now investigate, for one particular case (the Transport (DC) network), whether this trend continues with higher number of objectives. Results are shown in table 4.4. It can be seen that indeed the speedup factor increases with an increasing number of objectives. Nevertheless, these increases are relatively small.

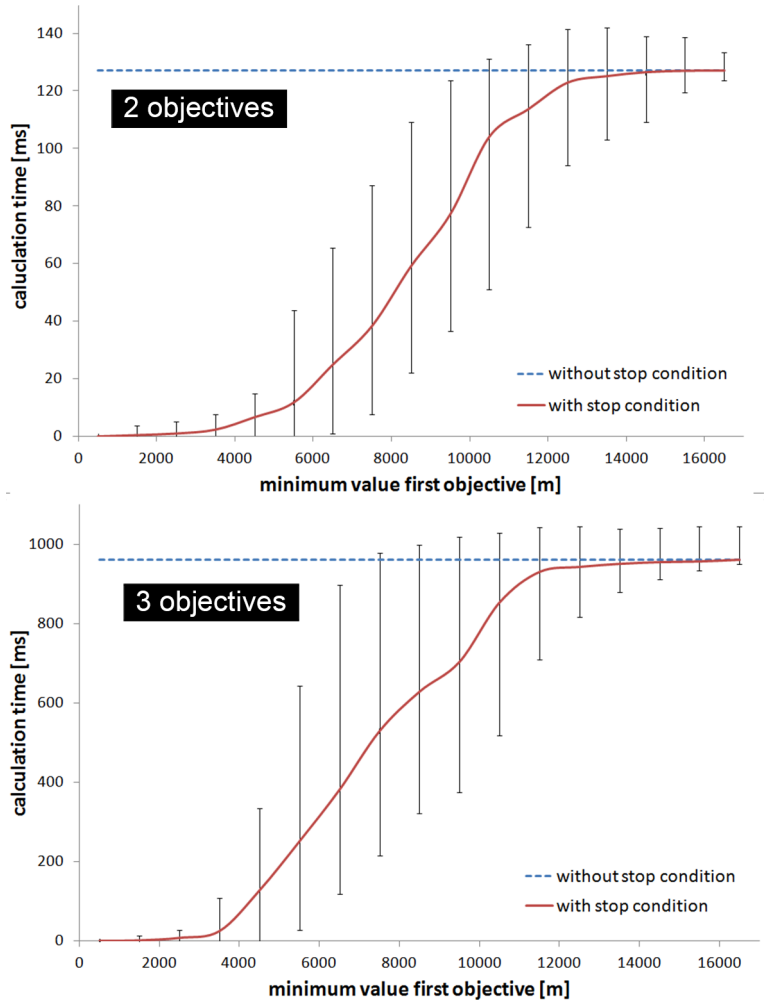

Figure 4.6: Calculation time of multiple objective shortest path algorithm with and without stop condition in the Transport (DC) network with 2 objectives (top) and 3 objectives (bottom).

# obj.	SUF(US/U)
2	2.35
3	2.55
4	2.69
5	2.76
6	2.87
7	2.97
8	3.05

Table 4.4: Speedup factors (SUF) of the unidirectional MOSP algorithm for different number of objectives in the Transport(DC) network.

4.5.3 The bidirectional MOSP algorithm

In this section it will be demonstrated how searching the network bidirectionally in most cases indeed speeds up the calculations. Besides examining the speedup factors for the different network configurations, we will see how the speed up factor reacts to an increasing number of objectives in transportation networks. Moreover, it will be indicated for which paths searching bidirectionally is really advantageous. Next to this, we will have a quick look at the cardinality of the solution set.

Table 4.5 shows the speedup factors of the bidirectional MOSP algorithm over the unidirectional algorithm with the stop condition ($SUF(B/US)$), for both 2 and 3 objectives. Clearly noticeable is the fact that the speedup factors in the transportation networks are higher than those in the other network configurations. Searching the network bidirectionally is certainly not a good choice when dealing with complete networks. Here, the bidirectional algorithm is even slower than its unidirectional counterpart. For square grid networks, only relatively small speedups are achieved. Looking at the random networks, some trends can be identified. When the number of nodes in the network increases, the speedup factor increases too. If, on the other hand, the number of nodes is constant and the average out degree of the nodes is increased, the speedup factor decreases. These trends are similar to those observed for the speedup factor of the unidirectional algorithm with the stop condition over the one without the stop condition.

Similar to the results of section 4.5.2 we looked at the calculation time of the bidirectional MOSP algorithm in the Transport (DC) network in function of the 'distance' (i.e. minimum value of the first objective) between the origin and the destination node. This is shown in figure 4.7, together with the calculation time of the unidirectional algorithm with the stop condition, which is the same as in figure 4.6. One can see that the calculation times for the bidirectional algorithm are indeed much smaller. The curves of the bidirectional algorithm increase monotonously without reaching an upper limit, like the curves of the unidirectional

	2 OBJECTIVES		3 OBJECTIVES	
network configuration	**CPU time (ms)**	**SUF(B/US)**	**CPU time (ms)**	**SUF(B/US)**
Transport (DC)	54	5.76	377	12.47
Transport (B)	132	5.91	13 976	14.46
Transport (RI)	801	7.77	54 718	16.16
Transport (NJ)	6 146	10.08	549 174	21.43
Complete 100	40	0.60	305	0.50
Complete 300	847	0.58	7 916	0.37
Square Grid 30	102	1.49	11 328	1.57
Square Grid 100	615	1.79	63 849	1.96
Random 10000(3)	77	3.29	72	3.54
Random 20000(3)	346	3.99	66	3.99
Random 100000(3)	1 078	4.52	2 277	4.67
Random 10000(4)	173	2.75	566	2.51
Random 10000(5)	308	2.01	1 213	1.30
Random 10000(6)	503	1.04	289	0.53

Table 4.5: The average execution times of the unidirectional MOSP algorithm with the stop condition and the speedup factors of the bidirectional algorithm over the unidirectional algorithm with the stop condition (SUF(B/US)) in different network configurations.

# obj.	SUF(B/US)
2	5.76
3	12.47
4	12.88
5	13.20
6	13.24
7	14.27
8	15.50

Table 4.6: Speedup factors (SUF) of the bidirectional MOSP algorithm over the unidirectional algorithm for different number of objectives in the Transport(DC) network.

algorithms do. This means that the highest speedups are achieved for the medium length paths. For the shortest paths the curves are nearly flat for both algorithms, resulting in smaller speedup factors. When the 'distance' between the origin and the destination node increases, the calculation time of the unidirectional algorithm increases faster than that of the bidirectional algorithm. This results in higher speedup factors. For the longest paths the unidirectional algorithm has reached its upper limit, while the calculation time of the bidirectional algorithm still increases, resulting in smaller speedup factors. The highest speedup factors are encountered for the medium length paths.

Subsequently, we investigated the impact of the number of objectives on the speedup factor. In table 4.5 results are shown for both 2 and 3 objectives, where it can be seen that the speedup factors are in most cases higher when there are 3 objectives. For the complete networks, where it is not advantageous to search bidirectionally, adding an objective increases the execution time of the bidirectional algorithm. For the random networks with a higher average out degree of the nodes, this is also the case.

We will now focus on the transportation networks and calculate the average speedup of the bidirectional algorithm over the unidirectional algorithm (with stop condition) for a different number of objectives. Results are shown in table 4.6. It is shown that indeed the speedup factor increases with an increasing number of objectives. There is a difference in the increase between 2 and 3 objectives and that between a higher number of objectives. This can be explained by the fact that for the first two objectives the actual distance and travel time information was used, while for the other objectives random values were selected.

Besides this, we investigated the average cardinality of the solution set. Looking at table 4.7 one can see that this cardinality is highly dependent on the network configuration with the highest cardinalities for the square grid networks and the lowest ones for the random networks. Moreover, the average cardinality is dependent on both the number of nodes in the network and the average out degree of the nodes. When the number of nodes in the network increases, the cardinality

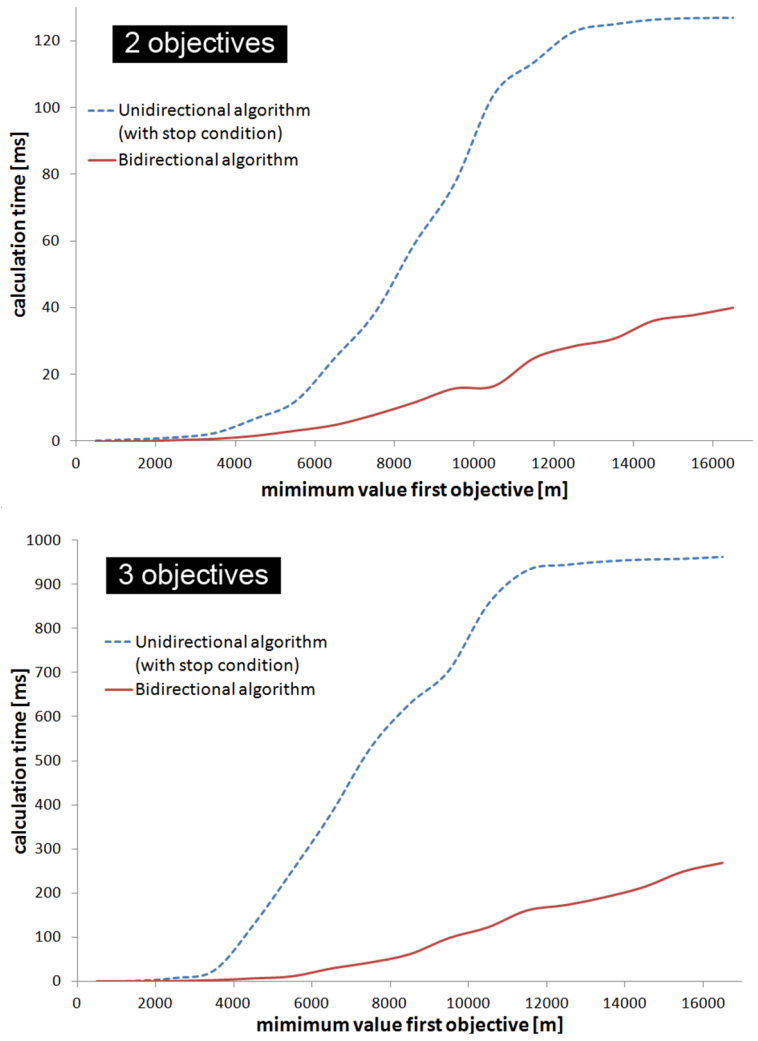

Figure 4.7: Comparison of unidirectional and bidirectional algorithm in function of the number of hops in the shortest path.

	avg. # L_{result}	
	2 OBJECTIVES	**3 OBJECTIVES**
Transport (DC)	14.97	32.67
Transport (B)	21.72	48.03
Transport (RI)	30.40	87.98
Transport (NJ)	83.23	151.15
Complete 100	12.83	58.79
Complete 300	17.98	120.29
Square Grid 30	23.52	242.49
Square Grid 100	173.64	502.37
Random 10000(3)	3.86	3.67
Random 20000(3)	3.94	7.87
Random 100000(3)	4.27	9.46
Random 10000(4)	4.89	10.98
Random 10000(5)	5.71	13.84
Random 10000(6)	6.56	17.11

Table 4.7: Average cardinality solution set for different network configurations.

of the solution set increases. Similarly, when the average out degree of the nodes increases, the cardinality of the solution set does also. This is as expected.

4.6 Conclusion

In this chapter the main goal was to speed up the multiple objective shortest path (MOSP) algorithm as presented by Martins [1], while still maintaining the guarantee to find all Pareto optimal paths. The improvements prove to be very useful, especially in large transportation networks.

First, a stop criterion for the unidirectional algorithm was introduced which finishes the search process as soon as the destination node contains all its labels, which can be translated to Pareto optimal paths. It has been proven that, once this stop condition holds, all Pareto optimal paths are found. Experiments have shown that a remarkable speedup can be achieved in transportation networks and that this speedup is the largest if the origin and the destination node are 'close' to each other. Moreover, the speedup factor increases when dealing with more objectives.

Secondly, a bidirectional MOSP algorithm was introduced, which searches the network from the origin and the destination simultaneously. It is proven that this algorithm always returns the complete Pareto optimal set of solutions. In the experiments, it can be seen that the speedup achieved by searching the network bidirectionally is dependent upon the network configuration, the number of nodes in

the network, the average out degree of the nodes, the relative position of the origin and the destination node, and the number of objectives.

From the experiments it can be concluded that the speedup measures presented in this chapter are particularly well suited for (large) transportation networks. The highest speedup is achieved by searching the network bidirectionally.

References

[1] E. Q. V. Martins. *On a multicriteria shortest path problem*. European Journal of Operational Research, 16(2):236 – 245, 1984.

[2] 9th DIMACS Implementation Challenge. *Shortest Paths*, 2006. Available from: http://www.dis.uniroma1.it/~challenge9.

[3] S. Demeyer, P. Audenaert, B. Slock, M. Pickavet, and P. Demeester. *Multimodal transport planning in a dynamic environment*. In Proceedings of IPTS2008, the Conference on Intelligent Public Transport Systems, pages 155–167, 2008.

[4] M. Ehrgott and X. Gandibleux. *A survey and annotated bibliography of multiobjective combinatorial optimization*. OR-Spektrum, 22:425–460, 2000.

[5] P. Serafini. *Some considerations about computational complexity for multiobjective combinatorial problems*. Recent advances and historical development of vector optimization, 294:222–232, 1986.

[6] P. Vinkce. *Problèmes multicritères*. Cahiers Centre Etudes Recherche Operationnelle, 16:425–439, 1974.

[7] P. Hansen. *Bicriterion path problems*. In G. Fandel and T. Gal, editors, Multiple Criteria Decision Making: Theory and Applications, volume 177 of *Lecture Notes in Economics and in Mathematical Systems*, pages 109–127. Springer Berlin / Heidelberg, 1980.

[8] M. Ehrgott and X. Gandibleux. *Multiple Criteria Optimization. State of the art annotated bibliographic surveys*. Kluwer Academic, Dordrecht, 2002.

[9] A. Raith and M. Ehrgott. *A comparison of solution strategies for biobjective shortest path problems*. Computers & Operations Research, 36(4):1299 – 1331, 2009.

[10] A. J. V. Skriver. *A classification of bicriterion shortest path (BSP) algorithms*. Asia Pacific Journal of Operational Research, 17:192–212, 2000.

[11] J. C. N. Clìmaco and M. M. B. Pascoal. *Multicriteria path and tree problems: discussion on exact algorithms and applications*. International Transactions in Operational Research, 19(1-2):63–98, 2012.

[12] J. C. N. Climaco and E. Q. V. Martins. *A bicriterion shortest path algorithm*. European Journal of Operational Research, 11(4):399 – 404, 1982.

[13] J. Azevedo, M. E. O. S. Costa, J. J. E. S. Madeira, and E. Q. V. Martins. *An algorithm for the ranking of shortest paths*. European Journal of Operational Research, 69(1):97 – 106, 1993.

[14] E. Q. V. Martins. *An algorithm for ranking paths that may contain cycles*. European Journal of Operational Research, 18(1):123 – 130, 1984.

[15] V. Jiménez and A. Marzal. *Computing the K Shortest Paths: A New Algorithm and an Experimental Comparison*. In J. Vitter and C. Zaroliagis, editors, Algorithm Engineering, volume 1668 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin / Heidelberg, 1999.

[16] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. *A new improvement for a K shortest paths algorithm*. Investigação Operational, 21:47–60, 2001.

[17] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. *Labeling algorithms for ranking shortest paths*. Technical report, CISUC, 2000.

[18] M. EQV., P. ao JM., R. MS., and S. JLE. *Ranking multiobjective shortest paths*. Pré-publicações do Departamento de Matemática, 07-11, 2007.

[19] J. Paixão, M. Rosa, and J. Santos. *A new ranking path algorithm for the multiobjective shortest path problem*. Pré-publicações do Departamento de Matemática, 08-27, 2008.

[20] V. Jiménez and A. Marzal. *A Lazy Version of Eppsteins K Shortest Paths Algorithm*. In K. Jansen, M. Margraf, M. Mastrolilli, and J. Rolim, editors, Experimental and Efficient Algorithms, volume 2647 of *Lecture Notes in Computer Science*, pages 179–191. Springer Berlin / Heidelberg, 2003.

[21] D. Eppstein. *Finding the k shortest paths*. SIAM Journal on Computing, 28(2):652–673, 1998.

[22] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. *Deviation algorithms for ranking shortest paths*. International Journal of Foundations of Computer Science, 10(03):247–261, 1999.

[23] E. W. Dijkstra. *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1):269–271, 1959.

[24] R. Bellman. *On a Routing Problem*. Quarterly of Applied Mathematics, 16:87 – 90, 1958.

[25] J. Brumbaugh-Smith and D. Shier. *An empirical investigation of some bicriterion shortest path algorithms*. European Journal of Operational Research, 43(2):216 – 224, 1989.

[26] A. Skriver and K. Andersen. *A label correcting approach for solving bicriterion shortest-path problems*. Computers & Operations Research, 27(6):507 – 524, 2000.

[27] V. Sastry, T. Janakiraman, and S. Mohideen. *New algorithms for multi objective shortest path problem*. Opsearch, 40(4):278–298, 2003.

[28] F. Guerriero and R. Musmanno. *Label Correcting Methods to Solve Multicriteria Shortest Path Problems*. Journal of Optimization Theory and Applications, 111:589–613, 2001.

[29] E. Q. V. Martins and J. L. E. Santos. *The Labeling Algorithm for the Multiobjective Shortest Path Problem*. Technical report, CISUC, 1999.

[30] X. Gandibleux, F. Beugnies, and S. Randriamasy. *Martins' algorithm revisited for multi-objective shortest path problems with a MaxMin cost function*. 4OR: A Quarterly Journal of Operations Research, 4:47–59, 2006.

[31] L. de Lima Pinto, C. T. Bornstein, and N. Maculan. *The tricriterion shortest path problem with at least two bottleneck objective functions*. European Journal of Operational Research, 198(2):387 – 391, 2009.

[32] L. L. Pinto and M. M. Pascoal. *On algorithms for the tricriteria shortest path problem with two bottleneck objective functions*. Computers & Operations Research, 37(10):1774 – 1779, 2010.

[33] C. T. Bornstein, N. Maculan, M. Pascoal, and L. L. Pinto. *Multiobjective combinatorial optimization problems with a cost and several bottleneck objective functions: An algorithm with reoptimization*. Computers & Operations Research, 39(9):1969 – 1976, 2012.

[34] J. M. A. Pangilinan and G. K. Janssens. *Evolutionary algorithms for the multiobjective shortest path planning problem*. In International journal of computer and information science and engineering, pages 54–59, 2007.

[35] R. G. Garroppo, S. Giordano, and L. Tavanti. *A survey on multi-constrained optimal path computation: Exact and approximate algorithms*. Computer Networks, 54(17):3081 – 3107, 2010.

[36] J. Paixão, M. Rosa, and J. Santos. *Labelling methods for the general case of the multi-objective shortest path problem - a computational study*. Pré-publicações do Departamento de Matemática, 07-42, 2007.

[37] B. S. Stewart and C. C. White, III. *Multiobjective A**. Journal of the Association for Computing Machinery (ACM), 38(4):775–814, 1991.

[38] Y. Disser, M. Müller-Hannemann, and M. Schnee. *Multi-criteria Shortest Paths in Time-Dependent Train Networks*. In C. McGeoch, editor, Experimental Algorithms, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer Berlin / Heidelberg, 2008.

[39] A. Raith. *Speed-up of labelling algorithms for biobjective shortest path problems*. In Proceedings of the 45th Annual Conference of the ORSNZ, pages 313–322, 2010.

[40] L. Fu, D. Sun, and L. R. Rilett. *Heuristic shortest path algorithms for transportation applications: state of the art*. Computers and Operations Research, 33(11):3324–3343, 2006.

[41] A. V. Goldberg, H. Kaplan, and R. F. Werneck. *Reach for A: Efficient point-to-point shortest path algorithms*. In In Workshop on Algorithm Engineering & Experiments, pages 129–143, 2006.

[42] T. A. J. Nicholson. *Finding the Shortest Route between Two Points in a Network*. The Computer Journal, 9(3):275–280, 1966.

[43] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*. In C. McGeoch, editor, Experimental Algorithms, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin / Heidelberg, 2008.

5

Time-Dependent and Stochastic Routing in Multimodal Transportation Systems

In this chapter a case study is presented of a multimodal routing system that takes into account both time-dependent and stochastic travel time information. This routing system uses the multimodal network model that was presented in chapter 2, in which it is possible to model the travel time information of each transport mode differently. This travel time information can either be static (i.e. time-independent) or time-dependent, and either deterministic or stochastic. Next to this, a Dijkstra-based routing algorithm is presented that deals with this variety of travel time information in a uniform way. This research focuses on a practical implementation of the system, which means that a number of assumptions were made, like, for example, the modeling of the stochastic distributions (see also section 2.4.4), comparing these distributions, etc. A trade-off had to be made between the performance of the system and the accuracy of the results. Experiments have shown that our system produces realistic routes in a short amount of time. It is demonstrated that time-dependent routing indeed results in a travel time gain in comparison to routing statically. By using the additional stochastic travel time information, even better (i.e. faster) and more reliable routes can be calculated. Moreover, it is shown that routing in the multimodal network may have its advantages over routing in a unimodal network, especially during rush hours.

5.1 Introduction

With the evolvement of GPS systems, novel routing algorithms for transportation networks have emerged. In order to better predict the travel time of a route, time-dependent travel time information should be taken into account. Furthermore, more accurate travel time predictions can be made by making use of stochastic information. This results in a route together with a stochastic distribution of its travel time. With increasing traffic volumes, often resulting in congestion, using multiple transportation modes (e.g. train, plane, ship, etc.) gains more interest. This explains the appearance of multimodal routing algorithms that take into account multiple modes of transportation to determine the best route between two locations.

In this chapter, a multimodal, time-dependent and stochastic routing system is presented. This means that it takes into account both the time-dependency and the uncertainty of the travel times. Moreover, multiple modes of transportation are considered in order to find a better route between an origin and a destination. To the best of our knowledge, routing systems incorporating all these characteristics have never been realized before. Nevertheless, both unimodal as well as multimodal systems exist that take into account either the time-dependency of the travel time or the uncertainty. In section 5.1.1 we will review some of these systems and note the differences and/or similarities with the system described in this chapter.

5.1.1 Related Work

The most common (early) routing systems presume the travel time data to be static, i.e. independent of the time of the day. In this setting, the well-known algorithm of Dijkstra [1] can be applied in order to find the shortest (i.e. fastest) route between two locations. Since traffic is not a static matter, more accurate travel times can be obtained by taking into account its time-dependent nature, i.e. the (travel time) cost of a link is dependent on the time of the day this link is traversed. To deal with time-dependent travel times, the concept of time-based graphs was introduced in which two major approaches can be distinguished: the time-expanded and the time-dependent approach [2]. While in the time-expanded approach a node exists for every event at a location and links represent time lapses between these events, in the time-dependent approach each geographic location is represented by a single node and all time-dependent travel time information is stored in the links themselves. For a detailed description of these time-based graph models we would like to refer to section 2.2.2 of chapter 2. Due to the large number of possible events in a road network (cars can leave at any time), we opted for the time-dependent approach.

In [3], it is shown that time-dependent shortest path computations indeed can reduce the travel time significantly. Delling and Wagner [4] give an overview

of the current state of time-dependent route planning together with a number of speedup techniques. It includes some of the concepts we used in our research, such as the modeling of time-dependent travel times, the augmented algorithm of Dijkstra, etc. A number of speedup techniques for time-dependent shortest path routing, such as hierarchical routing ([5] and [6]) and bidirectional A* routing [7], have been proposed. Nevertheless, since we are routing both time-dependently and stochastically, these cannot be directly applied in our system.

Road travel times are not only time-dependent, but also contain an amount of uncertainty. One can never be a hundred percent sure when to arrive at his/her destination, as the travel time is influenced by random factors (individual driver's behavior, weather conditions, traffic accidents, etc.). We opted to model the travel time by custom probability distributions (see section 2.4.4 of chapter 2) and developed a stochastic origin-destination shortest path algorithm. In the early days, some of the basic problems encountered when developing stochastic shortest path algorithms, were tackled ([8], [9] and [10]). Ji [11] presents three kinds of stochastic problems together with a genetic algorithm and adapted linear programming methods to solve these problems. Unfortunately, these algorithms have only been tested on small networks and are not scalable for very large (road) networks. A promising time-dependent stochastic algorithm is presented by Azaron and Kianfar [12]. They assume the travel times to have an exponential distribution, while we focus on the more accurate custom distributions collected from actual travel time measurements. Li et al. [13] take into account the stochastic properties of the travel time and study whether a long term equilibrium exists. While we mainly search for the earliest arrival time starting from a certain departure time, they optimize the departure time for a preferred arrival time. Moreover, they do not assign stochastic travel time distributions to the links, but look at a long term equilibrium. A multicriteria A* shortest path algorithm was presented by Chen et al. [14]. They assume the stochastic travel time of a link to be correlated with the travel times of the neighboring links. Aside from the fact that this algorithm could only be tested on small networks, they do not take into account the time-dependency of the travel times. Samaranayake et al. [15] provide a theoretical basis for enabling tractable solutions to the on-time arrival problem in a stochastic environment and present a stochastic shortest path algorithm that performs well in road networks. While we focus on a practical stochastic model, they approach it from a theoretical point of view. Moreover, their main focus is on the time-independent case.

By making use of multiple modes of transportation [16], travel times can be reduced. The main issue in multimodal transportation systems is the modeling of the (different) information of the different transport modes in a more or less uniform way. The most common solutions preserve the network of each mode and interconnect these by means of trans-shipment links, as described in chapter 2. For a complete overview of how multimodal networks can be modeled we would like

to refer to section 2.2.1 of chapter 2. Furthermore, when routing multimodally, other objectives gain importance [17], such as number of transfers, transfer cost, waiting time, etc.

