

# **Optimalisatie van client-side intermodale routeplanning**

**Brecht Van de Vyvere**

Promotoren: prof. dr. Erik Mannens, prof. dr. ir. Rik Van de Walle  
Begeleiders: Pieter Colpaert, dr. ir. Ruben Verborgh

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: informatica

Vakgroep Elektronica en Informatiesystemen  
Voorzitter: prof. dr. ir. Rik Van de Walle

Vakgroep Elektronica en Informatiesystemen,Vakgroep  
Industriële Technologie en Constructie  
Voorzitter: prof. dr. ir. Rik Van de Walle

Faculteit Ingenieurswetenschappen en Architectuur  
Academiejaar 2015-2016





# **Optimalisatie van client-side intermodale routeplanning**

**Brecht Van de Vyvere**

Promotoren: prof. dr. Erik Mannens, prof. dr. ir. Rik Van de Walle  
Begeleiders: Pieter Colpaert, dr. ir. Ruben Verborgh

Masterproef ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: informatica

Vakgroep Elektronica en Informatiesystemen  
Voorzitter: prof. dr. ir. Rik Van de Walle

Vakgroep Elektronica en Informatiesystemen,Vakgroep  
Industriële Technologie en Constructie  
Voorzitter: prof. dr. ir. Rik Van de Walle

Faculteit Ingenieurswetenschappen en Architectuur  
Academiejaar 2015-2016



# Voorwoord

*Als vrijwilliger bij iRail, een vereniging die digitale creativiteit rond mobiliteit ondersteunt, ben ik al vele malen in contact gekomen met de verschillende noden van routeplanners. Laatste jaren is de evolutie naar een transparanter open data-beleid van de overheid en publieke voersmaatschappijen gestart. Innovatieve projecten, zoals deze masterproef, tonen aan dat er nog veel onderzoek mogelijk is wanneer data opengesteld worden.*

*Graag wil ik via deze weg enkele personen bedanken.*

*Allereerst bedank ik mijn promotor Rik Van de Walle om mij de kans te geven om mijn masterproef over dit innovatief project te schrijven.*

*Mijn promotor Erik Mannens wil ik bedanken om onderzoek te mogen doen binnen het ‘Data Science Lab’-team. Het was een boeiende ervaring om te mogen meewerken aan een winnend project voor ISWC 2015.*

*Verder wil ik Pieter Colpaert bedanken voor de vele inzichten die ik tijdens deze masterproef verworven heb. Dankzij zijn begeleiding werd dit werk een alomvattend geheel waarbij onderzoek en communicatie centraal stonden. Ik wil hem dan ook bedanken voor het geven van feedback en het aanleren van correcte programmeerstijlen, als ook het nalezen van deze scriptie.*

*Ook wil ik mijn begeleider Ruben Verborgh bedanken om zijn kennis over hypermedia met mij te delen, en het geven van feedback op onderzoeksproblemen.*

*Ten slotte wil ik mijn familie en vrienden bedanken voor de vele steun en motivatie die ik tijdens mijn studies gekregen heb.*

# Toelating tot bruikleen

“De auteur geeft de toelating deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik.

Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.”

Brecht Van de Vyvere, januari 2016

# Optimalisatie van client-side intermodale routeplanning

door

Brecht Van de Vyvere

Scriptie ingediend tot het behalen van de academische graad van  
Master of Science in de industriële wetenschappen: informatica

Promotor: prof.dr.ir. Rik Van de Walle, prof.dr.ir. Erik Mannens  
Scriptiebegeleider: ing. Pieter Colpaert, dr.ir. Ruben Verborgh

Vakgroep Elektronica en Informatiesystemen, Vakgroep Industriële Technologie en Constructie

Voorzitter: Prof. Dr. Ir. Rik Van de Walle

Faculteit Ingenieurswetenschappen en Architectuur

Academiejaar 2015–2016

## Samenvatting

Linked Connections is een framework die client-side routeplanning toelaat door tijdstabellen van het openbaar vervoer te downloaden. Dit introduceert nieuwe interessante trade-offs tussen server en client, maar heeft een trage querytijd tot gevolg. Data wordt momenteel enkel gepubliceerd volgens bepaalde tijdsintervallen waardoor de client onnodig veel data moet downloaden. Wanneer je bijvoorbeeld vertrekt in Brussel, ben je nog niet geïnteresseerd in de gegevens van Rome. Deze masterproef maakt extra filtering mogelijk op basis van de locatie van het vertrekpunt van de gezochte route. Twee client-side algoritmen worden geïmplementeerd die van deze extra filter gebruik maken: de ene gebruikt een heuristiek, de andere accelereert de originele implementatie. Enkel deze laatste speedup-methode kan een optimale oplossing garanderen. Om deze feature te kunnen toevoegen aan de server moet er eenmalig hulpinformatie uit de tijdstabellen berekend worden. Dit onderzoek focust op het oplossen van het *Earliest Arrival Time*-probleem. Dit betekent dat de snelste route tussen twee stopplaatsen gevonden wordt rekening houdend met een bepaalde vertrektijd. Een versnelling van 37% is waargenomen met de speedup-methode om queries op de client op te lossen voor het Belgische spoorwegennetwerk. Een bijkomend voordeel is een datavermindering van 38%. Ondanks het toevoegen van een extra filter blijft de server nog steeds low-cost. Doordat de client sneller en met minder data een route kan berekenen en het voorbewerken beperkt blijft, biedt de voorgestelde speedup-methode een meerwaarde voor Linked Connections.

## Trefwoorden

Linked Connections, optimalisatie, routeplanning, client-side, GTFS

# Optimization of client-side intermodal route planning

Brecht Van de Vyvere

Supervisor(s): Pieter Colpaert, Ruben Verborgh, Erik Mannens, Rik Van de Walle

**Abstract**— **Linked Connections** is a framework to solve public transit route planning queries on the client by downloading data just in time. While this approach introduces new trade-offs between client/server that look promising, query execution time is slow, as much data needs to be downloaded. By introducing an extra control on the server that filters the transit connections on the neighbours of a departure stop, we implement and evaluate two client-side route planning algorithms: one by using a heuristic that tries to find the optimal solution by only using the extra control and another that uses the extra control as speedup for the original **Linked Connections** implementation. Extra filtering is made possible by calculating the minimum amount of time and transfers between each pair of stops in a pre-processing phase. When executing the Earliest Arrival Time problem with an amended **Linked Connections** client, query execution time is improved by 37% with the speedup and 57% with the heuristic. The latter found the optimal route for 78% of the tested queries. The speedup reduces data consumption by 38%, while the heuristic saves 27%. Adding more filter options requires more effort from the server, but reduces the client load tangible. The speedup algorithm guarantees a solution for the Earliest Arrival Time problem and requires few complexity and pre-processing time, which makes it an asset for the **Linked Connections** framework.

**Keywords**—**Linked Connections**, optimization, public transit, route planning, client-side, GTFS

## I. INTRODUCTION

ROUTE planning is at the beginning of an evolution, just like the Web did in the *nineties*. On the one hand, you have commuters that want to know which vehicles take them from point A to B. On the other hand, you have public transit companies that provide services that calculate which route you can take. Since a few years, more and more companies share their time schedules in the uniform formats *General Transit Feed Specification* (GTFS) and *GTFS-RealTime* as Open Data. Data is getting available, but the dots need to be connected. Not only with other transit data from around the world, but also with all the data that the Web has to offer.

Nowadays, route planners are build with an Application Programming Interface (API). This works well for certain applications, but extending it with new features is limited. Another limitation of an API is that the client can't control which information it wants to use and make smart decisions. This can be solved by using one of the core aspects of the Web: hypermedia. This means that clients are able to retrieve information by following links.

Now that certain necessities for route planning are explained, we will first discuss some related work. After that, we'll tell more about *Linked Connections* and how it tries to solve above issues. Next, we introduce two optimization methods that will be evaluated thereafter. Finally, some concluding remarks will be made.

## II. RELATED WORK

In the first two subsections we will discuss briefly what types of route planning algorithms are available. After that, we'll tell more about moving intelligence to the client and how it can be used for route planning.

### A. Dijkstra

Route planning has been solved traditionally with Dijkstra algorithm [1]. This means that public transit networks are modelled as a graph-based network and Dijkstra calculates the shortest path between a source node and every other node. Several optimization techniques have been introduced to achieve query times within milliseconds[2]. These techniques often result in long pre-processing time and/or large space consumption.

### B. CSA

Since 2013, the Connection Scan Algorithm (CSA) [3] makes route planning a data problem, instead of a mathematical one. A connection  $c$  represents a vehicle that departs in a certain departure stop  $c_{depstop}$  and arrives in a certain arrival stop  $c_{arrstop}$  without stopping. Also an indication of the departure time  $c_{deptime}$  and arrival time  $c_{arrrtime}$  is contained within a connection. Connections can be retrieved out of e.g. a *General Transit Feed Specification* (GTFS) feed. Like the name suggests, route planning queries can be solved by scanning a list of connections. The only restriction is that the connections have to be sorted by their departure time  $c_{deptime}$ . While scanning, a Minimum Spanning Tree (MST) is built holding connections that produce the fastest route from the departure stop at a certain departure time to every other stop. *Earliest Arrival Time* (EAT) queries can be solved easily with this MST. Other query types can also be calculated with CSA, but are more complex. For example, *profile* queries return multiple routes within a time interval. Due to restricted time of this research, we will only evaluate EAT queries in section V.

### C. Linked Data Fragments

Linked Data Fragments (LDFs) [5] is a vision that makes it possible to capture a certain part of a dataset. Every fragment type has a *selector* function that selects the data that belongs to it. Also *metadata* and *hypermedia controls* define a fragment. Its purpose is to let clients efficiently execute queries on these datasources. By placing these types of fragments on an axis (fig. 1), different trade-offs between client and server can be discussed. This axis shows the amount of effort that needs to be done by data consumers and data publishers. One of the advantages of LDFs is the possibility to cache the documents. This

enables servers to be low-cost.

#### D. Linked Connections

Linked Connections (LC) [4] is a framework that enables clients to execute route planning queries on data sources. Servers are only responsible for publishing LDFs that contain connections (section II-B), so called *Linked Connections Fragments* (LCFs). A LCF is a type of LDF and contains connections from a subset of stops  $S$  from a dataset. Hypermedia controls give clients the possibility to retrieve more information, e.g. where to find a next fragment or metadata about a stop. You can see in fig. 1 a LDF-axis representing LCFs. (1) represents the original implementation of LC. (2) and (3) are respectively the heuristic and speedup optimizations that will be further discussed in section IV. In next subsection, we explain how LC currently publishes its connections with *basic LCFs* (BLCFs).

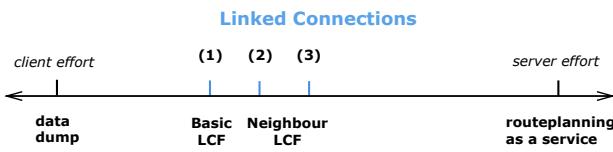


Fig. 1. Linked Data Fragments-axis for public transit route planning. On the left, you can see data dumps, e.g. GTFS files. On the right, you have web services that provide certain functionality to clients. Linked Connections tries to find new ways to plan routes by publishing transit data using different types of fragments.

#### D.1 Basic Linked Connections Fragment

A *basic LCF* (BLCF) contains every connection  $c$  that has a departure time between a certain time interval  $F_{start}$  and  $F_{end}$ . For example, these time intervals can be fixed to every  $X$  minutes or precalculated so every fragment has a similar size of connections. The subset of stops  $S$  of a BLCF contains every stop of the dataset. Figure 2 shows how BLCFs have a *nextPage*-link to the next fragment and that connections are ordered by departure time. A client can now download the fragment as data source and input the connections to CSA. Until a solution is found or certain conditions are reached, e.g. maximum query time, the client fetches next fragments. Route planning with only BLCFs has already been implemented before this research, so we'll call this the *original* method.

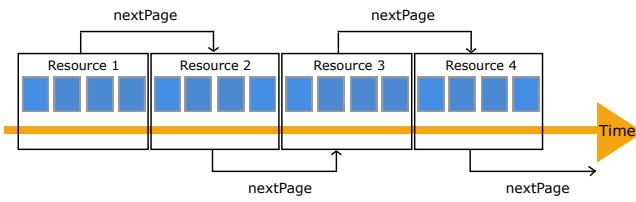


Fig. 2. A *basic LCF* represents all connections between a certain time range.

This way of publishing makes a server very low-cost, because the amount of possible fragments is limited. Each fragment corresponds with one URL. Equation (1) shows the amount of

URLs when all minutes of a day are spread by a fixed time interval  $F$ . If  $F$  is equal to 10 minutes, then there would be 144 URLs. Most public transit companies are not always active (for example, after midnight), so the amount of non-empty fragments is practically lower.

$$24 * 60 / F \quad (1)$$

This method causes the client to scan a lot of *useless* connections. This means that the connection is not added to the Minimum Spanning Tree (MST) of CSA. BLCFs return connections from every stop of the dataset between that time period. E.g.: when you depart in Brussels, you're not interested in connections from Madrid at that moment. To address this issue, we introduce a new type of LDF in section III.

### III. NEIGHBOUR LINKED CONNECTIONS FRAGMENT

In a *neighbour LCF*, connections are returned depending relative to the  $query_{depstop}$  and  $query_{deptime}$ .  $S$  represents the  $query_{depstop}$  and its neighbouring stops within a certain maximum amount of possible transfers  $K_{max}$ . A *transfer* occurs when there is a change of vehicle. Further, every stop  $s$  of  $S$  has its own time interval of connections that belong to the fragment. By also providing information about the departure stop, a connection  $c$  can be dropped if  $c_{deptime} < R(query_{depstop}, c_{depstop})$ . For the previous example: when departing from Brussels, connections that depart in Spain on the same moment won't be returned. The use of a NLCF allows better filtering, especially in the beginning. Less connections need to be scanned so CSA can advance faster in time.

Public transit is not symmetrically: going from stop A to B can be different in the opposite direction. Following two features need to be pre-processed for every pair stops ( $A, B$ ):

- The minimum amount of transfers  $K(A, B)$ .
- The minimum amount of time  $R(A, B)$ . This has not the same meaning of the fastest route, but is equal to the sum of the edges of the shortest path from A to B in a graph (section III-A).

#### A. Pre-processing

Pre-processing of above features (section III) requires two additional steps.

1. Calculate  $R(A, B)$  for every pair direct reachable stops  $A$  and  $B$ . A stop is direct reachable if it can be reached with one trip ( $K(A, B) = 0$ ). This information can be fetched by looping through the *stop\_times.txt* file of a GTFS feed or while generating connections.
2. Calculate  $R(A, B)$  and  $K(A, B)$  for every stop  $A$  in the dataset to every indirect ( $K(A, B) > 0$ ) reachable stop  $B$ .

Table I shows the amount of pre-processing time to calculate  $R(A, B)$  and  $K(A, B)$  for every pair of stops ( $A, B$ ) of a dataset.

### IV. OPTIMIZATION

This research has tested two optimizations. First, we introduce a method (section IV-A) that uses a heuristic to determine a next NLCF. The second optimization (section IV-B) uses the benefits of the heuristic method to speedup the original method.

Dataset	Amount of stops	Pre-processing time (min)
NMBS	597	6,35
NS	2532	21,19

TABLE I

TIME TO CALCULATE INDIRECT REACHABLE STOPS FOR EVERY STOP IN A GTFS FEED.

### A. Heuristic

The heuristic method ((2) on fig. 1) solves queries by only using NLCFs. Every stop  $s$  of the NLCF uses the same time range  $T$  (fig. 3).  $T$  and  $K_{max}$  can be changed dynamically on the server. The optimal route is found if every connection  $c$  of that route has following properties:

- $(query_{depstop} + R(query_{depstop}, c_{depstop})) \leq c_{depstop} < c_{depstop} + T$ :  $c$  departs within the time range of stop  $c_{depstop}$
- $c_{depstop} \in S$  with  $S$  determined by a maximum amount of allowed transfers  $K_{max}$

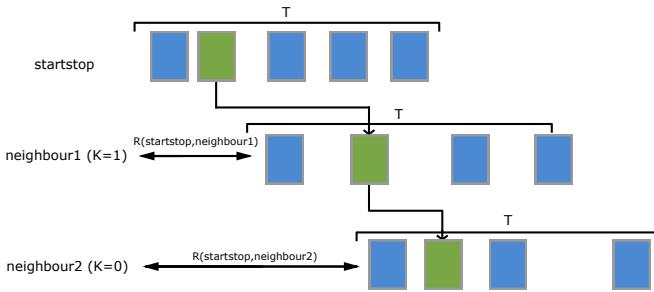


Fig. 3. Overview of connections that are returned with a Neighbour LC Fragment for the heuristic optimization. Note that the amount of stops is limited by a maximum amount of allowed transfers  $K_{max}$ .

If the fastest route isn't found then a next NLCF needs to be fetched by the client. The heuristic method does this by holding a priority queue of reached stops. Every *useful* connection  $c$  is analysed and gets a priority defined by eq. (2):

$$\text{priority}(c) = \alpha * c_v + \beta * c_{arrstop}[\text{importance}] + \gamma * \cos(\theta) \quad (2)$$

Parameters that are used:

- *Velocity*  $c_v$  is distance over time of the connection.
- *Importance* of a stop is indicated by the amount of direct reachable stops.
- *Cosine similarity* shows if  $c$  is going in the right direction.  $\theta$  is the degree between the connection  $c$  vector and the vector between  $query_{depstop}$  and  $query_{arrstop}$ .

Each parameter is divided by the maximum value that occurs in the fragment so its value is normalized.  $\alpha$ ,  $\beta$  and  $\gamma$  are weight factors that can be altered to give certain parameters more impact.

The heuristic method results in a suboptimal solution if a next stop is chosen that doesn't belong to the optimal route and the destination gets reached. It depends of the chosen stop how many minutes longer the route takes. The results (section VI) will show that most queries can be solved with the first NLCF.