For more accurate routing, multicriteria algorithms should be used [18], similar to the algorithm that was presented in chapter 4. For simplicity, we opt to omit these other objectives in this proof-of-concept and focus on the travel time, which is usually the most important objective. Nevertheless, we incorporate the waiting time as part of the total travel time. The multimodal system presented by Bielli et al. [19] resembles the system presented in this article the most. It incorporates multiple modes of transportation and deals with the different travel time information of each of these modes. Moreover, an origin-destination algorithm is presented to find the fastest path between two locations. The main difference with the research presented here is that Bielli et al. do not take into account the stochastic nature of the travel time information.

In order to accurately predict the travel time between two locations, we made use of both time table information and travel time data extracted from cellular networks [20]. This data was provided by industrial companies within the IBBT ICON project MobiRoute [21].

5.1.2 Context and Resources

To better understand this chapter, and appreciate the approach taken to tackle the problems, it is necessary to describe in some detail the context of our work. Since a few years, a major industrial player conquered the commercial field of traffic information by providing up-to-date and real-time information on the current status of traffic flow on the Belgian road network and provide a limited service of traffic forecasting. An interesting way to achieve this has proven to be very successful in measuring the actual throughput on any given moment of the day: they have a business agreement with one of the major cellular phone providers in the country, from which they receive real-time handover timings of cellular phones. This means that all phone subscribers are continuously tracked (anonymously), allowing to trace their routes and to measure the time it took to drive each road segment. This is done by complex calculations involving triangulation and the mapping of the signals on road maps. In practice, there are lots of phone users driving, which provide timings on how long it takes to drive a certain segment of our road network. This allows not only for knowledge about current traffic jams, but collecting all this data provides an enormous amount of historical travel time data, which enables accurate predictions for future jams.

The company contacted our research group, in order to develop a route planner which takes into account the historical data, and provides accurate estimates on the travel times together with uncertainty intervals. Indeed, it is observed that a

large number of people using route planners are not interested in the shortest route from A to B, but the fastest route, together with an uncertainty interval which indicates the chances of delay. The company provided us with a road network consisting of 53010 nodes and 96286 directed links between these nodes. Together with this, a database of timing measurements was provided in the form of data from 10 Tuesdays throughout the year. For every 15 minutes, an average of the measurements was calculated. When measurements were missing (due to external measurement system failures), the values were interpolated.

Next to this, time table information of the Belgian railroad network was provided. This way, a multimodal system can be built, making use of the time-dependent and stochastic data of the road network and the time-dependent deterministic data of the railroad network.

The aim was to develop an industrial-strength (i.e. realistic and robust) commercial platform for probabilistic multimodal routing. To that aim, we had to make sure that the data structures and algorithms were both robust and scalable. If necessary, a trade-off was made between theoretical issues and practical feasibility.

An overview of the system that was developed is given in figure 5.1. We start from both the network data and the (historical) travel time data. After preprocessing this travel time information in order to form practical cost objects (i.e. 'data processing'), a graph is built (i.e. 'graph modeling'). This graph is modeled to incorporate all modes of transportation more or less uniformly. Subsequently, a routing algorithm (i.e. 'routing algorithm') was developed that finds the fastest routes in this graph and is able to cope with the variety of the available travel time information. Starting from an origin, a destination and a departure time, this algorithm finds the 'fastest' route together with a stochastic distribution of the travel time on this route.

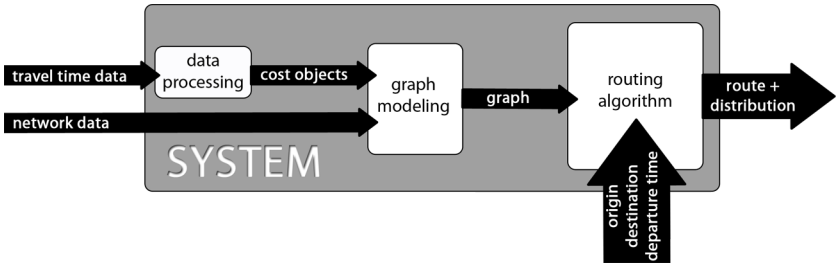

Figure 5.1: Overview of the multimodal time-dependent and stochastic routing system.

5.1.3 Outline

In this chapter we present a case study of a practical industrial-strength multimodal routing system, which efficiently calculates the routes while taking into account the characteristics of the travel time data, such as time-dependency and uncertainty. In the next section the time-dependent multimodal network model is repeated briefly, in which a lot of attention is given to the cost modeling as costs can either be static or time-dependent and deterministic or stochastic. Subsequently (see section 5.3), a novel time-dependent and stochastic shortest path algorithm is presented, which is based on the algorithm of Dijkstra. In section 5.4 it is shown that indeed a time gain can be realized by routing time-dependently, stochastically and multimodally. Moreover, when the stochastic travel time information is used, more reliable paths are calculated. It is demonstrated that, by making use of our data structures, these paths (with additional stochastic information) can be calculated in an acceptable time. Finally, this chapter is brought to an end with a number of conclusions. It should be noted that the routing system presented here has been commercialized and a stripped down (bi-modal) version for the Belgian (road-railroad) network can be found online [22].

5.2 The Multimodal Time-Dependent and Stochastic Network Model

To better understand the remainder of this chapter, this section briefly recapitulates the multimodal network model that was presented in chapter 2. Moreover, a number of small practical changes are introduced. This network model results in one large network that contains a number of mode-specific networks interconnected by trans-shipment links. Each of these mode-specific networks is modeled in more or less the same way and the differences between the different transportation modes are modeled in the cost objects that are assigned to the links.

5.2.1 The Multimodal Network

As stated in chapter 2, we opted for a layered and time-dependent approach. The layered network model preserves the mode-specific networks. These networks of the different transportation modes are connected by means of trans-shipment links between 'geographically co-located' nodes of different modes. Moreover, in the time-dependent network model all travel time information is modeled in the cost objects that are assigned to the links. This model allows to work with different cost models for the different transportation modes as will be illustrated in the following subsection.

Two minor adaptations have been added to the network model. Firstly, in order

to model transfers in public (unimodal) transportation networks, extra nodes and links were added that represent bus stops or stations and their platforms. One node depicts the station (or bus stop) itself and dummy nodes are added for every service (e.g a bus line) that stops at this location. Getting off and on the bus can then be modeled as a link between the dummy node and the station node.

In reality, for example in a road network, the number of geographically co-located nodes is small. Cars often cannot park in the stations themselves, but have to use a nearby parking. Therefor, we opted to identify the trans-shipments differently. The nodes of every transportation mode are now connected to the m closest nodes in the densest network (usually this is the road network), as in this network almost all geographic locations are reachable. Trans-shipments between two different (non-road) transportation modes always pass through this (road) network, which is close to reality. The trans-shipment links now represent walking from the end point of one mode of transportation (e.g. a parking) to the begin point of another mode (e.g. a train station). It should be noted that when the multimodal network contains a walking mode, the original definition of trans-shipment links can still be used.

In the end, one large multimodal network is constructed which consists of all mode-specific networks connected together by trans-shipment links. Let us consider a multimodal transport network with m modes of transportation. The mode-specific network of mode i ($i = 1, \ldots, m$) then can be represented by $G_i = (V_i, E_i)$ with V_i a set of nodes and E_i a set of links $(v, u) : v, u \in V_i$. The trans-shipment links are represented by an equivalence relation $R = \{(u, v) | u \in V_i; v \in V_j; 1 \leq i, j \leq n; u \; close \; to \; v\}$. The complete multimodal network is denoted by $G = (V, E)$ with $V = \bigcup V_i$ and $E = (\bigcup E_i) \bigcup R$. In this case study we opted to omit the access layer and assume that the transportation modes at the origin and the destination node are known.

5.2.2 Cost Modeling

As mentioned before, all mode-specific information is modeled in the cost objects. In this case study, costs represent travel times. These travel times can either be static or time-dependent. If the time to traverse a link is not dependent on the hour of the day, it is considered to be static and can be represented as a single value. If, on the other hand, this cost is dependent on the hour of the day the specific link is taken, it can no longer be modeled as a single value, but as a travel time function representing the travel time cost in function of the hour of the day. Moreover, some of the travel time information has an amount of uncertainty attached to it. This leads to stochastic travel time information. Instead of a single value, a time cost at a certain hour of the day now is represented by a stochastic distribution. An overview of these different time costs is given in table 5.1, in which the two

	STATIC	TIME-DEPENDENT
DETERMINISTIC	single value	function of values
STOCHASTIC	single distribution	function of distributions

Table 5.1: Travel Time Costs

characteristics (static/time-dependent and deterministic/stochastic) are combined. We will now discuss a practical example of each of these cases in more detail.

Trans-shipment costs are mostly considered both static and deterministic. It always takes the same amount of time and this time is independent of the hour of the day. Thus these costs can be represented by single values.

In the railroad network (and all other networks which are bound to time tables), we consider the costs to be time-dependent and deterministic. The travel time, which consists of both a waiting time in the station and a driving time, is dependent on the hour of the day, but considered to be deterministic and thus independent of external factors. Time tables can easily be translated to a travel time function as indicated in figure 5.2 where the lowest points represent a train leaving the station and the lines model the waiting in the station.

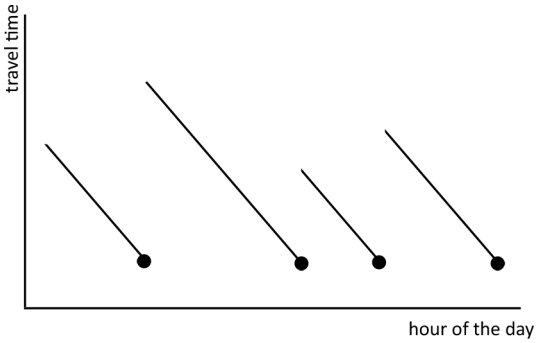

Figure 5.2: Travel Time Function in Railroad Network.

Since waiting times are proportional to the hour of the day, this function can be represented by a number of departure times, together with their corresponding driving time. Suppose the driving time at departure time T is $f(T)$, then the travel time $f(t)$ at time t between two known departure times T_1 and $T_2(T_1 < t < T_2)$ can be calculated as follows:

$$f(t) = f(T_2) + (T_2 - t) \tag{5.1}$$

A binary search algorithm can be applied to find the neighboring departure times.

If memory consumption is not an issue, a speedup can be realized by calculating the travel time for every minute (i.e. the finest detail of the time table) and storing it in an array which then can easily be accessed by translating the specific hour of the day to the corresponding index. This leads to faster travel time calculations, but has a major impact on the memory consumption. For a complete description of this model we would like to refer to the second part of section 2.4.3 of chapter 2. Since the networks that are used in our proof-of-concept system are relatively small, we can make use of these memory-intensive arrays to store the travel time information, with the advantage of fast lookup operations.

In an underground network, subway trains leave the station every k minutes, but are not bound to a specific time table. This introduces an amount of uncertainty which leads to a stochastic and static (assumed day and night are equal) time cost. Suppose for example, a subway train leaves every 6 minutes and driving to the next station takes 2 minutes. Then the travel time of this link lies somewhere between 2 and 8 minutes depending on the time one arrives in the subway station. We will make use of a cumulative distribution which contains for each probability the maximum travel time. In the example we have a 100% chance to travel 8 minutes or less, while we have 50% chance of traveling less than 5 minutes. In order to both save memory and simplify the calculations, we will represent a stochastic distribution by m values, corresponding to m predefined percentiles. This number m results in a trade-off between accuracy and performance. More percentiles increase the accuracy of the results, but the more storage space is needed. For a complete description of the stochastic cost model we would like to refer to section 2.4.4 of chapter 2.

In a road network, travel time costs are both time-dependent and stochastic. Due to possible traffic jams during rush hours, the travel time in a road network is dependent on the hour of the day. Moreover, external factors (for example traffic accidents) add even more uncertainty to the estimated time of arrival. The time-dependent and stochastic travel time costs are modeled as a function of distributions. In our use case, travel time data was collected for every quarter of an hour during multiple (similar) days. Stochastic distributions are constructed from this data for each quarter of an hour for each day of the week (i.e. Monday, Tuesday, ...). For the times between these quarters, linear interpolation is used between the corresponding percentiles. The x-th percentile of the travel time distribution $F(x,t)$ at time t between two timestamps (i.e. quarters) T_1 and $T_2 (T_1 < t < T_2)$ then can be calculated as follows:

$$F(x,t) = F(x,T_1) + \frac{F(x,T_2) - F(x,T_1)}{T_2 - T_1}(t - T_1) \tag{5.2}$$

A more accurate travel time function can be obtained by adding more timestamps, at the cost of more intensive memory usage.

5.3 The Routing Algorithm

As indicated in the previous section, a multimodal graph consists of a set of nodes V and a set of links E, which contains mode-specific links and trans-shipment links. Each of these links has assigned to it a travel cost, which can either be a single value (static deterministic), a function of values (time-dependent deterministic), a distribution (static stochastic) or a function of distributions (time-dependent stochastic).

In this section, we present a shortest path algorithm that is equipped to deal with this variety of travel time information. As both static and deterministic information can easily be translated to time-dependent and stochastic functions respectively (see section 2.4.5 of chapter 2), we have developed an algorithm which assumes all travel time information to be time-dependent and stochastic. Furthermore, in this case study (with only two modes of transportation), we opted for the so-called 'taxi-model', in which it is assumed that a cab is present at any location. This way, trans-shipment to the road network is always possible. There are multiple possible solutions to model the availability of a car: making the network dynamic and only adding trans-shipment links in the locations where a car is present, modeling it in the routing algorithm, etc.

The problem that is addressed here can be described as follows. Given an origin and destination node together with a departure time, find the path between the origin and the destination that has the smallest distribution of the arrival time, according to a comparison measure that will be defined further on. The network, in which the routing happens, contains multiple modes of transportation and the travel time costs can either be static or time-dependent, and either deterministic or stochastic.

The algorithm presented here is based on the algorithm of Dijkstra (see section 1.3.1), a label-setting algorithm with labels representing the shortest distance between the origin and the specific node. During initialization the label of the origin node is set to zero and all other labels are set to infinity. The set T contains all (non-permanent) nodes whose labels have been updated by the algorithm. In every iteration the node with the lowest label is removed from T and made permanent. Then, all labels of the neighboring nodes are updated. More specifically, if the sum of the label of the investigated (permanent) node and the link cost is smaller than the previous label of the neighboring node, then the label is changed to this sum. This process is repeated until the destination node has been made permanent. The algorithm of Dijkstra requires all link costs to be non-negative values, which is the case as we are working with travel times here.

The algorithm described above needs to be altered in two ways. Firstly, the time-dependent character of the links costs needs to be taken into account when determining the travel time on a link. Secondly, labels are no longer single val-

ues but distributions. We need to define how to combine and compare stochastic distributions with each other.

In order to make this algorithm time-dependent, some small adaptations are needed. Labels now represent the earliest time at which one can arrive in a specific node starting at the origin node at a specific departure time. In the initialization phase the origin label is set to this departure time. Furthermore, the travel time of a link is dependent on the time at which this link is traversed. For a link (u, v) this time is equal to the label of the start node $l(u)$. The value of a tentative new label can then be calculated as follows:

$$l'(v) = l(u) + c(u, v, l(u)) \tag{5.3}$$

with $c(u, v, l(u))$ the travel time on link (u, v) at time $l(u)$. When the algorithm is finished, the label of the destination node represents the earliest arrival time, starting at a predefined departure time in the origin node.

Making this algorithm stochastic introduces new challenges. Labels are no longer single values, but probabilistic distributions of travel times, represented by a number of percentiles. Two operations need to be defined: comparing labels and updating labels. Deciding which of two labels is the best is no unambiguous process. To simplify the calculations, we opted to compare a single percentile. In most cases the 50% percentile suffices, but if the user is interested in a more certain solution a higher percentile should be used, such as the 90% percentile. A more accurate comparison would involve all percentiles and applying an algorithm similar to the multi-objective one (see chapter 4), but this would require a higher amount of computing power.

To define the sum of two labels we investigate two extreme cases, namely one in which the distributions of the links are completely correlated and one in which they are completely uncorrelated (and stochastically independent). While in the former the pointwise sum can be used, the latter needs the convolution product to combine two labels. In the stochastic case, the label of node v can be represented as an array of m percentiles $[l_{p_i\%}(v)], i = 0, \ldots, m$ (note: in the remainder of this article we will omit the boundaries of i and use square brackets to denote an array over this index i). The pointwise sum of two labels $[l_{p_i\%}(v)]$ and $[l_{p_i\%}(w)]$ then can be defined as:

$$[l_{p_i\%}(v)] + [l_{p_i\%}(w)] = [l_{p_i\%}(v) + l_{p_i\%}(w)] \tag{5.4}$$

Calculating the convolution product is more complex. In calculus, the convolution product of two functions f and g is defined as

$$(f \star g)(t) = \int_{-\infty}^{+\infty} f(x)g(t - x)\mathrm{d}x \tag{5.5}$$

When the distributions are represented as a number of percentiles, the convolution product of two labels $[l_{p_i\%}(v)]$ and $[l_{p_i\%}(w)]$ can be determined as follows.

For each i and j, $s_{ij} = l_{p_i\%}(v) + l_{p_j\%}(w)$ is calculated. The requested percentiles then can be extracted from the distribution of these s_{ij} values. It should be noted that determining percentiles in large arrays can be sped up by making use of order statistics [23]. The order statistic algorithm returns the x-th element of a sorted array without sorting the complete array. This is illustrated in figure 5.3.

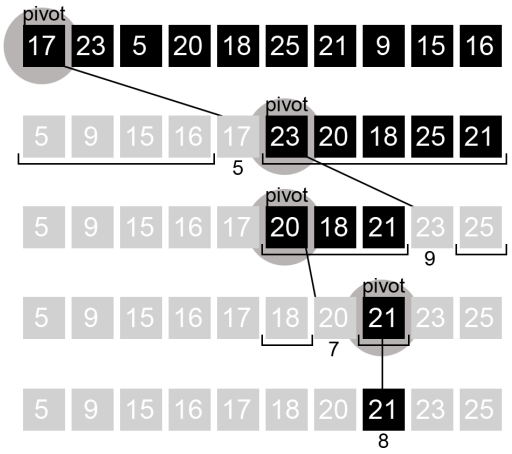

Figure 5.3: Example of the order statistic algorithm. Here, the 8-th element is determined of an array that is sorted in increasing order. In every iteration a pivot element is identified and the remaining elements of the part of the array that contains this searched element are sorted partially around this pivot. In this way the area where the element is situated is diminished.

In reality, some of the links are correlated with each other, while others are not. For example, two consecutive sections on a highway are closely correlated. A traffic jam in one of the sections increases the chance of a traffic jam in the other section. On the other hand, a section on the highway has nearly no correlation with a small road besides it. As determining all correlations between all links of the network and subsequently combining the costs of these links appropriately is very burdensome, we will work with these two extreme cases. The user then can decide how much he wants to tempt his fate making use of these calculated boundaries.

Below, the pseudocode of the algorithm as described above is given. It should be noted that $[p]$ represents a constant distribution with value p.

FindBestPathBetween(o,d)

```
1     forall(v ∈ V)
2         l(v) := [∞];
3     l(o) := [departure_time];
4     P := EMPTY_SET;
5     T := {o};
6     while(!(T is empty) && !(d ∈ P)){
7         u := T.removeMin();
8         P.add(u);
9         for(n:  neighbor(u)){
10            l_new:= l(x) ⋄ translate(c(u,n,l(u)));
11            if(l_new < l(n)){
12                l(n) := l_new;
13                T.add(n);
14            }
15        }
16    }
```

The first three lines are the initialization phase. The label of the origin node is set to a constant distribution of the departure time (line 3), while all other labels are set to the infinity distribution (line 1-2). Subsequently, the temporary set is initialized with a singleton containing the origin node (line 4). As long as the destination node is not permanent and there are still elements in the temporary set (line 6), the following instructions are executed. Firstly, the node with the smallest label u is removed from the temporary set (line 7) and added to the permanent set (line 8). For each neighbor n of this node, a new tentative label is determined by combining the label of u with the time-dependent and stochastic cost of the link between u and n (line 10). If the cost of the link is static or deterministic, it is translated to a time-dependent and stochastic one. We assume the distributions to be either completely correlated or completely uncorrelated, as described above. The operator $\diamond$ can thus either represent a pointwise sum or a convolution product. When this tentative label turns out to be smaller (w.r.t. the comparison operator as described above) than the previous label of n, it is changed and the temporary set is updated (line 11 to line 13). It should be noted that this algorithm is similar to the algorithm of Dijkstra in which the labels are defined differently.

5.4 Results

In this section it is shown that indeed a travel time gain can be realized by routing time-dependently (in comparison to routing statically), stochastically (in comparison to deterministically) and multimodally (in comparison to unimodally). Moreover, when routing stochastically, additional information about the travel time is provided to the user. After a general description of the setup of the system and the experiments (see section 5.4.1), some light is shed on the execution times of the different algorithms (see section 5.4.2). Subsequently, we will investigate the deterministic case and show how better (i.e. faster) routes can be calculated by making use of time-dependent information (see section 5.4.3). Section 5.4.4 deals with the core issue of our research, namely stochastic routing. It will be demonstrated that, indeed, by making use of additional stochastic information, more reliable routes can be calculated. Here, we will examine the two extreme cases, namely assuming all links are completely correlated and assuming all links are completely uncorrelated. In section 5.4.5, we will have a look at multimodal routing in comparison with unimodal routing. It will be shown that making use of multiple modes of transportation may have its advantages in particular cases.

5.4.1 General Descriptions of Setup

The routing system as described in this article was implemented in Java (version 1.6.0-18) on a machine with the following specifications: Intel ® Core ™ 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM.

Currently, the network contains only two modes of transportation, namely road and railroad, but we are gathering data of the other transport modes to further expand it. For the experiments we made use of a Belgian network (as for this network we have very detailed travel time information at our disposal) composed of a road network of 53 010 nodes and 96 286 links and a railroad network of 551 nodes and 1 716 links. For each train station, three trans-shipment links were added connecting it to the road network.

Stochastic distributions, represented as 5 percentiles (i.e. the 10%, 30%, 50%, 70% and 90% percentile) were calculated for the road links based on historical measurements of travel times.

Railroad time tables were analyzed and translated to the data structure as presented in figure 2.6 of chapter 2. We opted for an array with a travel time for every minute of the day, as there is sufficient memory available for these networks. While for the railroad network a time-dependent granularity of 1 minute was used, the road network employs a granularity of 15 minutes, due to the data restrictions of the measurements. Interpolation, as shown in section 5.2, is applied for the times in between timestamps.

In the experiments the different algorithms are compared to one another. To

	Calculation Time (ms)
static deterministic	107
time-dependent deterministic	209
static stochastic (correlated)	307
static stochastic (uncorrelated)	376
time-dependent stochastic (correlated)	433
time-dependent stochastic (uncorrelated)	489

Table 5.2: Average calculation times of the different algorithms

compare the (travel time) costs of the resulting paths, a difference measure was defined. The amount of difference (Δ), in percentage, of the cost of the path calculated by algorithm (2) compared to the cost of the path calculated by algorithm (1) is

$$\Delta = \frac{cost_{(1)} - cost_{(2)}}{cost_{(1)}} \tag{5.6}$$

Both the average and the maximum values of this parameter will be reported.

All the experiments of the following subsections were executed on the multimodal network as described above, unless explicitly stated otherwise.

5.4.2 Calculation time

First and foremost, we investigated the performance of the algorithms. Table 5.2 shows the average calculation times (in milliseconds) of all algorithms that were used during the experiments.

The static deterministic algorithm is in fact the standard Dijkstra algorithm in a network where all travel time costs are assumed to be constant. When the link costs are time-dependent travel times, we opted to use the values of a quiet timeslot (i.e. a timeslot outside of the rush hours). This represents driving when there is little traffic, such as driving at night. When the travel time information is stochastic, the median values (i.e. 50% percentile) are used.

The time-dependent deterministic algorithm takes into account the time-dependent travel time information but still assumes the travel times to be single values. For links with stochastic distributions assigned to them, the median values were used.

The static stochastic algorithm makes use of a single stochastic distribution for each link. When the travel times are time-dependent, again we opted to use a distribution of a timeslot in the night.

In the time-dependent stochastic algorithm, all available travel time information was used to calculate the best (fastest) routes. The stochastic algorithms were

executed for two extreme cases, namely assuming that the links are completely correlated (using the pointwise sum) and assuming that they are completely uncorrelated (using the convolution product).

The results of table 5.2 are averages of 10 000 shortest path calculations with randomly chosen origin-destination pairs and random departure times. The calculation times where measured with a timer that has a precision of nanoseconds and an accuracy of microseconds. It can be seen that taking into account the time-dependent information slightly slows down the calculations (static vs. time-dependent). Moreover, for the stochastic algorithms, assuming the links are completely uncorrelated consumes more calculation time than assuming that they are completely correlated. This is as expected as the convolution product is a more complex operation than the pointwise sum.

The results show that for these algorithms the two most crucial operations are determining the travel time of a link at the time this link is traversed and combining this travel time information. While in the static deterministic case the constant cost values of the links can just be added up, in the time-dependent cases the exact travel time needs to be determined by making use of the arrival time in the start node of the link in question. For the stochastic algorithms combining the distributions even adds more computational complexity to the problem. These calculation times are acceptable, as they stay below 500 ms while providing better (i.e. faster) and more reliable routes.

5.4.3 Deterministic Routing: time-dependent vs. static routes

In this section, it will be demonstrated how better (i.e. faster) routes can be calculated by making use of the time-dependent travel time information. We will start with a small example to show that routing time-dependently indeed has its advantages. In the unimodal road network a route was calculated from Ghent to Liège for different departure times. This route passes through Brussels which is a known bottleneck of the Belgian road network. Parts of these routes, namely those around Brussels, are depicted in figure 5.4, when leaving at 6 am, 7 am and 8 am. It can be seen that, during rush hour, the ring road around Brussels is avoided. At 7 am, the highway before Brussels is left earlier in order to avoid the first part of the ring way. At 8 am, a road going through the center of Brussels proves to be faster than waiting in the traffic jams on the ring road.