The next method (section IV-B) combines the use of this first NLCF together with BLCFs to guarantee the fastest route.

### B. Speedup

The speedup method accelerates the original method by using one NLCF initially. This NLCF is split in multiple pages with hypermedia links holding them together (fig. 4). This way, the size of pages remain compact. The NLCF uses one global time interval  $T$  for all the stops.

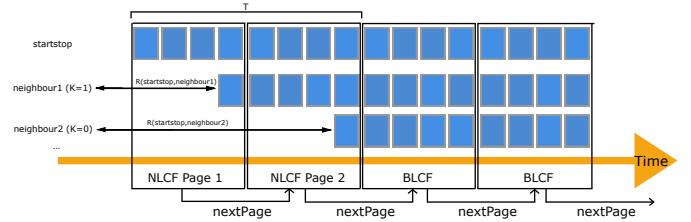


Fig. 4. Overview of the speedup technique. First a NLCF is used to filter connections by the departure time and departure stop of the query. After a certain time period  $T$ , basic LCFs are used.

To guarantee that the fastest route can be found, all stops of the dataset need to be included ( $K_{max}$  represents every stop) in the NLCF. After  $T$ , hypermedia *nextPage*-links refer back to BLCFs. The size of  $T$  depends off the dataset. For the NMBS, the Belgian railway company, most stops can be reached within 120 - 150 minutes. After that time, NLCF pages are similar to BLCFs. This speedup method is the most centered line (3) on the LDF-axis (fig. 1) because it requires three HTTP parameters: the departure time, departure stop and page number. This causes less caching possibilities for the server thus higher server effort.

## V. EVALUATION

*Earliest Arrival Time* queries have been tested. Such queries exist out of three components: the stop where the route starts  $query_{depstop}$ , the destination  $query_{arrstop}$  and a time of departure  $query_{deptime}$ . The result is the fastest route between those two stops starting from the departure time. The amount of transfers is not considered as criteria. A web application has been developed to compare all three methods: the original, heuristic and speedup. 900 queries are generated out of the NMBS GTFS feed, which contains 644 stops and 127116 connections for one day. All tests have been run on a Macbook Pro 2015 with 8 GB RAM and Intel Core i5 2,7 Ghz CPU. There are still software issues, such as stations that are hard to reach, which causes the client to stop searching. The results are representative for local, in Belgium located, stations. The time interval of a basic LCF is limited to every 20 minutes, representing 1100 connections on average. Neighbour LCFs of the heuristic method use a time interval of 100 minutes. The NLCF of the speedup method uses a time interval of 120 minutes split in pages of 30 minutes.

## VI. RESULTS

### A. Speed

Figure 5 shows the time to plan a route according to the distance of the route in bird flight. The original method (in blue)

and speedup method (in red) both grow linear with the distance. The speedup method accelerates the original method with seconds, because less connections need to be scanned. For close distance queries, the speedup is easily a second faster. The difference in time becomes more clear when looking to long distance queries: the speedup creates a gap of a multiple seconds. On average, the speedup method makes querying 37% faster than the original method. The heuristic method in yellow follows a different pattern. Almost all query times are located on a horizontal line of 2 seconds. This means that these queries are solved with the first NLCF. Only a few queries are solved with multiple NLCFs. Note that 22% of the results of the heuristic method are not displayed due to a suboptimal solution. The optimal routes are 57% faster than the original method.

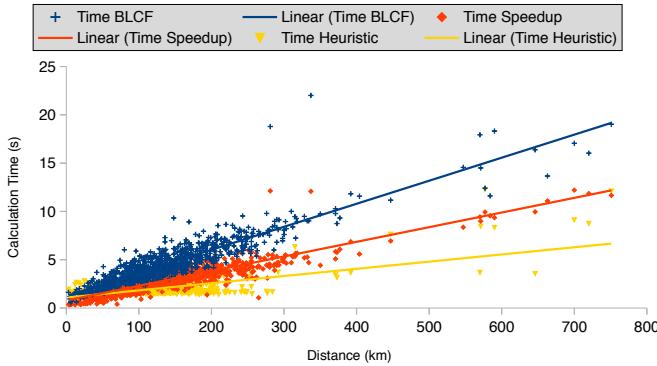


Fig. 5. Time to calculate queries according to the distance of the route.

### B. Data usage

Another important aspect is data usage. The use of NLCFs makes better filtering possible. You can see in fig. 6 how much data consumption has been lowered with the optimization methods. The speedup has the biggest advantage. It uses mostly less data than the heuristic method, because the NLCF is paged: the surplus of downloaded connections is smaller with pages then downloading the whole NLCF. The speedup method reduces the amount of data with 38% on average. Less data usage is also an advantage in a setup where internet connectivity is sparse. The heuristic method uses less data than the original method for distances smaller than 200 kilometer, because connections are better filtered by the server. Some queries between 200 and 300km use less data with the heuristic. These queries benefit from the advancing time frame of the neighbours. When multiple NLCFs are necessary, the data usage of the heuristic method is double or more compared to the other methods.

Table II gives a summary of the results regarding speed, data consumption and useful connnections.

## VII. CONCLUSION

In this paper, we have implemented three methods to solve route planning queries with Linked Connections. As base line, we have tested how fast route planning works with only *basic* Linked Connections Fragments. We introduced two different methods that use an extra filter based on the departure stop and

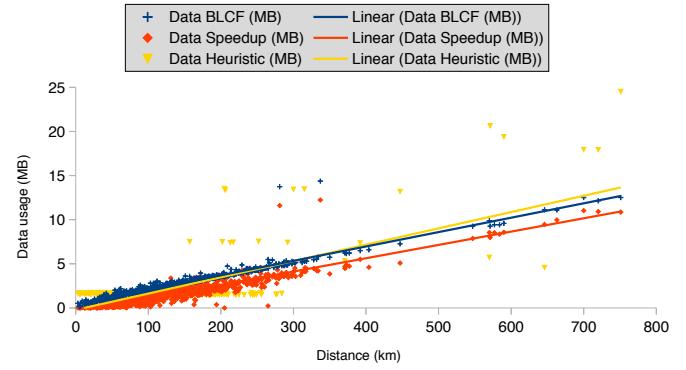


Fig. 6. Overview of data consumption. An extra filter by departure stop enables the server to drop connections that are useless and don't need to be sent.

	Basic	Speedup	Heuristic
Speed (s)	4.54	2.87	1.96
Data usage (MB)	2.69	1.68	1.97
Useful connections (%)	0.026	0.047	0.064
Optimal	yes	yes	no

TABLE II

departure time of the query. Pre-processing time is limited to minutes. The heuristic method is the fastest, but only returns the optimal result for 78% of the tested queries. The speedup method makes querying 37% faster than the original method while guaranteeing the optimal solution. Also bandwidth is saved by 38%. Adding an extra filter to the server helps the client not significantly, but in a tangible way. Due to the low complexity and pre-processing time, the proposed speedup can be a benefit to the existing Linked Connections framework.

## REFERENCES

- [1] Daniel Delling and Thomas Pajor and Renato F. Werneck, *Round-Based Public Transit Routing*, Transportation Science, pages 591-604, 2015
- [2] Hannah Bast and Matthias Hertel and Sabine Storandt, *Scalable Transfer Patterns*, 2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)
- [3] Dibbelt, Julian and Pajor, Thomas and Strasser, Ben and Wagner, Dorothea, *Intriguingly Simple and Fast Transit Routing*, Experimental Algorithms, pages 43-54., Springer, 2013.
- [4] Colpaert, Pieter and Llaves, Alejandro and Verborgh, Ruben and Corcho, Oscar and Mannens, Erik and Van de Walle, Rik, *Intermodal public transit routing using Linked Connections*, Proceedings of the 14th International Semantic Web Conference: Posters and Demos, oct. 2015
- [5] Verborgh, Ruben and Vander Sande, Miel and Colpaert, Pieter and Coppen, Sam and Mannens, Erik and Van de Walle, Rik, *Web-Scale Querying through Linked Data Fragments*, Proceedings of the 7th Workshop on Linked Data on the Web, apr. 2014
- [6] GTFS reference, <https://developers.google.com/transit/gtfs/reference>

# Inhoudsopgave

<b>1 Inleiding</b>	<b>1</b>
1.1 Probleemstelling . . . . .	2
1.2 Onderzoeksraag . . . . .	4
1.3 Hypotheses . . . . .	4
1.4 Oriëntatie . . . . .	5
<b>2 Literatuurstudie</b>	<b>6</b>
2.1 Dienstregeling openbaar vervoer . . . . .	6
2.2 Semantisch web . . . . .	8
2.3 REST . . . . .	16
2.4 Routeplanning-algoritmen . . . . .	18
2.4.1 Dijkstra . . . . .	18
2.4.2 Connection Scan Algorithm . . . . .	19
<b>3 Linked Connections</b>	<b>22</b>
3.1 Principe . . . . .	22
3.1.1 Technologieën . . . . .	23
3.2 Convertor naar connecties . . . . .	24
3.2.1 Connectiescript . . . . .	25
3.2.2 JSONLD-stream . . . . .	28
3.3 Client . . . . .	29
3.3.1 Merger . . . . .	30
3.4 Conclusie . . . . .	32
<b>4 Optimalisatie</b>	<b>33</b>
4.1 Geofiltering . . . . .	33

4.1.1	Geoparameter . . . . .	33
4.1.2	Criteria . . . . .	34
4.2	Minimale tijdsduur . . . . .	35
4.2.1	Direct bereikbare stops . . . . .	36
4.2.2	Buren . . . . .	36
4.3	Neighbour Linked Connections Fragment . . . . .	37
4.4	Heuristiek . . . . .	39
4.4.1	Formule . . . . .	40
4.4.2	Voordelen . . . . .	42
4.4.3	Nadelen . . . . .	42
4.5	Speedup . . . . .	43
4.5.1	Hypermedia . . . . .	44
<b>5</b>	<b>Resultaten</b>	<b>45</b>
5.1	Opstelling . . . . .	45
5.1.1	Lokale server versus productieserver . . . . .	46
5.1.2	Fragmentgrootte . . . . .	46
5.2	Hypotheses . . . . .	48
5.2.1	Efficiëntie . . . . .	51
5.2.2	Dataverbruik . . . . .	52
5.2.3	Serverbelasting . . . . .	52
5.3	Overzicht . . . . .	52
<b>6</b>	<b>Conclusie en future work</b>	<b>54</b>
6.1	Conclusie . . . . .	54
6.2	Future work . . . . .	56
<b>A</b>	<b>Code</b>	<b>58</b>
<b>B</b>	<b>Demo</b>	<b>60</b>

## Hoofdstuk 1

# Inleiding

Routeplanning is momenteel een van de moeilijkste onderzoeksonderwerpen. Dit komt hoofdzakelijk door twee problemen:

- Een eerste probleem is dat de data moet bestaan. Zo heeft het Belgische spoorwegennetwerk pas in 2015 hun tijdstabellen opgesteld in een uniform formaat. Om 100% correcte routeplanning te doen is realtime informatie noodzakelijk. Er zijn maar zeer weinig ov-bedrijven die dit hebben in het juiste formaat en nog minder die dit vrijgegeven.
- Een tweede probleem hierbij is dat deze data geen Open Data is. Sinds augustus 2015 is de Belgische overheid “open-by-default”. Dit houdt in dat alle gegevens van de overheid publiek moeten zijn, tenzij expliciet verklaard wordt waarom deze niet open kan, bijvoorbeeld wegens privacy-schending. Sommige openbaar vervoersbedrijven (ov-bedrijven) zoals De Lijn en de NMBS, geven hun tijdstabellen pas vrij onder 1 op 1 contract waardoor het voor ontwikkelaars moeilijk is om met deze data aan de slag te gaan.

Deze twee problemen zorgen ervoor dat het zeer moeilijk is om een oplossing te vinden die duurzaam met deze verschillende datasets kan omgaan.

Er komt meer bij routeplanning kijken dan van punt A naar B te geraken via de snelste of kortste weg. We leven in een wereld vol verandering. Nieuwe technologieën zoals the Internet Of Things (IOT) zorgen voor nieuwe opportuniteiten. Meer en meer zal data over jou en jouw omgeving een centrale rol spelen. Ook bij routeplanning is personalisatie belangrijk. Dit kan gaan van interessante gebouwen in de buurt tot toegankelijkheid van perrons voor mindervalide mensen.

Open Data is sinds kort in een sterke opmars. Zo is er geschat<sup>1</sup> dat België een nettowinst van 900 miljoen euro ontloopt door bepaalde datasets niet open te stellen. Er komt een bewustzijn dat het duurzaam oplossen van bepaalde problemen met data moet gebeuren.

## 1.1 Probleemstelling

Tot voor kort waren er twee mogelijkheden om een routeplanning applicatie te bouwen:

- ofwel beschikt de client over alle data lokaal. Zo kunnen alle behoeften van de client voldaan worden. Dit is kostelijk voor de client, want alles moet zelf berekend worden. Meestal bevat deze niet over de nodige geheugencapaciteit om routes te berekenen over grote datasets.
- ofwel wordt er een server opgezet die een bepaalde functionaliteiten aanbiedt, dit onder de vorm van een Application Programming Interface (API). Zoals je kan zien in lijst 1.1 kunnen er meerdere parameters meegegeven worden: waar is het startpunt, wanneer wil je vertrekken, waar wil je aankomen...

```
|| http://mijn-api.org?start={...}&bestemming={...}&vertrektijd={?}&
||   transportmodes={...}&extraFeature={...}&...
```

Lijst 1.1: Klassieke webservice interface

Er zijn tientallen mogelijke modes zoals de bus, boot of trein. Een andere uitdaging van routeplanning is het overstappen tussen twee perrons. Overstappen is voor een bejaarde niet hetzelfde als voor een marathon-loper. Kortom, routeplanning moet met meerdere factoren rekening kunnen houden. Om dit allemaal te berekenen wordt ervoor geopteerd om de server deze berekeningen te laten doen. Enkele voordelen hiervan:

- Clients met weinig rekenkracht kunnen snel routeplanningsadvies bekomen.
- Er is weinig bandbreedte nodig: 1 HTTP request is voldoende.

Er zijn ook nadelen hieraan verbonden:

- Personalisatie is zeer moeilijk.

---

<sup>1</sup><http://www.decroo.belgium.be/nl/groen-licht-voor-federale-open-data-strategie-overheidsdata-voortaan-vrij-beschikbaar>

- Een service uitbreiden is moeizaam door de vele factoren die mee rekening gehouden worden.

In figuur 1.2 zie je deze mogelijkheden weergegeven op een Linked Data Fragments-as. Dit is een conceptueel framework om de balans tussen client en server weer te geven. Later (zie sectie 2.2) zal dit beter uitgelegd worden.



Figuur 1.1: Linked Data Fragments-as: huidige oplossingen om routes te plannen. client beschikken ofwel over alle data in een dump, ofwel over een service die het routeplannen voor zich neemt.

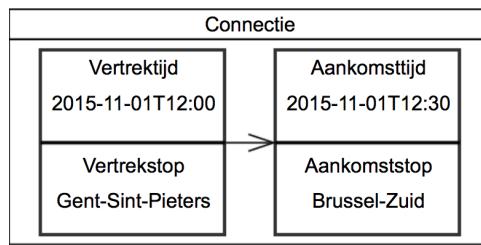
Linked Connections is een framework die berekeningen voor routeplanning op de client toelaat door databronnen net op tijd te downloaden. Deze manier van werken introduceert nieuwe trade-offs die onderzocht kunnen worden. Figuur 1.2 geeft dit weer op een LDF-as.



Figuur 1.2: Linked Connections bieden nieuwe trade-offs tussen client en server aan.

De basiseenheid is een connectie (zie figuur 1.3). Een connectie is de verbinding tussen een vertrek- en eindstop zonder onderbreking, met respectievelijk een vertrek- en aankomsttijd. Een route bestaat uit een combinatie van deze connecties. Connecties zijn gelinkt als ze links bevatten naar andere informatie, zoals connecties die hierop volgen of interessante koffiebars in de buurt.

De huidige Linked Connections implementatie publiceert connecties op basis van een tijdsfilter. Dit zorgt ervoor dat de client onnodig veel data moeten downloaden. Als je bijvoorbeeld in Brussel vertrekt, hebben connecties die vertrekken uit Rome op hetzelfde moment geen nut voor de client. Deze masterproef zal onderzoeken hoe dit opgelost kan worden door nieuwe trade-offs



Figuur 1.3: Voorbeeld van een connectie. Deze bestaat uit een vertrek- en aankomstplaats, resp. met vertrek- en aankomsttijd

te introduceren.

## 1.2 Onderzoeksvraag

Deze masterproef zal hoofdzakelijk een antwoord bieden op volgende vraag:

- *Hoe kunnen we client-side routeplannen met gelinkte connecties sneller maken?*

Client-side routeplannen kan sowieso sneller gemaakt worden zoals bij huidige routeplanners het geval is. We willen alle mogelijkheden van het web gebruiken om het publiceren van die gelinkte connecties op de server zo kosteloos mogelijk te maken met daarbij enkele filtermogelijkheden om het routeplannen op de client sneller te maken. Verschillende *trade-offs* zullen onderzocht moeten worden.

Enkele subvragen die we hierbij kunnen stellen, zijn:

1. Welke extra filter(s) moeten toegevoegd worden?
2. Welke metadata kan een meerwaarde bieden voor de client?
3. Kunnen we garanderen dat de snelste route gevonden wordt bij extra filtering?

## 1.3 Hypotheses

Volgende hypotheses zijn waargenomen:

1. Hoe meer connecties, hoe meer bandbreedte. Bijgevolg meer tijd.
2. Het toevoegen van een extra filter zal client-side routeplanning minstens dubbel zo snel maken.

Deze hypotheses zullen in hoofdstuk 5 en hoofdstuk 6 besproken worden.