Table 5.3 shows the actual travel time gain realized by the time-dependent deterministic algorithm in comparison to the static deterministic algorithm. Here, 10 000 paths were calculated between random origin-destination pairs at a random departure time with both the static and the time-dependent deterministic algorithm. Subsequently, both the time-dependent deterministic and the time-dependent stochastic costs of the resulting paths were calculated. For this, the

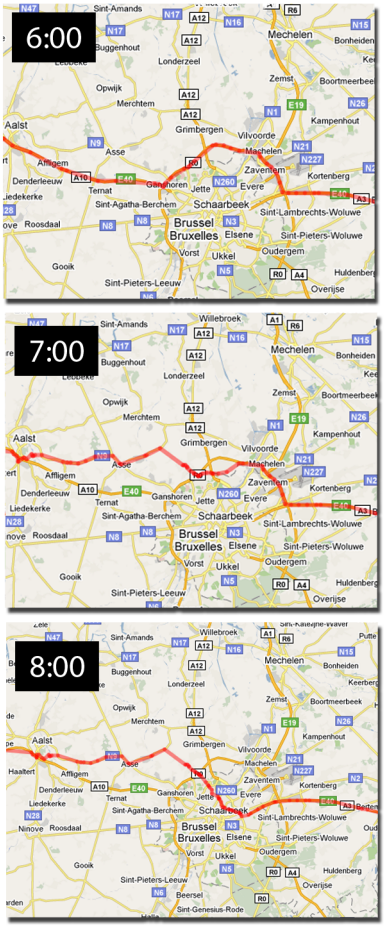

Figure 5.4: Time-Dependent Routes in Belgian Road Network.

travel time cost	**average Δ (%)**				
time-dependent deterministic	14.25				
	10%	30%	50%	70%	90%
time-dependent stochastic (correlated)	-2.59	-0.42	1.19	3.24	3.68
time-dependent stochastic (uncorrelated)	0.78	1.27	1.66	2.04	2.35
	maximum Δ (%)				
time-dependent deterministic	56.96				
	10%	30%	50%	70%	90%
time-dependent stochastic (correlated)	25.92	32.09	35.61	42.75	49.30
time-dependent stochastic (uncorrelated)	34.31	34.53	34.65	35.91	36.93

Table 5.3: Comparison of the time-dependent travel times (both deterministic and stochastic) of the paths calculated by the static deterministic algorithm (1) and the time-dependent deterministic algorithm (2)

actual travel time costs were removed from the paths and recalculated assuming that all links of the paths are time-dependent deterministic and time-dependent stochastic respectively. The differences between these travel time costs (i.e. the recalculated costs of the paths calculated by both algorithms) are reported in the table.

First of all, the time-dependent deterministic travel time costs were compared. The paths calculated by the time-dependent deterministic algorithm are on average 14.25% shorter than the paths of the static deterministic algorithm, with a maximum value of 56.96%. This proves that, indeed, making use of the time-dependent travel time information in the routing algorithm has its benefits. In 85.12% of the cases, a shorter route was found by the time-dependent deterministic algorithm. In the other cases, the paths are equal to the ones calculated by the static algorithm.

Next, we looked at the stochastic costs, for both extreme cases, and compared the corresponding percentile values with one another. Despite the fact that the average values are relatively small, if we look at the maximum values, a remarkable travel time gain can be realized by routing time-dependently instead of statically. Moreover, the amount of difference between the two algorithms is higher for the higher percentile values. This means that for users who want more reliability, routing time-dependently really pays off. For the lower percentiles, routing time-dependently is on average even slightly worse than routing statically assuming the links are completely correlated. If we compare the case in which it is assumed that all links are completely correlated (using pointwise sum) and the case in which

they are assumed to be completely uncorrelated (using convolution product), we see that the difference between the lowest and the highest percentiles is larger when using the pointwise sum. This is as expected, since in the pointwise sum, the extreme values are combined with each other, while the convolution product takes into account all values of the distributions, smoothing out the extreme values.

5.4.4 Stochastic Routing

This section deals with stochastic routing. We will demonstrate that by making use of the additional stochastic travel time information, more intelligent routes can be calculated. After an example that illustrates the difference between the two extreme cases (i.e. the links are completely correlated vs. completely uncorrelated), two aspects will be investigated. Firstly, we will compare the static stochastic algorithm with the time-dependent stochastic algorithm and report on the travel time gain that can be realized by routing time-dependently. Subsequently, a comparison is made between the time-dependent deterministic algorithm and the time-dependent stochastic algorithm to illustrate the strength of using the additional stochastic information to produce more intelligent routes.

With respect to combining stochastic distributions, two extreme cases were investigated: completely correlated links (with the pointwise sum) and completely uncorrelated links (with the convolution product). In figure 5.5 an example is given of a day overview for one specific path in both scenarios. The time-dependent stochastic shortest route was calculated for each time of the day in the unimodal (road) network and the resulting percentiles were plotted. In this example distributions were compared w.r.t. their 50% percentiles.

The rush hours can be readily observed, namely a major one in the morning (from 7 am to 9 am) and a smaller one in the evening (from 4 pm to 6 pm). Moreover, it should be noted that the 50% percentile values are similar with only differences of a couple of seconds. The main difference between the two calculation methods lies in the other percentiles. While the standard deviation in the uncorrelated case is small, the percentiles of the correlated case are more scattered. This can be explained intuitively as the convolution product takes into account all values of the distributions, resulting in extreme values being weakened by the others. In the completely correlated case (pointwise sum) a high probability of congested traffic on one link of the network results in a higher probability on all other links following this link. Extreme percentile values have a direct influence on the corresponding percentile of the resulting distribution.

As mentioned before, the percentiles of the actual distribution lie somewhere between these two extremes. This means that for the percentiles under the 50% percentiles (which is used here for comparison) the uncorrelated case is an upper bound and the correlated case a lower bound. The opposite is true for the per-

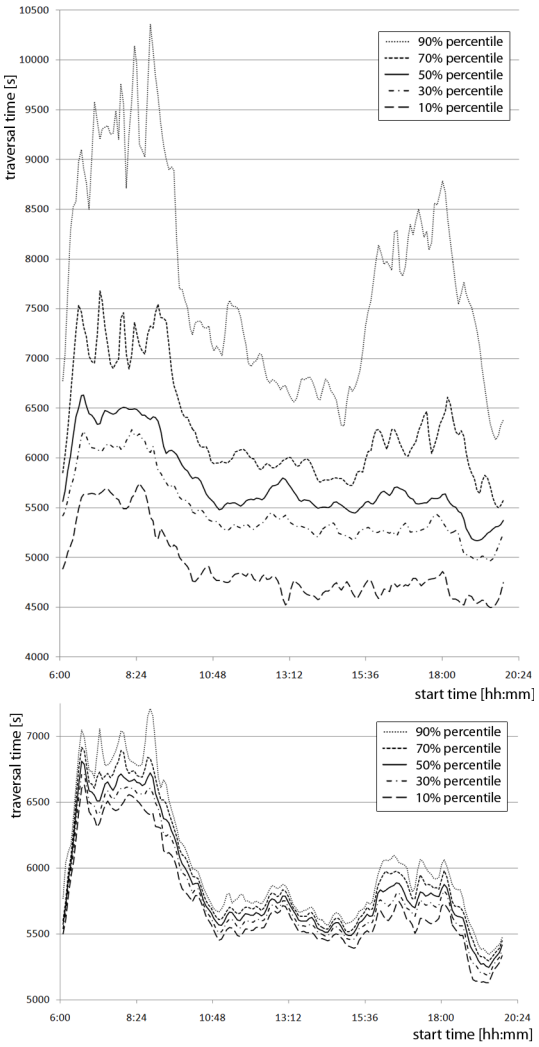

Figure 5.5: Stochastic Day Overview.
Top: correlated links (pointwise sum) - Bottom: uncorrelated links (convolution product)

assuming links are ...	**average Δ (%)**				
	10%	30%	50%	70%	90%
completely correlated	2.20	2.31	2.14	3.81	7.29
completely uncorrelated	1.23	1.59	1.85	2.04	2.31
	maximum Δ (%)				
	10%	30%	50%	70%	90%
completely correlated	35.90	36.02	37.16	39.44	50.39
completely uncorrelated	33.17	34.44	35.82	36.82	37.72

Table 5.4: Comparison of the travel times of the paths calculated by the static stochastic algorithm (1) and the time-dependent stochastic algorithm (2)

centiles above the 50% percentile. If the user wants more certainty of its arrival time, i.e. he wants to be sure to be at his destination at the latest at a certain chosen time, the 90% percentile can be used to compare labels. In this case the pointwise sum calculations always serve as a lower bound.

In the remainder of this section stochastic distributions will be compared by their 90% percentile values.

5.4.4.1 Static Stochastic vs. Time-Dependent Stochastic

Table 5.4 shows the average and maximum amount of difference between the travel times of the paths calculated by the static stochastic algorithm and the time-dependent stochastic algorithm for the two extreme cases. These values were calculated from 10 000 shortest path calculations between random origins and destinations at random departure times. Similar to the statements made in section 4.3, indeed a travel time gain can be realized by making use of the time-dependent character of the travel time information. Despite the relatively low average values, the maximum travel time gain is noteworthy. To compare labels with one another, we made use of the 90% percentiles, so the resulting paths are optimized with respect to this percentile. The results show that indeed the highest travel time gains are realized for this percentile.

Next we looked at the amount of better paths (w.r.t. the 90% percentile values of the distributions) that were calculated by routing time-dependently. In the situation that all links are assumed to be completely correlated, in 81.82% of the cases a better path is found, while in the other situation (all links are completely uncorrelated), this comes down to 65.06% of the cases. This proves that it is worth routing time-dependently when the information is available.

In conclusion, for users who are interested in reliable routes (making use of the 90% percentiles), by routing time-dependently instead of statically a travel

assuming links are ...	**average Δ (%)**				
	10%	30%	50%	70%	90%
completely correlated	-1.36	-0.97	-0.71	-0.75	2.69
completely uncorrelated	0.88	0.80	0.74	0.66	0.75
	maximum Δ (%)				
	10%	30%	50%	70%	90%
completely correlated	22.78	18.94	14.27	18.36	81.32
completely uncorrelated	19.32	15.31	13.77	13.77	51.03

Table 5.5: Comparison of the travel times of the paths calculated by the time-dependent deterministic algorithm (1) and the time-dependent stochastic algorithm (2)

time gain of up to 50% can be realized in the case all links are correlated, and nearly 38% in the case the links are completely uncorrelated.

5.4.4.2 Time-Dependent Deterministic vs. Time-Dependent Stochastic

In this section we will investigate the advantages of using the stochastic travel time information in the algorithm and compare the time-dependent deterministic algorithm with the time-dependent stochastic one. Again the shortest paths were calculated for 10 000 randomly chosen origin-destination pairs at random departure times. The stochastic travel time was then calculated for the resulting shortest paths and all percentile values were compared to one another. The average and maximum amounts of difference are shown in table 5.5.

The average values of table 5.5 are small, meaning that on average not that much travel time gain can be realized by routing stochastically. In the case all links are assumed to be correlated, for the lower percentiles the paths calculated by the stochastic algorithm are even worse than those calculated by the deterministic algorithm. However, the paths calculated by the stochastic algorithm are more reliable than those calculated by the deterministic algorithm. Looking at the 90% percentiles (for which these paths were optimized, i.e. minimized), we see that for this percentile indeed a travel time gain is realized by the stochastic algorithm. This means that more reliable paths are calculated.

If we look at the maximum difference of the 90% percentile values, we see that travel time gains can be realized of up to 81.32% and 51.03% for the correlated and the uncorrelated case respectively, which is quite remarkable. Users who are interested in reliable routes can really benefit from routing stochastically.

Subsequently, for each origin-destination pair the 90% percentiles of the travel time distributions of the paths found by both algorithms were compared with each other. When it is assumed that all links are uncorrelated, 53.30% of the paths

are actually better, while only 12.75% of the paths are worse. For the case that all links are correlated, these numbers are 37.61% and 33.42% respectively. As more reliable routes are produced, routing stochastically indeed is valuable. So, in 87.25% (completely uncorrelated) and 66.57% (completely correlated) of the cases the paths are as well as or better than the paths calculated by the deterministic algorithm. Moreover, the resulting routes are more reliable. This means that making use of the stochastic travel time information when determining the shortest path, indeed is valuable.

5.4.5 Multimodal vs. Unimodal Routing

The system presented here is multimodal in the sense that both roads and railroads are taken into account. In this section we will determine whether traveling multimodally indeed has advantages (with respect to the travel time) over traveling unimodally. We start with a small example, which is similar to the example of section 5.4.3, as it also concerns the bottleneck around Brussels in the Belgian road network. In figure 5.6 the multimodal routes between Leuven (on the right) and Ghent (on the left) are depicted for different departure times. These routes were calculated with the time-dependent deterministic algorithm. While in the example in section 5.4.3 the traffic jams around Brussels were avoided by taking other routes in the road network, here the traffic jams are avoided by making use of another mode of transportation, viz railroads. Outside the rush hour, for example at 6 am, the road is chosen to be the best mode of transportation. Nevertheless, when traffic is jammed around Brussels (in the case of leaving at 7:30 am) the train becomes more favorable. Combinations of both transport modes are also possible. For example, at 7 am the best route is the one leaving by car in Leuven and taking the train in Brussels to continue to Ghent.

Next, a more systematic comparison was performed. Routes were calculated between 10 000 random origin-destination pairs in both the unimodal (road) and the multimodal network (making use of the time-dependent stochastic algorithm). As the example indicates that during rush hours the multimodal network might be more favorable than outside these rush hours, these two cases were examined separately, namely departing at 7:30 am (during the rush hour) and departing at 2 pm (outside of the rush hour). Results are shown in table 5.6.

The average travel time gain by using the multimodal network is almost negligible outside of the rush hours. Moreover, in approximately 5% of the cases (5.58% and 4.46% for the completely uncorrelated and correlated case respectively) a better path is found in the multimodal network. During these hours, taking the car is in most cases more advantageous than using multiple modes of transportation. It should be noted that, if a unimodal path is the best option to get from the origin to the destination, this path will be returned by the algorithm, even

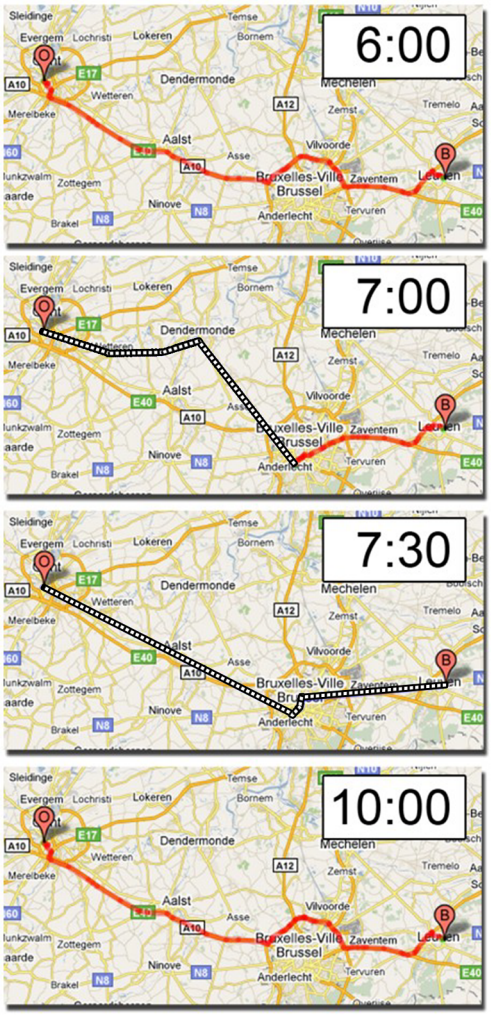

Figure 5.6: Multimodal Routes (Leuven-Ghent). straight line = road - dotted line = railroad

DURING RUSH HOUR (7:30 am)

assuming links are ...	**average Δ (%)**				
	10%	30%	50%	70%	90%
completely correlated	-1.32	-0.41	-0.09	1.50	1.69
completely uncorrelated	0.72	0.79	0.83	0.89	0.89
	maximum Δ (%)				
	10%	30%	50%	70%	90%
completely correlated	38.37	40.16	39.83	41.73	41.89
completely uncorrelated	41.31	41.28	41.02	41.51	41.86

OUTSIDE RUSH HOUR (2 pm)

assumed links are ...	**average Δ (%)**				
	10%	30%	50%	70%	90%
completely correlated	-0.23	-0.03	0.02	0.12	0.31
completely uncorrelated	0.16	0.17	0.18	0.19	0.19
	maximum Δ (%)				
	10%	30%	50%	70%	90%
completely correlated	4.20	6.33	7.14	19.77	24.33
completely uncorrelated	14.01	14.19	14.67	15.33	15.88

Table 5.6: Comparison of the travel times of the paths in the unimodal network (1) and the multimodal network (2)

when routing in a multimodal network. When traveling during the rush hours, multimodal transportation may result in travel time gains of up to more than 40%. Moreover, during the rush hours in approximately 15% of the cases (14.13% and 14.87% for the completely uncorrelated and correlated case respectively) a better path (according to the 90% percentiles) is found in the multimodal network.

All multimodal paths were inspected more closely and it was observed that the origin and/or destination of these routes are situated close to train stations, as expected. If the origin and the destination are situated far from train station, a multimodal route would consist of a rather long path from the origin to the closest train station, followed by a train ride, followed by a rather long path from the last train station to the destination. In most cases, this cannot compete with a direct route by car.

Next, we investigated for which origin-destination pairs a better route was found in the multimodal network. More specifically, it is determined whether routing multimodally is more rewarding for short range routes or for long range routes. For this, 10 000 paths (with random origin-destination pairs) were calculated in both the unimodal and the multimodal network and both during and outside the rush hour. In this experiment the time-dependent stochastic algorithm was used. These paths were ordered according to their Dijkstra-rank (defined as the number of nodes that have been made permanent when the algorithm finishes) and divided into 10 categories. For each of these categories the percentage of paths that are multimodal (i.e. paths that are better in the multimodal network compared to unimodal ones) was then calculated. Results are shown in figure 5.7. Here we assumed that all links are completely correlated. Similar results are obtained when all links are uncorrelated. It is clear that routing multimodally has more advantages for long range paths, and during the rush hours. When the origin and the destination are situated further apart (i.e. a higher Dijkstra-rank), the percentage of multimodal paths increases steadily. Outside the rush hour, this phenomenon is not that prominent. Here, the main promoter for multimodal transportation is whether the origin and the destination are situated close to train stations.

As the travel time gain of multimodal routing over unimodal routing differs depending on the departure time, we want to investigate how this evolves during the day. The fastest route was calculated between one origin (Ghent) and one destination (Leuven) for a number of different departure times (with the time-dependent stochastic algorithm, assuming all links are completely uncorrelated). The 90% percentile values for these different departure times are depicted in figure 5.8. When the travel time in the multimodal network is equal to that in the unimodal network, this means that the route in the multimodal network is in fact unimodal as no travel time gain is realized by routing multimodally. Again, it is seen that routing multimodally is most advantageous during the rush hours. So, for commuters, taking the train is a good alternative.

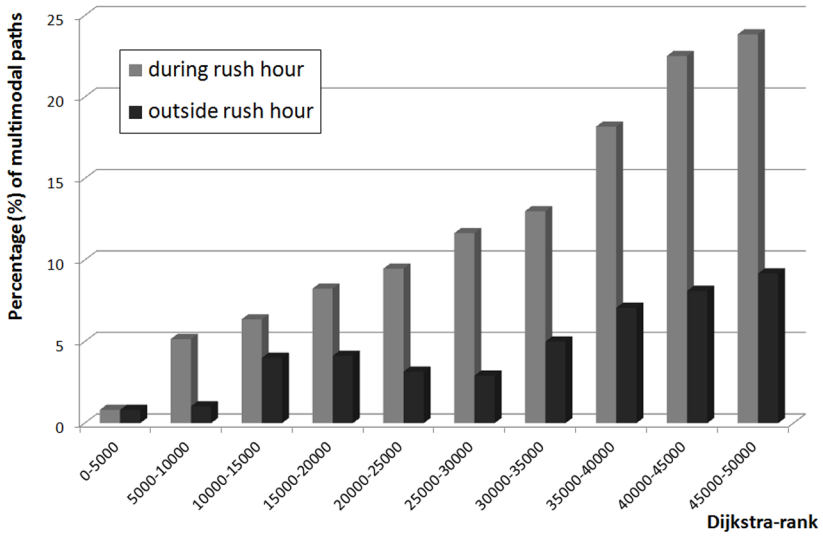

Figure 5.7: Percentage of multimodal paths in function of the Dijkstra-rank.

In conclusion, while in most cases the road network seems to be more rewarding, multimodal transportation (in this case study: taking the train) proves to be a good alternative to travel between major cities (i.e. close to train stations) and long distances during rush hours.

5.5 Conclusion

In this chapter a case study was presented of a novel practical multimodal routing system. Next to the time-dependent travel times, additional information about the (un)certainty of the results is calculated. A lot of attention was given to the design of efficient data structures to store the needed travel time information.

Making use of these data structures an algorithm was developed to calculate the shortest path between two locations, both time-dependently and stochastically. Experiments have shown that by routing time-dependently, stochastically and multimodally indeed a travel time gain can be realized by avoiding the congested roads. Moreover, in the case of stochastic routing, additional stochastic information is provided, which gives the user an idea of the (un)certainty of the travel time of the resulting route.

Two extreme cases, viz completely correlated links and completely uncorrelated links, were investigated, from which a more realistic estimation of the exact stochastic distribution can be deduced, as the real stochastic distribution can be

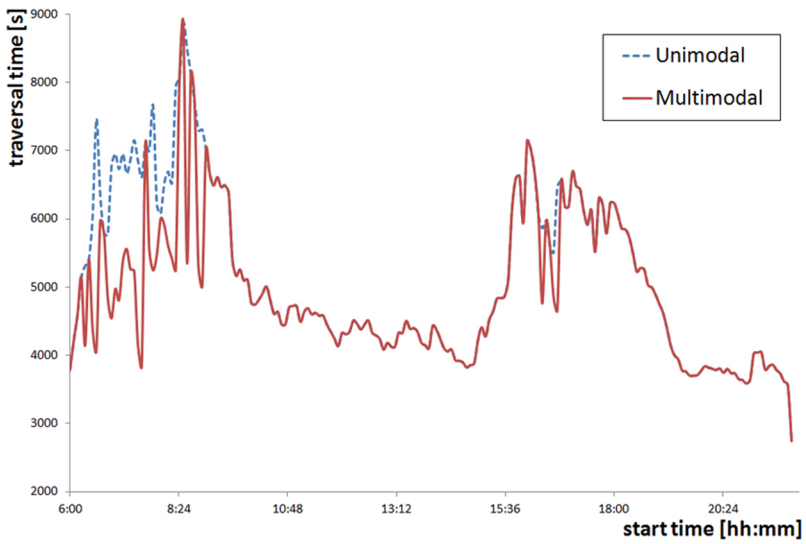

Figure 5.8: Comparison of travel times in unimodal and multimodal networks (day overview)

found somewhere in between. A stripped down version of the system presented in this chapter can be accessed online [22]. Moreover, the research presented here has been commercialized as an industrial-strength routing engine.

References

[1] E. W. Dijkstra. *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik, 1(1):269–271, 1959.

[2] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. *Efficient models for timetable information in public transportation systems*. Journal of Experimental Algorithmics, 12:1–39, 2008.

[3] U. Demiryurek, F. Banaei-Kashani, and C. 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, GIS '10, pages 474–477, 2010.

[4] D. Delling and D. Wagner. *Time-Dependent Route Planning*. In R. Ahuja, R. Mhring, and C. Zaroliagis, editors, Robust and Online Large-Scale Optimization, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer Berlin / Heidelberg, 2009.

[5] D. Delling and G. Nannicini. *Core Routing on Dynamic Time-Dependent Road Networks*. INFORMS Journal on Computing, 24(2):187–201, 2012.

[6] T. Kieritz, D. Luxen, P. Sanders, and C. Vetter. *Distributed time-dependent contraction hierarchies*. In Proceedings of the 9th International Conference on Experimental Algorithms, SEA'10, pages 83–93, 2010.

[7] G. Nannicini, D. Delling, D. Schultes, and L. Liberti. *Bidirectional A* search on time-dependent road networks*. Networks, 59(2):240–251, 2012.

[8] H. Frank. *Shortest Paths in Probabilistic Graphs*. Operations Research, 17(4):583–599, 1969.

[9] C. E. Sigal, A. A. B. Pritsker, and J. J. Solberg. *The Stochastic Shortest Route Problem*. Operations Research, 28(5):1122–1129, 1980.

[10] R. W. Hall. *The Fastest Path Through a Network with Random Time-Dependent Travel Times*. Transportation Science, 20(3):182–188, 1986.

[11] X. Ji. *Models and algorithm for stochastic shortest path problem*. Applied Mathematics and Computation, 170(1):503 – 514, 2005.

[12] A. Azaron and F. Kianfar. *Dynamic shortest path in stochastic dynamic networks: Ship routing problem*. European Journal of Operational Research, 144(1):138 – 156, 2003.

[13] H. Li, M. C. J. Bliemer, and P. H. L. Bovy. *Strategic Departure Time Choice in a Bottleneck with Stochastic Capacities*. In Transportation Research Board 87th Annual Meeting, 2008.

[14] B. Y. Chen, W. H. K. Lam, A. Sumalee, and Z. Li. *Reliable shortest path finding in stochastic networks with spatial correlated link travel times.* International Journal of Geographical Information Science, 26(2):365–386, 2012.

[15] S. Samaranayake, S. Blandin, and A. Bayen. *A Tractable Class of Algorithms for Reliable Routing in Stochastic Networks*. Transportation Research Part C, 20:199 – 217, 2012.

[16] S. E. Grasman. *Dynamic approach to strategic and operational multimodal routing decisions*. International Journal of Logistics Systems and Management, 2(1):96–106, 2006.