## 1.4 Oriëntatie

In het volgend hoofdstuk komt een uitgebreide literatuurstudie over het semantisch web en technologieën die een belangrijke rol vervullen bij routeplanning. In hoofdstuk 3 worden de basisprincipes van Linked Connections uitgelegd. Hierna worden twee optimalisatietechnieken geïntroduceerd in hoofdstuk 4. Vervolgens wordt de performantie getest van de huidige Linked Connections implementatie en de optimalisaties (hoofdstuk 5). Ten slotte wordt een antwoord geformuleerd op de vraag hoe we client-side routeplanner sneller kunnen maken en welke aspecten toekomstig onderzoek vergen (hoofdstuk 6).

## Hoofdstuk 2

# Literatuurstudie

Dit hoofdstuk geeft informatie over concepten, technologieën, dataformaten etc. die van belang zijn voor deze masterproef. De eerste sectie geeft een overzicht van de structuur van de data die gebruikt wordt. Vervolgens wordt het semantisch web toegelicht om deze data ook voor machines begrijpbaar te maken. Daarna volgt een korte verduidelijking van de principes van REST. Ten slotte worden enkele algoritmen voor routeplanning besproken.

### 2.1 Dienstregeling openbaar vervoer

Interoperabiliteit van datasets is belangrijk om routeplanning over verschillende vervoersmaatschappijen toe te laten. Als je jouw reis wil verderzetten van een bus naar een trein zul je hoogstwaarschijnlijk een ander transportbedrijf gebruiken. De tijdstabellen moeten als het ware dezelfde “taal” spreken om die samen te laten werken.

#### GTFS

In 2005 introduceerde Google een specificatie om uniformiteit van transportdata te verzekeren, genaamd General Transit Feed Specification (GTFS) [5]. GTFS is een set van regels die statische tijdstabellen moeten volgen. Deze specificeert welke bestanden noodzakelijk en optioneel zijn en welke datavelden hierin horen. Deze bestanden zijn opgesteld volgens het Comma Separated Values (CSV) formaat. Niet alle bestanden zijn noodzakelijk voor deze masterproef. Zie [5] voor de volledige documentatie. Hieronder volgt een uitleg van de meestgebruikte termen en bestanden.

## Terminologie

Een *stop* is een plaats waar een voertuig stopt, dit kan gaan om een perron, bushalte etc. Elk ov-bedrijf bestaat uit een aantal *routes*. Dit zijn vastgelegde trajecten, bijvoorbeeld het traject tussen Gent Flanders Expo en Wondelgem waartussen tram 1 van De Lijn rijdt. *Trips* zijn de effectieve trajecten die afgelegd worden door een voertuig. Zo zijn er meestal meerdere trips die eenzelfde route afleggen. Tram 1 legt meerdere keren per dag eenzelfde route af. Er zijn ook meerdere trips voor een route, omdat niet elke trip op dezelfde plaatsen stopt. Het kan gerust gebeuren dat er een stopplaats wordt overgeslagen. Deze trips worden gegroepeerd per dag en worden voorgesteld door een *service*. Wil je weten welke routes op een bepaalde dag X rijden, dan moet je ophalen welke services die dag rijden. Dan kun je terugvinden welke routes deze representeren. Een andere belangrijke term is *stoptime*. Dit houdt in wanneer een voertuig tijdens een trip op een bepaalde stopplaats aankomt en terug vertrekt. Een *overstap* (Engels: *transfer*) neemt plaats wanneer van voertuig veranderd moet worden. Hierbij moet er rekening gehouden worden met een bepaalde *minimale overstaptijd*. Een overstap gebeurt wanneer er veranderd wordt van trip.

Zoals je kan opmerken is routeplanning een verwarring woord in de context van GTFS. Dit kan gezien worden als het combineren van verschillende trips.

## Gebruikte bestanden

Volgens GTFS zijn er zes bestanden verplicht en zeven bestanden optioneel. Deze masterproef doelt enkel op het verbeteren van de basisfunctionaliteit van routeplanning waardoor data over tarieven, spoorvormen etc. niet nodig zijn.

Van elke GTFS *feed* worden er vijf bestanden gebruikt:

- trips.txt. Dit bestand zorgt voor de mapping tussen trip, een route en service.
- routes.txt. Bevat een overzicht van alle routes.
- calendar\_dates.txt. Dit bestand bevat voor elke dag welke services er rijden. Normaal wordt een calendar.txt bestand gebruikt om services te mappen op de dagen dat ze rijden en wordt calendar\_dates.txt enkel gebruikt voor uitzonderingsdagen. In de realiteit gebruiken de meeste ov-bedrijven enkel dit bestand om de dagen op te lijsten. Daarom werd er voor gekozen om enkel hierop toe te spitsen.

- stoptimes.txt Dit bestand bestaat uit een verzameling stopplaatsen met telkens de aankomst- en vertrektijden bij.
- stops.txt bevat een lijst met informatie over alle stopplaatsen. Dit speelt een belangrijke rol om interoperabiliteit tussen verschillende datasets te voorzien. Later hierover meer (sectie 6.2).

Deze bestanden bevatten enkel gegevens voor statische tijdstabellen. Om realtime informatie mogelijk te maken is een extra laag bovenop GTFS gemaakt, genaamd GTFS-RT.

### **GTFS-RT**

General Transit Feed Specification RealTime (GTFS-RT) bevat actuele informatie over bepaalde trips, routes, stops... van de corresponderende GTFS feed. Dit kan gaan van vertragingen en onvoorzien omstandigheden tot de exacte huidige positie van een voertuig. Deze data wordt met Protocol buffers, een binair formaat, geserialiseerd om zo compact mogelijk te zijn. Uiteindelijk worden de GTFS-RT bestanden ter beschikking gesteld via een webserver.

Nu we hebben gezien hoe transportdata wordt vrijgegeven, kunnen we afvragen hoe verschillende datasets gecombineerd kunnen worden. Eén van de methodes om dit op te lossen is er voor te zorgen dat machines begrijpen waarover deze data gaat. GTFS data kan gebruikt worden in programma's, maar enkel een mens begrijpt wat er bedoeld wordt met bepaalde headers. In volgende sectie bekijken we welke mogelijkheden het semantische web hiervoor biedt.

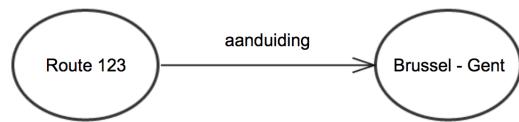
## **2.2 Semantisch web**

Het semantisch web is een verzameling technologiën (URI, RDF, SPARQL, ontologien...) die het mogelijk maakt om informatie op het web machine-leesbaar te maken. Concepten, termen en relaties binnen een bepaald domein worden met elkaar gelinkt waardoor het mogelijk is om informatie te achterhalen dat aanvankelijk niet aanwezig was. Sommige technologiën zoals SPARQL worden niet gebruikt voor de uiteindelijke implementatie van deze masterproef, maar als middel om analogie te vinden met de huidige problemen van routeplannen.

### **RDF**

Resource Description Framework (RDF) is een conceptueel model om bronnen weer te geven. Een feit wordt als atomaire eenheid beschouwd om kennis weer te geven. Zo is het mogelijk om

nieuwe feiten te destilleren als verschillende feiten over dezelfde zaken gaan. Dit is een andere manier om objecten weer te geven dan de typische objectgeoriënteerde omgeving van klassen en attributen. Een feit<sup>1</sup> bestaat uit drie elementen: subject, predikaat en object. Dit kan je als een zin lezen met een onderwerp, werkwoord en lijdend voorwerp, bijvoorbeeld “route 123 heeft als aanduiding ‘Brussel - Gent’ ”. Hierbij is ‘route 123’ het onderwerp, ‘aanduiding’ de relatie en ‘Brussel - Gent’ de waarde van het onderwerp. Het object kan een vaste waarde of een ander object zijn.



Figuur 2.1: RDF representatie van een triple

Een RDF-model is dus een gerichte graaf waarbij de knopen en verbindingen benoemd zijn. Er ontstaat als het ware een web van verschillende concepten die met elkaar verweven worden. Data die op zo’n manier opgebouwd is, wordt *Linked Data* genoemd. Om in de praktijk te kunnen refereren naar bepaalde bronnen, zal er identificatie van concepten nodig zijn.

### Bronidentificatie

Met RDF hebben we een model om bronnen met elkaar te linken en zo feiten te creëren. Om geen verwarring tussen verschillende bronnen te hebben, wordt er gebruik gemaakt van HTTP Universal Resource Identifiers (URI’s) op het web. Dit heeft dezelfde structuur als een Universal Resource Locator (URL) die in de adresbalk van een browser ingegeven wordt. URI’s worden gebruikt om te identificeren, terwijl URL’s gebruikt worden om documenten te localiseren. Het kan dus zijn dat een URI een URL is als deze naast identificeren ook gebruikt wordt om meer informatie over te vinden. Het opzoeken van meer informatie over een bepaald onderwerp noemt men *derefereren*. Neem nu een boek met als ISBN nummer 123 uit een bibliotheek, deze kan geïdentificeerd worden via <https://bibliotheek.org/boeken/123>. Als andere bronnen, zoals de auteur, informatie bevatten over dit boek, dan kan er zonder verwarring verwezen worden hiernaar.

Net zoals bij objectgeoriënteerd programmeren worden er klassen met bepaalde attributen gemaakt om concepten uit de echte wereld zo reëel mogelijk weer te geven. In de wereld van het

---

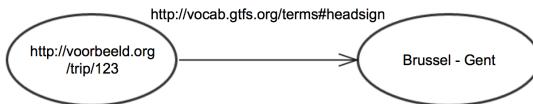
<sup>1</sup>Deze drieledige structuur wordt ook *tuple* of *tuple* genoemd.

semantisch web wordt hiervoor gebruik gemaakt van vocabularia.

### Vocabulary

Een vocabulary, ook wel ontologie genoemd, is een verzameling klassen en eigenschappen die binnen een bepaalde context samenhangen. Zo is RDF zelf ook een vocabulary waarbij bronnen ofwel een subject, predikaat of object voorstellen. Op dit niveau horen alle bronnen simpelweg tot dezelfde context van triples. Om zaken uit de echte wereld samen te steken, zijn er twee vocabularia die daarbij kunnen helpen: Resource Description Framework Schema (RDFS) en Web Ontology Language (OWL). Met deze vocabularia worden er extra elementen geïntroduceerd waarmee er gespecificeerd kan worden of een bron een klasse, eigenschap, waarde of datatype voorstelt.

Zo werd er voor GTFS data ook een vocabulary<sup>2</sup> opgesteld. Als we dit toepassen op het voorbeeld van daarnet, zien we dat een triple wordt voorgesteld door 2 URI's en een waarde voor het object.



Figuur 2.2: Triple met URI's

Om een toepassing te kunnen bouwen wordt data in een bepaald formaat gegoten. Extensible Markup Language (XML) en JavaScript Object Notation (JSON) zijn de meest bekende om webapplicaties te bouwen. Voor beiden is er een uitbreiding voor Linked Data gemaakt: RDF/XML en JSON-LD. Doordat JSON als standaard dataformaat op het web beschouwd wordt zullen we enkel dieper ingaan op JSON-LD.

### JSON voor Linked Data

Als je de openingsuren van een winkel opzoekt in een zoekmachine, gebeurt het vaak dat je een overzicht met informatie bekomt zonder te moeten verderklikken naar een website. Meestal wordt er gebruik gemaakt van JSON-LinkedData (JSON-LD) om deze informatie aan zoekmachines duidelijk te maken. JSON-LD is dus een serialisatieformaat voor gelinkte data. Het

---

<sup>2</sup>vocab.gtfs.org/terms

grote voordeel van JSON-LD is de compatibiliteit met bestaande tools die met JSON werken. Een JSON-LD document kan bijvoorbeeld als apart script bestand toegevoegd worden aan een website om deze semantisch te verrijken. Een JSON-LD document bestaat uit twee delen: een context die bepaalt hoe de data geïnterpreteerd moet worden en de data zelf. Lijst 2.1 bevat informatie over het routevoorbeeld (figuur 2.2) in JSON.

```
{
  "trip_id": "trip 123",
  "aanduiding": "Brussel - Gent"
}
```

Lijst 2.1: Voorbeeld van een trip in JSON.

Om deze informatie semantisch te beschrijven, wordt er context toegevoegd. In lijst 2.2 gebruiken we als context de Linked GTFS ontologie. Het type ‘trip’ wordt toegevoegd met het sleutelwoord ‘@type’. Er is ook een *prefix* ‘trip:’ toegevoegd om de leesbaarheid te verhogen. Het is een goede gewoonte om elke bron een eigen identificering geven met behulp van ‘@id’.

```
{
  "@context": "http://vocab.gtfs.org/terms",
  "@type": "Trip",
  "trip": "http://voorbeeld.org/trips/",
  "@id": "trip:123",
  "shortName": "trip 123",
  "headsign": "Brussel - Gent"
}
```

Lijst 2.2: Voorbeeld van een trip in JSON-LD.

Dit JSON-LD object bevat nu enkel informatie over de entiteit ‘trip:123’. Als er meerdere trips zijn met dezelfde context is het handiger om gebruik te maken van benoemde grafen (Engels: *Named Graphs*).

### Benoemde grafen

Triples kunnen onderverdeeld worden in grafen. Dit is handig om eigenschappen, metadata ... toe te kennen aan een groep triples. In sectie 2.3 zal blijken dat dit voor hypermedia API’s een belangrijke rol speelt. De oorspronkelijke triple is nu een *quad* geworden met

als vorm  $<\text{graaf}><\text{subject}><\text{predikaat}><\text{object}>$ . Een voorwaarde is dat de graaf zelf ook geïdentificeerd wordt met een URI zodat er van buitenaf hiernaar gerefereerd kan worden.

In JSON-LD wordt er gebruik gemaakt van het sleutelwoord ‘@graph’ waarin een array van alle entiteiten van een graaf komt. Lijst 2.3 toont hoe makkelijk het is om metadata toe te voegen aan de graaf.

```
{
  "@context": "http://vocab.gtfs.org/terms",
  "dc": "http://purl.org/dc/terms/",
  "dc:publisher": "Brecht Van de Vyvere",
  "@id": "http://voorbeeld.org/graaf/1",
  "@graph":
  [
    {
      "@id": "trip:123",
      "@type": "Trip",
      "trip": "http://voorbeeld.org/trips/",
      "shortName": "trip 123",
      "headsign": "Brussel - Gent"
    }, ...
  ]
}
```

Lijst 2.3: Voorbeeld van een trip in JSON-LD met benoemde graaf.

JSON-LD data kan omgezet worden naar triples om die dan vervolgens in te laden in een triplestore. Hierop kunnen dan vragen afgevuurd worden om informatie te bekomen.

## SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) is een zoektaal specifiek voor data in RDF formaat. Hiermee kan gelijk welke vraag gesteld worden aan een *triplestore*. In lijst 2.4 zie je een voorbeeld van een SPARQL-query die de URI’s van alle verschillende luchthavens in Italië opvraagt.

```

PREFIX dbpedia-owl: <http://dbpedia.org/ontology/> .
PREFIX dbpprop: <http://dbpedia.org/property/> .

SELECT DISTINCT ?entity
WHERE {
```

```

    ?entity a dbpedia-owl:Airport ;
      dbpprop:cityServed dbpedia:Italy .
}

```

Lijst 2.4: Voorbeeld van een SPARQL-query.

Dit is nu nog een relatief simpele vraag voor een SPARQL-*endpoint*. Het grote probleem [7] van deze endpoints is dat slechts 30% effectief 99% online blijft in een maand. Er is namelijk geen restrictie op het soort queries die uitgevoerd kunnen worden. Voor reële toepassingen is het ontoelaatbaar dat wanneer veel clients complexe queries afvuren, de server kan uitvallen.

We zullen eerst het spectrum van *Linked Data fragments* (LDF's) bespreken. Daarna bekijken we een oplossing om SPARQL-endpoints schaalbaar te kunnen queryen.

### Linked Data Fragments

Een Linked Data dataset is een collectie triples die door iemand wordt vrijgegeven. Meestal zijn we enkel geïnteresseerd in bepaalde delen van deze collectie, bijvoorbeeld met een SPARQL-query beschik je enkel over de data die aan de query voldoet. Door een URL te dereferenceren beschik je enkel over informatie die over dit onderwerp gaat. Deze verschillende interfaces hebben gemeen dat ze een bepaald fragment over eenzelfde dataset voorstellen. In figuur figuur 2.3 zie je de LDF's as.



Figuur 2.3: Linked Data Fragments-as

Bij elk soort LDF hoort een bepaalde *selector*. Dit is een functie die bepaalt of een triple tot het gewenste deel van de dataset behoort of niet. Voor een datadump is deze functie altijd TRUE voor gelijk welke triple. Triples die horen bij een SPARQL-query hebben de query als selector.

Naast data, volgens een bepaalde selector, zijn er nog twee andere eigenschappen die LDF van elkaar onderscheiden:

- Hypermedia controls. Dit zijn URL's die meer informatie bevatten over bepaalde entiteiten

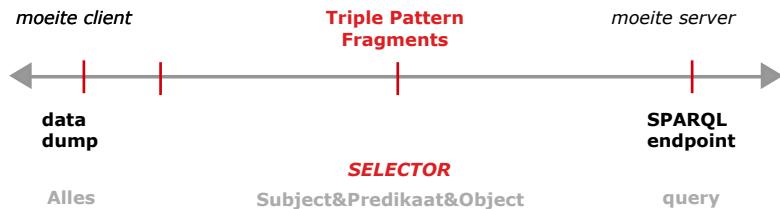
in het fragment, m.a.w. informatie over gerelateerde Linked Data Fragmenten. Voor datadumps en het resultaat van een SPARQL query komt dit neer op het derefereren van de URL die een bepaald object voorstelt.

- Metadata is informatie over de data zelf, bijvoorbeeld het aantal triples of datum van aanmaak/queryen. Deze informatie wordt in triples toegevoegd aan het datafragment.

Het interessante aan de LDF-as is dat verschillende interfaces met elkaar vergeleken kunnen worden. LDF's is dus geen technologie, maar een visie hoe de balans tussen client en server kan afgewogen worden. Met data, controls en metadata in het achterhoofd werd in 2014 een oplossing bedacht om het queryen van Linked Data schaalbaar te maken (sectie 2.2).

### Triple Pattern Fragments

Bij SPARQL-endpoints (sectie 2.2) is er een reële kans dat deze niet 100% online blijven bij een groot aantal requests [7] doordat er geen beperking is in de grootte en complexiteit van queries. Triple Pattern Fragments (TPF's) introduceert nieuwe afwegingen tussen client en server (figuur 2.4).



Figuur 2.4: Linked Data Fragments-as met Triple Pattern Fragments.

Deze nieuwe LDF-visie houdt twee zaken in:

- Servers bieden een simpele, *low-cost* interface aan.
- Clients zijn verantwoordelijk voor het oplossen van bepaalde queries.

### Server

De server is verantwoordelijk voor het teruggeven van triples die voldoen aan een bepaald triple patroon. Zo'n patroon is de combinatie van een subject, predicaat en object waarvan minstens een element een waarde heeft. Een voorbeeld van een HTTP template van een TPF-server is te zien in lijst 2.5. Dankzij hypermedia controls is het mogelijk om een LDF gepagineerd

terug te geven. Dit wordt gedaan met behulp van de Hydra hypermedia vocabularium<sup>3</sup>: hydra:totalItems, hydra:itemsPerPage, hydra:firstPage, hydra:nextPage. In volgende subsectie zal het nut van metadata over het totaal aantal items aangetoond worden.

```
|| http://triplepatternfragments.org?subject={...}&predikaat={...}&object={...}
```

Lijst 2.5: Triple Pattern Fragments-interface

Het berekenen van triples die aan zo'n patroon voldoen is relatief simpel voor een server. Dankzij het beperkt aantal parameters kan er aan HTTP caching gedaan worden. Hierdoor is het mogelijk om veel clients tegelijk aan te kunnen en is het probleem van SPARQL-endpoints aan de kant geschoven.

### Client

Bij SPARQL kon de client gelijk welke vraag stellen aan de server en kreeg daar (hopelijk) antwoord op. Nu is de client verantwoordelijk voor het oplossen van een query door meerdere simpele vragen te stellen aan de server. Dit gaat ten koste van bandbreedte.

Om het algoritme uit te leggen, gebruiken we het voorbeeld met Italiaanse luchthavens van lijst 2.4. De WHERE-clausule bestaat uit triple-patronen: de eerste stelt het LDF voor van alle luchthavens en het tweede patroon stelt alle entiteiten voor die ten dienste van Italië staan. De eerste stap van het algoritme haalt voor elk patroon de eerste pagina van het eerste fragment op. Dankzij metadata over het fragment weet de client welk patroon het minste triples bevat zonder alle pagina's te moeten downloaden. Als er in het voorbeeld 1000 luchthavens zijn (eerste patroon), maar slechts 10 entiteiten zijn die Italië dienen (tweede patroon) dan wordt het tweede patroon als startpatroon gekozen. Over het resultaat van dit patroon wordt er geïterereerd en worden de variabelen hiervan ingevuld bij de andere WHERE-clausules. Dit proces noemt men het *joinen* van triples. Nu wordt hetzelfde algoritme recursief uitgevoerd op de ingevulde, overgebleven WHERE-clausules. Als de *count*-metadata van deze join één is, weten we dat dit een goed antwoord is.

Zoals je kan zien is TPF's een *greedy*-algoritme doordat telkens het kleinste LDF van een bepaald patroon gekozen wordt. Eén van de voordelen van deze manier van werken is dat het resultaat *gestreamd* kan worden. In tegenstelling tot SPARQL hoeft de client niet te wachten op het volledige resultaat om resultaten te tonen aan de gebruiker.

---

<sup>3</sup><http://www.w3.org/ns/hydra/core#>

### Analogie met routeplanning

Een SPARQL-endpoint is vergelijkbaar met de API van een routeplanner: een server staat in voor het berekenen van een oplossing van een complexe vraag. Een client kan bepaalde vragen stellen aan een server. Deze laatste berekent oplossingen en stuurt deze terug. Een SPARQL-server vraagt als input een query terwijl een routeplanningsserver complexe queries probeert op te lossen via een interface met HTTP parameters. Beide problemen kunnen opgelost worden door de intelligentie te verschuiven van de server naar de client en ervoor te zorgen dat de server een simpele, cachebare interface aanbiedt. Waar Triple Pattern Fragments een oplossing biedt voor het queryen van RDF-triples op het web, zal later (hoofdstuk 3) aangetoond worden hoe dit succesverhaal kan toegepast worden voor routeplannen met Linked Connections.

## 2.3 REST

In deze sectie bespreken we enkele richtlijnen voor het bouwen van duurzame Web API's.

REST staat voor Representational State Transfer. Dit is een set van beperkingen om een architectuur te bekomen die makkelijk uitbreidbaar en gedistribueerd is. Wanneer een Web API aan alle beperkingen voldoet, is deze *RESTful* genoemd. Hieronder volgt een overzicht van deze beperkingen.

### Beperkingen

- Client-server: een client en server moeten losgekoppeld zijn. De server biedt een uniforme interface aan dat door verschillende soorten clients (app's, websites...) gebruikt kan worden.
- Toestandsloos: de server houdt geen toestand bij van de client. Als twee requests  $R_1$  en  $R_2$  dezelfde informatie opvragen, moet  $R_2$  hetzelfde antwoord krijgen als  $R_1$ . De informatie die de server van de client krijgt zou voldoende moeten zijn om een antwoord terug te kunnen geven.
- Cachebaar: antwoorden van de server moeten opgeslagen worden. Als twee dezelfde aanvragen worden verstuurd, mag enkel de eerste effectief berekend worden<sup>4</sup>. De tweede

---

<sup>4</sup>Rekening houdend met de ingestelde cache-regels die bijhouden hoelang bepaalde document gecachet mogen worden.

aanvraag krijgt als het ware een kopie van de eerste. Dit heeft als voordeel dat de client sneller antwoordt krijgt en dat de server geen dubbel werk moet doen.

- Gelaagd systeem: een server bestaat uit verschillende lagen om zo modulair mogelijk te zijn. Hierdoor is het mogelijk om bepaalde functionaliteiten te verspreiden over verschillende servers om overbelasting te vermijden.

Om een uniforme interface te kunnen aanbieden moet de server rekening houden met een aantal bijkomende beperkingen:

- Resources: niet alleen bronnen moeten identificeerbaar (zie ook sectie 2.2) zijn met onveranderlijke URI's, maar ook collecties. Een URI van een bron bestaat uit de URI van de collectie waartoe het behoort + '/' + de identificatie van de bron zelf.

```
|| 'http://voorbeeld.org/boeken/1'.
```

Lijst 2.6: Een REST collectie 'boeken' bevat een boek met als identificatie 1.

Er kunnen meerdere representaties van zo'n bron of collectie zijn: HMTL, JSON-LD, XML, Turtle... Met behulp van *content negotiation* kan het gewenste formaat verkregen worden. Dit mechanisme zorgt ervoor dat een bron, dat één URI heeft, meerdere representaties kan hebben. Met behulp van de *Accept* HTTP-hoofding kan de client meegeven welke representaties voorrang krijgen.

- Acties: Om Create/Read/Update/Delete (CRUD) operaties toe te passen op zo'n resource wordt er gebruik gemaakt van volgende HTTP methodes: GET, POST, PUT en DELETE. Complexere acties, zoals sharen, filteren etc., bestaan meestal uit een werkwoord die de actie omschrijft. Om bronidentificatie niet te ondermijnen wordt dit werkwoord na de bron URI met een '?' geplaatst. Lijst 2.7 toont een voorbeeld van zo'n complexe actie.

```
|| http://voorbeeld.org/trips?zoek="Brussel - Gent"
```

Lijst 2.7: Zoek-actie op een bron.

- Hypermedia: Om de interface van een API uniform te maken, is een van de beperkingen die opgelegd wordt *Hypermedia as the Engine of Application State*, kortweg HATEOAS.

Hypermedia is simpelweg het volgen van links. Een Hypertext Markup Language (HTML) document zit vol met links naar foto's, video's, andere pagina's etc. HTML toont welke acties mogelijk zijn en als mens bepalen we zelf waar we geïnteresseerd in zijn. Met dit idee in gedachte zijn Hypermedia API's ontworpen. Zo'n API geeft mee aan de client welke acties allemaal mogelijk zijn en dan is het aan de client om hier slim mee om te gaan. Deze functionaliteit wordt beschreven in een semantisch formaat zoals JSON-LD. Net zoals een website aangepast kan worden zonder dat de boodschap verloren gaat, kan een Hypermedia API aangepast worden zonder dat de client hierbij geherprogrammeerd moet worden.

## 2.4 Routeplanning-algoritmen

Routeplannig heeft als doel om een of meerdere routes te berekenen die bepaalde eisen vervullen. Er zijn verschillende soorten queries: een vroegste aankomsttijd (Engels: *Earliest Arrival Time* (EAT))-query berekent de snelste route vanaf een bepaald moment en plaats om een bestemming te bereiken, een *multicriteria* query houdt rekening met meerdere criteria zoals het maximaal aantal overstappen, rolstoeltoegankelijkheid etc. Een andere veelgebruikte query voor routeplanning voor openbaar vervoer is een profielquery (Engels: *profile query*). Deze bepaalt verschillende routes binnen een bepaald tijdsinterval zodat de gebruiker alternatieven heeft. We zullen nu de twee meestgebruikte algoritmen bespreken om aan routeplanning te doen.

### 2.4.1 Dijkstra

Het algoritme van Dijkstra was tot voor kort de standaard voor het berekenen van het kortste pad tussen twee plaatsen. Een transportnetwerk wordt hierbij voorgesteld als een graaf met stops als knopen en bijvoorbeeld tripsequenties als verbindingen. Wanneer deze graaf gebouwd is, kan er een vertrekpunt gekozen worden en met Dijkstra het snelste pad berekend worden. Tijdens de opleiding industrieel ingenieur werd het algoritme van Dijkstra uitvoerig besproken. Daarom zullen we enkel optimalisaties van Dijkstra bespreken, omdat deze tot interessante inzichten kunnen leiden.

Het idee is om een geoptimaliseerde graaf vooraf te berekenen. Hierop kan er dan met Dijkstra zeer snel gequeryt worden. Enkele technieken die dit mogelijk maken, zijn:

- Voorafberekenen van *hubs*. Een hub is een subset van stops, bijvoorbeeld wanneer deze

dicht bij elkaar liggen. Bij continentaal routeplannen kunnen metropolen bijvoorbeeld als hubs worden beschouwd.

- Het berekenen van connecties wordt in verschillende ronden beschouwd. Eerst worden routes beschouwd tussen stations die rechtstreeks, zonder overstappen, bereikbaar zijn. Daarna wordt er met maximaal één overstap rekening gehouden enzovoort. Dit principe wordt gebruikt in Round-Based Public Transit Routing (RAPTOR) [4] .
- Voordat Dijkstra wordt toegepast, wordt de graaf geëxpandeerd in de tijd. Dit heeft als nadeel dat het aantal knopen en verbindingen stijgt waardoor de graaf nog groter wordt. Volgens paper [1] wordt bijvoorbeeld de graaf met alle stations (338 000) van Noord-Amerika geëxpandeerd naar een graaf van 100 miljoen knopen.
- Public Transit Labeling [3] maakt gebruik van etikettering. Zo krijgt elke verbinding twee etiketten, een voorwaartse en achterwaartse aangezien het om een gerichte graaf gaat, die aanduiden welke hubs bereikbaar zijn via deze verbinding.
- Schaalbare overstappatronen (Engels: *Scalable Transfer Patterns* (TP)) is de manier waarop Google Maps werkt. Het idee is dat publieke vervoersnetwerken ingedeeld kunnen worden in afgescheiden kleinere netwerken. Vooraf wordt er voor elke paar stops berekend wat de optimale routes zijn en waar overgestapt moet worden. Deze worden voorgesteld als overstappatronen. Dankzij deze voorstelling kan er een minimale graaf opgebouwd worden waardoor queryen zeer snel wordt. Het probleem hierbij was dat dit ofwel ten koste van het geheugengebruik, ofwel ten koste van de berekeningstijd gaat. In een zeer recente paper [1] is een verbetering van deze techniek voorgesteld die niet alleen een snelle voorbewerkingstijd heeft, maar ook weinig geheugen inneemt.

#### 2.4.2 Connection Scan Algorithm

Een andere recent ontwikkeld algoritme is het Connection Scan Algorithm (CSA). Routeplanning wordt in plaats van een algoritmisch probleem, een dataprobleem. Als bouwsteen wordt een connectie genomen. Dit is een verbinding tussen twee stopplaatsen zonder dat het voertuig tussenin stopt. Op een bepaald moment en plaats vertrekt een voertuig en stopt zoveel tijd later op een andere plaats. Neem je bijvoorbeeld de trein van Gent-Sint-Pieters naar Oostende en deze stopt in Brugge, dan bestaat deze route uit twee connecties (Gent-Sint-Pieters naar Brugge en Brugge naar Oostende).

CSA kan de Earliest Arrival Time van een route in lineaire tijd berekenen door connecties te scannen. Als invoer verwacht CSA een op vertrektijd gesorteerde lijst van connecties. Zo moet er pas gescand worden vanaf de ingestelde vertrektijd van een query. De minimale tijd voor elke andere stop wordt bijgehouden waardoor een minimale overspannende boom opgebouwd wordt. Een connectie is geldig wanneer de vertrekstop bereikbaar is én de aankomsttijd van de aankomststop kan verbeteren. Het algoritme stopt wanneer de bestemming bereikt is en alle connecties met als vertrektijd kleiner of gelijk aan de minimale aankomsttijd van de bestemming gescand zijn. Dan pas kan met zekerheid bepaald worden wat de snelste route is. Merk hierbij op dat in deze basisimplementatie geen rekening gehouden wordt met maximum aantal overstappen. Algoritme 1 toont hoe CSA in pseudocode. Met behulp van twee rijen wordt de vroegste aankomsttijd van elke stop bijgehouden en welke connectie hiervoor verantwoordelijk is. Wanneer de optimale route gevonden is, kan het pad gereconstrueerd worden met de *in\_connectie*-rij. Merk op dat enkel connecties  $c$  gescand worden met  $c.\text{vertrektijd} \geq \text{query.vertrektijd}$  en gestopt kan worden wanneer  $c.\text{aankomsttijd} > \text{aankomsttijd}[\text{query.bestemming}]$ .

---

**Algorithm 1** Pseudocode: Connection Scan Algorithm

---

```

// Initialisatie
for elke stop  $s$  do
    aankomsttijd[s] =  $\infty$ ;
    in_connectie[s] = null;
end for

aankomsttijd[query.startpunt] = query.vertrektijd;

for elke connectie  $c$  do
    if aankomsttijd[c.vertrekstop] < c.vertrektijd && aankomsttijd[c.aankomststop] > c.aankomststop then
        aankomsttijd[c.aankomststop] ← c.aankomsttijd
        in_connectie[c.eindstop] = c
    end if
end for

```

---

Enkele voordelen van CSA zijn:

- CSA werkt zeer snel door de lokaliteit van connecties. Deze kunnen makkelijk in het geheugen geladen en overlopen worden.

- Routeplanning is een dataprobleem geworden. Hierdoor kan bijvoorbeeld overstappen (Engels: *foot paths*), dit is het veranderen van stop of vervoersmiddel, gemodelleerd worden aan de hand van data. Ook interoperabiliteit tussen verschillende datasets is een dataprobleem: identifiers van stops, routes etc. moeten persistent voorgesteld kunnen worden. Door het voorzien van de juiste data kan CSA met deze zaken rekening houden.

Een groot nadeel van CSA is schaalbaarheid. Bij grote netwerken kan de lijst met gesorteerde connecties enorm oplopen. Om dit probleem op te lossen werd een uitbreiding gemaakt, genaamd *accelerated Connection Scan Algorithm* (ACSA) [6]. Dit heeft dus als doel om queries sneller te kunnen berekenen op zeer grote netwerken. ACSA verdeelt de graaf in meerdere cellen met als voorwaarde dat het kortste pad bewaard blijft. Voor elk zo'n cel wordt er een set van connecties bijgehouden die tot optimale routes leidt. Ook wordt er een set van connecties bijgehouden voor lange afstanden. Dit zijn routes die buiten de cel liggen. CSA is uitgebreid zodat de snelste route kan berekend worden over een ongesorteerde set van connecties. Deze sets zijn afkomstig van de verschillende cellen tussen start- en eindpunt. Vervolgens wordt er een tijdsinterval van mogelijke vertrektijden bepaald waarin de snelste route zou kunnen liggen. Indien de route niet gevonden is, wordt er opnieuw gezocht met een groter tijdsinterval. Connecties die bekijken zijn, krijgen een aanduiding. Met behulp van binair zoeken kan voor elke cel in  $O(\log(n))$  bepaald worden welke connecties nog niet gezien zijn.

In volgend hoofdstuk bespreken we wat Linked Connections is en hoe het gebruik maakt van CSA om routes te berekenen.