[17] T. Grabener, A. Berro, and Y. Duthen. *Time dependent multiobjective best path for multimodal urban routing*. Electronic Notes in Discrete Mathematics, 36:487 – 494, 2010.

[18] K. Zografos and K. Androutsopoulos. *Algorithms for Itinerary Planning in Multimodal Transportation Networks*. Intelligent Transportation Systems, IEEE Transactions on, 9(1):175 –184, 2008.

[19] M. Bielli, A. Boulmakoul, and H. Mouncif. *Object modeling and path computation for multimodal travel systems*. European Journal of Operational Research, 175(3):1705 – 1730, 2006.

[20] N. Caceres, J. Wideberg, and F. Benitez. *Review of traffic data estimations extracted from cellular networks*. IET Intelligent Transport Systems, 2(3):179 –192, 2008.

[21] IBBT. *ICON project MobiRoute*, 2010. Available from: http://www.ibbt.be/en/projects/overview-projects/p/detail/mobiroute-2.

[22] S. Demeyer. *Multimodal Dynamic and Stochastic Route Planning*, 2010. Available from: http://dna.intec.ugent.be/mdsr.

[23] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

6

Conclusions and Future Perspectives

"The important thing is not to stop questioning.
Curiosity has its own reason for existing."

–Albert Einstein (1879 - 1955)

6.1 Overall Conclusions

In the early 90s, the first personal GPS navigation system was launched on the market. This caused the establishement of a number of specialized navigation companies, like Garmin and TomTom, and online routing applications, such as Mappy and Google Maps. It also had an impact on the scientific world as it gave rise to a renewed interest in the design of fast and efficient shortest path algorithms for transportation networks. Furthermore, today's transportation (both personal and freight) is not restricted to one mode of transportation. Multiple transport modes may be used during a single trip. This is denoted with the term multimodal transportation. One of the reasons for the rise of multimodal transportation is the fact that the roads are getting more and more congested, forcing the drivers to consider the other modes of transportation. In this PhD dissertation research was presented on the planning and routing of multimodal transportation. Two challenges can be distinguished: the modeling of multimodal transportation networks and the design of efficient routing algorithms for these networks.

In chapter 2, a multimodal transportation network model was presented. This model is flexible in the sense that transportation modes can easily be added or

removed. Each mode-specific transportation network is represented by a single layer. Trans-shipments are then modeled as links between geographically co-located nodes of different layers. Furthermore, an access layer was added to find routes independently from the transportation modes of the origin and the destination. Since all cost information is modeled as properties of the links, the common shortest path routing algorithms can be applied directly in this model. Additional flexibility was added to the costs, as in this network model, each link may have a different cost model. To represent static and deterministic costs, single values are assigned to the links. If the costs are time-dependent (and deterministic) the cost objects represent functions. We assumed that these functions are piecewise linear, so that they can be represented by a finite number of values. This introduced a trade-off between the memory consumption and the accuracy of the function. If, on the other hand, the costs are (static) stochastic, a probability distribution is needed. In this research, a stochastic distribution is represented by a number of percentile values (i.e. values for which $x\%$ of the values of the distribution is smaller). This simplifies both the representation and the calculations, but comes again with a trade-off between the memory consumption and the accuracy of the information. If the cost information is both time-dependent and stochastic, it is represented by a (piecewise linear) function of distributions, which can be stored in a two-dimensional array. Moreover, multiple objective cost objects, represented by an array of objectives, may be assigned to the links. The major advantage of this network model is the flexibility in both the number of transport modes and the types of the link costs. This allows the use of common shortest path routing algorithms with only minor modifications.

Besides the multimodal network model, a number of shortest path algorithms were presented. Two challenges were tackled in this research: reducing the execution time of the algorithms and designing algorithms for the different types of costs. In chapter 3, where it was assumed that all costs are single values, two novel goal-directed algorithms were presented, together with a number of optimizations. While the predecessor algorithm only uses local information to guide the search towards the destination, in the accounting algorithm, an additional history of each path is kept. The advantage of these algorithms is that they only use estimated distances to determine whether to investigate a node, posing less restrictions on the estimation function. As the basic predecessor algorithm only finds a small fraction of the paths, a tolerance factor was introduced. Fine-tuning this factor is a tradeoff between the accuracy of the results and the performance of the algorithm. A lower tolerance factor means lower execution times, but less accurate results, and vice versa. By investigating the areas around the origin and the destination completely, more accurate results can be found by the predecessor algorithm. This is also realized by, instead of comparing a node with its predecessor, comparing it to its k-th predecessor. By keeping track of the number of steps in the right/wrong direction,

the accounting algorithm allows larger detours. This results in higher execution times, but more accurate results. Furthermore, two optimizations were presented: queue optimization and the feedback loop. Queue optimization (i.e. comparing a node with the previously investigated node) results in very low execution times, but this comes at the cost of less accurate results. However, when higher tolerance factors are used, adequate results are produced in a limited amount of time. The feedback loop mechanism, on the other hand, is guaranteed to always return a path between the origin and the destination, while only limited additional execution time is needed. We are convinced that these algorithms can compete with state-of-the-art goal-directed algorithms, such as the branch pruning algorithm, and are even a better alternative when no straightforward estimation function is available.

Subsequently, the case of optimizing multiple objectives simultaneously was investigated. Two major contributions can be observed in this research. Firstly, a stop condition was presented for the multiple objective shortest path algorithm of Martins. It has been demonstrated experimentally that, while this latter algorithm has a nearly constant execution time, applying the stop condition results in lower execution times, especially for paths between nodes which are close to each other. It is on average more than 2 times faster than the algorithm without the stop condition. Moreover, higher speedups (up to 3 times faster) are realized with an increasing number of objectives. Secondly, a bidirectional version of the algorithm of Martins was presented, focusing on the definition of the stop condition. Experiments demonstrate that routing bidirectionally indeed has its advantages with respect to the execution time of the algorithm. For transportation networks the bidirectional algorithms may be 10 times (for 2 objectives) or 20 times (for 3 objectives) faster than the unidirectional algorithm. Again, higher speedups are perceived for a higher number of objectives. Besides experimental results, it has been proven theoretically that once these stop conditions (for both the unidirectional and the bidirectional algorithm) hold, the set of Pareto optimal solutions has been found. The research presented in chapter 4 is a first step towards speeding up multiple objective shortest path calculations. The next step is to investigate whether other speed up techniques may be applied as well in these algorithms.

The last part of this PhD research focused on optimizing the travel time in multimodal networks. Here, it was assumed that each mode of transportation may have a different type of cost (static or time-dependent / deterministic or stochastic). A Dijkstra-based algorithms was developed that copes with these multiple cost types uniformly. Experiments were carried out for a case study on the Belgian road-railroad network. While the time-dependent and stochastic algorithms are slower than the basic algorithm of Dijkstra (with single value costs), more accurate and more reliable results are produced. It is demonstrated that by making use of the time-dependent information faster routes are calculated, especially during rush hour when traffic jams can be avoided. Furthermore, with respect to the

stochastic travel time information, two extreme cases were investigated, namely assuming that all links are correlated and assuming they are not. In reality, some of the links are correlated and some are not, but calculating all correlations and taking them into account when routing is too time-consuming. Nevertheless, these two extremes provide lower and upper bounds of the actual stochastic distribution of the travel time on the route. As we opted to compare distributions by a single percentile value, the shortest paths are optimized for this percentile. When using the 90% percentile values, more reliable routes are thus calculated. Furthermore, it was demonstrated experimentally that, indeed, using multiple modes of transportation in one single trip has its advantages, especially during rush hours and between locations that are not closeby. We are convinced that the additional information on the reliability of the travel times is a huge added value. However, from a practical point of view an easy-to-interpret presentation format should be developed in order for this system to be used in daily life environments.

6.2 Future Research Perspectives

There is a saying that research always gives rise to novel research possibilities. This is certainly true for the research that was presented in this dissertation. This section briefly describes some of the future research perspectives that arose when finishing the research of the previous chapters.

In the network model of chapter 2, a number of assumptions were made to simplify both the representation and the shortest path calculations. For instance, no turn restrictions (e.g. no left turns on highways, no u-turns) were taken into account. Adding these would result in a more realistic network. In chapter 3 of [1] multiple solutions to model turn restrictions are presented. The most promising technique is to split up the nodes for which turn restrictions hold and modeling the restrictions in these subnetworks. The advantage of this approach is that no additional information needs to be stored in the nodes and/or links, which allows the known algorithms to be applied without any modifications.

Furthermore, in this network model it was assumed that all modes of transportation are available at any point in time. For example, even if no car was available at the station, the road network still was considered to continue the trip, where it was assumed that taxis were available. This could be overcome by making the network dynamic and only adding the trans-shipment links from the other transport modes to the car layer when there is actually a car available at that location. Moreover, parking information should be provided. When switching from car to the other transport modes, the possibility of parking the car should be investigated. Unfortunately, this information was not available and thus was not added in the current system.

The goal-directed shortest path algorithms, that were presented in chapter 3,

can be further sped up by letting them investigate the network bidirectionally. First experiments with the bidirectional versions of the predecessor and the accounting heuristic show that, indeed the calculation time is diminished by searching the network from the origin and the destination simultaneously, but that these algorithms produce less accurate results. Further research is needed to find a good balance between the performance of the algorithms and the accuracy of the results.

With respect to the multiple objective shortest path algorithms, one of the future research possibilities is to investigate how the execution time of the algorithms is influenced by the correlation between the different objectives. First experiments have indicated that the calculation times are lower when the objectives are correlated. It would be interesting to see how the correlation of the objectives really affects the operation of the multiple objective shortest path algorithms.

Furthermore, a k shortest path algorithm was designed, based on the concepts of the multiple objective shortest path algorithms. Each node may have k labels and the algorithms searches the network until the k shortest paths to the destination node have been found. Again, a bidirectional version of this algorithm could be developed, speeding up the calculations.

In chapter 5, stochastic travel time costs were introduced. Recent research in transportation networks only focuses on static and time-dependent travel times. Very promising techniques precalculate additional network information and make use of hierarchies to speed up the calculations [2, 3]. The most promising of these techniques is the contraction hierarchy [3], of which a time-dependent approach has already been proprosed [4]. The next step would be to develop stochastic contraction hierarchies. Other interesting research opportunities lie in incorporating stochastic travel time information in other state-of-the-art algorithms, such as hub labeling [5], transfer patterns [6], etc.

Finally, to enhance the research on routing algorithms for multimodal transportation networks, more network data should be made available. Current researchers are struggling to obtain qualitative and realistic data sets. Almost all traffic is being monitored, but due to legislative restrictions, these measurements are often shielded for public (and scientific) use. Open source solutions exist (cfr. OpenStreetMap), but only have limited cost information. For example, time-dependent and/or stochastic travel time information is mostly absent in these networks. More available transportation network data would boost the research in this field.

References

[1] S. Vanhove. *Alternative routing algorithms for road networks*. Ghent University, Department of Applied Mathematics and Informatics, 2012.

[2] G. Jagadeesh and T. Srikanthan. *Route computation in large road networks: a hierarchical approach*. IET Intelligent Transport Systems, 2(3):219 –227, 2008.

[3] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. *Exact Routing in Large Road Networks Using Contraction Hierarchies*. Transportation Science, 46(3):388–404, 2012.

[4] G. V. Batz, D. Delling, P. Sanders, and C. Vetter. *Time-dependent contraction hierarchies*. In Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX), pages 97–105. SIAM, 2009.

[5] I. Abraham, D. Delling, A. Goldberg, and R. Werneck. *A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks*. In P. Pardalos and S. Rebennack, editors, Experimental Algorithms, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer Berlin Heidelberg, 2011.

[6] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. *Fast routing in very large public transportation networks using transfer patterns*. In Proceedings of the 18th annual European conference on Algorithms: Part I, ESA'10, pages 290–301, 2010.

A

Ant colony optimization for the routing of jobs in optical grid networks

This chapter focuses on the design of a number of anycast algorithms for the routing of jobs in optical grid networks, represented as graph structures. Generic shortest path routing algorithms are used as a part of the anycast algorithms to determine which resource and/or link should be selected.

⋆ ⋆ ⋆

S. Demeyer, M. De Leenheer, J. Baert, M. Pickavet, and P. Demeester.

Published in Journal of Optical Networking, volume 7 (2), February 2008

Abstract Grid networks provide users with a transparent way to access computational and storage resources. The introduction of (dense) wavelength division multiplexing techniques have made optical networks the technology of choice for data-intensive grid traffic. In a grid network scenario, users are generally more interested in the successful completion of their jobs than in the location where the actual processing occurs. Job routing and scheduling in current generation grid networks are managed by resource brokers, which assign each job to a resource and route the job in a unicast way. An anycast approach using grid-aware network algorithms would bypass the need for a resource broker and increase scalability.

We propose several anycast algorithms for job routing in optical grid networks, based on the concept of ant colony optimization, which draws parallels between the behavior of ants gathering food and the routing of packets inside a network. Simulation results show an increased performance of our algorithms over more classical unicast-based protocols, even though this is accompanied by a slight increase in complexity.

A.1 Introduction

A.1.1 Concepts

In modern times the demand for more computational and storage resources continues to grow, while at the same time, a vast amount of resources remains underused. Grid networks [1] attempt to provide an efficient way of using this excess capacity.

In 1998 Foster and Kesselman [2] defined the computational grid as "... a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." In grid networks, resources are geographically dispersed and shared among all clients. One of the main advantages of grid networks is the ability to handle peak loads, since local processing would require additional infrastructure, which is often financially unfeasible. In a grid network, off-site resources with free capacity can take care of the peak loads, without additional infrastructure costs (beyond the network cost).

An essential factor in the successful realization of grid networks is the transport network. Optical networks can offer an effective solution to this problem, given the high-performance (with respect to bandwidth, delay, and signal strength) and low cost of current optical technology. For instance, (dense) wavelength division multiplexing [(D)WDM] [3] allows multiple signals to be sent over a single link by assigning a dedicated wavelength to each signal [4, 5].

Several studies on optical network infrastructure for grid networks have been carried out [6–8]. Moreover, the emergence of optical grid projects such as the European Phosphorus project [9] (which has set up an optical grid test bed to develop multidomain, end-to-end connectivity) indicates the relevance of this research.

The scheduling of jobs in grids has been studied thoroughly [10]. Most of this research focuses on resource brokering by a central component, which allows highly optimized scheduling decisions but suffers from scalability issues. To overcome this, we intend to route the jobs without the use of such a broker. Multiple paradigms exist to route packets in a network: unicast, multicast, or anycast. In unicast routing one source communicates with a single destination. Multicast routing has one source and multiple destinations, which all receive the messages from the source. Anycast routing [11, 12] also sends the packets to multiple destinations of which at least one, and preferably only one, should receive the message

sent by the source. The source typically does not know anything about the possible destinations, as this is not important, as long as the packets it sends arrive in one of the destinations. This idea can be extended to grid networks, since users are generally more interested in the successful completion of their jobs than in the location where the actual processing occurs. Indeed, as long as a job is executed successfully while adhering to its predetermined requirements (e.g., meet a given deadline), the decision of when and where to start a grid job is usually left to the grids management functions. The routing and scheduling algorithms can consequently exploit this flexibility to realize certain global objectives, such as minimal blocking probability or maximization of the resource utilization. Since anycast routing can send a packet to multiple destinations, the load can be balanced by distributing it proportionally over the different destinations. The packets do not need to be sent to all destinations, so there is no additional network overhead.

In general, two approaches exist to deal with anycast routing: network level anycast and application level anycast [13]. The former makes use of the information available in the routing tables, while the latter sends the request to a central point that selects a destination. Application level anycasting in grid networks corresponds to the traditional approach of using a resource broker. As we intend to avoid the use of these brokers, this paper will focus on the network level anycast routing. Network level anycast routing algorithms exist [14, 15], but in general they are not adapted to grid networks.

Anycast routing can be dealt with in several ways. In this paper, we will present a set of algorithms based on the principles of ant colony optimization (ACO) [16–19], a form of swarm intelligence that has proven its effectiveness in many business and routing problems [20]. ACO is a technique inspired by the natural behavior of ants in a colony. By leaving pheromone trails, ants are able to communicate the shortest way to a food resource. Scouting ants explore the environment, and when food is found they return to the base camp by following their own pheromone trails. As subsequent ants are attracted to these pheromone trails and the shortest path to the food will be visited most (shortest round-trip time), in the end only this shortest path will remain. This is illustrated in figure A.1. The idea of ACO can be translated to optical grid networks where an ant is a digital item that, while also gathering information, leaves information in the routers on its path. Using this information left by the ants, packets can be routed on the shortest path.

A.1.2 Contributions and Objectives

Grid networks offer a solution to overcome the resource shortage of single computers. Since very large amounts of data need to be sent over a network in a predictable way in grids, optical networks are the technology of choice. In this paper, our attention goes to optical grid networks.

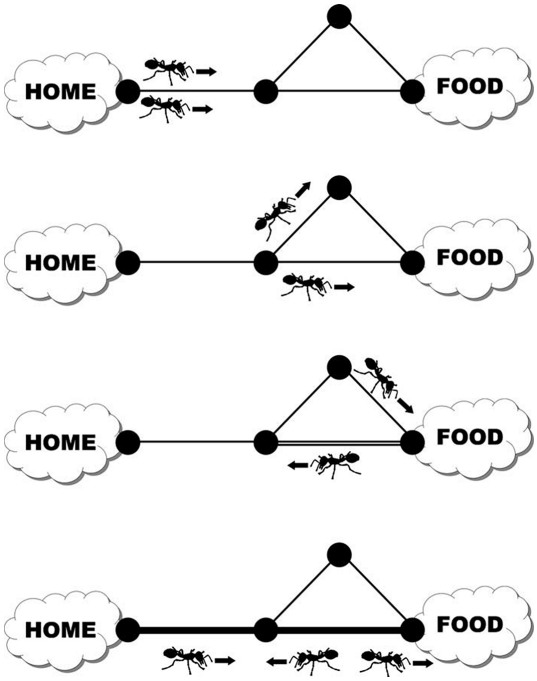

Figure A.1: Ant Colony Optimization.

A way to route packets in an optical grid network efficiently is presented, without the use of a resource broker. Unicast algorithms are useless in grid networks as packets have no fixed destination address since clients are not aware of the locations of the resources. Multicast algorithms can handle multiple dispersed destinations, but this is accompanied by traffic overhead since every packet is sent to all resources. As clients want their packets to be sent to one of the resources that can process the job successfully, we will focus on anycast algorithms. As it is a connectionless protocol, anycast routing eliminates the use of a resource broker. We will demonstrate that this routing protocol performs load balancing of network usage, resulting in a higher acceptance probability.

Grid networks have multiple dynamic properties such as varying resource and link availability. System efficiency is consequently dependent on the ability of algorithms to adapt to varying state parameters. This is where ACO shows its potential, since ants are traveling continuously to explore and signal changing resource and network states. As we consider an optical grid scenario, the control traffic (carrying the ants) will not harm the capacity of the links. In contrast to Xuan et al. [14] and Jia et al. [15], the proposed ACO anycast algorithms will be grid-aware through the information dispersed by the ants over the nodes. Finally, note that ACO is heuristic; it provides no guarantees concerning the optimality of the technique, although a discussion on complexity is presented. Moreover, we will demonstrate that the ACO-based algorithms show increased performance over traditional grid routing algorithms.

The remainder of this paper is structured as follows. Section A.2 presents the model used in this paper, starting with the network model. Subsequently we will discuss ants, which travel through the network to disperse the routing information. Next, several algorithms to route jobs in an optical grid network are presented. Two selection algorithms will be discussed: one for the resource selection and one for the link selection. We will also explain how state updates from the resources will be communicated to the routers in the network. Section A.3 is concluded by a discussion on the complexity of the different proposed algorithms. Section A.4 presents simulation results of the algorithms in an optical grid network. Finally, we present our conclusions.

A.2 Model

A.2.1 Network Model

In the network model a number of resources (where data can be stored, calculations can be made, etc.) and clients (that generate jobs) are dispersed over the grid network. At the core, the grid network consists of routers. All resources, clients, and routers are connected by means of optical fibers.

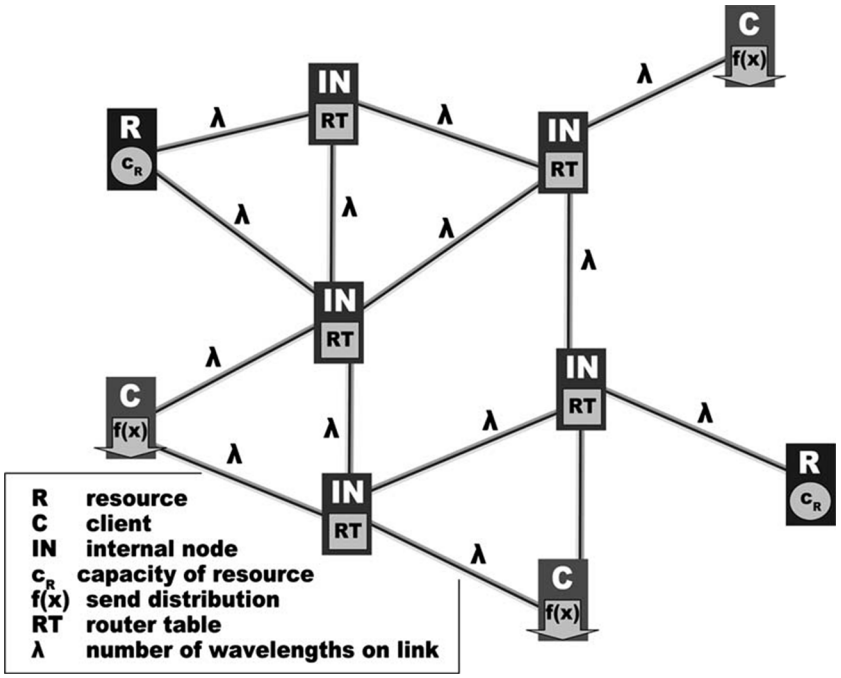

Figure A.2: Optical Grid Network.

Figure A.2 shows an optical grid network with the abstractions we made. A resource has a certain amount of capacity (c_R) to execute a number of jobs simultaneously, independently of the type of jobs. A client sends tasks into the network at certain times. There are multiple distributions $[f(x)]$ possible to generate these jobs, such as deterministic, bursty, Poisson,.... The optical links have a number of wavelengths (λ) available on which information can be sent. The time needed for a packet to traverse a link is dependent upon the length of that link (propagation time) and the size of the packet (transmission time). The propagation time is almost negligible compared to the transmission time, since bits are traveling at the speed of light and the links to be traversed are relatively short. For the transmission time we assumed a synchronous digital hierarchy (SDH) network in which information travels at 155 Mbytes/s (STM-1).

Besides resources, clients, and links we can also find internal nodes in the network. These nodes act as routers. To route the packets from client to destination, information needs to be stored about the resources and the possible routes to get there. This can be seen in figure A.3. Every node knows the capacity of all nearby resources, together with the amount of this capacity that is already occupied by previous tasks. Since this information is dispersed by the resources themselves,

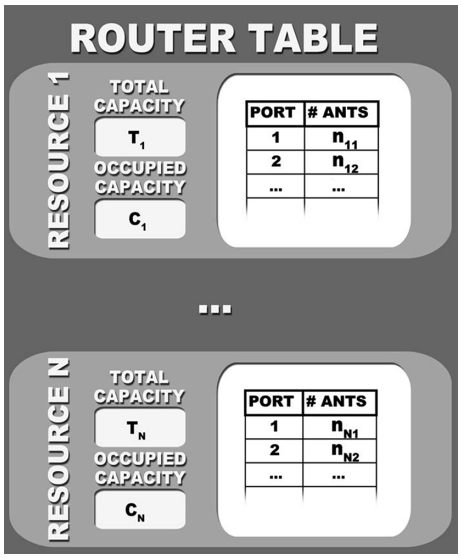

Figure A.3: Router Table.

it is not always up to date, but it is an indication for the attractiveness of the resources. Next to this resource information, the nodes contain information about the passing ants. For every resource a table is maintained for every outgoing link (associated with a port in the router) for the number of ants that have passed this link to reach the specific resource. The information in these tables is updated by the backward ants (see subsection A.2.2) and thus is always up to date.

A.2.2 Ants

In this subsection, the actual implementation of ACO, in which ants explore the network, is discussed. In optical grid networks, ants are small packets that travel from the clients to the resources and back and help to distribute the information needed in the algorithms (see Section 3). We can distinguish two types of ants: forward ants and backward ants.

A forward ant travels from a client to one of the resources. In the resource the ant discovers first it will be transformed into a backward ant and return to its origin. While foraging the network, a forward ant will execute the following pseudo code in every node/router.

```
1     if(node connected to resource) {
2         visit (resource);
3         store (information);
4     }else{
5         if(routerTable = NOT empty){
6             p0 = random();
7             if(p0 < threshold){
8                   link = algorithm();
9             }else{
10                  link = random();
11            }
12        }else{
13            link = random();
14        }
15        store (information);
16        send();
17    }
```

When the node is connected to a resource, the ant will visit the resource to gather the needed information (lines 2 and 3). If the node is not connected to a resource, the router table is investigated. In line 7 a random number is compared to a threshold as a way of controlling the algorithms dynamics. A low threshold will encourage the ants to discover new roads and not to follow the path indicated by the router tables. This can be compared to the ants likelihood of following existing pheromone tracks. The algorithm() function in line 8 uses one of the algorithms described in section A.3 to route the ant, i.e., selecting which resource to choose and which path to follow using the data mentioned. Before traveling to the next link, the forward ant will gather information about the node and the next link. This information will be stored in the ant and will be carried along (line 15).

When a forward ant has reached a resource and has gathered the needed information about the resource, it has accomplished its task and will be transformed into a backward ant. A backward ant will return to the client on the same route of the forward ant, and while on its way is updating the nodes router tables.

The following pseudo code explains what happens with a backward ant entering a node.

```
1  if (!(local && (lifetime ≥ sizeNeighborhood))){
2        if(node connected to client C &&
                                sourceClient=C){
3            update (router);
4        }else{
5            update (router);
6            link=ant.getNextLink();
7            send();
8        }
9  }
```

If the ant has reached the router connected to the client, it only has to update the information in this router. If the backward ant is in an intermediary node, it also has to determine on which link its forward ant traveled (line 6), which is stored internally in the ant.