## Hoofdstuk 3

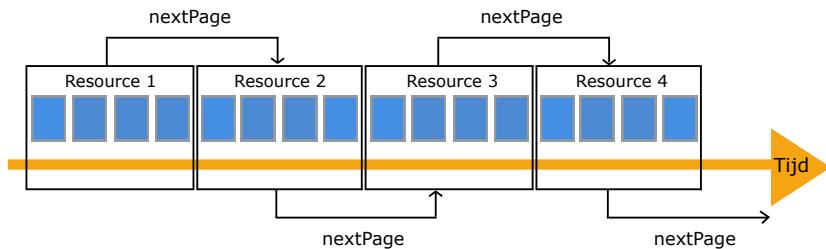
# Linked Connections

In de literatuurstudie hebben we gezien dat CSA een alternatief biedt op Dijkstra om routes te berekenen. Door het gebruik van connecties staat routeplanning dichter bij de data zelf. Dit hoofdstuk legt in de eerste sectie uit hoe Linked Connections gebruik maakt van CSA en welke technologieën hierbij gebruikt worden. Vervolgens wordt bekeken hoe deze connecties gegenereerd worden uit een dataset. Ten slotte wordt een *merger* besproken die een eerste stap zet tot een gedistribueerd routeplanningsysteem.

### 3.1 Principe

Linked Connections [2] is een framework dat *client-side* routeplanning mogelijk maakt. Een server is verantwoordelijk voor het publiceren van connecties (zie sectie 1.1). Deze connecties kunnen dan door een client opgevraagd worden om dan met CSA (subsectie 2.4.2) een route te kunnen berekenen. De visie van Linked Data Fragments vertelt dat we een bepaalde selector op onze dataset moeten toepassen om de dataset in fragmenten in te delen. Dit laat toe om simpele webservers te bouwen waarbij clients queries kunnen oplossen door de juiste fragmenten net op tijd te downloaden. Een eenvoudige selector is de tijdsfunctie die connecties met een verstrektijd binnen een bepaald tijdsinterval beschouwt als een fragment. Zo'n fragment noemen we een *basic Linked Connections Fragment* (BLCF). Met behulp van hypermedia-links (sectie 2.3) worden deze fragmenten aan elkaar verweven. Dit laat toe dat de server de selector op de dataset dynamisch kan aanpassen, bijvoorbeeld van 10 minuten naar een halfuur, afhankelijk van de hoeveelheid connecties binnen dat tijdsinterval. De Hydra-ontologie wordt gebruikt om de serverfunctionaliteit machine-verstaanbaar te maken voor de client. Figuur figuur 3.1 toont een

overzicht hoe de connecties gepresenteerd worden door de server.



Figuur 3.1: Connecties zijn onderverdeeld in fragmenten volgens een bepaald tijdsinterval, zogenaamde basic Linked Connections Fragmenten (BLCF). Met `hydra:nextPage` links kan makkelijk het volgende fragment gevonden worden.

De client kan met behulp van `hydra:nextPage` links makkelijk het volgende fragment vinden. De connecties van een fragment worden als invoer doorgegeven aan CSA (subsectie 2.4.2). Deze bouwt een minimaal opspannende boom met de snelste tijd om stops te bereiken. Wanneer het resultaat gevonden is, kan de client stoppen met ophalen van fragmenten. Hieronder volgt een overzicht van de technologieën waarvan Linked Connections gebruik maakt.

### 3.1.1 Technologieën

Het publiceren van connecties gebeurt met behulp van drie technologieën:

- De REST-principes (zie ook sectie 2.3) worden gevuld om ervoor te zorgen dat de fragmenten, de REST resources, cachebaar zijn. Er is namelijk een eindige verzameling URL's, afhankelijk van het ingestelde tijdsinterval. Lijst 3.1 toont een HTTP template van deze REST interface.

```
|| http://voorbeeld.org/connecties/?startTijd=?
```

Lijst 3.1: URI template van een basic Linked Connections Fragment.

Als  $F$  het tijdsinterval in minuten is van de fragmenten, dan kan het aantal mogelijke URI's berekend worden met formule (3.1). Is  $F = 10\text{min}$ , dan zijn er 144 verschillende fragmenten.

$$24 * 60 / F \quad (3.1)$$

De meeste publieke vervoersmaatschappijen rijden niet 24/24 in de praktijk dus het aantal fragmenten die connecties bevat, kan een stuk lager ingeschat worden. Met behulp van HTTP omleidingen kan dit opgelost worden.

- Hypermedia zorgt niet enkel voor het vinden van volgende fragmenten, maar dient ook voor het voorzien van extra *features* met Linked Data. Neem nu bijvoorbeeld een connectie. Deze heeft een bepaalde vertrek- en aankomststop. Er kunnen links toegevoegd worden die meer informatie geven over deze stops, bijvoorbeeld over de vorm van platforms of welke *points of interest* er in de buurt zijn.
- Het semantisch web zorgt voor de semantische interoperabiliteit van de gelinkte connecties en de hypermedia API. Gelinkte connecties maken o.a. gebruik van de GTFS<sup>1</sup> en Linked Connections<sup>2</sup> ontologiën. De API-functionaliteit maakt gebruik van de Hydra-ontologie. Deze beschrijft dat de client kan zoeken door de vertrektijdparameter in te vullen in de *URI-template*. Kortom, semantiek laat toe om generische clients te bouwen. Doordat zoveel mogelijk gebruik gemaakt wordt van Linked Data kan er gequeryd worden over het web met bijvoorbeeld *Triple Pattern Fragments* (TPF) (sectie 2.2). Er bestaan al talrijke servers online die mogelijke uitbreidingen kunnen zijn. Met behulp van GeoNames<sup>3</sup> kan er meer informatie gevonden worden over bepaalde geolocaties. Ook voor *Points of Interest* is een Linked Data portaal te vinden<sup>4</sup>.

In volgende sectie bespreken we hoe connecties voor deze masterproef gegenereerd zijn.

## 3.2 Convertor naar connecties

Door het wereldwijde gebruik van GTFS hebben we deze masterproef toegespitst op het converteren van een GTFS *feed* naar connecties. Doordat meerdere bestanden tegelijk behandeld moeten worden, werd er in de ontwerpfase beslist om met een databank te werken. Het is belangrijk op te merken dat connecties niet gesorteerd moeten zijn. Deze worden later in de databank van een Linked Connections server ingeladen die zelf sorteert tijdens het queryen.

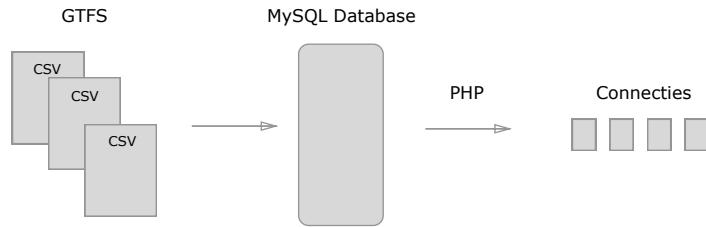
<sup>1</sup><http://lov.okfn.org/dataset/lov/vocabs/gtfs>

<sup>2</sup>Linked Connections ontologie: <http://semweb.mmlab.be/ns/linkedconnections>

<sup>3</sup>Momenteel is er al een TPF-server met gelinkte GeoNames beschikbaar: <http://data.linkeddatafragments.org/geonames>

<sup>4</sup>Ook hiervoor is een Linked Data portaal aanwezig die door intelligente clients gebrowset kunnen worden: <http://data.linkeddatafragments.org/linkedgeodata>

Figuur 3.2 toont een globaal overzicht om connecties te genereren. De eerste stap bestaat uit het inladen in een MySQL-databank. Daarna worden connecties met PHP scripts gegenereerd en weggeschreven naar een apart bestand. In volgende subsectie gaan we dieper in de logica om connecties te genereren met queries.



Figuur 3.2: Overzicht hoe connecties worden gegenereerd uit een GTFS feed. Deze wordt ingeladen in een MySQL databank. Met de scripttaal PHP worden connecties gegenereerd.

### 3.2.1 Connectiescript

Nadat de bestanden van een GTFS feed in gelijknamige tabellen van databank ingeladen zijn, kan het genereren van connecties starten. Deze connecties worden dag per dag berekend. Ofwel worden start- en einddatum meegegeven als parameters, ofwel worden start- en einddatum van de GTFS feed genomen.



Figuur 3.3: Per dag worden de corresponderende service ID's, trip ID's en route ID's opgehaald. De vertrek-/aankomststopplaats met respectievelijk vertrek-/aankomsttijd van connecties worden berekend uit een verzameling stoptijden.

Een connectie vereist informatie uit verschillende tabellen. Deze moeten per dag  $d$  stapsge- wijs opgehaald worden:

1. Een eerste stap is het ophalen van alle services die actief zijn op  $d$ . Deze services zitten in de *calendar* en *calendar\_dates* tabellen.

2. De *trips* tabel bevat de mapping tussen een service, route en trip. Elke service komt overeen met een route en meerdere trips. Een route is niet essentieel voor een connectie, maar bevat wel interessante informatie zoals de routebeschrijving en type voertuig.
3. De laatste stap is het overlopen van stoptijden met *stoptimes.txt*-bestand. Voor elke trip komt een reeks stoptijden overeen. Uit de combinatie van twee opeenvolgende stoptijden van eenzelfde trip wordt een connectie berekend. Lijst 3.2 bevat een voorbeeld van twee opeenvolgende stoptijden van een bepaalde trip. De vertrektijd en vertrekstop van een eerste stoptijd wordt samengenomen met de aankomsttijd en aankomststop van de tweede stoptijd.

```

trip_id , arrival_time , departure_time , stop_id , stop_sequence
16 , 07:18:00 , 07:18:00 , 8200100 , 1
16 , 07:33:00 , 07:33:00 , 8200110 , 2

```

Lijst 3.2: Voorbeeldstoptijden uit GTFS *stop\_times.txt*.

Hiermee zijn alle stappen doorlopen en kan een connectie weggeschreven worden. Lijst 3.3 toont hoe een connectie eruit ziet bij output. Merk op dat een '@context' niet is toegevoegd en de connectie ook niet in een graaf zit. In subsectie 3.2.2 wordt hierover meer uitleg gegeven.

```

{
  "@id": "http://example.org/connections/1"
  "@type": "http://semweb.mmlab.be/ns/linkedconnections#Connection",
  "http://vocab.gtfs.org/terms#trip": "16",
  "http://vocab.gtfs.org/terms#route": "route-1",
  "http://semweb.mmlab.be/ns/linkedconnections#departureTime": "2015-10-21
    T06:18:00.000Z",
  "http://semweb.mmlab.be/ns/linkedconnections#departureStop": "8200100",
  "http://semweb.mmlab.be/ns/linkedconnections#arrivalTime": "2015-10-21T06
    :33:00.000Z",
  "http://semweb.mmlab.be/ns/linkedconnections#arrivalStop": "8200110"
}

```

Lijst 3.3: Voorbeeld van een connectie voor een bepaalde trip in JSON-LD formaat.

Als je de tijdsaanduidingen van een stoptijd (lijst 3.2) vergelijkt met de vertrek- en aankomsttijd van een connectie (lijst 3.3) zie je dat de connectietijd een uur vroeger lijkt dan de

stoptijd. Dit komt doordat de tijd van connecties in *Coordinated Universal Time (UTC)* staat. Dit is herkenbaar aan de 'Z' die van achteren staat. Er moet altijd rekening gehouden worden met de tijdszone van het gebied van de GTFS feed. Lijst 3.2 stelt een feed uit België voor in wintertijd. België volgt in de wintertijd UTC+1 en in de zomertijd UTC+2. In de wintertijd moet er dus een uur afgetrokken worden om een correcte UTC tijd te bekomen.

Een van de moeilijkheden voor het genereren van connecties was het ophalen van trips die over meerdere dagen actief zijn. GTFS lost dit op door de tijd na middernacht op te tellen bij 24u. Twee uur na middernacht staat bijvoorbeeld weergegeven als 26u. Volgens GTFS rijden deze stoptijden allemaal op dezelfde dag, maar voor Linked Connections zijn dit effectief twee verschillende dagen, omdat er met exacte tijden gewerkt wordt. Het script moet dus bij de startdatum rekening houden met trips van de dag ervoor die na middernacht nog actief zijn. Om dit op te lossen werden er twee extra vlaggen toegevoegd in de databank of de vertrekken/of aankomsttijd van een stoptijd voor of na middernacht plaatsvinden. Met aparte queries kunnen de stoptijden na middernacht opgevraagd worden.

Een voordeel van een databank is dat er makkelijk features kunnen toegevoegd worden met behulp van extra kolommen of tabellen. Het is nadelig dat alle data vooraf ingeladen moeten worden, want dit kost tijd. Na het inladen kan het berekenen van connecties effectief beginnen. In tabel 3.1 staat een overzicht van de drie gebruikte datasets in deze masterproef met de tijd om in te laden. Voor zeer grote datasets, zoals De Lijn, is inladen een *bottleneck*.

	Tijd inladen databank (min)	Grootte (MB)	Periode (weken)
NMBS	3.1	1.1	12
NS	8.35	21.4	56
De Lijn	100	44	10

Tabel 3.1: Tijd om een GTFS feed in te laden in een MySQL databank.

Het genereren van connecties zelf gaat een stuk sneller. In tabel 3.2 zie je dat voor kleine datasets (zoals NMBS en NS) het minder dan een minuut duurt om de connecties voor een dag te berekenen. De Lijn scoort zeer slecht door de grootte van de dataset. In dit voorbeeld werden connecties voor dinsdag 1 december 2015 berekend.

Ondertussen is er een veel snellere convertor <sup>5</sup> gemaakt die connecties zonder databank kan genereren. In de laatste kolom van tabel 3.2 staat de tijd met de nieuwe convertor vermeld.

<sup>5</sup>Zie: [github.com/linkedconnections/gtfs2lc](https://github.com/linkedconnections/gtfs2lc)

	Connecties	Aantal stops	Tijd met databank (min)	Tijd met nieuwe convertor (min)
NMBS	55479	597	0,21	0,18
NS	41796	2531	1,30	0,25

Tabel 3.2: Tijd om connecties te genereren voor één dag.

Deze werkt veel sneller en vereist geen inlaadfase meer.

Connecties worden achtereenvolgens geschreven naar een apart bestand. Hiervoor wordt er gebruik gemaakt van een nieuwe specificatie, genaamd *JSONLD-stream*<sup>6</sup>.

### 3.2.2 JSONLD-stream

Om connecties als Linked Data te publiceren werd ervoor gekozen om JSON-LD te gebruiken. JavaScript Object Notation (JSON) wordt beschouwd als het standaard dataformaat op het web dankzij de compactheid, leesbaarheid en vele handige tools die hiervan gebruik maken. Wanneer een verzameling objecten dezelfde context heeft, zijn er twee mogelijkheden om deze te publiceren:

- een gemeenschappelijke context voorzien en alle objecten in bijhorende graaf stoppen (lijst 2.3). Deze methode vereist dat de graaf in het geheugen geladen moet worden vooraleer verdere operaties mogelijk zijn.
- elk object een context geven (lijst 2.2). Met deze methode kan object per object gestreamd worden, maar telkens de context toevoegen zorgt voor een grote overhead.

Een oplossing hiervoor is de JSONLD-streamspecificatie. Deze geeft aan dat een context van een object impliciet van toepassing is op alle andere objecten van hetzelfde document. Zo moet er maar eenmalig een context opgegeven worden. De voorbewerker kan volgens dit formaat connecties als Linked Data wegschrijven zonder extra data in het geheugen te moeten houden. Dit stroomlijnen is belangrijk om de data later lijn per lijn te kunnen inladen in de databank van een LC server.

Vooraleer we intermodaliteit bekijken, bespreken we eerst hoe een Linked Connections client kan aangemaakt worden.

---

<sup>6</sup>Zie: <https://github.com/pietercolpaert/jsonld-stream>

### 3.3 Client

De client is verantwoordelijk voor het berekenen van de snelste route met CSA. Deze kan met een paar lijnen code aangemaakt worden (lijst 3.4).

```

var planner = new Client({"entrypoints" : ["http://
example.linkedconnections.org/"]});

planner.query({
    "startstop": "Brussel-Zuid",
    "eindstop": "Gent-Sint-Pieters",
    "starttijd": new Date("2015-11-05T10:00")
}, function (stroom) {
    stroom.on("result", function (pad) {
        // pad bevat verzameling connecties die snelste route voorstellen
    });
    stroom.on("data", function (connectie) {
        // connectie is gebruikt geweest voor minimaal opspannende boom
    });
});

```

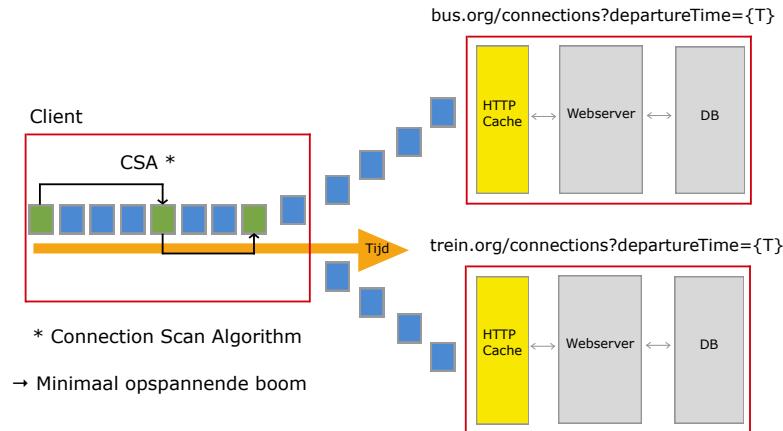
Lijst 3.4: Code om een LC client op te zetten in JavaScript.

Er zijn twee fases voordat routes berekend kunnen worden:

- Configuratiefase: de LC client moet weten welke servers er bereikt moeten worden door de webadressen toe te voegen aan de *entrypoints*-rij. Ook moet een query ingesteld worden. Met *startstop*, *starttijd* en *eindstop* wordt respectievelijk het vertrekpunt, vertrektijd en bestemming van de query bedoeld. Deze benamingen zullen in het vervolg van deze masterproef vaak gebruikt worden. Momenteel zijn dit de enige routeplanningparameters die ondersteund worden.
- Nadat de planner geïnitialiseerd is, zal deze connecties opvragen en doorgeven aan CSA. CSA geeft een *stroom* terug die notificaties afvuurt wanneer een bepaalde actie ondernomen is. Zo wordt een *data*-notificatie uitgestuurd wanneer een connectie gebruikt is voor de minimaal opspannende boom.

Voor deze masterproef was het enkel mogelijk om een connectiestroom (Engels: *connection stream*) van één *entrypoint* op te vragen. In tegenstelling tot reguliere routeplanners, die alle data lokaal staan hebben, is het de bedoeling van Linked Connections om schaalbaar verschillende

servers te kunnen opzetten voor elke modus en/of land. Figuur 3.4 toont een overzicht van een opstelling met meerdere servers. Rechts staan twee servers, de ene verantwoordelijk voor busdata, de andere voor treindata.



Figuur 3.4: Opstelling van een client en twee Linked Connections servers die elk verantwoordelijk zijn voor een modi.

CSA verwacht als invoer een gesorteerde rij van connecties. Elke connectiestroom van een server is zelf gesorteerd, maar niet onderling. De *merger* die hiervoor zorgt, wordt in volgende sectie besproken.

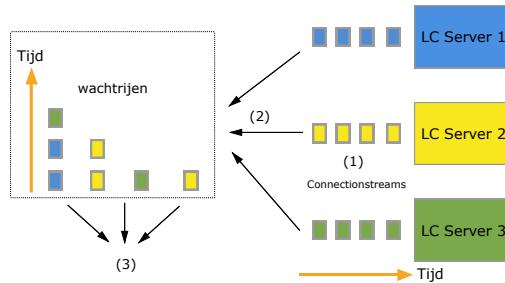
### 3.3.1 Merger

Een merger zorgt ervoor dat één of meerdere gesorteerde connectiestromen dynamisch samengevoegd kunnen worden. Het resultaat is een connectiestroom die zelf ook gesorteerd is op vertrektijd. Met dynamisch wordt bedoeld dat de client op elk moment kan beslissen om connectiestromen toe te voegen of te verwijderen.

Connecties komen onder de vorm van een *stream* binnen. Een stream verwerkt data en stuurt deze door met notificaties. Deze notificaties worden opgevangen door notificatieluisterraars. Dit werkt op een asynchrone manier waardoor het niet vanzelfsprekend is om aan de sorteringsvoorwaarde te voldoen en zeker niet om de client tijd te geven om andere stromen toe te voegen of te verwijderen. De volgorde waarin deze notificaties worden opgeworpen is onvoorspelbaar waardoor een mechanisme ontworpen moet worden om te verzekeren dat elke stroom aan bod is gekomen.

Figuur 3.5 toont een overzicht van een merger. De merger luistert (1) naar de verschillende

onderling gesorteerde datastromen. Wanneer er data vrijgegeven wordt, wordt dit toegevoegd aan de wachtrijen (2). Deze wachtrijen hebben als voorwaarde dat ze zelf gesorteerd zijn.



Figuur 3.5: Overzicht merger

Om te kunnen garanderen dat de totale connectiestroom (3) gesorteerd is, wordt er geëist dat de wachtrijen connecties bevatten van elke connectiestroom (1). Doordat de stromen asynchroon werken, moet elke stroom telkens gepauzeerd worden. Nadat elke stroom data heeft vrijgegeven kan de merger connecties teruggeven (3) met dezelfde vertrektijd voor elke stroom. Deze minimale vertrektijd wordt apart bijgehouden. Wanneer alle connecties met deze minimale vertrektijd verwerkt zijn, wordt deze geüpdatet. Een andere reden waarom de datastromen telkens worden gepauzeerd is door het feit dat de client beslissingstijd moet hebben. Zo kan er op elk moment beslist worden om een bepaalde connectiestroom uit te schakelen of toe te voegen. Een use case zou bijvoorbeeld het toevoegen van busdata zijn wanneer het doel binnen een bepaalde straal bereikbaar is.

Lijst 3.5 toont hoe een merger geïnitialiseerd kan worden. Merk op dat bij elke connectiestroom een naam toegevoegd is. Zo kan de merger controleren dat er voor elke stroom minstens één connectie in de wachtrijen aanwezig is. De client kan elk moment beslissen om een connectiestroom toe te voegen aan de merger. Het vroeger stoppen van een connectiestroom wordt buiten de merger uitgevoerd. De merger ontvangt een notificatie dat de stroom gestopt is en zal hiermee rekening houden.

```

var connectiestromen = [
    [ 'stroom1', connectiestroom1 ],
    [ 'stroom2', connectiestroom2 ],
    ...
];

var samengevoegde_connectiestroom = new Merger(connectiestromen,

```

```
    query.vertrektijd);  
  
    samengevoegde_connectiestroom.voegtoe('stroom3', connectiestroom3);
```

Lijst 3.5: Voorbeeldcode van een merger. Deze voegt meerdere stromen van connecties samen. Er kunnen connectiestromen dynamisch worden toegevoegd.

## 3.4 Conclusie

We hebben gezien hoe connecties gegenereerd kunnen worden uit een GTFS feed en hoe een Linked Connections server deze als *basic Linked Connections Fragments* aanbiedt aan de client. Deze laatste maakt dan gebruik van CSA om de snelste route te berekenen. Een eerste stap naar intermodaliteit werd gezet door de introductie van een merger van connectiestromen.

In volgend hoofdstuk onderzoeken we hoe client-side routeplanning sneller gemaakt kan worden door nieuwe afwegingen tussen client en server te onderzoeken.

## Hoofdstuk 4

# Optimalisatie

In dit hoofdstuk wordt besproken hoe Linked Connections sneller gemaakt kan worden door connecties beter te filteren op de server. Eerste sectie bespreekt welke mogelijkheden er hiervoor zijn. Hierna volgt een verduidelijking hoe deze informatie bekomen wordt met extra voorbewerking van de dataset. Vervolgens wordt een nieuw LDF-type gedefinieerd die connecties bevat rekening houdend met de extra filter. Ten slotte worden twee client-side algoritmen voor routeplanning voorgesteld die hiervan gebruik maken.

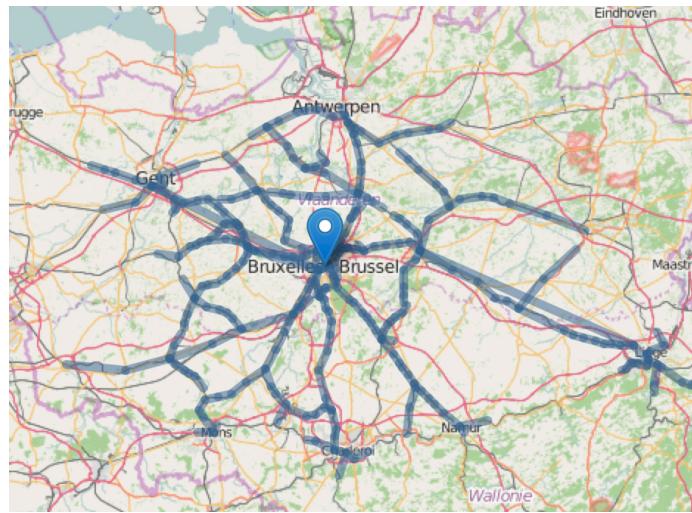
### 4.1 Geofiltering

Het CSA-algoritme bouwt met de binnenkomende connecties een minimale omspannende boom. Dit kan voor centraal gelegen vertrekplaatsen visueel voorgesteld worden als een cirkel die telkens groter wordt (figuur 4.1). CSA scant connectie per connectie waardoor de minimaal omspannende boom ook maar aan dit tempo kan groeien. Ofwel start een connectie vanuit de vertrekstop, ofwel stelt deze een *join* voor met een voorgaande connectie.

Het idee van filtering bij Linked Connections is om enkel gefilterde connecties op basis van de vertrektijd én vertrekstop terug te geven. Extra functionaliteit toevoegen aan de web-interface van de server kan door extra HTTP-parameter(s) toe te voegen of door extra HTTP-requests te sturen. In deze masterproef wordt enkel gewerkt met extra HTTP-parameter(s).

#### 4.1.1 Geoparameter

Een eerste inspiratie was om connecties onder te verdelen op dezelfde manier als kaarten van Open Street Maps werken. Hierbij wordt er met *tiles* gewerkt. Een *tile* is een rechthoekig



Figuur 4.1: Visualisatie van CSA. De minimaal opspannende boom kan voorgesteld worden als een groeiende cirkel rondom het vertrekpunt.

stuk van een kaart die op basis van drie parameters kan geïdentificeerd worden: het  $x$ - en  $y$ -coördinaat en een *zoomlevel*  $z$ . Bij statische data zoals kaarten is dit zeer handig, omdat deze stukjes cachebaar zijn. Bij transportdata moet er ook rekening gehouden worden met tijd, waardoor zo'n systeem niet haalbaar is.

Resources zijn pas cachebaar als er een beperkte set van URL's zijn. Aangezien elke publieke transportmaatschappij een beperkte set stops heeft, werd ervoor gekozen om het vertrekpunt van een query toe te voegen als extra HTTP parameter (lijst 4.1). Connecties die in de buurt van de vertrekstop liggen, hebben zeker initieel een grotere kans om gebruikt te worden. In volgende subsectie bespreken we de criteria om connecties terug te geven.

```
|| http://voorbeeld.org?starttijd={...}&startstop={...}
```

Lijst 4.1: Interface LDF met starttijd en startstop als parameters.

### 4.1.2 Criteria

Door de vertrektijd én vertrekstop mee te geven aan de server kan de server slimmer connecties teruggeven. De bedoeling is om een subset van stops te bekomen. Hiervoor zijn er verschillende mogelijkheden:

- De server kiest een bepaalde straal  $s$  waarbinnen stops mogelijk zijn. Zo is de startstop van

de query het middelpunt van de cirkel. Enkel connecties waarvan de vertrekstop binnen deze cirkel ligt worden in rekening gebracht. Met dit criteria wordt er op geografische wijze gefilterd.

- Een andere mogelijkheid gebruikt het minimaal aantal overstappen (sectie 2.1) om een stop  $s$  te kunnen bereiken vanuit de startstop. Dit benoemen we vervolgens  $K(\text{startstop}, s)$ . Deze informatie moet vooraf berekend worden (subsectie 4.2.2). Moeilijk bereikbare stops vereisen meestal een of meerdere overstappen. Door een maximum aantal mogelijke overstappen  $K_{\max}$  op te leggen, kan de server een subset van stops bekomen.

Naast het selecteren van een subset van stops zorgt volgende criterium voor extra filtering in de tijd:

- Afhankelijk van de vertrektijd zijn er verschillende routes mogelijk tussen elk paar stops in een GTFS feed. Deze kunnen een verschillende tijdsduur hebben. Voor elk paar stops  $s_i$  en  $s_j$  wordt de minimale tijdsduur  $R(s_i, s_j)$  bijgehouden. Een connectie  $c$  die vertrekt vanuit een stop  $s$  is pas mogelijk indien  $c.\text{vertrektijd} \geq \text{starttijd} + R(\text{startstop}, s)$ .

Net zoals bij basic LCF's is een tijdsinterval vereist waarbinnen de vertrektijd van connecties mogelijk zijn. De grootte van dit tijdsinterval noemen we vervolgens  $T$ . In deze masterproef worden twee technieken besproken die zowel van  $T$  als  $R$  gebruik maken. Het gebruik van  $K$  is optioneel. Deze worden respectievelijk in sectie 4.4 en sectie 4.5 besproken. Volgende sectie geeft meer uitleg hoe  $R$  bepaald kan worden voor elke paar stops van een GTFS dataset.

## 4.2 Minimale tijdsduur

Om de minimale tijdsduur  $R$  tussen elk paar stops te berekenen, introduceren we twee nieuwe termen:

- $K = 0 \Rightarrow \text{direct bereikbare stop}$ : dit is een stop dat zonder overstappen bereikbaar is vanuit de vertrekstop. In GTFS termen betekent dit dat de stops verbonden zijn met één trip.
- $K > 0 \Rightarrow \text{indirect bereikbare stop}$ : dit is een stop die minstens  $K$  (met  $K > 0$ ) overstappen vereist vanuit de startstop.
- $\text{buur}$ : dit is een stop die direct of indirect bereikbaar is vanuit de startstop.

Kortom, voor elke stop  $s$  moeten elke mogelijke buur  $b$  bepaald worden samen met informatie over het minimaal aantal overstappen  $K(s, b)$  en minimale tijdsduur  $R(s, b)$  om deze te bereiken. Hiervoor worden er twee extra voorbewerkingsfases toegevoegd.

#### 4.2.1 Direct bereikbare stops

Alle direct bereikbare stops voor elke stop van de dataset kan makkelijk afgeleid worden uit het *stoptimes.txt* bestand. Hierin staan alle mogelijk trips die voorkomen. Vooraf moet deze gesorteerd worden zodat stopsequenties in juiste volgorde voorkomen. Hierna moet er éénmalig door het bestand gelopen worden om voor elke stop een minimaal opspannende boom van direct bereikbare stops te bekomen. De gewichten van de verbindingen stellen de minimale tijd voor om die buur te bereiken. Deze informatie wordt weggeschreven naar een CSV-bestand voor volgende fase.

#### 4.2.2 Buren

Om informatie over alle buren van een stop te bekomen, moeten de indirect bereikbare stops berekend worden uit de minimaal opspannende bomen uit subsectie 4.2.1. Een indirect bereikbare stop  $b$  van stop  $s$  vereist een minimaal aantal overstappen  $K(s, b) > 0$  en een minimale tijdsduur  $R(s, b) > 0$ . Als er  $n$  stops zijn in een dataset, moeten er  $n * n$   $K$ 's en  $R$ 's berekend worden. Dit zou gemakkelijk met Floyd-Warshall kunnen, maar bij grote netwerken zoals bij De Lijn vormt het geheugen een probleem. De Lijn heeft namelijk afgerond 35000 stopplaatsen waardoor een matrix van 35000 x 35000 in het geheugen moet gehouden worden. Er is geen sprake van symmetrie bij openbaar vervoer: van een stop  $A$  naar een stop  $B$  gaan kan andere routes, en dus tijdsduren, omvatten als van  $B$  naar  $A$ . Pseudocode hiervan is te vinden in algoritme 2. Er kan ingesteld worden tot hoeveel overstappen maximaal gezocht moet worden. Het algoritme maakt gebruik van een wachtrij van stops die nog moeten bekeken worden. Wanneer een stop bekeken wordt, worden zijn rechtstreekse buren afgelopen. Een buur die nog niet ontdekt is, wordt toegevoegd aan de wachtrij. Elke stap probeert de tijdsduur en/of aantal overstappen te minimaliseren van een stop naar een buur. Het object dat bekomen wordt voor een bepaalde stop staat afgebeeld in lijst 4.2. Het is mogelijk om hier later extra metadata aan toe te voegen. Dit wordt weggeschreven naar een aparte buren-tabel van een databank op de Linked Connections server.

```
{
    stop_id : 123,
    aantal_direct_bereikbare_stops: 58,
    // andere metadata over de stop
    buren: [
        {
            stop_id: 253, // stop identifier van buur
            R: 1800, // minimale tijdsduur in seconden
            K: 2 // minimaal aantal overstappen
        }, {
            ...
        }
    ]
}
```

Lijst 4.2: Metadata over een stop met bijhorende buren.

Zoals je kan zien in algoritme 2 wordt ook het aantal rechtstreeks bereikbare stops bijgehouden voor elke buur. Sectie 4.4 zal hierover meer uitleg geven. Tabel 4.1 toont hoelang het duurt om buren te berekenen volgens de grootte van het netwerk van de NMBS en NS. Hierbij is  $K_{max} = 20$  waardoor alle mogelijke stopporen berekend worden voor deze datasets.

Dataset	Aantal stops	Tijd (min)
NMBS	597	6,35
NS	2532	21,19

Tabel 4.1: Tijd om buren van alle GTFS stops te berekenen.

In volgende sectie bespreken we hoe deze informatie gebruikt wordt om client-side routeplannen sneller te maken.

### 4.3 Neighbour Linked Connections Fragment

Wanneer de client een request met vertrektijd én vertrekstop opvraagt, geeft de server connecties terug die zowel uit de vertrekstop als uit de buren vertrekken. Zo'n fragment op basis van vertrektijd en vertrekstop noemen we *Neighbour Linked Connections Fragment* (NLCF). Het idee is dat een NLCF meer nuttige informatie bevat dan een basic LCF door het gebruiken van

---

**Algorithm 2** Berekenen van Minimale Ospannende Boom (MOB) van burenstops voor een GTFS stops.txt-bestand.  $K_{max}$  is het maximaal aantal overstappen waarmee rekening gehouden wordt.

---

```

 $K_{max} \leftarrow$  natuurlijk getal
for elke stop  $s$  in stops.txt do
    // Initialisatie
    var mob = {};
    var wachtrij = []; // nog te bezoeken stops
    for elke direct bereikbare stop  $b$  van  $s$  do
        // K: minimum aantal overstappen
        // R: minimum tijdsduur
        mob[b.stop_id] = { K : 0, R : b.R, aantal_rechtstreekse_buren: start.buren.length };
        if ( $K_{max} > 1$ ) then
            wachtrij.push(b.stop_id);
        end if
    end for
    while ( wachtrij niet leeg is ) do
        var start = wachtrij.shift();
        for elke direct bereikbare stop  $b$  van  $start$  do // Zit nog niet in MOB
            if (!mob[b.stop_id]) then
                mob[b.stop_id] = { K: start.K + 1, tijdsafstand: start.R + b.R, aantal_rechtstreeks_buren: start.buren.length };
                if (b.K < T) then
                    wachtrij.push(b); // nog verder te onderzoeken
                end if
            else
                // Minimum aantal overstappen
                if (start.K + 1 < mob[b.stop_id].K) then mob[b.stop_id].K = start.K + 1;
                end if
            end if
        end for
    end while
end for

```

---

informatie over de vertrekstop en zijn buren. Hierbij zijn er twee in te stellen parameters:

- Het maximaal aantal overstappen  $K_{max}$ . Moeilijk bereikbare stops hebben een grotere kans op meer overstappen. De vraag is of het nuttig is om hiermee rekening te moeten houden.
- Het tijdsinterval  $T$  waarin connecties van een stop worden teruggegeven.