A.3 Algorithms

A.3.1 Selection Algorithms

In section A.2 we discussed how ants disperse information in grid networks. Here we will explain how jobs sent by clients use this information to reach one of the resources. To deal with changing network topologies, an ant travels along with each job to update the information in the network.

In every router, two problems have to be solved: which resource will the packet be sent to and which link to traverse first in order to reach the chosen resource. An overview of these selection procedures is shown in figure A.4. The resource selection mechanism is responsible for determining the best resource. This can be the closest resource or the resource with the highest spare capacity. Once this decision has been taken, one has to decide which link to traverse first in order to reach the resource. As we want to introduce a load balancing mechanism, not always the shortest path will be used to send the packets to a specific resource. For both resource and link selection there are multiple options (figure A.4).

A.3.1.1 Resource Selection

We propose four alternatives for resource selection: using the algorithm of Dijkstra, using the information stored by the ants in the router tables, a weighted choice with the information in the router tables and best link. The latter one is not really

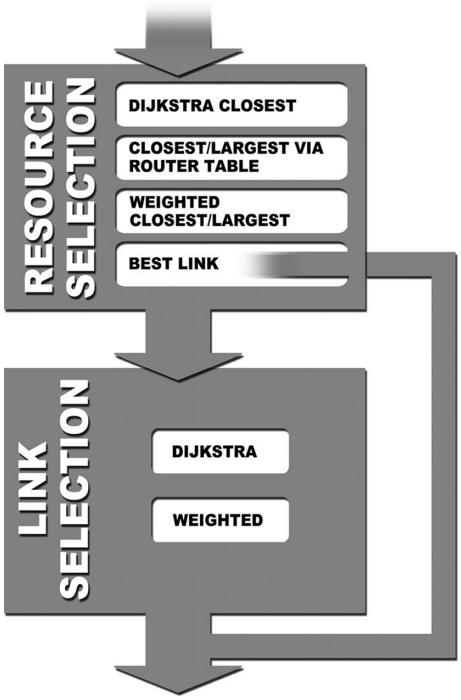

Figure A.4: Selection Algorithms.

a resource selection procedure but determines the overall best link to reach one of the resources.

When using the well-known algorithm of Dijkstra each node knows in advance which one is the closest resource, which thus will be selected. Here the calculations to determine the closest resource only have to be executed when the information in the router table is updated, more specifically when a new resource is discovered or when a resource goes offline.

A resource can also be determined by the use of the router table (subsection A.2.1), which keeps track of the free capacity of every resource and per resource and per link the number of ants that choose that link to reach the resource. According to the ACO principle the closest resource is the one that has been reached most, i.e., the resource with the highest number of ants independent of the path they followed to reach the resource. Additionally, the resource with the highest spare capacity can easily be deduced from the router table.

The previous selection procedures choose the resource unambiguously. In contrast, when using a weighted choice, a level of uncertainty is introduced. Again the information in the router table is used to select a resource, but every resource is now assigned a probability proportional to the information in the router table. To find the "closest resource" the following formula is used:

$$p_k = \frac{\sum_j ant(k,j)}{\sum_i (\sum_j ant(i,j))}$$

where p_k is the probability that resource k is the closest resource. $Ant(i,j)$ defines the number of ants that traversed link j to reach resource i. So this probability is equal to the number of ants that reached resource k divided by the total number of ants that crossed the router. A unit interval is split according to these probabilities, and the section in which the randomly chosen number is situated determines the selected resource. An analog procedure can be used to select the "resource with the most free capacity", but instead of the number of ants the remaining free capacity is used.

The last resource selection procedure determines the best link to traverse in order to reach one of the resources, without explicitly selecting a single resource. In fact this is a link selection algorithm, but since no resource selection is needed in advance it is categorized here. The link that has been crossed the most will be chosen. If we want to introduce a level of uncertainty this selection can happen in a weighted manner, as explained previously.

A.3.1.2 Link Selection

Once a resource has been selected, we have to decide on which link the job has to travel first to reach this resource. We examined two ways to perform this link se-

lection. Internal nodes can calculate the shortest routes to the resources in advance using the algorithm of Dijkstra. For each resource the node now knows which link a job has to travel on to reach the resource as fast as possible. These links only have to be recalculated if the network topology changes.

Another way makes use of the router table. By determining how many ants crossed each link to reach the selected resource, the link that is most likely part of the shortest path can be selected. To introduce a way of load balancing we opted here for a weighted choice, according to the procedure discussed in subsection A.3.1.1.

A.3.2 State Updates

When a job arrives in a resource and is executed the internal state of the resource changes. The amount of free capacity diminishes (start execution) or increases (end execution). This state update is communicated by the resources itself. This can be done globally or locally. Local state updates reduce the amount of control traffic and will prove their benefit in scalability studies. Globally dispersed information induces better choice making as information about all resources is known, but this is accompanied by a traffic overhead.

When the information is spread globally the information update will be flooded to all nodes in the network. At any time, every node has the correct status of all resources. When updates are local, only the nodes that are close to the resource know about the exact status of the resource. The area that is updated is called the environment.

Local algorithms have the advantage of scalability. When the network is really big, every resource has a part of the network it has to notify. There is no message overhead to notify nodes far away that probably never will be visited anyway. A disadvantage of these local updates is a performance reduction. All nodes only have local information and no global view of the network. So the network load will not always be distributed equally. However the idea of local updates conforms more to the idea of a decentralized algorithm in the grid network.

A.3.3 Complexity

In this subsection, we examine the additional overhead introduced by the novel routing algorithms. We focus on two aspects: the amount of memory routers needed to store the routing table and a quantitative analysis of the processing complexity to execute the different routing algorithms.

The structure of the routing table (figure A.3) shows memory is required to store two values for every resource that can be reached, together with a table containing a number of entries consisting of two values. The maximum number of entries in this table is the number of outgoing links (m_i for resource i). Assuming

	Average LC [1]	Number of Resources	RT [2] (kB)
Ring network	2	200	4.80
Mesh network	4	200	8.00
Random network	6	200	11.20
Simulation network	3	5	0.16

[1]LC is the link connectivity.
[2]RT is the router table.

Table A.1: Router Table Memory Needs

N resources are in the network, the formula for the amount of memory needed in a router is

$$\frac{2cN + 2c\sum_i m_i}{8}$$

where c represents the number of bits needed to represent a single value. If we assume we are dealing with standard integers, c is 32 bits.

Table A.1 shows the amount of memory needed for four different network topologies: a ring network, mesh network, random network, and the simulated topology. As is clear from the table, for a considerable amount of resources in the network, the amount of memory needed to store a single routing table remains very reasonable. For a much larger number of resources, aggregation techniques could prove useful to reduce the routing tables size at the expense of slightly less accurate routing decisions.

In the simulations we used a software simulation model in which all information is stored centrally. To save on memory space we opted for a smaller network with only five resources and an average link connectivity of 3 (simulation network in table A.1). In this way, less space is needed to store the router tables of all routers.

Next we will examine the number of calculations needed for the different selection procedures. This is presented in table A.2, where N represents the number of resources in the network, while m_i denotes the number of outgoing links in a router. The first column indicates if the selection procedure concerns resources (R) or links (L). Random denotes the number of calculations needed to determine a random number and selection() denotes the number of calculations to pick the maximum value (max) or the item that corresponds to the random number (rand). For more information we refer to subsection A.3.1.

We see that the ACO selection procedures are more complex than the algorithm of Dijkstra, in which, after primary calculations, only a table has to be consulted. When using the algorithm of Dijkstra the shortest paths have to be calculated once with a running time of $O(|V|^2 + |E|)$ in the simplest implementation [21]. (V

		Number of Calculations
R	Closest via router table	$N(m_i - 1) + selection(max)$
R	Largest via router table	$selection(max)$
R	Weighted closest	$((m_i - 1) + (N - 1)(m_i - 1) + 1)$ $+random + selection(rand)$
R	Weighted largest	$((N - 1) + 1) + random + selection(rand)$
R/L	Best Link	$m_1(N - 1) + selection(max)$
L	Weighted	$((m_i - 1) + 1) + random + selection(rand)$

Table A.2: Number of Calculations in Selection Procedures

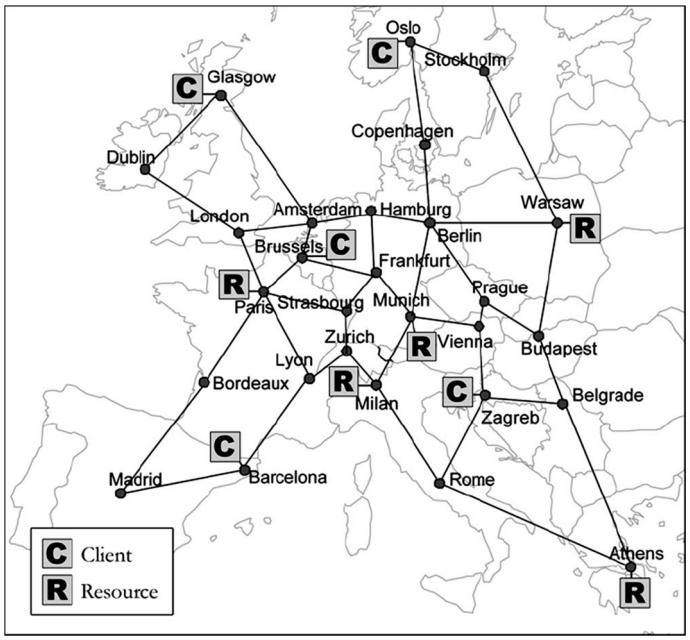

Figure A.5: Network Topology (European optical network).

represents the set of vertices, E represents the set of edges). Once the closest resource is determined,and this is stored inside the router, no additional calculations are needed to route a job. Only when the network topology changes drastically, are new calculations needed. When using the ACO algorithms, calculations have to be executed every time a job enters a router. Additionally ants are foraging the network for initialization and at runtime.

A.4 Evaluation

A.4.1 Setup

To test the algorithms proposed in section A.3 we make use of a standard European optical network topology that we transformed into a grid network by adding some clients and resources. The links are modeled as SDH links with a capacity of 155.52 Mbytes/s. The network we used to gather most of the presented results is shown in figure A.5, but multiple networks were used to test the algorithms.

The network in figure A.5 has 28 nodes and 41 links. There are five clients and five resources dispersed randomly in the network. These nodes are fixed for all tests. This network is intentionally kept small as simulations are performed on a

single machine, and we want to reduce the memory needed for the simulations. To maintain a realistic distribution between the number of clients and resources and the number of internal nodes, only five clients and resources are used. All other aspects are modeled in a way that resembles a real-life network: realistic network load, realistic acceptance rates, etc.

Moreover, the presented selection procedures make no assumptions about the network properties (e.g., structure, topology, size,). Simulation on a small network gives us the advantage that all aspects can be studied thoroughly. The results for the network, depicted in figure A.5, present an accurate image of changes in connectivity.

Even if the network would be large, ants forage the complete network, i.e., all possible nodes have been visited after a certain amount of time. The ACO algorithm will always converge if no failures exist in the network. When a local algorithm is used, the size of the neighborhood needs to be large enough so that at least one resource can be reached from all routers. This can be implemented by gradually increasing the size of the neighborhood when this condition is not satisfied.

Clients generate jobs by a Poisson process. We chose a load factor so that the network was not idle most of the time as well as not overloaded. We used an interarrival time a little bigger than the time to send a job, so most of the jobs can be sent and the network is not idle most of the time. Resources have enough buffer space to execute most of the jobs arriving. Only when too many packets are sent to the same resource will they be dropped.

There are two kinds of drops: packets can be dropped because all outgoing links of a node are overloaded, or packets can be dropped because the buffer of the resource they arrive in is full. Figure A.6 shows for some simulations the number of drops in a network and in the resources for varying job sizes. The job size unit is chosen in the way that if we consider only one single outgoing link it takes at least the average interarrival time until the next job can be sent, if the job sent has size one. If we denote the interarrival time by λ and the time to send a job by τ, i.e., the size of the job in bits divided by the capacity of the links (155.52 Mbytes/s), the job unit can be represented by λ/τ. The figure shows that resources are not the restricting factor since most of the packets are dropped on the links and not in the resources.

To simulate the operation of the algorithms a discrete event simulator was used. Events are executed one after each other while incrementing the time by discrete hops. As the simulator never has to wait for events to happen (all events are stored in the event list), discrete event simulations generally take less time than real-life situations.

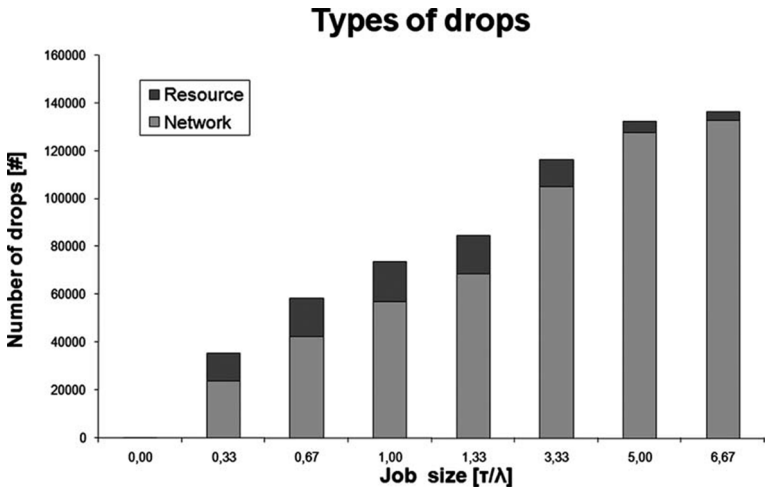

Figure A.6: Types of Drops.

A.4.2 Results

This subsection presents the simulation results. First we consider the resource and link selection algorithms. Next we will discuss the local versus the global algorithms. Besides that, the number of unused links and the influence of the network topology on the results are examined. In the results the job size unit is generalized so that when a job has size one and is sent on a single outgoing link, this link is free again after at least the average interarrival time by which jobs are generated (τ/λ).

In figure A.7 the different resource selection procedures are compared when the link selection procedure is fixed, i.e., a weighted link selection. These algorithms are compared with the well-known algorithm of Dijkstra. We see that in general the acceptance rate of the jobs diminishes as the jobs get bigger. The algorithms that determine the closest resource perform better than those that determine the resource with the most free capacity. The best link algorithm, where no single resource is selected, performs as well as the algorithms that select the closest resource. The figure shows that a weighted resource selection has a bigger acceptance rate than a selection procedure that determines the resource unambiguously using the router table. The algorithm in which the closest resource is determined using Dijkstra (and the link selected in weighted manner) outperforms all others. We can conclude that all ACO algorithms perform better than the algorithm of Dijkstra and that the best resource selection procedure is the one in which the closest resource is determined by the algorithm of Dijkstra, when link selection happens in a weighted manner.

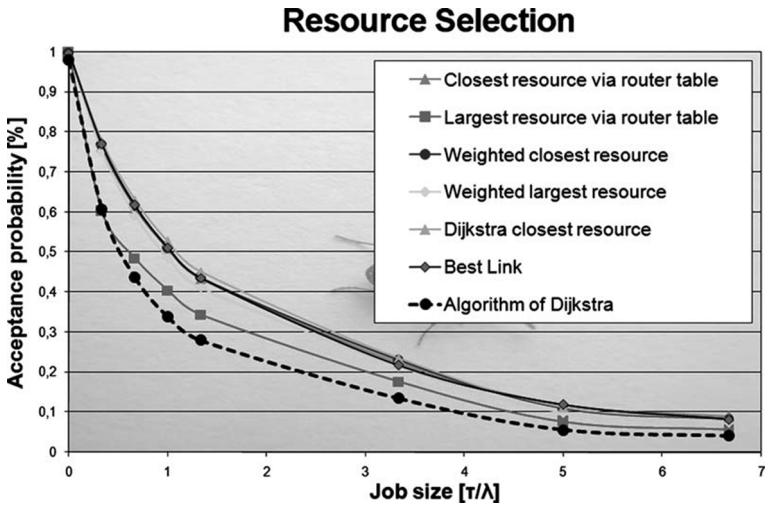

Figure A.7: Resource Selection.

After selecting a resource, the best link to reach this resource is determined. Here the resource selection procedure is fixed, i.e., the closest resource making use of the router table. We will examine two ways to perform the link selection: via the algorithm of Dijkstra or in a weighted manner. Figure A.8 shows a comparison of these algorithms. For comparison reasons the algorithm of Dijkstra is depicted too. The figure shows that a weighted link selection performs better than one using Dijkstra since this selection procedure is more dynamic. Just as before we see that the ACO algorithms outperform the algorithm of Dijkstra. So the best ACO algorithm with regard to the acceptance probability is the one in which the closest resource is selected making use of the algorithm of Dijkstra, and the link selection happens in a weighted manner.

Figure A.9 compares the update mechanisms: local versus global. Both resource and link selection happen in a weighted way as this best approaches the idea of ACO. We see that the algorithms with local update perform better than the ones with global update. The figure also shows that with local update the algorithms that select the resource with the most free capacity perform better than those looking for the nearest resource when jobs are small. This is due to the fact that algorithms with local update will always choose a resource that is in the neighborhood of the client. In this way these algorithms can be seen as a combination of the closest and the largest resource.

Besides the acceptance probabilities, we looked at the number of unused links in the different algorithms, which provides insight into the network utilization. Table A.3 presents the number of unused links for some of the algorithms. The

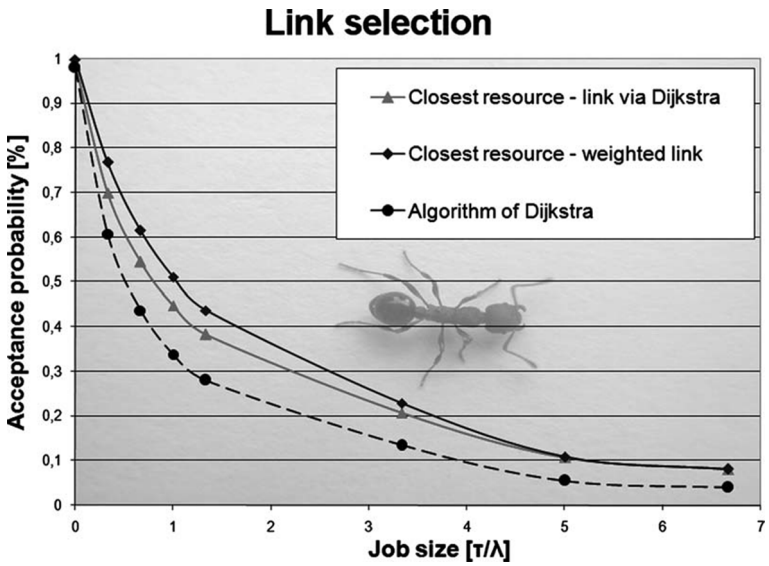

Figure A.8: Link Selection.

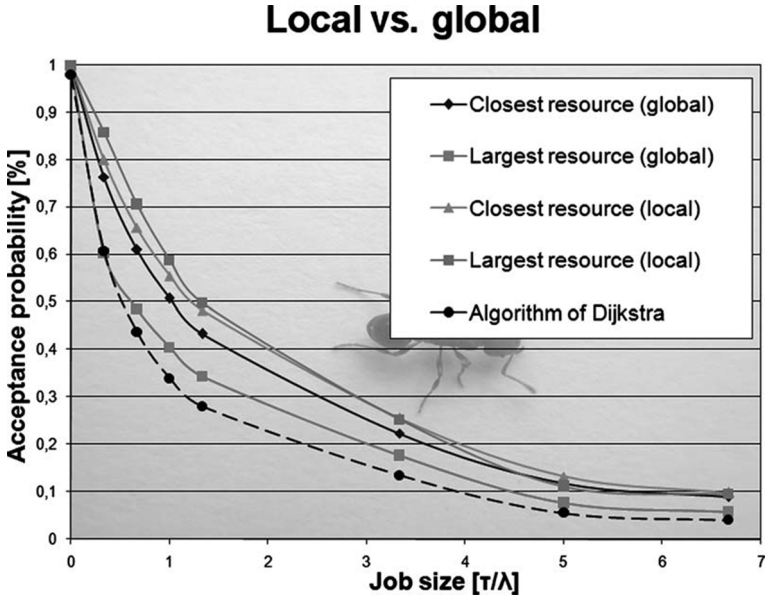

Figure A.9: Local versus Global.

	Number of Unused Links	Average Number of Hops
Algorithm of Dijkstra (closest)	31	1.56
Algorithm of Dijkstra (largest)	13	2.36
Closest resource (local)	5	2.50
Largest resource (local)	4	2.74
Closest resource (global)	4	2.55
Largest resource (global)	3	2.75
Best link	2	2.77

Table A.3: Number of Unused Links/Average Number of Hops

first two algorithms make use of the algorithm of Dijkstra to determine the shortest path to the closest resource or to the one with the most spare capacity. All ACO algorithms make use of weighted selection criteria. First of all we can see that the ACO algorithms use more links than the known algorithm of Dijkstra. The latter one has one predefined route to reach each resource, while in the former one multiple possible routes exist. The local algorithms use fewer links than the global ones, because jobs only travel to the resources in the neighborhood. The algorithm in which the overall best link is chosen performs best with regard to overall network usage. It will best disperse the load over the complete network.

Table A.3 also shows the average number of hops jobs have to travel to reach a resource. Obviously, jobs looking for the closest resource do not travel as far as jobs going to the resources with the highest free capacity. We can see that jobs routed by one of the ACO algorithms remain in the network longer than the jobs routed by the algorithm of Dijkstra. The small differences between the local and the global algorithms are due to the fact that the network is quite small.

Finally we examined the influence of the network topology by running identical experiments on similar networks with a different connectivity. The additional networks are based on the basic European network presented in figure A.5 (average link connectivity: 3 links). One is less connected (ring network) with an average link connectivity of 2.4 links, and the other one is more connected (triangular network) with an average link connectivity of 4.4 links. Figure A.10 compares these different network topologies. The algorithm selects the closest resource using the router table and has a weighted link selection. We can see that the acceptance probability increases according to an increasing level of connectivity, thanks to the fact that more routes to the resources exist. The results presented earlier in this subsection were verified in the triangular network. The overall observations remain the same but the acceptance probabilities are shifted accordingly.

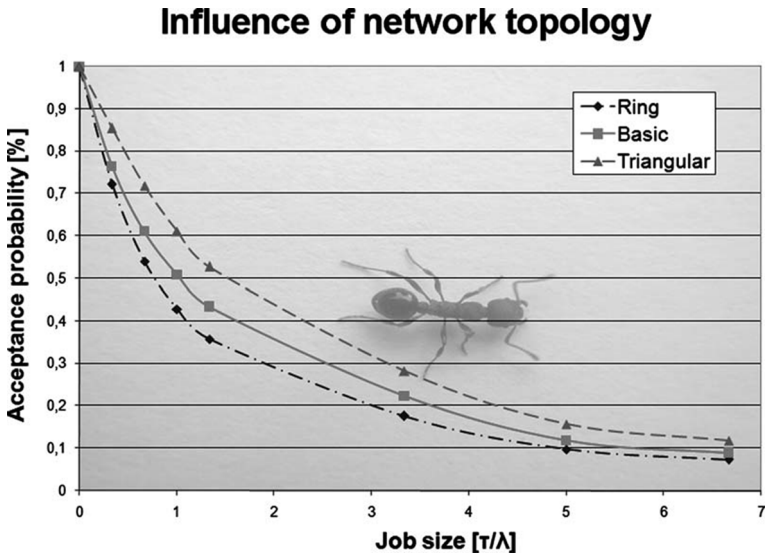

Figure A.10: Influence of Network Topology.

A.5 Conclusion

In this paper we proposed several ACO-based algorithms for routing and job scheduling in optical grids. A simulation analysis was used to demonstrate the efficiency and scalability of the algorithms. Improvements in network usage (by load balancing) are shown, together with an increase in job acceptance probability when compared to traditional shortest path routing. However, ACO-based algorithms exhibit slightly increased travel times and have a higher complexity. To cope with the latter problem, we introduced the notion of locality in routing, which also addresses issues of scalability. Overall, the improved performance of the ACO algorithms is due to their ability to adapt to a dynamic grid network environment.

Acknowledgment

This research was partly funded by the European Commission through the projects e-Photon/One+ and Phosphorus and by the Flemish government through the project FWO G.0315.04. M. De Leenheer thanks the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT) for its financial support through a Ph.D. grant.

References

[1] B. Volckaert, P. Thysebaert, M. D. Leenheer, F. D. Turck, B. Dhoedt, and P. Demeester. The Journal of The Communications Network (Proceedings of FICTE2004).

[2] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[3] C. Brackett. *Dense wavelength division multiplexing networks: principles and applications*. IEEE Journal on Selected Areas in Communications, 8(6):948 – 964, 1990.

[4] R. Ramaswami and K. N. Sivarajan. *Routing and wavelength assignment in all-optical networks*. IEEE/ACM Transactions on Networking, 3(5):489 – 500, 1995.

[5] C. Chen and S. Banerjee. *A new model for optimal routing and wavelength assignment in wavelength division multiplexed optical networks*. In Proceedings IEEE INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation., pages 164 – 171, 1996.

[6] D. Simeonidou and R. Nejabati. *Photonic infrastructure for grid enabling networks*. In Proceedings of 2004 6th International Conference onTransparent Optical Networks, 2004., pages 59 – 64, 2004.

[7] D. Simeonidou, R. Nejabati, G. Zervas, D. Klonidis, A. Tzanakaki, and M. O'Mahony. *Dynamic optical-network architectures and technologies for existing and emerging grid services*. Journal of Lightwave Technology, 23(10):3347 – 3357, 2005.

[8] S. D and N. R. *Optical networks infrastructure for grid*. In Global Grid Forum, 2002.

[9] A. Binczewski. *Phosphorus*. Available from: http://ist-phosphorus.eu.