De query die hierbij gebruikt wordt op de server staat vermeld in lijst 4.3. Voordat deze query wordt uitgevoerd worden alle buren binnen  $K_{max}$  opgehaald uit subsectie 4.2.2.

```

||| SELECT *
  FROM connecties
 WHERE (vertrekStop = startstop && vertrekTijd > starttijd && vertrekTijd <
        starttijd + T)
 OR  (vertrekStop = buur1 && vertrekTijd > R(startstop,buur1) && vertrekTijd <
      R(startstop,buur1) + T)
 OR  (vertrekStop = buur2 && vertrekTijd > R(startstop,buur2) && vertrekTijd <
      R(startstop,buur2) + T)
 OR ...

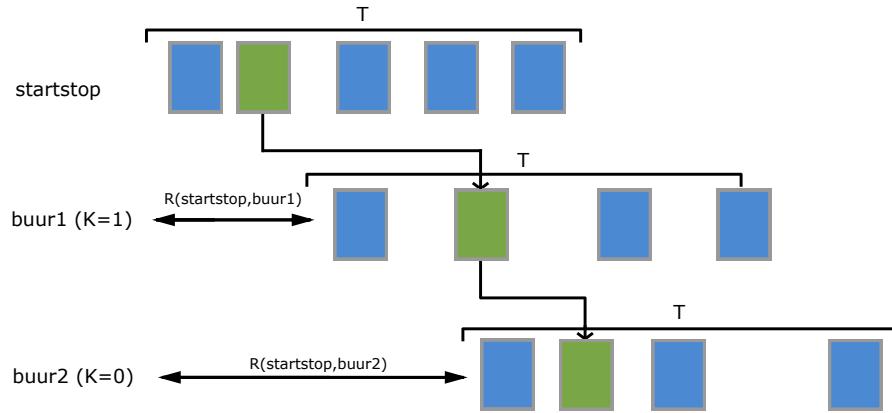
```

Lijst 4.3: SQL query template van NLCF.

Een Linked Connections server is nu verantwoordelijk voor BLCF's en NLCF's. De optimatisatiemethodes in volgende secties gebruiken elk een verschillende NLCF-interpratie.

## 4.4 Heuristiek

De heuristische techniek maakt enkel gebruik van neighbouring LCF's. Deze NLCF's hebben een vorm zoals in figuur 4.2 te zien is. Naast  $R(startstop, buur)$  heeft elke buur een even groot tijdsinterval  $T$ . Wanneer alle connecties van het snelste pad binnen dit tijdsinterval ligt voor elke stop, dan kan met één zo'n fragment het resultaat gevonden worden. Als het antwoord niet gevonden is, moet een volgende vertrekstop gekozen worden. Hiervoor wordt er gebruik gemaakt van een heuristiek. Deze bepaalt aan de hand van enkele parameters welke stop het meest kans heeft om op het pad van de snelste route te liggen. De stop met de hoogste score wordt gekozen als vertrekstop van de volgende NLCF.



Figuur 4.2: Neighbour Linked Connections Fragment van heuristiek-methode bevat connecties van startstop en buren binnen een bepaald tijdsinterval die rekening houdt met de minimale tijdsduur tussen startstop en buur.

#### 4.4.1 Formule

Een connectie is nuttig als het toegevoegd wordt aan de minimaal opspannende boom door CSA. Enkel nuttige connecties worden geanalyseerd door de heuristiekmethode. Deze maakt gebruik van formule (4.1):

$$\text{prioriteit(connectie)} = \alpha * \text{connectie}_v + \beta * \text{connectie}_{\text{aankomststop}}[\text{aantal\_rechtstreekse\_stops}] + \gamma * \cos(\theta) \quad (4.1)$$

Volgende parameters worden gebruikt:

- De snelheid  $v$ , gedefinieerd als het quotiënt van afstand en tijd van een connectie. Een connectie die in korte tijd een grote afstand aflegt, heeft een grotere kans om tot de snelste route te horen.
- Het aantal rechtstreekse stops van de aankomststop van de connectie geeft een indicatie van de belangrijkheid van de stop (tabel 4.2). Het gebeurt vaak dat een snelste route niet de meest rechtstreekse weg is. Dit is vooral het geval bij treinen. Er is bijvoorbeeld een rechtstreekse trein tussen Gent en Lichervelde, maar vaak is het dat de snelste route via Brugge gaat. Brugge ligt niet op de rechtstreekse weg tussen Gent en Lichervelde dus moet er metadata zijn die de belangrijkheid van bepaalde stations aanduidt. Er moet een soort van knoophierarchie van stops komen. In tabel 4.2 zie je een overzicht

van enkele stations met hun aantal rechtstreekse stops en drukte<sup>1</sup>. Brussel is het meest centrale punt in België. Dit is zichtbaar in het aantal rechtstreekse stops én gemiddeld aantal stoptijden ten opzichte van de andere stations. De stations van Gent en Antwerpen zijn gelijkwaardig in grootte. Dit is hier ook te zien in de verhouding tussen het aantal rechtstreekse stops en gemiddeld aantal stoptijden. De laatste drie stations bestaan uit twee stations (Lichtervelde, Deinze) waar er mogelijkheid is om over testappen. Tielt is slechts een doorgangsstation. Dit weerspiegelt zich zowel in de eerste als tweede waarde. We kunnen concluderen dat het aantal rechtstreekse stops een goede parameter voor de belangrijkheid van een station is.

Station	Aantal rechtstreekse stops	Gemiddeld aantal stoptijden per dag
Brussel-Centraal	350	634
Gent-Sint-Pieters	175	309
Antwerpen-Centraal	154	467
Deinze	88	44
Lichtervelde	77	53
Tielt	39	19

Tabel 4.2: Overzicht van enkele stations van de NMBS met hun aantal rechtstreeks bereikbare stops en gemiddeld aantal stoptijden per dag.

- De cosinusgelijkheid is de cosinus van de hoek  $\theta$  tussen de connectievector en de vector van de rechtstreekse weg tussen start- en eindpunt. Dit bepaalt de mate waarin de connectie in de richting van de bestemming gaat.

Deze parameters moeten eerst genormaliseerd worden zodat elke parameter een *score* (met  $0 \leq score \leq 1$ ) heeft. Wanneer de client een fragment opgehaald heeft, wordt er een bepaald aantal connecties eerst geanalyseerd voor de normalisatie. Voor elke parameter wordt de maximale waarde binnen dat fragment bepaald. Later wordt voor elke parameter van een nuttige connectie het quotiënt genomen met de corresponderende maximale waarde. Daarna wordt deze genormaliseerde waarde vermenigvuldigd met een bepaalde gewichtsfactor  $\alpha, \beta \dots$ . De sommatie bepaalt de prioriteit van het station. De geanalyseerde connectie met bijhorende

<sup>1</sup>De drukte van een station is bepaald met de open dataset van NMBS stations: <http://github.com/irail/stations>

prioriteit wordt aan een prioriteitswachtrij toegevoegd om later in  $O(1)$ -tijd het belangrijkste station te kunnen opvragen.

Als de eindbestemming nog niet gevonden is en alle connecties gescand zijn, wordt de belangrijkste connectie uit de prioriteitswachtrij gehaald. Op basis van de aankomsttijd en aankomststop van deze connectie bepaalt de client het volgende fragment.

#### 4.4.2 Voordelen

- Het aantal mogelijke URL's blijft beperkt. Doordat er rekening gehouden wordt met een subset van stops door het maximaal aantal overstappen  $K_{max}$  en minimale routeduur  $R_{min}$  tussen startstop en buur, kan het tijdsinterval groter ingesteld worden dan bij BLCF's. Formule (4.2) toont het aantal URI's in functie van het gekozen tijdsinterval  $F$  voor de fragmenten en het aantal stops  $S$  van de dataset.

$$(24 * 60/F) * S \quad (4.2)$$

- Routes naar makkelijk bereikbare stops kunnen met één request gevonden worden. Een bestemming is makkelijk bereikbaar wanneer er frequent routes tussen deze bestemming en startstop zijn en/of weinig overstappen vereist zijn. De kans dat de snelste route in het eerste fragment valt, wordt groter naarmate  $T$  en  $K_{max}$  groter gekozen worden.

#### 4.4.3 Nadelen

- Bij meerdere requests wordt er minder efficiënt omgegaan met connecties dan gewenst: een groot deel van de connecties van de volgende vertrekstop bevinden zich geografische in de richting vanwaar vertrokken werd en zijn dus niet nuttig.
- Een NLCF introduceert een trade-off tussen het maximaal aantal overstappen  $K_{max}$  van een buur en het tijdsinterval  $T$  waarbinnen connecties worden teruggeven. Alle connecties die hieraan voldoen behoren tot hetzelfde fragment. Voor query's met als bestemming een makkelijk bereikbare stop is het beter dat de fragmentgrootte niet te groot is. Voor query's naar moeilijk bereikbare stops is het beter dat dit groter is, want hoe minder request gestuurd moeten worden, hoe sneller.
- Het werken met een heuristiek zorgt voor een suboptimale oplossing. Wanneer geen route gevonden is met het eerste fragment, moet een volgende vertrekstop gekozen worden volgens de prioriteit. Connecties worden teruggegeven rekening houdend met de minimale

routeduur vanuit de nieuwe vertrekstop. Indien een verkeerde stop gekozen is en de bestemming wordt bereikt, is het resultaat suboptimaal.

De heuristiekmethode geeft niet altijd het gewenste resultaat terug wanneer een verkeerde volgende vertrekstop gekozen wordt. Volgende techniek (sectie 4.5) garandeert de optimale oplossing door gebruik te maken van een combinatie van NLCF's en BLCF's.

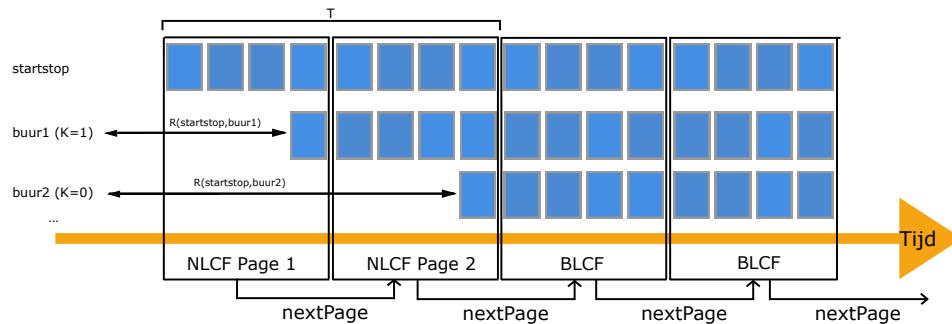
## 4.5 Speedup

De *speedup*-techniek maakt enkel in het begin gebruik van één NLCF. Indien het resultaat nog niet gevonden is, wordt verder gezocht met basic LCF's. Om te kunnen garanderen dat de snelste route binnen deze connecties ligt, wordt slechts één NLCF gebruikt die voldoet aan volgende voorwaarde:

- $K_{max} = \infty \Rightarrow$  er wordt rekening gehouden met alle stops van de dataset.

Er wordt een tijdsinterval  $T$  gekozen waarbinnen rekening gehouden wordt met de minimale routeduur tussen de startstop en een buur  $R(startstop, buur)$ . Indien de snelste route nog niet gevonden is binnen  $T$  wordt er overgeschakeld naar BLCF. Figuur 4.3 toont een overzicht van deze speedup-techniek.

Doordat het eerste NLCF procentueel meer nuttige connecties bevat (tabel 5.6), moeten er minder connecties gescand worden binnen  $T$  waardoor een versnelling (Engels: *speedup*) in berekeningstijd merkbaar is.



Figuur 4.3: Overzicht van resources van speedup-optimalisatietechniek.

#### 4.5.1 Hypermedia

De originele implementatie (subsectie 3.1.1) maakt gebruik van *hydra:nextPage* links om volgende fragmenten te vinden. Dit principe kunnen we toepassen op een NLCF. Op deze manier kunnen we een grotere  $T$  waarde instellen zonder dat kleinere routes onnodig veel connecties moeten downloaden. Uit de resultaten van basic Linked Connections Fragments (subsectie 5.1.2) blijkt dat queries gemiddeld het snelst opgelost geraken als de pagina's een grootte hebben van ongeveer 1000 connecties. Subsectie 4.5.1 toont hoe deze nieuwe interface ook een pagina-nummer heeft om het NLCF op te splitsen.

```
|| http://example.org?departureTime={...}&departureStop={...}&page={...}
```

Lijst 4.4: De interface van de speedup-techniek bestaat uit drie parameters .

Met deze techniek worden de nadelen (subsectie 4.4.3) van de heuristische techniek weggetrokken:

- Er is een grotere efficiëntie, omdat een groter tijdsinterval  $T$  gekozen kan worden door het pagineren van het NLCF.
- De optimale oplossing wordt gevonden. De NLCF geeft een gefilterde lijst van connecties voor alle mogelijke buren (dankzij  $K_{max}$ ) terug die gesorteerd zijn volgens vertrektijd. Alle mogelijke connecties met vertrektijden binnen de query vertrektijd en  $T$  worden beschouwd. Als de snelste route hierin ligt, zal deze gevonden worden. Als de snelste route nog niet gevonden is, wordt simpelweg verdergezocht met basic LCF's met vertrektijd na  $T$ . Deze garandeert ook een optimale oplossing.
- Connecties worden slechts een keer opgestuurd. Bij de heuristische techniek was het mogelijk dat een connectie meerdere keren teruggegeven werd door het kiezen van een nieuwe vertrekstop.

## Hoofdstuk 5

# Resultaten

In dit hoofdstuk bespreken we verschillende testen die een antwoord zullen bieden op de onderzoeksraag en hypotheses. In sectie 5.1 worden eerst enkele praktische zaken besproken. Daarna worden de drie besproken technieken uit sectie 3.1, sectie 4.4 en sectie 4.5 onderling vergeleken.

### 5.1 Opstelling

Deze masterproef maakt gebruik van de officiële GTFS dataset van de Belgische spoorwegemaatschappij NMBS.

Zowel de client als de server werden getest op dezelfde machine: een Macbook Pro 2015 editie met 8 GB RAM en Intel Core i5 2,7 Ghz processor. De client is een webapplicatie waarin benchmark-queries kunnen lopen met behulp van Javascript. Als browser werd Firefox gebruikt. In appendix B is een screenshot zichtbaar van de applicatie. Hierin is het mogelijk om twee stations en de gewenste techniek (enkel BLCF's, heuristiek en speedup) te selecteren. De vertrekdatum en -tijd staan vast op 1 december 2015 om 7u. Tijdens het berekenen van een Earliest Arrival Time-query zijn er enkele statistieken zichtbaar voor de gebruiker: het aantal gescande connecties, nuttige connecties en HTTP requests en querytijd in seonden. Onder deze statistieken staat een kaart waarop nuttige connecties weergegeven worden. Nadat de query is opgelost, komt er een overzicht van gebruikte connecties voor de snelste route zichtbaar.

### 5.1.1 Lokale server versus productieserver

Doordat de testen werden uitgevoerd in een lokale omgeving zullen we testen of er een merkbaar verschil in snelheid is met een server in productie<sup>1</sup>. Er werden een honderdtal queries op de dataset van de NMBS uitgevoerd. Beide servers hebben als tijdsinterval tien minuten.

Type	Aantal queries	Querytijd (s)	Requests	Querytijd/request (s)	Connecties
lokaal	30	103.196	583	0.17	10368
productie	30	68.5509	550	0.12	9915

Tabel 5.1: Vergelijking tussen een lokale server en een productieserver.

Figuur tabel 5.1 toont aan dat er een verschil in snelheid is tussen een lokale en een productieserver. De servers bevatten connecties van verschillende dagen waardoor de queries dezelfde start- en eindstop en vertrektijd hebben, maar verschillende vertrekdatum. Dit is de reden waarom het aantal requests en connecties niet gelijk zijn. Het resultaat toont aan dat lokale server 30% trager dan een productieserver is. Volgende subsectie toont aan welke fragmentgrootte gemiddeld het snelst is.

### 5.1.2 Fragmentgrootte

*Basic Linked Connections Fragments* (BLCF's) hebben een vast tijdsinterval waarin connecties worden teruggegeven. Dit komt overeen met een gemiddeld aantal connecties per fragment. Tabel 5.2 toont een overzicht van verschillende fragmentgroottes die mogelijk zijn met het gemiddeld aantal connecties die hiermee overeenkomt.

Om te bepalen wat de optimale fragmentgrootte is, werden honderd queries uitgevoerd met telkens een andere fragmentgrootte. De resultaten in figuur 5.1 tonen aan dat een tijdsinterval tussen 10 en 30 minuten de gemiddeld beste querysnelheid geeft. Dit komt overeen met een paginagrootte van 500 tot 2000 connecties. Figuur 5.1b toont het verband tussen het aantal gedownloade fragmenten en het ingestelde tijdsinterval.

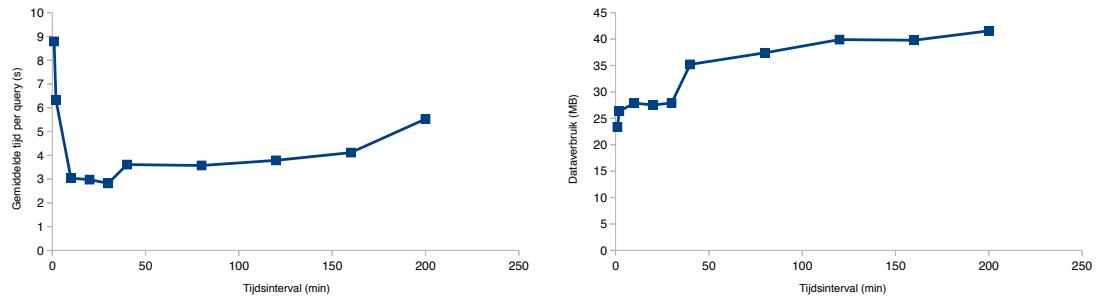
De grootte van een fragment is afhankelijk van het aantal aanwezige connecties. Zo is er een gemiddelde vaste grootte per connectie (tabel 5.3). Deze zal later gebruikt worden als referentie-waarde.