[10] B. Volckaert, P. Thysebaert, M. D. Leenheer, F. D. Turck, B. Dhoedt, and P. Demeester. *Network Aware Scheduling in Grids*. In Proceedings of the ninth European Conference on Networks and Optical Communications, 2004.

[11] C. Partridge, T. Mendez, and W. Milliken. *Host Anycasting Service*. RFC 1546 (Informational), 1993. Available from: http://www.ietf.org/rfc/rfc1546.txt.

[12] S. Deering and R. Hinden. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, 1998. Available from: http://tools.ietf.org/html/rfc2460.

[13] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei. *Application-Layer Anycasting*. In Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution, INFOCOM '97, pages 1388 – 1396, 1997.

[14] D. Xuan, W. Jia, and W. Zhao. *Routing Algorithms for Anycast Messages*. In Proceedings of the 1998 International Conference on Parallel Processing, ICPP '98, pages 122–, 1998.

[15] D. Xuan, W. Jia, and W. Zhao. *Integrated routing algorithms for anycast messages*. IEEE Communications Magazine, 38:48 – 53, 2000.

[16] M. Dorigo, V. Maniezzo, and A. Colorni. *Ant system: optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 26(1):29 –41, 1996.

[17] B. Barn and R. Sosa. *AntNet: Routing Algorithm for Data Networks based on Mobile Agents.* Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 5(12):75 – 84, 2001.

[18] M. Dorigo and G. D. Caro. *The ant colony optimization meta-heuristic*. In in New Ideas in Optimization, pages 11 – 32. McGraw-Hill, 1999.

[19] G. Pavani and H. Waldman. *Grid Resource Management by means of Ant Colony Optimization*. In 3rd International Conference on Broadband Communications, Networks and Systems, 2006. BROADNETS 2006, pages 1 – 9, 2006.

[20] I. Kassabalidis, M. El-Sharkawi, R. Marks, II, P. Arabshahi, and A. Gray. *Swarm Intelligence for Routing in Communication Networks*. In IEEE Globecom, pages 3613 – 3617, 2001.

[21] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

B

The Index-based Subgraph Matching Algorithm (ISMA): fast subgraph enumeration in large networks using optimized search trees.

This chapter presents research that was carried out in the field of bioinformatics. A graph algorithm was developed that finds all predefined subgraphs in large (biological) networks. Besides the design of an efficient network structure and the algorithm, a number of data structures proposed that speed up the calculations.

S. Demeyer, T. Michoel, J. Fostier, P. Audenaert, M. Pickavet, P. Demeester

Accepted for publication in PLoSONE, March 2013

Abstract Subgraph matching algorithms are designed to find all instances of predefined subgraphs in a large graph or network and play an important role in the discovery and analysis of so-called network motifs, subgraph patterns which occur more often than expected by chance. We present the index-based subgraph matching algorithm (ISMA), a novel tree-based algorithm. ISMA realizes a significant speedup compared to existing algorithms by carefully selecting the order in which the nodes of a query subgraph are investigated. In order to achieve this, we developed a number of data structures and maximally exploited symmetry char-

acteristics of the subgraph. We compared ISMA to a naive recursive tree-based algorithm and to a number of well-known subgraph matching algorithms. Our algorithm significantly outperforms the other algorithms, especially on large networks and with large query subgraphs. An implementation of ISMA in Java is freely available at https://sourcefourge.net/projects/isma.

B.1 Introduction

Over the last decade, network theory has come to play a central role in our understanding of complex systems in fields as diverse as molecular biology, sociology, economics, the internet, and others [1]. The central question in all these fields is to understand behavior at the level of the whole system from the topology of interactions between its individual constituents. In this respect, the existence of *network motifs*, small subgraph patterns which occur more often in a network than expected by chance, has turned out to be one of the defining properties of real-world complex networks, in particular biological networks [2]. Network motifs act as the fundamental information processing units in cellular regulatory networks [3] and they form the building blocks of larger functional modules (also known as network communities) [4–6]. The discovery and analysis of network motifs crucially depends on the ability to enumerate all instances of a given query subgraph in a network or graph of interest, a classical problem in pattern recognition [7], that is known to be NP complete [8].

Subgraph matching algorithms are usually classified as either exact algorithms, which require a strict correspondence between the query graph (i.e. the subgraph) and any match in the target graph, or inexact algorithms, where some deformation of the query graph is allowed when searching for a match [7]. Here we are only concerned with exact algorithms. Some of the most well-known algorithms that realize this are the algorithm of Ullmann [9], the VF [10, 11] and the VF2 algorithm [12, 13]. The algorithm of Ullmann uses the adjacency matrix representation of the networks. A number of auxiliary matrices are defined to determine the set of subgraph isomorphisms iteratively. In the VF algorithms, on the other hand, the networks are represented by graphs. A state space representation is used in which each state depicts a (partial) mapping between nodes of both networks. The algorithm recursively builds a network of states by adding to the present states a pair of nodes that can be mapped on each other. After computing a set of candidate pairs, for each pair it is checked whether it meets the feasibility rules. Only then a new state is created. The difference between the VF and the VF2 algorithm is that the exploration of the search space has been improved in the VF2 algorithm to reduce memory requirements. This means that it is faster and can also be applied in larger graphs.

Most of the other exact algorithms typically find subgraph isomorphisms in a

database of graphs. To realize the subgraph matching efficiently, a preprocessing step on this database is introduced. Messmer and Bunke [14] proposed a method consisting of building a decision tree from the database of graphs by a form of indexing. This structure can then be used to find all subgraph instances. This preprocessing step has been further optimized by Weber et al. [15]. Another algorithm, the GraphGrep algorithm [16], uses hash-based fingerprinting to index the database of graphs. GIndex [17] makes use of frequent substructures for its indexing. The GADDI algorithm [18] on the other hand deals with larger graphs and uses an indexing based on a neighborhood structure, similar to the TALE algorithm [19]. Another way to deal with exact subgraph matching is to reformulate it as a constraint satisfaction problem and solving it with constraint programming, which is a good approach if there are other constraints that need to be taken into account as well [20, 21].

Motivated by problems in biology, where it is necessary to find subgraph instances in graphs with certain characteristics on the links, which define the type of interaction between cellular components (e.g. protein-protein, protein-DNA or protein phosphorylation, etc.) [6, 22, 23], we developed a novel exact subgraph matching algorithm, which uses a search tree to find all instances of a query subgraph in an edge-colored graph without using an additional, usually time consuming, preprocessing step. The algorithms that resemble our algorithm most are the algorithm of Ullmann [9], the VF [10, 11] and the VF2 algorithm [12, 13]. Note that our problem differs from for example the SAGA algorithm [24] in which the nodes instead of the edges contain certain characteristics.

At the heart of our algorithm are custom designed data structures (for both the network and the algorithm) which provide, at each step in the subgraph matching procedure, rapid indexing of the candidate nodes for inclusion in a subgraph instance. By carefully selecting the order in which the motif nodes (denoted by an index) are investigated, these sets of candidate nodes are kept as small as possible. This allows to cut unfavorable branches in the search tree as soon as possible and leads to a dramatic speedup compared to existing algorithms. In this paper, we present a formal description of the Index-based Subgraph Matching Algorithm (ISMA), the data structures and how symmetries in the query subgraph are dealt with. This paper is organized as follows. After giving a general overview of the problem, together with the definitions of the concepts that are used in this article, a naive recursive algorithm is presented. The weaknesses of this algorithm are then identified and an improved recursive ISMA algorithm is proposed. As iterative algorithms may achieve a performance gain and require less stack space and function call overhead, an iterative version of the ISMA algorithm is presented, for which a number of custom data structures were designed. We also present comparisons to related subgraph matching algorithms using a variety of biological and non-biological networks.

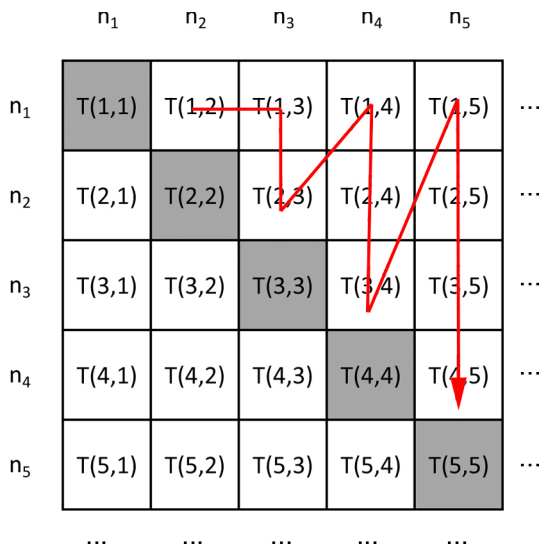

Figure B.1: ***The motif adjacency matrix.*** *In this article, motifs are denoted by a motif specification which can be deduced from its adjacency matrix as indicated by the red arrow.*

B.2 Methods

B.2.1 General Description

In biological networks, the same set of nodes (typically genes or proteins) can be connected in different ways, representing different physical interaction mechanisms, which may be directed or not [25]. In order to find matches for so-called composite motifs (subgraph patterns with more than one interaction type [22, 23]), the ISMA algorithm is designed to find all occurrences of a given query subgraph in graphs G_t with annotated edges. More precisely, $G_t = \{V, E\}$ with V the set of vertices (or nodes) and E the set of edges (or links), where each link is represented by a triplet (u, v, T) with u and v the start and end node respectively, and T the type of the link. Hence, in contrast to ordinary graphs, the links now also have a type, which identifies a number of characteristics of the link such as whether it is directed or undirected. It should be noted that parallel links are allowed in G_t if and only if they are of a different type.

A motif or query subgraph M is defined as follows. It is a small graph of k nodes with no (anti-)parallel links. This means that a motif has at most $K = \frac{k(k-1)}{2}$ links. The assumptions of no (anti-)parallel links is not a strict condition and the research presented here can easily be extended to motifs with (anti-)parallel links. There are four possible configurations between two nodes n_1 and n_2: no

link, a directed link from n_1 to n_2, a directed link from n_2 to n_1, or an undirected link between n_1 and n_2. The motif nodes (n_i) are ordered and can hence be referred to by a unique index i. In the remainder of this article, this index will be used to refer to the motif nodes themselves. Motifs are specified by a list of K link types as follows:

$$[T(1,2), T(1,3), T(2,3), T(1,4), T(2,4), \ldots, T(k-1,k)]$$

with $T(i,j)$ the link type between the i-th and the j-th node of the motif. It is defined that if no link exists between two nodes in a motif the corresponding link type is set to $null$ (or '0'). This motif specification can easily be deduced from the adjacency matrix of the graph as indicated in figure B.1.

Link types may be specified by upper case characters (A, B, etc.). In the case the links are directed, the reverse of a link can be represented by the corresponding lower case character. A number of examples of motifs and their specification are given in figure B.2.

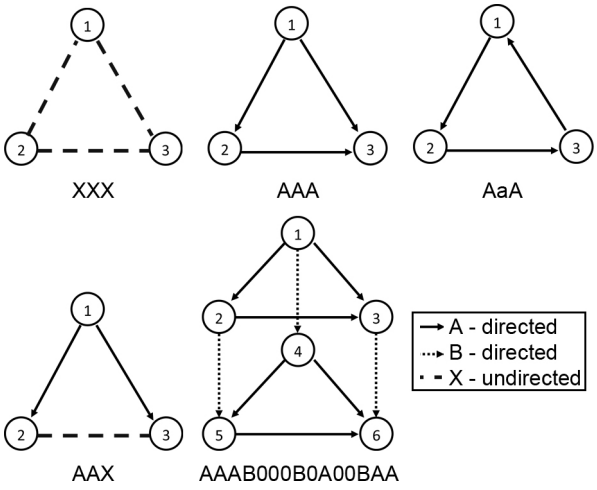

Figure B.2: ***Examples of motifs and their specifications.*** *Here nodes are denoted by their index. Look at for example the motif AAAB000B0A00BAA. Its motif specification can be deduced as follows: a directed link of type A from node 1 to node 2, a directed link of type A from node 1 to node 3, a directed link of type A from node 2 to node 3, a directed link of type B from node 1 to node 4, no link between node 2 and node 4, no link between node 3 and node 4, no link between node 1 and node 5, a directed link of type B from node 2 to node 5, no link between node 3 and node 5, a directed link of type A from node 4 to node 5, no link between node 1 and node 6, no link between node 2 and node 6, a directed link of type B from node 3 to node 6, a directed link of type A from node 4 to node 6, a directed link of type A from node 5 to node 6.*

B.2.2 The Naive Recursive Subgraph Matching Algorithm (RSMA)

In this section, a naive recursive subgraph matching algorithm is described which is implemented in a motif clustering software tool (Cyclus 3D) [26]. We describe this algorithm in detail here, as it will form the basis for the ISMA algorithm. It is a depth first tree search procedure in which the motif nodes are investigated in the order in which they are listed in the specification. This means that the algorithm will first map a network node on the first motif node, then on the second motif node, and so on.

It should be noted that this algorithm resembles the VF2 algorithm. However, it is not completely similar. In the VF2 algorithms first a set of candidate pairs (i.e. a network node and a motif node on which this network node can be mapped) is calculated and subsequently this set is filtered according to the feasibility rules. One of these rules, for example, checks whether the links have the correct attributes (i.e. link types). By a careful design of the network data structure in the ISMA algorithm all candidate nodes are feasible nodes, which means that no additional checking operation is needed.

The pseudocode of the recursive subgraph matching algorithm (RSMA) is given below. The algorithm (i.e. the function `findMotifs`) takes 3 input parameters: a motif specification `mspec`, the motif instance `instance` that tracks which network nodes have been mapped on the motif nodes, and the network G_t. In each recursive call, it is first checked whether the instance is complete, i.e. whether all motif nodes have network nodes mapped on them. If this is the case, the motif instance is exported. Otherwise, the next motif node (specified by its index) to be investigated is identified by the function `next()`. Here, `next()` returns the smallest index that has not been investigated yet. Subsequently, a set is determined of all network nodes that can be mapped on this motif node. This set contains all network nodes that are connected to the network nodes that were already mapped in the instance by links of the correct type (according to the motif specification). For the first motif node, this set simply consists of all nodes of the network G_t. One by one, the network nodes in this set are mapped onto the motif node, after which the `findMotifs` routine is called recursively. This way, all instances in G_t corresponding to the motif specification are enumerated.

Recursive Subgraph Matching Algorithm

Abbreviations:

mn = motif node
nn = network node

```
findMotifs(mspec, instance, G_t){
    if(instance is complete){
        export(instance);
        return;
    }
    mn = next();
    set = determineSet(instance, mn, G_t);
    forall(nodes nn in set){
        instance.put(mn, nn);
        findMotifs(mspec, instance, G_t);
        previous();
        instance.remove(nn);
    }
}

i = 0;
next(){
    return i++;
}

previous(){
    i--;
}

determineSet(instance, mn, G_t){
    if(index==1){
        return V;
    }else{
        sets = EMPTY;
        forall(nodes ni in instance){
            motifIndex = instance.getIndex(ni);
            linkType =
                mspec.getLinkType(motifIndex, mn);
            set = ni.getNeighborsofType(linkType);
            sets.add(set);
```

```
35              }
36              return intersection(sets);
37          }
38      }
```

The RMSA algorithm has some efficiency issues. When the set of candidate network nodes is determined for the first motif node, the complete set of network nodes is returned (line 22). It is possible that a lot of these network nodes don't even have the correct links (according to the motif specification) departing from them and thus are bad candidates to be mapped on the first motif node. This means that the search tree is very broad near the root. It would be better to narrow this down and determine a set of good candidate nodes by checking the types of the links that are departing from the network nodes and only select those nodes that have links of the same type as the links from the first motif node. Moreover, this set of candidate nodes can be further reduced by selecting another motif node to be investigated first. By changing the order in which the motif nodes are investigated (in the `next()`-function), the sets of candidate network nodes can be kept as small as possible. Smaller sets lead to less branches in the search tree and thus faster calculation times.

B.2.3 Operation of the ISMA algorithm: an example

In this section, we will sketch the operation of the index-based subgraph matching algorithm by means of an example. In the three following sections, the algorithm is described in full detail.

Suppose we want to find all occurrences of the motif ABC (with link types A and B directed, and link type C undirected) in the network depicted in figure B.3. In the initialization phase, the best motif node to be investigated first is determined. As we want to narrow down the search tree, this should be the motif node with the least number of possible network nodes that can be mapped on it. These network nodes are the nodes that have the same types of outgoing links as the motif node. The set of possible network nodes for each motif node can then be determined by calculating the intersection of the sets of start network nodes of the corresponding link types. As calculating these intersections can be quite time-consuming, we opted to calculate the number of start nodes for each link type of the motif and select the motif node from which a link departs of the type with the lowest number of start nodes. This is shown in table B.1. In practice, these sets of network (start) nodes are retrieved in constant time since they are stored in the data structure of the network itself (see section on data structures). It should be noted that if some of the links are directed, the occurrences of the reverse links should also be taken into account. In our example, this means that the sets of starting network nodes from the link types a and b also need to be determined. The link type with the

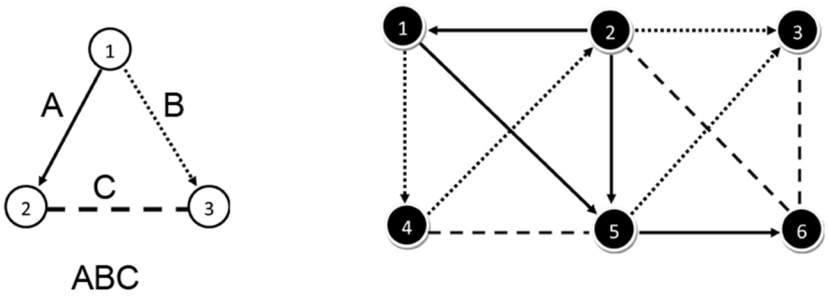

Figure B.3: ***Example motif and network.*** *The motif (left) which is searched for in the example network (right). Links of type A or B are directed; links of type C are undirected.*

link type	{**nn**}	**# nn**	**link type**	{**nn**}	**# nn**
A	{1,2,5}	3	a	{1,5,6}	3
B	{1,2,4,5}	4	b	{2,3,4}	3
C	{2,3,4,5,6}	5			

Table B.1: The initialization phase. For each link type the set of network nodes is depicted together with its cardinality

lowest cardinality of its corresponding set determines the first motif node, namely the motif node from which a link of this type departs.

If there are multiple link types with the same cardinality, there are two possibilities: randomly picking one of these link types or actually calculating the sets of possible network nodes that can be mapped on the motif nodes. Here, we will apply the latter and calculate for the concerned motif nodes the intersection of the sets of starting network nodes of the links of which the types are specified in the motif. In our example, we encounter the same cardinality for the link types A, a and b, which correspond to motif nodes 1 (for A), 2 (for a) and 3 (for b). For the motif node 1, the set of possible network nodes is the intersection of the start sets of link type A ({1,2,5}) and link type B ({1,2,4,5}), thus set {1,2,5}. Similarly, for motif node 2 (link types a and C) and motif node 3 (link types b and C) this results in {5,6} and {2,3,4} respectively. From this we can conclude that motif node 2 is the best option to be investigated first as its set of candidate network nodes only has 2 elements, namely network nodes 5 and 6.

We will map one of these nodes, network node 5, on motif node 2. Now it needs to be determined which of the 2 remaining motif nodes (1 or 3) is the best option to be investigated next. To do this, we determine the cardinality of the sets of network nodes that are neighbors of network node 5 according to the correct link types in order to be mapped on the specific motif nodes. For motif node 1 (connected to motif node 2 with link type a) this set of network nodes is {1,2},

while for motif node 3 (connected to motif node 2 with link type C) this set is $\{4\}$. As there is only one network node that can be mapped on motif node 3, we will consider this motif node to be investigated next.

Network node 4 is mapped on motif node 3. Now, we need to determine which network nodes can be mapped on the last motif node, namely node 1. This is the intersection of the set of neighbors of network node 5 according to link type a ($\{1,2\}$) and the set of neighboring network nodes of node 4 according to link type b ($\{1\}$). This results in singleton $\{1\}$. We now have a complete instance that can be exported.

As there are no other network nodes that can be mapped on motif node 1 (all nodes of the set $\{1\}$ have been mapped), and no other network nodes can be mapped on motif node 3 (all nodes of the set $\{4\}$ have been mapped), we will map the next network node on motif node 2, namely node 6 (from the set $\{5,6\}$ determined at initialization). Again, it will be determined which of the remaining motif nodes (1 or 3) will be investigated first. For motif node 1, the set of possible network nodes (neighbors of network node 6 according to link type a) is $\{5\}$. For motif node 3 (connected to motif node 2 with link type C) this set is $\{2,3\}$. Now motif node 1 is the best option to be investigated first as there is only one network node that can be mapped on it. Network node 5 is mapped on motif node 1. To determine which network nodes can be mapped on the last motif node 3, the intersection is calculated between the set of neighbors of network node 6 according to link type C ($\{2,3\}$) and the set of neighbors of network node 5 according to link type B ($\{3\}$). This results in the singleton $\{3\}$. Network node 3 is mapped om motif node 3 and the complete instance can be exported.

There are no more network nodes that can be mapped on motif node 3 (all nodes of the set $\{3\}$ have been mapped), and no more network nodes that can be mapped on motif node 1 (all nodes of the set $\{5\}$ have been mapped). Moreover, we iterated over all network nodes that can be mapped on motif node 2 (all nodes of the set $\{5,6\}$ have been mapped). This means that the algorithm can terminate and has found all instances of the motif ABC in the network. Two motif instances have been found, one with the network nodes 1, 5 and 4 mapped on motif nodes 1, 2 and 3 respectively, and one with the network nodes 5, 6 and 3.

Similar to the naive recursive algorithm, this algorithm is also a depth-first search algorithm. When m motif nodes have network nodes mapped on them, first all possibilities for the remaining $k - m$ nodes (assuming the motif has k nodes) will be checked, before mapping the next network node on the m-th motif node. As mentioned before, in this algorithm motif nodes are not always investigated in the same order. In the above example, in both instances motif node 2 was investigated first, but for the first instance motif node 3 was the next to be investigated, while for the second instance this was motif node 1. By carefully selecting the order in which the motif nodes are investigated, the sets of network nodes that can be

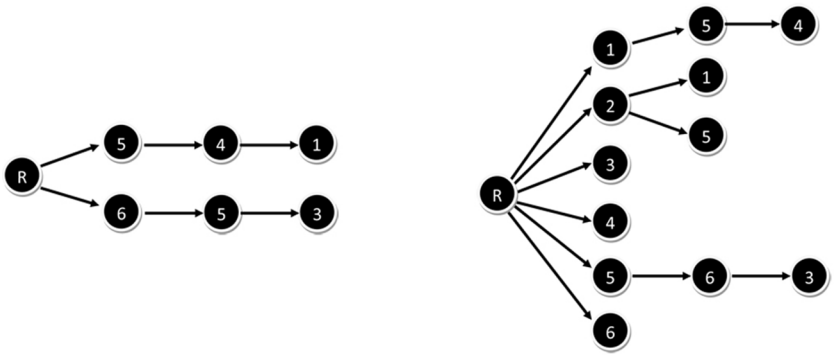

Figure B.4: ***Search tree.*** *The search tree of the ISMA algorithm (left) and the standard recursive algorithm (right) applied to the example network. A search tree indicates which network nodes have been mapped on the motif nodes.*

mapped on the motif nodes are kept as small as possible, reducing the number of branches in the search tree.

The search trees of both the RSMA and the ISMA algorithm are depicted in figure B.4. It can be seen that the ISMA algorithm indeed has a significantly smaller search tree, namely 6 nodes instead of 12.

B.2.4 The Recursive Index-based Subgraph Matching Algorithm

Below, the pseudo code of the recursive index-based subgraph matching algorithm is presented. One can see that the main algorithm (i.e. the `findMotifs`-function) is similar to the one of the naive recursive algorithm. The `determineSet`-function only differs in the case in which no network nodes have been mapped on the motif nodes yet. Instead of returning the complete set of network nodes, now a set of good candidate network nodes for this motif node is returned. These are the networks nodes that have the same links (or more accurately link types) departing from them as the specific motif node.

The `next`-function returns the best motif node to be investigated next. This is the motif node that has the smallest set of candidate network nodes, as this leads to a smaller search tree. In the case the motif instance is empty (i.e. no network nodes have been mapped on the motif nodes), we will determine for each link type the number of networks links of this type. The link type with the lowest cardinality determines the best motif node. As stated in the previous section, when multiple link types have the same cardinality (i.e. the boolean variable `multiple` is true), there are two options: randomly selecting one of these link types or calculating the sets of possible network nodes that can be mapped on the motif nodes. The code depicts the latter one. For each motif node the set of candidate network nodes is

determined. These are the network nodes with the same outgoing link types as the motif node. The motif node with the smallest set of candidate nodes is the best option to be investigated first. In the case where some of the motif nodes have network nodes mapped on them, we will determine for each of the unmapped motif nodes the number of neighbors of the mapped network nodes that can be mapped on this motif node. The minimum number then determines which motif node will be investigated next. By always selecting the motif node with the smallest set of candidate network nodes, the number of branches in the search tree is kept as small as possible, leading to faster calculations.

Recursive ISMA Algorithm

```
1     findMotifs(mspec, instance, G_t){
2         if(instance is complete){
3             export(instance);
4             return;
5         }
6         mn = next(instance, mspec);
7         set = determineSet(instance, mn, G_t, mspec);
8         forall(nodes nn in set){
9             instance.put(mn, nn);
10            findMotifs(mspec, instance, G_t);
11            instance.remove(nn);
12        }
13    }
14
15    next(instance, mspec){
16        min = LARGE_NUMBER;
17        if(instance is empty){
18            multiple = FALSE;
19            linkTypes = mspec.getLinkTypes();
20            forall(types t in linkTypes){
21                #NBS = #{G_t.getStartNodesofType(t)};
22                if(#NBS<min){
23                     min = #NBS;
24                     i = t.getStartNode();
25                     multiple = FALSE;
26                }else if(#NBS == min){
27                     multiple = TRUE;
28                }
29            }
30            if(multiple){
```

```
31                forall(motif nodes mn){
32                    #NBS = #{determineSet(instance,
                                        mn, G_t, mspec)};
33                    if(#NBS<min){
34                         min = #NBS;
35                         i = mn;
36                    }
37                }
38            }
39        }else{
40            forall(unmapped motif nodes mn){
41                forall(mapped network nodes nn){
42                    #NBS =number of neighbors of nn
                           that can be mapped on mn;
43                    if(#NBS<min){
44                         min = #NBS;
45                         i = mn;
46                    }
47                }
48            }
49        }
50        return i;
51    }
52
53    determineSet(instance, mn, G_t, mspec){
54        if(instance is empty){
55            linkTypes = mspec.getLinkTypesFrom(mn);
56            forall(types t in linkTypes){
57                set = G_t.getStartNodesofType(t);
58                sets.add(set);
59            }
60            return intersection(sets);
61        }else{
62            sets = EMPTY;
63            forall(nodes ni in instance){
64                motifIndex = instance.getIndex(ni);
65                linkType=mspec.getLinkType(motifIndex,
                                            mn);
66                set=ni.getNeighborsofType(linkType);
67                sets.add(set);
68            }
```

```
69              return intersection(sets);
70          }
71      }
```

B.2.5 Data Structures

As shown in the example and the recursive ISMA algorithm, the execution time of the recursive algorithm can be reduced by an intelligent choice of the order in which the nodes of a motif are investigated. This way, unfavorable branches of the search tree are pruned as soon as possible. Moreover, by an intelligent design of the network data structure an additional speedup can be realized for both the RSMA and the ISMA algorithm.

This section starts with an overview of the optimizations to the network data structure which enable fast retrieval of the network nodes adjacent to links of a certain type. Subsequently, a number of data structures are presented in order to realize an iterative version of the ISMA algorithm (which is more efficient and requires less stack space). The checklist keeps track of the order in which the motif nodes are investigated. A motif iterator is a collection of iterators that iterate over the possible network nodes for each of the motif nodes. The priority queue map is used in the algorithm to determine which of the motif nodes is the most lucrative to be investigated next. Investigating a motif node here means determining a set of network nodes that can be mapped on it, and adding each of these nodes to the instance one after the other.

To better understand the algorithm specific data structures, figure B.6 presents an example in which the possible content of the data structures is given for one instant during the execution of the algorithm.

B.2.5.1 Optimizations to the network data structure

The network data structure contains a network (i.e. graph). The main structure consists of a list of nodes. Each of these nodes has a list of neighbors, together with the links connecting them. The data structure has been optimized for two specific operations in the ISMA algorithm: the retrieval of the start nodes of all links of a specific type and the retrieval of all neighbors of a node that are connected to that node with a link of a specific type. This has been accomplished by 2 changes:

- In the network structure a map is added with the link type as key and the set of the start nodes of all links of this link type as value. As the number of link types in a network is relatively small, finding the start nodes of all links of a specific type now only requires a small amount of execution time.

  However, this structure requires some additional memory. In a map, for each link type a set of reference to nodes (i.e. start nodes of links of this type) is

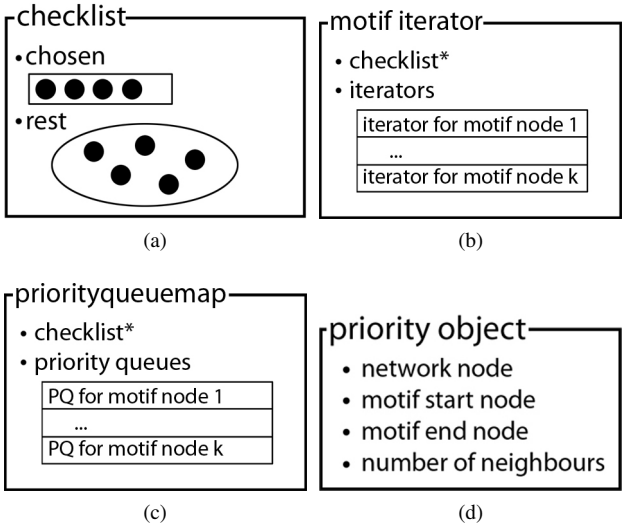

Figure B.5: ***Data Structures.*** *(a) The checklist. In the ISMA algorithm, the circles represent motif nodes (b) The motif iterator. (c) The priority queue map. (d) The priority object. It is assumed that the motif has k nodes.*

stored. Assuming that the network has $|E|$ links and that there are $|T|$ link types, this means that the additional memory needed is equal to $(|E| + |T|)$ references plus the memory overhead of one map and $|T|$ sets.

- For every node a map is kept with the link types as keys and the sets of all neighbors according to this link types as values. This speeds up the operation of finding all neighbors that are connected with a link of a specific type.

  As in 'non-optimized' networks nodes also contain references to their neighbors, this structure only needs a small amount of additional memory, namely $|T|$ references to the link types plus the memory overhead of one map and $|T|$ sets.

B.2.5.2 Checklist

A checklist is a data structure that keeps track of the order in which elements (motif nodes) are chosen from a collection. It contains an ordered list of the chosen elements, together with the set of all elements that have not been chosen yet. It should be noted that in the recursive algorithm, this information is kept in the stack. The check list data structure is illustrated in figure B.5a.

In the example (see figure B.6b) first motif node 2 was removed from the rest set and added to the chosen list. Subsequently motif node 4 was removed from the

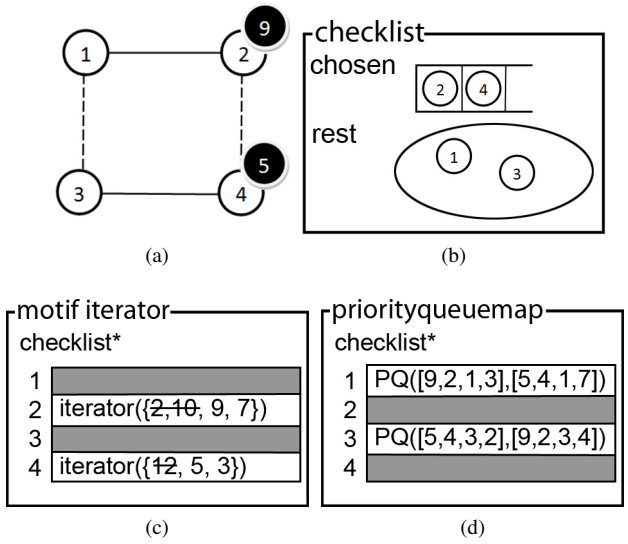

Figure B.6: ***Example of data structures.*** *We are looking for a 4-node motif. In the motif instance (a) network node 9 is mapped on motif node 2 and network node 5 is mapped on motif node 4. These motif nodes were added (in the correct order) to the chosen list of the checklist (b), while the other two motif nodes (1 and 3) remain in the rest set. The motif iterator (c) contains two iterators that are of importance, namely the ones for the motif nodes of the chosen list. These iterate over the possible network nodes that can be mapped on the motif nodes. The priorityqueuemap (d) only contains valuable priority queues for the motif nodes 1 and 3. Each priority queue contains a priorityobject for each network node that is already mapped in the instance.*

rest set and added to the chosen list. Two nodes (1 and 3) remain in the rest set.

Following functions are defined on an checklist:

- `numberChecked()` : returns the number of chosen elements
- `lastChecked()` : returns the last element that has been chosen
- `check(element)` : removes `element` from the `rest`-set and adds it to the list of chosen elements
- `uncheck()` : removes the last element from the `chosen`-list and adds it to the `rest`-set
- `checked()` : returns the list of checked (i.e. chosen) elements
- `rest()` : returns the `rest`-set

B.2.5.3 Motif iterator

The motif iterator contains an iterator for each of the motif nodes that iterates over the possible network nodes that can be mapped on this motif node. Additionally, in order to know the order in which the motif nodes have been investigated, it contains a pointer to a checklist. When a motif node has not been investigated yet, the corresponding iterator is set to $null$. This data structure is depicted graphically in figure B.5b.

Whereas for the 'first' motif node the iterator stays the same during the complete execution of the algorithm, the iterators for the other motif nodes will change. Every time a motif node is determined as the best to be investigated next, a new iterator (that iterates over the set of possible network nodes that can be mapped on this motif node) is added to the motif iterator. The motif iterator will always first iterate over the set of nodes that can be mapped on the motif node that was last checked in the check list. Once it has iterated over all these network nodes, it will iterate further on the set of network nodes that can be mapped on the previous motif node according to the checklist. This explains the need of the checklist.

In the example the motif iterator (figure B.6c) has an iterator over the set (of network nodes) $\{2, 10, 9, 7\}$ for motif node 2 and an iterator over the set $\{12, 5, 3\}$ for motif node 4. The other iterators are of no importance at this stage in the algorithm. At this moment network node 9 is mapped on motif node 2, which means that network nodes 2 and 10 were already mapped on motif node 2. Similarly network node 5 is mapped on motif node 4 meaning that network node 12 was already mapped on this motif node. When the algorithm continues, after finding network nodes that can be mapped on motif nodes 1 and 3, network node 3 will be mapped on motif node 3. When this iterator finishes, it is removed from

the motif iterator and the search continues by mapping network node 7 on motif node 2.

Following functions are defined on an index iterator:

- `put(motifnode, iterator)`: adds the `iterator` to the corresponding `motifnode`
- `hasNext()`: returns a boolean value indicating if any of the iterators has a next element. It will first check the iterator of the `checklist.lastChecked()`. If it is empty, it will check the iterator of the previous motif node. And so on.
- `next()`: returns the next element of the index iterator. This is the next element of the current motif node in the index list

B.2.5.4 Priorityqueuemap

The priorityqueuemap is an instrument to determine the best possible motif node to be investigated next. It contains a priority queue for each motif node. Moreover, in order to know which of the motif nodes have not been investigated yet, a pointer to the checklist object is kept. This data structure is illustrated in figure B.5c. It is similar to the motif iterator, but the iterators are substituted by priority queues (PQ).

In order to keep the search tree as small as possible, we want to select the (uninvestigated) motif node with the smallest set of possible network nodes that can be mapped on it. This set is the intersection of all the neighbor sets (a neighbor set is the set of all the network nodes that are connected to a mapped network node according to a specific link type) of nodes that can be mapped on this motif node. Since calculating this intersection can be time-consuming and the maximum cardinality of this intersection is the cardinality of the smallest of these neighbor sets, in the priority queues we will keep track of how many neighbors each mapped network node has according to the types of each of its outgoing links (according to the motif specification). By only taking into account the (cardinality of the) sets of neighbors of the mapped network nodes and not the intersection of these sets for one motif node, we do not produce the optimal (i.e. the smallest) search tree, but it is a good compromise between efficiency (calculating the intersection is time-consuming) and optimality.

The objects that are stored in the priority queues were designed specifically for the ISMA algorithm. We opted to name them priority objects (see figure B.5d). Priority objects consist of 4 fields: a network node, a motif start node, a motif end node and the number of neighbors of the network node according to the type of the link between the motif start node and end node. The network node is the node of the current instance that has been mapped on the motif start node. A priority

object contains 2 motif nodes, a start and an end node, from which the link type can be deduced. It should be noted that for one priority queue all the 'motif end node' fields are equal, and thus could be omitted. To determine the next best motif node, for each of the priority queues of the motif nodes that have not been chosen yet the minimum number of possible neighbors is retrieved. The overall minimum determines which motif node will be chosen next, as the maximum cardinality of the set of possible network nodes is minimal for this motif node.

In the example (see figure B.6d) priority objects are indicated by a list of 4 elements ([network node, motif start node, motif end node, number of neighbors]). Here only the priority queues of motif node 1 and 3 are of importance. Each priority queue contains 2 priority object, one for each of the mapped network nodes. Motif node 3 would be selected as the best motif node to be investigated next, since network node 5 has the least number of neighbors that can be mapped on it.

The following functions are defined on a priorityqueuemap:

- `add(priorityqueueObject)`: adds the `priorityqueueObject` to the correct priority queue according to its motif end node
- `poll()`: removes and returns the overall best element (i.e. priority object) of the priority queues

B.2.6 The Index-based Subgraph Matching Algorithm (ISMA)

As mentioned previously, the recursive ISMA algorithm outperforms the naive recursive algorithm by always selecting the motif node with the smallest set of possible network nodes that can be mapped on it. In this way, the number of branches in the search tree is minimized.

Moreover, iterative algorithms can improve the performance and consume less stack space and function call overhead. In the previous section a number of data structures were presented in order to realize an iterative version of the ISMA algorithm. In this section the actual algorithm is discussed.

The pseudo code of the iterative ISMA algorithm is given below.

Index-based Subgraph Matching Algorithm (ISMA)

Abbreviations:

`CL` = checklist
`MI` = motif iterator
`PQM` = priorityqueuemap
`PO` = priority object

```
1     findMotifs(mspec, G_t){
2         instance = EMPTY_INSTANCE;
3         mn = determineFirstMotifNode();
4         CL.check(mn);
5         {startset} = determineSet(instance, mn,
                                    G_t, mspec);
6         MI.put(mn, startset.iterator());
7
8         while(MI.hasNext()){
9             nn = MI.next();
10            backtrack();
11            instance.putNode(CL.lastChecked(), nn);
12            if(instance is complete){
13                export(instance);
14                continue on line 8;
15            }
16            forall(i in CL.rest()){
17                PQM.add(new PO(nn, CL.lastChecked(),
                                 i, #NBS));
18            }
19            mn = PQM.poll().getEndNode();
20            CL.check(mn);
21            set = determineSet(instance, mn, G_t);
22            MI.put(mn, set.iterator());
23        }
24    }
```

In the initialization phase (line 2-6) we will stipulate which of the motif nodes is best suited to be handled first (line 3). To determine this, for each link type present in the motif the number of occurrences in the network is counted. The start (motif) node of the link of the type with the least instances in the network is chosen to be the first motif node. If multiple motif nodes have an outgoing link of this type or multiple link types have the same number of instances, there are two options. One could randomly select one of these link types to determine the first motif node fast, but at the risk of creating a unnecessary branches in the search tree. The other option is to calculate the actual sets of candidate network nodes for these motif nodes by intersecting the sets for each of the link types departing from this motif node. Then the motif node with the smallest set of candidate network nodes is selected. This comes down to calculating the following intersection value (IV) for all motif nodes mn from which links of the specific types depart:

$$IV(mn) = \#IS(mn) = \#\{ \bigcap_{j|j\in M, j\neq mn} \{k|(k,l,type(mn,j)) \in E\}\}$$

This is in fact the cardinality of the intersection (IS) of all sets of network start nodes of the links of the types that depart from mn. For mn all types of the links departing from it are determined. For each of these link types the set of starting nodes in the network is collected and the cardinality of the intersection of all these sets is calculated. The minimal value (for the different motif nodes) of this parameter then determines which motif node will be handled first. The determination of the first motif node is heuristic in the sense that we want to find a good first motif node as soon as possible, while we cannot guarantee that the search tree we build is indeed the smallest one possible. Once it is known which motif node is the first to be investigated, a start set of network nodes is calculated (line 5). It consists of all network nodes that can be mapped onto this motif node. It should be noted that the `determineSet`-function is identical to the one used in the recursive ISMA algorithm. The chosen motif node is checked in the checklist (line 4), and an iterator over the determined start set is added to the motif iterator (line 6).

The main part of the algorithm executes the following as long as there are network nodes in the motif iterator (line 8). It retrieves the next network node from the motif iterator, and adds it to the instance on the position of the last checked motif node (line 9 and 11). If the instance is complete (i.e. all motif nodes have network nodes mapped on them), it is exported and the algorithm continues by retrieving the following network node from the motif iterator (line 12-15). If this is not the case, for all the motif nodes that have not been handled yet it is determined how many neighbors of this network node can be mapped on them and the results are added, in the form of a priority object, to the priorityqueuemap (line 16-18). Subsequently, the next best motif node is determined by retrieving the best priority object from the priorityqueuemap (line 19). This motif node is checked in the checklist, and an iterator over a set of network nodes is added to the motif iterator (line 20-22). This set is the result of the function `determineSet` that is identical to the one used in the recursive ISMA algorithm. For a complete description of this function we would like to refer to the section of the recursive algorithm.

The algorithm terminates when there are no more network nodes left in the motif iterator. This means that there are no more network nodes that are good candidates to be mapped on the motif nodes. All motif instances have thus been found.

It should be noted that, when the next motif node is retrieved from the motif iterator (line 9), the data structures are updated to allow backtracking. This is indicated by the `backtrack`-procedure (line 10). There are three possible situations. When a new iterator was added in the previous iteration, no updates are needed, and the algorithm can continue. If the motif iterator retrieves a motif node from the same iterator as in the previous iteration, both the motif instance and the priorityqueuemap need to be updated. The network node that was mapped in the

previous iteration needs to be removed from the instance, so that a new network node can be mapped. Moreover, all priority objects that are associated with this previous network node need to be removed from the priorityqueuemap. If, on the other hand, the network node that is returned comes from an iterator that is associated with a motif node that was handled previously, all data structures are updated. In the checklist the motif nodes that have been investigated completely (i.e. the algorithm has iterated through all network nodes that can be mapped on them in the current situation) need to be unchecked. In the priorityqueuemap all priority objects that have these motif nodes as start node are removed. Moreover, all network nodes that are mapped on motif nodes of the `rest`-set of the checklist are removed from the instance, as well as the network node that is mapped on the current motif node (i.e. `lastChecked()`). The priority object associated with this last network node are also removed from the priorityqueumap.

B.2.7 Dealing with Symmetry

In this section it will be explained how the ISMA algorithm can be further optimized when dealing with symmetric motifs. A motif is called to be symmetric if it is identical to a motif from which the nodes are permutated in a certain way. The most common symmetries in motifs are reflections, rotations, translations and combinations of these three. By making use of the symmetry characteristics of a motif, the search tree of the (iterative) ISMA algorithm can be pruned further. Once a network node has been mapped on a motif node that takes part in a symmetric permutation, it should not be mapped again on the other nodes of this symmetry if this would lead to the same motif instance. In this paper, we will focus on two kind of symmetries, namely reflections (or mirror symmetry) and cyclic rotations, as these can easily be exploited to speed up the calculations. In the future, we plan to take into account all sorts of symmetries (like the algorithm of [27] does). At this time, for all other symmetries, duplicate instances will be eliminated once they have been found.

A motif contains a reflection symmetry if and only if two or more nodes can be swapped without changing the motif's configuration. For example, the motif AAX in figure B.7 has a reflection symmetry between node 2 and node 3. A cyclic rotation symmetry is a symmetry in which nodes can be moved in circles. An example of a motif with this type of symmetry is AaA (figure B.7) where the motif with nodes $[1, 2, 3]$ is equal to the motif with nodes $[2, 3, 1]$ and the one with nodes $[3, 1, 2]$.

The idea behind 'dealing with symmetry' is that once we have mapped one network node on a motif node that is part of a symmetry, we do not want it mapped again on another motif node of the symmetry. Suppose that we have a reflection symmetry with s motif nodes and that there are t network nodes that can be mapped

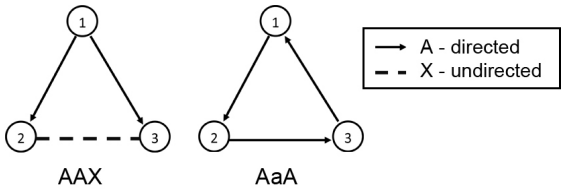

Figure B.7: ***Examples of reflection and cyclic rotation symmetries.*** *The motif on the left has a reflection symmetry between nodes 2 and 3. The motif on the right has a cyclic rotation symmetry between the three nodes.*

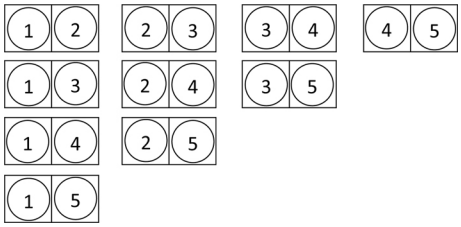

Figure B.8: ***Reflection symmetry.*** *Enumeration of all possibilities to map 5 network nodes on 2 reflection symmetric motif nodes. The squares represent motif nodes, the circles represent network nodes. Once a network node has been mapped on a motif node that is part of a reflection symmetry, it will never be mapped on one of the other nodes of the symmetry.*

on them, then mapping the network nodes on these symmetric motif nodes comes down to choosing s distinct elements out of a set of t elements, not taking into account the order of the elements, which is in fact a combination of s elements out of a set of t elements. One way to enumerate all these combinations is by first summing up all sets with the first element, then all sets with the second element that have not been encountered yet, etc. This is illustrated in figure B.8. From this, it can be seen that, once one network node is mapped on a symmetric motif node (in the figure the first motif node), it will never be mapped again on one of the other symmetric motif nodes that are investigated thereafter.

For cyclic rotation symmetries this is slightly different. Here one motif node needs to be chosen as the 'first' node of the rotation and the idea is that once a network node is mapped on this 'first' node, it cannot be mapped on one of the other motif nodes in the symmetry, while a network node that is mapped on the 'second' (or higher) motif node still could be mapped on the other nodes of the symmetry. This is illustrated in figure B.9 where all possibilities are given to map 5 network nodes on 3 motif nodes that form a cyclic rotation. Once a

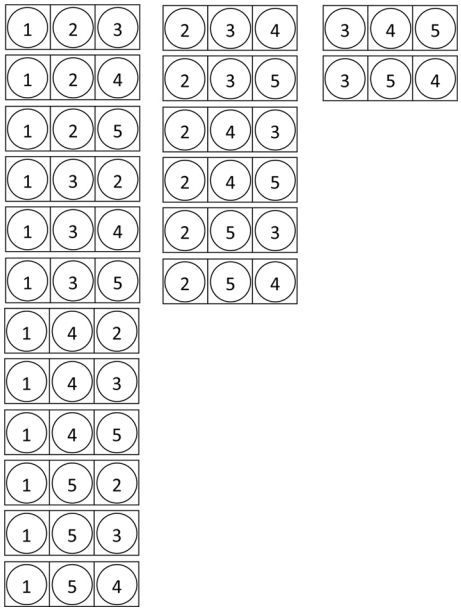

Figure B.9: ***Cyclic rotation symmetry.*** *Enumeration of all possibilities to map 5 network nodes on 3 motif nodes that are part of a cyclic rotation. The squares represent motif nodes, the circles represent network nodes. Once a network node is mapped on the 'first' motif of the symmetry, it will never be mapped on the other nodes of the symmetry. For the other motif nodes of the symmetry, all possibilities still need to be explored.*

network node has been mapped on the 'first' motif node of the rotation, it will not be mapped again on one of the other nodes in the rotation, while network nodes that are mapped on the 'second' motif node, still can be mapped on the 'third' motif node later on.

In order to realize a speedup by making use of these symmetry characteristics, some changes were made to the motif data structure and the algorithm. Additional information is stored in the motif, namely for each motif node a list is kept of the motif nodes with which it is symmetric. In the case of reflection symmetry, for each node of the symmetry this list contains all other nodes of the symmetry. In the case of cyclic rotations for the motif node, that has been chosen as the 'first' node, this list contains all other symmetric nodes, while the lists of the other motif nodes only contain one element, namely the 'first' motif node. Next to these changes in the motif definition, a novel data structure, called symmetry sets, was developed. For each symmetric motif node it contains a set of all possible network nodes that have not been mapped on it yet. These sets are used to help determining the new set of candidate network nodes (line 21 of the ISMA algorithm). If the motif node is

symmetric, the (symmetry) sets of all nodes that are symmetric to it (according to the symmetry structure that was added in the motif definition), are added to the set of sets in the `determineSet` procedure. As these sets only contain the network nodes that have not been mapped yet, they make sure that all network nodes that have been mapped on symmetric motif nodes are eliminated when calculating the intersection.

In order to deal with symmetry (reflection or cyclic rotation) and further speed up the calculations, the ISMA algorithm is adapted in three ways:

- Every time it is determined which motif node will be investigated next (line 19), it is checked whether this node is part of a symmetry with motif nodes that have not been investigated yet. If this is the case, the set of network nodes that can be mapped on this motif node is added to the symmetry sets data structure. This is realized by adding following code after line 21.

```
21a if(mn.isSymmetric()){
21b   symmetrySets.put(mn, set);
21c }
```

- Every time a network node is retrieved from the motif iterator (line 9), the symmetry sets are updated. This means that, if this network node is mapped on a symmetric motif node, this network node is removed from the set associated with this motif mode. In this way it will not be mapped on the other motif nodes of the symmetry. This is realized by adding following code after line 11.

```
11a if(CL.lastChecked().isSymmetric()){
11b   symmetrySets.remove(CL.lastChecked(), nn);
11c }
```

- When the `determineSet` procedure is called, it is checked whether the motif node (i.e. `index`) is symmetric to nodes that have been investigated before. If this is the case, the corresponding (symmetry) sets are retrieved from the symmetry sets data structure and added to the set of sets, from which the intersection is calculated (line 55 of the recursive ISMA algorithm). This is realized by adding following code after line 54 in the `determineSet`-procedure.

```
54a forall(mn in CL.checked()){
54b       if(mn.symmetricTo(motifnode)){
```

```
54c          sets.add(symmetrySets.get(mn);
54d        }
54e }
```

As mentioned before, the above adaptations narrow down the search tree of the ISMA algorithm by making use of the characteristics of the reflection and the cyclic rotation symmetries. Besides this, for all other types of symmetry, every time a new instance has been found, it will be checked whether it is symmetric to a previously found instance. If this is the case, the instance will not be exported (line 13). In order to check this, all motifs that are symmetric to this motif (except for reflections and cyclic rotations) need to be identified. For a motif with k nodes, all symmetric motifs can be found by enumerating all permutations of the k motif nodes, connecting them according to the motif specification, and comparing this newly formed motif to the original one. From this list of symmetric motifs, the reflection and cyclic rotation symmetries are eliminated, as they are already accounted for in the algorithm. By mapping the motif nodes of an instance to all symmetric motifs, it can easily be checked whether the instance is symmetric to a previous one.

In order to enumerate all permutations, a 17th century algorithm, called 'plain changes' by the English bell ringers, was used. In computer science, it is known as the Steinhaus-Johnson-Trotter algorithm [28–30], and it has been improved by Even [31]. It is a powerful algorithm that generates an ordering of all permutations and is able to find all $n!$ permutations of n elements by swapping two adjacent elements $n! - 1$ times. Due to the small differences between two consecutive permutations, this algorithm can be implemented in a constant time per permutation.

B.3 Results and Discussion

This section starts with a description of the software that incorporates the index-based subgraph matching algorithm. Subsequently, a number of results are presented that indicate the strength of the algorithm in comparison with other subgraph matching algorithms.

B.3.1 Software

A software implementation of the iterative ISMA algorithm is freely available at https://sourceforge.net/projects/isma/. It is presented in the form of a Java .jar -file (ISMA.jar) and can be used from a command line interface as follows:

```
   java -jar "<directory>/ISMA.jar"
-folder "<folder of input files>"
```

```
-linkfiles "<list of <typename u/d filename>
separated by spaces>"
-motif "<motif>"
-output "<reference to output file>"
```

The first two words indicate we want to execute a .jar-file. Subsequently, it is indicated where the .jar-file in question is situated. The program takes four arguments: folder, linkfiles, motif and output. The folder argument contains the directory where all input files are situated. It avoids retyping it for every inputfile. The linkfiles are the files that compose the network. Each linkfile contains all links for one specific type. It is denoted in the command by three arguments: the name of the link type (mostly an upper case character), a character indicating whether the links are directed (`d`) or undirected (`u`) and the name of the file. The different linkfiles are separated by spaces. The next argument is the motif. This is the motif specification as defined previously in this article. The last argument determines where the output should be stored.

The input files contain all links of one type. These links are represented by the name of the start node and the name of the end node separated by a tab. Every line contains one link. Example input files can be found online. The output file has one line for every motif that has been exported. A motif is represented as follows: `Motif [<motifspecification>]: [<node 1>, <node 2>, ...]`.

As the above notation is quite complicated, we will explain it in more detail by means of an example.

```
   java -jar "ISMA/ISMA.jar"
-folder "ISMA/input/"
-linkfiles "A d linksAtype.txt B d linksBtype.txt
C u linksCtype.txt"
-motif "ABC"
-output "ISMA/output/results.txt"
```

In this example `ISMA.jar` can be found in `ISMA/`. All input files are present in the folder `ISMA/input/`. The network contains links of three types: directed links of type A, directed links of type B and undirected links of type C. All these links can be found in the files `linksAtype.txt`, `linksBtype.txt` and `linksCtype.txt` respectively. The motif that is searched for is `ABC`, which is the motif we used in the example (see figure B.3). The result of the ISMA algorithm (i.e. a list of all motifs found) is written to the file `results.txt` in the directory `ISMA/output/`.

In the 'Files' tab of this SourceForge project, all input (network) files that

PGS network	
# nodes	1 255
# S links	667
# G links	8 102
# P links	3 688

XYZ network	
# nodes	7 810
# X links	36 391
# Y links	40 630
# Z links	3 390

Table B.2: Network configurations of biological networks.

were used in the experiments are available. For a complete description of these networks, we would like to refer to the following section.

B.3.2 Results

To demonstrate the strength of the ISMA algorithm, we compared it to the naive recursive subgraph matching algorithm (RSMA) as well as the algorithm of Ullmann [9], the VF algorithm [10, 11] and the VF2 algorithm [12, 13], which are state-of-the-art subgraph matching algorithms. We used two networks with multiple edge types as test networks. The first is an integrated network of physical (P, undirected), genetic (G, undirected) and signaling (S, directed) interactions between kinases and phosphatases in yeast [32, 33], also used in [26] (see left panel of Table B.2 for basic network characteristics). The second consists of protein-protein interactions in yeast (X, undirected, obtained from the BioGRID [34] database), protein-protein interactions in human (Y, undirected, obtained from the BioGRID [34] and STRING [35] databases), and orthology relations between human and yeast proteins (Z, bipartite, from the InParanoid database [36]) (see right panel of Table B.2 for basic network characteristics). Experiments were carried out on a 64-bit machine with a processor of the type Intel(R) Core(TM) 2 Duo CPU P8600, 2.40 GHz and 4 GB of RAM. Both the Recursive Subgraph Matching algorithm (RSMA) and the Index-based Subgraph Matching Algorithm (ISMA) were implemented in Java (version 1.6.0_18). The Ullmann, VF and VF2 algorithms are all present in the VFlibrary, a (sub)graph matching library implemented in C. These algorithms are known for finding subgraph isomorphisms in Attributed Relational Graphs (ARGs) fast, which is exactly what we are dealing with here.

We first searched for a number of three-node motifs in the PGS-network, that are useful in the clustering algorithm of [26], and monitored the execution time for each of the algorithms. In table B.3 an overview is given of these execution times (in milliseconds). Here, it can be seen that there is a major difference between the algorithms of the VFlibrary and the (naive) RSMA and (iterative) ISMA algorithm. Despite the fact that they are implemented in C, which is a language with less execution overhead, the VFlibrary algorithms are remarkably slower. One of the decisive reasons for this discrepancy is the way how the network is stored in

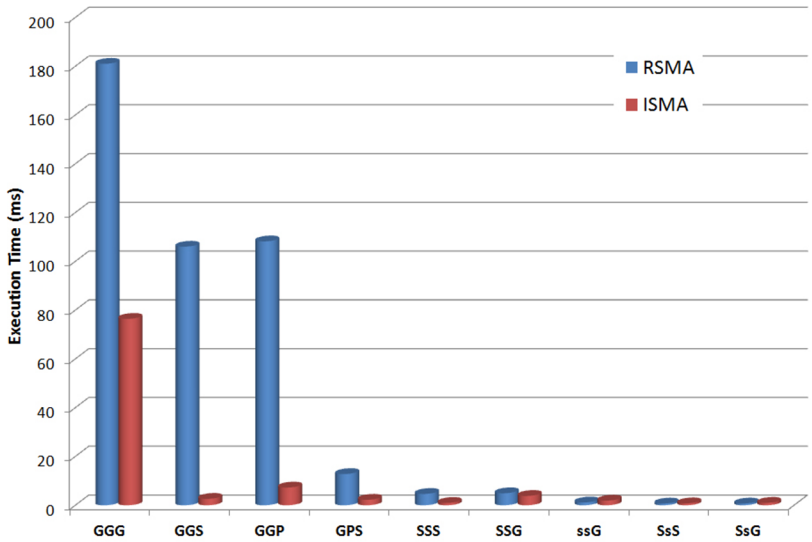

Figure B.10: ***Execution times.*** *Comparison of the execution times (in ms) of the RSMA and the ISMA algorithm for finding 3-node motifs in the PGS network.*

memory. For the ISMA and RSMA algorithm, the network structure has been optimized for fast retrieval of all start nodes of a certain link type (see Data Structures section). Looking at the RSMA and the ISMA algorithm separately (figure B.10), we see that the ISMA algorithm indeed has lower execution times than the RSMA algorithm.

Subsequently, we looked for a number of larger motifs in this network. Table B.3 shows the execution times of the algorithms when searching for all 8-cliques and 10-cliques (i.e. complete graphs of 8 and 10 nodes respectively). While for the ISMA algorithm, the execution times are around 1 second, they run up to more than two hours for the other algorithms. Here, the RSMA algorithm even performs worse that the algorithms of the VFlibrary. It should be noted that the difference between the ISMA algorithm and the other algorithms is most extreme for 'complete' motifs (i.e. cliques). However, for sparser motifs still significant speedups are realized, as will be shown in the experiments with the XYZ-network.

Moreover, we looked for a motif that is not present in the PGS-network, viz the 4-node motif PGSPGS. Results are shown in table B.3. While the execution times are relatively low, the difference between the algorithms of the VFlibrary and the RSMA and ISMA algorithms is not as remarkable as in the previous cases.

Next we searched in the XYZ-network for so-called *interologs*, 4-node subgraphs (with motif specification XZ00ZY) consisting of conserved protein-protein interactions between orthologous protein pairs in yeast and human [37]. General-

motif	# motifs	Ullmann	VF	VF2	RSMA	ISMA
GGG	9008	3653.5	4825.1	3421.1	181.0	76.4
SSS	81	851.0	145.6	80.4	4.6	0.9
GPS	47	1142.0	1061.9	769.4	12.8	2.3
SsS	3	815.2	140.3	76.7	0.8	0.8
SSG	103	861.4	155.7	87.0	4.9	3.8
SsG	31	838.4	145.5	83.0	1.0	1.0
GGS	312	1659.8	1167.6	866.7	106.0	2.6
GGP	418	1680.4	1201.5	898.7	108.2	7.3
ssG	112	868.2	163.6	94.5	1.9	1.9
8-clique	226	1 465 250	4 826 735	3 140 065	>2h	954
10-clique	1	4 759 462	>2h	>2h	>2h	1 009
PGSPGS	0	1020.1	251.3	170.8	83.9	14.2
XZ00ZY	2558	535 657.0	111 610.0	42 514.0	153.7	93.9
XXXZ000Z0Y00ZYY	4745	1 828 683.0	825 828.0	182 734.0	2354.6	270.1
AAAZ000Z0B00ZBB	840	726 732.0	166 608.0	34 053	595.2	123.2

Table B.3: The execution times (in milliseconds) for the different subgraph matching algorithms. It should be noted that the experiments were interrupted after 2 hours.

izing this concept, we also searched for conserved triangles (6 nodes, motif specification XXXZ000Z0Y00ZYY). Despite of the fact that protein-protein interactions are undirected, in the experiments we also assumed the links to be directed. The directions were determined by the input files as the first and the second proteins were considered tails and heads respectively. In this directed network, we looked for the 6-node motif AAAZ000Z0B00ZBB. Here as well, we observe dramatic reduction in execution times for ISMA (and to a lesser extent RSMA) compared to the VF library algorithms, as these tend to be quite slow for larger motifs.

In conclusion, by an intelligent design of the network data structure a remarkable speedup is realized for the RSMA and ISMA algorithm in comparison to the VFlibrary algorithms. Moreover, this speedup is increased even more by carefully selecting the order in which the motif nodes are investigated (i.e. ISMA vs. RSMA).

To quantify the relative speedup realized by the ISMA algorithm, we defined the calculation time multiplicator (CTM) as

$$CTM_{ISMA}(algorithm) = \frac{execution\ time(algorithm)}{execution\ time(ISMA)}$$

For clarity reasons, in the remainder of this article we will use $CTM(algorithm)$ in stead of
$CTM_{ISMA}(algorithm)$.

These calculation time multiplicators are given in table B.4. It shows that the highest speedup factors are achieved for the motifs with an average number of occurrences. These figures show that the larger the motif, the larger the speedup that can be realized (XZ00ZY vs. XXXZ000Z0Y00ZYY) and that these speedups are higher when there are more occurrences in the network (XXXZ000Z0Y00ZYY vs. AAAZ000Z0B00ZBB). In conclusion, 2 factors contribute in reducing the calculation type: the network data structure that has been optimized for fast retrieval of all links of a certain type and the order in which the motif nodes are investigated in order to reduce the search tree. While the former one explains the difference between the VFlibrary algorithms and the RSMA and ISMA algorithm, the latter one causes the execution time of the ISMA algorithm to be smaller than that of the RSMA algorithm.

As explained in the Methods section, the ISMA algorithm achieves its speedup compared to the RSMA algorithm by reducing the size of the search tree. We counted the number of nodes in the search tree for both algorithms when searching for 3-node motifs in the PGS-network (see figure B.11). On average, the search trees of the ISMA algorithm are around 100 times smaller than the corresponding search trees of the RSMA algorithm. It should be noted that the size of the search tree (and the execution time) is dependent on both the motif configuration and the network in which these motifs are searched for.

motif	**# motifs**	$CTM(Ullmann)$	$CTM(VF)$	$CTM(VF2)$	$CTM(RSMA)$
GGG	9008	47.8	63.1	44.8	2.4
SSS	81	873.7	149.5	82.5	4.8
GPS	47	505.5	470.1	340.6	5.7
SsS	3	1000.2	172.1	94.1	1.0
SSG	103	225.0	40.7	22.7	1.3
SsG	31	831.7	144.3	82.3	1.0
GGS	312	630.6	443.6	329.3	40.3
GGP	418	231.2	165.3	123.6	14.9
ssG	112	457.4	86.2	49.8	1.0
8-clique	226	1 535.9	5 059.5	3 291.5	$>$7 547.2
10-clique	1	4 717.0	$>$ 7135.8	$>$7 135.8	$>$7 135.8
PGSPGS	0	72.1	17.8	12.1	5.9
XZ00ZY	2558	5704.5	1188.6	452.8	1.7
XXXZ000Z0Y00ZYY	4745	6771.3	3057.9	676.6	8.7
AAAZ000Z0B00ZBB	840	5900.3	1352.7	276.5	4.8

Table B.4: The calculation time multiplicators (CTM) of the ISMA algorithm compared to other algorithms for a number of motif configurations.

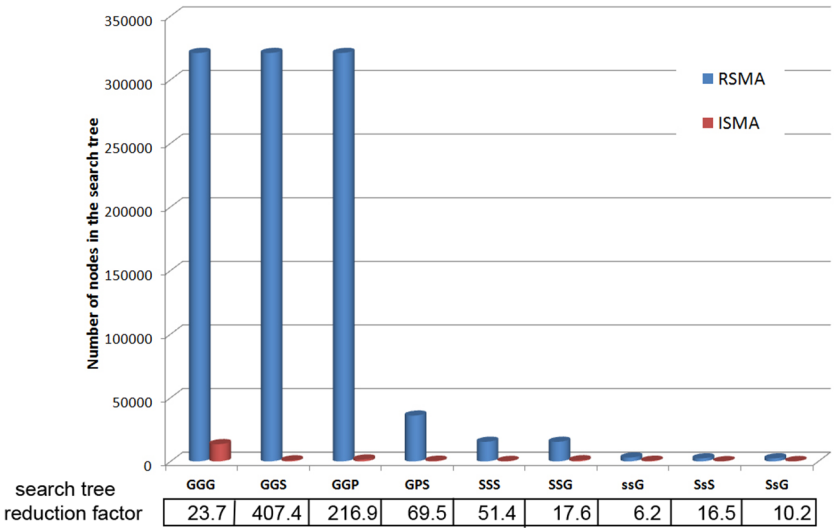

Figure B.11: ***Size search tree.*** *Comparison of the number of nodes in the search tree of the RSMA and the ISMA algorithm for finding 3-node motifs in the PGS network. The search tree reduction factor is defined as the size of the search tree of RSMA divided by the size of the search tree of ISMA.*

network	# nodes	# links	# motifs
Wiki-Vote	7115	103689	608389
p2p-Gnutella08	6301	20777	2383
p2p-Gnutella30	36682	88328	1590
CA-CondMat	23133	186936	173361
CA-HepTh	9877	51971	28339

Table B.5: Network configurations of the non-biological networks, together with the number of 3-node cliques present in these networks.

Although the development of the ISMA algorithm was motivated by the problem of identifying composite motifs in biological networks with multiple interaction types, or more generally Attributed Relational Graphs, it can of course be applied equally well to non-biological networks. We illustrate this by searching for all 3-node cliques in a number of networks from the SNAP database[1], where we treated all networks as undirected. Assuming all link are of the type X, this means the motif XXX is searched. Table B.5 shows the configurations of the networks that were used in the experiments, together with the number of 3-node cliques that were found in these networks. In table B.6 the CTM's are given of the ISMA

[1] http://snap.stanford.edu/data/index.html

network	CTM(Ullmann)	CTM(VF)	CTM(VF2)	CTM(RSMA)	STRF
Wiki-Vote	126.8	211.9	142.8	11.1	41.2
p2p-Gnutella08	131.9	27.3	12.9	1.6	26.5
p2p-Gnutella30	706.4	35.7	6.4	1.1	17.7
CA-CondMat	870.1	565.2	241.3	2.3	15.0
CA-HepTh	293.4	195.0	83.5	1.5	11.1

Table B.6: The calculation time multiplicators (CTM) of the ISMA algorithm compared to other algorithms for a number of non-biological networks. The last column shows the search tree reduction factors (STRF), i.e. the ratio of the number of nodes in the search tree of the RSMA algorithm to the search tree of the ISMA algorithm.

algorithm over the algorithms of the VFlibrary and over the recursive algorithm. Moreover, the search tree reduction factors (i.e. the number of nodes in the search tree of RSMA divided by the number of nodes in the search tree of ISMA) are depicted. This table shows that, similar to the experiments on biological networks, the execution times of the algorithm presented in this article are much lower that those of the algorithms of the VFlibrary. Again, this can be explained by the

network data structure that allows fast retrieval of all links of a certain type. If we take into account the large numbers of (XXX) motifs that are present in these networks, the CTM's are relatively small in comparison to those in biological networks. This is also confirmed in the search tree reduction factors. Here the search tree reduction factors are on average around 20, which is small in comparison with the average search tree reduction factor of around 100 for the biological networks. The reason for this is that here, when determining the set of network nodes that can be mapped on a motif node, all neighbors of the mapped network nodes need to be taken into account in stead of only the neighbors according to one specific link type.

B.4 Conclusion

Motivated by problems in the analysis of biological networks composed of multiple directed and undirected interaction types, we have developed a novel exact subgraph matching algorithm that is optimized for graphs with specific link characteristics. By carefully selecting the order in which the nodes of a network motif are investigated and by designing appropriate data structures, a remarkable speedup can be realized. In each iteration of the algorithm, sets of network nodes are determined that can be mapped on the remaining motif nodes. Always selecting the motif node with the smallest corresponding set of network nodes leads to less branches closer to the root of the search tree and consequently a reduced search tree.

In order to realize an iterative version of this algorithm, a number of data structures were developed: a checklist that keeps track of the order in which the motif nodes are investigated, a motif iterator to iterate over all the network nodes that can be mapped on a motif node, and a priorityqueuemap in order to select the best motif node to be investigated next.

Incorporating motif symmetries can lead to further increases in computational efficiency. When present, ISMA explicitly takes into account two specific symmetries, namely the reflections and cyclic rotations, to further speed up the algorithm. For all other motif symmetries, duplicate instances are eliminated once they have been created. In future versions of ISMA, we plan to take into account additional types of symmetries to prune the search tree.

Applications on real network data from the biological as well as non-biological domain, showed that the ISMA algorithm indeed leads to dramatic speedups compared to existing exact subgraph matching algorithms for attributed relational graphs. A comparison with a naive recursive tree-based subgraph matching algorithm shows that to a large extent, this speedup is indeed due to tree-pruning strategy implemented in ISMA, with search trees in ISMA being on average 100 times smaller than those of the recursive algorithm in our experiments on biological networks

and on average 20 times smaller in our experiments on non-biological networks.

Taken together, we believe ISMA is an interesting new exact subgraph matching algorithm which will be important for the discovery and analysis of small and large network motifs in ever growing biological networks, with potential applications in other domains as well.

References

[1] B. Jasny, L. Zahn, and E. Marshall. *Connections*. Science, 325(5939):405, 2009.

[2] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. *Network motifs: simple building blocks of complex networks*. Science, 298:824–827, 2002.

[3] U. Alon. *Network motifs: theory and experimental approaches*. Nature Reviews Genetics, 8:450–461, 2007.

[4] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. *Topological generalizations of network motifs*. Physical Review E, 70:031909, 2004.

[5] R. Dobrin, Q. K. Beg, A.-L. Barabási, and Z. N. Oltvai. *Aggregation of topological motifs in the Escherichia coli transcription regulatory network*. BMC Bioinformatics, 5:10, 2004.

[6] T. Michoel, A. Joshi, B. Nachtergaele, and Y. Van de Peer. *Enrichment and aggregation of topological network motifs are independent organizational principles of integrated interaction networks*. Molecular BioSystems, 7:2769–2778, 2011.

[7] D. Conte, P. Foggia, C. Sansone, and M. Vento. *Thirty Years Of Graph Matching In Pattern Recognition*. International Journal of Pattern Recognition an Artificial Intelligence, 18(3).

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] J. R. Ullmann. *An Algorithm for Subgraph Isomorphism*. Journal of the ACM, 23(1):31–42, 1976.

[10] L. Cordella, P. Foggia, C. Sansone, F. Tortorella, and M. Vento. *Graph matching: a fast algorithm and its evaluation*. Proceedings of the 14th International Conference on Pattern Recognition, 2:1582 –1584, 1998.

[11] L. Cordella, P. Foggia, C. Sansone, and M. Vento. *Performance evaluation of the VF graph matching algorithm*. Proceedings of the International Conference on Image Analysis and Processing, pages 1172 –1177, 1999.

[12] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. *An improved algorithm for matching large graphs*. 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen, pages 149–159, 2001.

[13] L. Cordella, P. Foggia, C. Sansone, and M. Vento. *A (sub)graph isomorphism algorithm for matching large graphs*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(10):1367 –1372, 2004.

[14] B. Messmer and H. Bunke. *A decision tree approach to graph and subgraph isomorphism detection*. Pattern Recognition, 32(12):1979 – 1998, 1999.

[15] M. Weber, M. Liwicki, and A. Dengel. *Faster subgraph isomorphism detection by well-founded total order indexing*. Pattern Recognition Letters, 33(15):2011 – 2019, 2012.

[16] R. Giugno and D. Shasha. *GraphGrep: A fast and universal method for querying graphs*. Proceedings of the 16th International Conference on Pattern Recognition, 2:112 – 115, 2002.

[17] X. Yan, P. S. Yu, and J. Han. *Graph indexing: a frequent structure-based approach*. Proceedings of the 2004 ACM SIGMOD, pages 335–346, 2004.

[18] S. Zhang, S. Li, and J. Yang. *GADDI: distance index based subgraph matching in biological networks*. Proceedings of the 12th International Conference on Extending Database Technology, pages 192–203, 2009.

[19] Y. Tian and J. M. Patel. *TALE: A Tool for Approximate Large Graph Matching*. Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pages 963–972, 2008.

[20] J. Larrosa and G. Valiente. *Constraint satisfaction algorithms for graph pattern matching*. Mathematical. Structures in Computer Science, 12(4):403–422, 2002.

[21] S. Zampelli, Y. Deville, and C. Solnon. *Solving Subgraph Isomorphism Problems with Constraint Programming*. Journal of Constraints, 15:327–353, 2010.

[22] E. Yeger-Lotem, S. Sattath, N. Kashtan, S. Itzkovitz, R. Milo, R. Y. Pinter, U. Alon, and H. Margalit. *Network motifs in integrated cellular networks of transcription-regulation and protein-protein interaction*. Proceedings of the National Academy of Sciences, 101:5934–5939, 2004.

[23] H. Yu, Y. Xia, V. Trifonov, and M. Gerstein. *Design principles of molecular networks revealed by global comparisons and composite motifs*. Genome Biology, 7:R55, 2006.

[24] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. *SAGA: a subgraph matching tool for biological graphs*. Bioinformatics, 23(2):232–239, 2007.

[25] X. Zhu, M. Gerstein, and M. Snyder. *Getting connected: analysis and principles of biological networks.* Genes & development, 21(9):1010–1024, 2007.

[26] P. Audenaert, T. Van Parys, F. Brondel, M. Pickavet, P. Demeester, Y. Van de Peer, and T. Michoel. *CyClus3D: a Cytoscape plugin for clustering network motifs in integrated networks*. Bioinformatics, 27(11):1587–1588, 2011.

[27] J. Grochow and M. Kellis. *Network motif discovery using subgraph enumeration and symmetry-breaking*. Research in Computational Molecular Biology, pages 92–106, 2007.

[28] H. Steinhaus. *One hundred problems in elementary mathematics*. Pergamon Press, Oxford,UK., 1963.

[29] S. M. Johnson. *Generation of Permutations by Adjacent Transposition*. Mathematics of Computation, 17(83):282–285, 1963.

[30] H. F. Trotter. *Algorithm 115: Perm*. Communications of the ACM, 5(8):434–435, 1962.

[31] S. Even. *Algorithmic combinatorics*. Macmillan, 1973.

[32] A. Breitkreutz, H. Choi, J. R. Sharom, L. Boucher, V. Neduva, B. Larsen, Z. Y. Lin, B. J. Breitkreutz, C. Stark, G. Liu, J. Ahn, D. Dewar-Darch, T. Reguly, X. Tang, R. Almeida, Z. S. Qin, T. Pawson, A. C. Gingras, A. I. Nesvizhskii, and M. Tyers. *A global protein kinase and phosphatase interaction network in yeast*. Science, 328:1043–1046, 2010.

[33] D. Fiedler, H. Braberg, M. Mehta, G. Chechik, G. Cagney, P. Mukherjee, A. C. Silva, M. Shales, S. R. Collins, S. van Wageningen, P. Kemmeren, F. C. P. Holstege, J. S. Weissman, M.-C. Keogh, D. Koller, K. M. Shokat, and N. J. Krogan. *Functional organization of the S. cerevisiae phosphorylation network*. Cell, 136:952–963, 2009.

[34] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. *BioGRID: a general repository for interaction datasets*. Nucleic Acids Research, 34:535–539, 2006.

[35] L. Jensen, M. Kuhn, M. Stark, S. Chaffron, C. Creevey, J. Muller, T. Doerks, P. Julien, A. Roth, M. Simonovic, et al. *STRING 8 – a global view on proteins and their functional interactions in 630 organisms*. Nucleic Acids Research, 37(suppl 1):412–416, 2009.

[36] A. C. Berglund, E. Sjolund, G. Ostlund, and E. L. L. Sonnhammer. *InParanoid 6: eukaryotic ortholog clusters with inparalogs*. Nucleic Acids Research, 36:263 – 266, 2008.

[37] H. Yu, N. Luscombe, H. Lu, X. Zhu, Y. Xia, J. Han, N. Bertin, S. Chung, M. Vidal, and M. Gerstein. *Annotation transfer between genomes: protein–protein interologs and protein–DNA regulogs*. Genome Research, 14(6):1107–1118, 2004.