Uit figuur 5.1 kan ook afgeleid worden dat de snelheid van routeplanning afhankelijk is

<sup>1</sup>Linked Connections server te vinden op: <http://belgianrail.linkedconnections.org/connections/>

t (min)	NMBS	
	Grootte (MB)	Connecties
1	0.012	56
2	0.027	115
5	0.069	264
10	0.14	570
20	0.27	1144
30	0.41	1704
40	0.55	2281
80	1.10	4564
120	1.90	7356
160	2.21	9332
200	2.77	11419

Tabel 5.2: Grootte (MB) en aantal connecties afhankelijk van tijdsinterval van basic LCF's.



- (a) De gemiddelde tijd (s) om queries uit te voeren in(b) De hoeveelheid opgevraagde data (MB) in functie van  
functie van het tijdsinterval van basic LCF's. het tijdsinterval (min) van basic LCF's.

Figuur 5.1: Snelheid en grootte hangen af van de ingestelde tijdsinterval van basic Linked Connections Fragments.

	Kilobyte	Megabyte
Grootte connectie	0,26158	0,00026158

Tabel 5.3: Grootte van één connectie.

van de combinatie van het aantal connecties en het aantal requests. Voor korte routes zijn kleinere fragmentgroottes beter. Lange routes zijn veel sneller met minder requests en een groot tijdsinterval. Om geen groot contrast te hebben tussen korte en lange routes, moet er dus een gulden middenweg genomen worden van rond de 1500 connecties per fragment. Hierbij moet er ook zuinig omgegaan worden met data. Grote overschotten van connecties die niet gescand moeten worden, zijn te vermijden.

Voortaan zullen we als referentie pagina's van ongeveer 1000 connecties nemen. In tabel 5.4 staat een overzicht van de verschillende paginagroottes voor de drie technieken die getest zijn. Het aantal mogelijke overstappen  $K$  is gemaximaliseerd zodat elke stop bereikbaar is vanuit een bepaalde vertrekstop. Voor de NMBS is  $K_{max} = 5$ .

Techniek	BLCF (min)	NLCF (min)	Fragmentgrootte (min)
Basic LCF's	20	-	-
Speedup	20	30	120
Heuristiek	-	100	100

Tabel 5.4: Tijdsintervallen (min) voor de verschillende technieken.

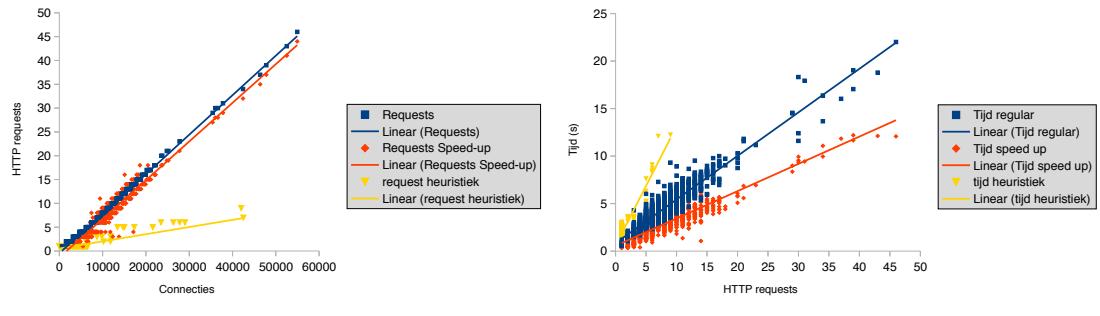
De reden waarom het tijdsinterval van NLCF's beperkt blijft tot 100-120 minuten is omdat dit kostelijker is voor de server om te berekenen. Zo moet er een extra databank-request gestuurd worden om de buren op te vragen van een stop. Na 120 minuten zijn de meeste stopplaatsen bereikbaar voor de NMBS waardoor NLCF pagina's niet meer opwegen in snelheid.

## 5.2 Hypotheses

In deze sectie wordt volgende hypothese aangekaart (sectie 1.3):

- *Hoe meer connecties gescand moeten worden, hoe meer bandbreedte dit vereist. Bijgevolg kost een route berekenen meer tijd.*

Figuur 5.2a toont aan dat het aantal requests lineair afhankelijk is met het aantal connecties. Ook figuur 5.2b toont een lineair verband tussen de tijd en het aantal requests. Voordat connecties effectief opgevraagd worden, stuurt de client een request om de juiste context op te vragen. Enkel requests met connecties zijn in rekening gebracht. Door de kleine grootte (1495 kB) en het feit dat dit bij alle technieken wordt gedaan, worden deze requests verwaarloosd.



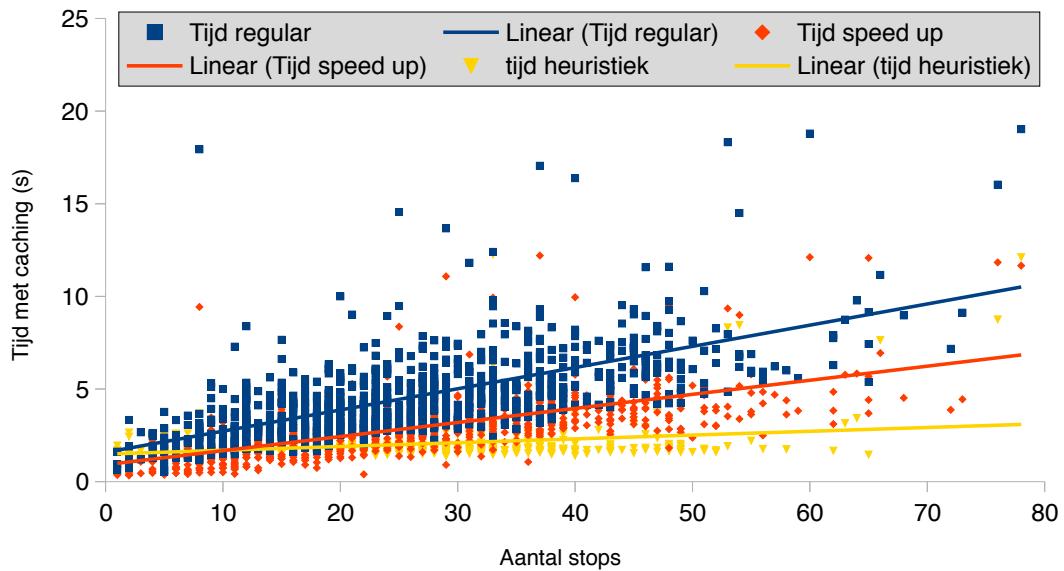
(a) HTTP requests in functie van aantal connecties. (b) Verband tussen aantal HTTP requests en tijd om route te berekenen met client-side caching.

Zowel Figuur 5.2a als figuur 5.2b) bevestigen de hypothese dat hoe meer connecties nodig zijn, hoe meer bandbreedte vereist wordt. Bijgevolg is er meer tijd nodig om een route te berekenen. Volgende paragraaf beantwoordt de tweede hypothese die in de inleiding aan bod kwam (sectie 1.3):

- *Het toevoegen van een extra filter om enkel nuttige connecties op te vragen zal client-side routeplanning minstens dubbel zo snel maken.*

Om de snelheid te meten van de verschillende technieken werden er 900 query's uitgevoerd. Een script selecteerde aanvankelijk 1000 query's door random telkens twee stops te selecteren. Elke query heeft dezelfde dag en tijdstip van vertrek (2015-12-01T07:00:00Z). Uiteindelijk zijn er 100 query's moeten wegvalLEN door softwarefouten. Momenteel stopt de client met zoeken wanneer geen connecties meer worden teruggegeven door de server. Bij slecht bereikbare, voornamelijk buitenlandse, stations komt dit frequent voor. Dit is te wijten aan het statisch toewijzen van fragmenten aan een bepaald tijdsinterval. In sectie 6.2 wordt dit verder besproken. Dit is vooral een probleem bij de speedup-techniek, omdat deze rekening houdt met de minimale tijdsduur. Bij buitenlandse stations is de kans op een lege eerste pagina veel groter. Wegens tijdsgebrek zijn deze query's manueel gefilterd op fouten om zo correct mogelijk de drie technieken met elkaar te kunnen vergelijken. Dit verklaart waarom slechts 0,2% van alle mogelijke routes getest werd. Dit geeft wel een representatief beeld voor routes tussen Belgische stations van de NMBS.

Op figuur 5.2 is te zien hoe sterk de verschillende technieken schalen in snelheid ten opzichte van het aantal stops. De snelste techniek is die met de heuristiek. Daarna volgt de speedup en de oorspronkelijke techniek met basic LCF's. Bij de heuristische techniek zijn de queries waarvoor



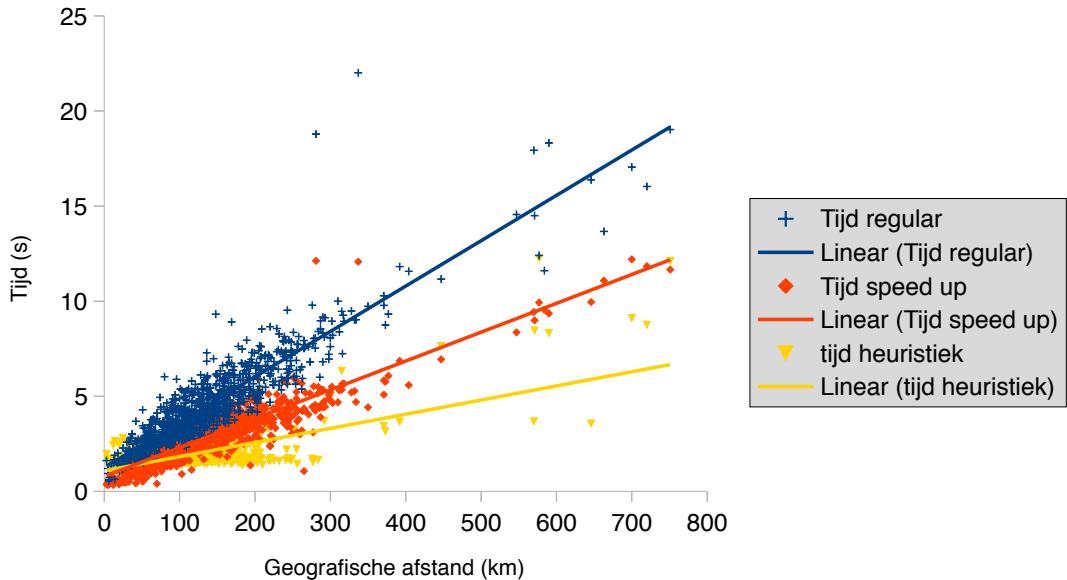
Figuur 5.2: De tijd om een route te berekenen in functie van het aantal stops voor de NMBS.

geen optimale route gevonden werden, weggelaten. Dit zijn queries waarvoor meer dan 20 requests gestuurd zijn zonder enig resultaat. Meestal zijn dit queries van of naar buitenlandse stations door het weinige verkeer hiernaartoe. Deze moeilijkere queries brengen de grootste tijd met zich mee dus de werkelijke snelheid ligt gemiddeld wat hoger. Voor 78 % van de queries is het gelukt om de optimale oplossing te vinden met de heuristische methode. De huidige implementatie die enkel basic LCF's gebruikt is duidelijk de traagste. De speedup-techniek is zoals verwacht iets sneller. Bij uitschieters is er een verschil van enkele seconden te merken tussen originele en de speedup-techniek. In tabel 5.5 staat een overzicht van de gemiddelde snelheid per query. Enkel de heuristiekmethode verdubbelt de snelheid ten opzichte van de originele methode. De speedup maakt routeplanning 37% sneller.

	BLCF's	Speedup	Heuristiek
Gemiddelde snelheid per query (s)	4.54	2.87	1.96

Tabel 5.5: Overzicht van de gemiddelde snelheid per query voor de methode met enkel BLCF's en de speedup en heuristiek methodes.

Figuur 5.3 toont aan dat er een lineair verband bestaat tussen de de querytijd en afstand tussen begin- en eindpunt.



Figuur 5.3: Verband tussen de geografische afstand tussen begin-en eindpunt en de tijd om de snelste route te berekenen.

De snelheid bij client-side routeplanning hangt af van zowel het aantal gedownloade connecties als het aantal requests. De speedup-methode stuurt slechts enkele requests minder dan de originele methode, maar het verschil in snelheid is merkbaar. De heuristiek-methode is voor langere routes het snelst doordat slechts één request gestuurd moet worden.

### 5.2.1 Efficiëntie

CSA gebruikt maar een beperkt aantal van de gescande connecties om een minimaal omspannende boom te berekenen. De gegevens uit tabel 5.6 tonen aan dat de speedup-techniek de efficiëntie van het aantal nuttige connecties verdubbelt in vergelijking met de originele implementatie. Merk hier ook op dat de heuristiek enkel rekening houdt met 80% van de queries, die relatief makkelijker op te lossen waren, en dus een hogere score bekomt dan de andere twee technieken.

	BLCF's	Speedup	Heuristiek
Nuttige connecties (%)	0.026	0.047	0.064

Tabel 5.6: Percentage nuttige connecties ten op zichte van alle gescande connecties voor elke techniek.

### 5.2.2 Dataverbruik

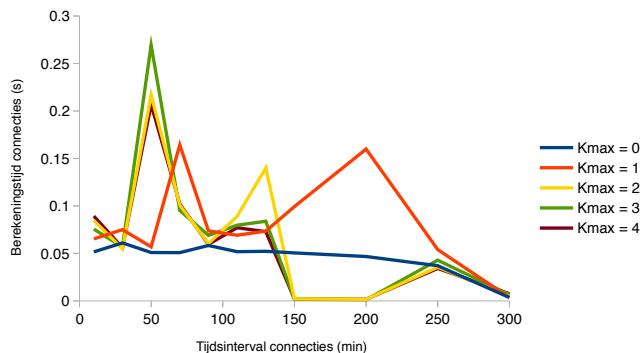
Naast efficiëntie is ook dataverbruik een belangrijk aspect bij routeplanning. Tabel 5.7 geeft een overzicht hoeveel data er gemiddeld verbruikt wordt met hun onderlinge percentages.

	BLCF's	Speedup	Heuristiek
Dataverbruik (MB)	2.69	1.68	1.97
Percentage	100	62	73

Tabel 5.7: Percentage dataverbruik voor elke techniek.

### 5.2.3 Serverbelasting

In deze laatste test onderzoeken we de tijd die de server nodig heeft om connecties op te halen uit een databank. Hiervoor hebben we voor elke mogelijke  $K_{max}$  en verschillende fragmentgroottes  $F$  de gemiddelde tijd berekend om connecties op te halen. Zoals je kan zien in figuur 5.4 heeft de server minder last als er meer connecties moeten teruggegeven worden. Hoe groter  $K_{max}$  en  $F$ , hoe meer connecties voldoen en hoe sneller antwoord kan teruggegeven worden.



Figuur 5.4: Tijd die server nodig heeft om connecties terug te geven in functie van tijdsintervallen van fragmenten. Dit werd voor verschillende maximum aantal overstappen  $K_{max}$  getest.

## 5.3 Overzicht

Tabel 5.8 toont een overzicht van de snelheid en dataverbruik om één query gemiddeld te berekenen. Ook staat er nog eens vermeld dat de heuristische methode een suboptimale oplossing berekent.

Methode	Snelheid (s)	Dataverbruik (MB)	Optimaal
BLCF's	4.54	2.69	ja
Speedup	2.87	1.68	ja
Heuristiek	1.96	1.97	nee

Tabel 5.8: Overzicht van de resultaten om één query te berekenen per methode.

## Hoofdstuk 6

# Conclusie en future work

Als slot overlopen we eerst de belangrijkste realisaties van deze masterproef. Daarna overlopen we de hypotheses waarop we aan de hand van de resultaten een antwoord gevonden hebben.

### 6.1 Conclusie

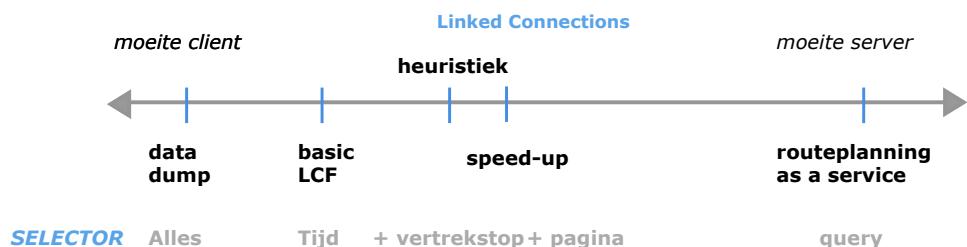
Deze masterproef heeft onderzocht of de huidige implementatie van Linked Connections sneller kan door het toevoegen van extra filters. Er werden twee soorten Linked Connections Fragments gedefinieerd: basic Linked Connections Fragments (BLCF's) en neighbour Linked Connections Fragments (NLCF's). Het eerste type bevat connecties die enkel gefilterd zijn volgens een bepaald tijdsinterval. Dit zorgt ervoor dat de client overbodig veel data moet downloaden. Daarom werd er een tweede type toegevoegd zodat connecties ook gefilterd kunnen worden op basis van de vertrekstop van de query. Hiervoor is een extra voorbewerkingsfase noodzakelijk die afhankelijk van de grootte van de dataset, weinig tijd vergt. Deze voorbewerking berekent de minimale tijd en het minimaal aantal overstappen tussen elk paar stopplaatsen. Het toevoegen van extra functionaliteit aan de server maakt HTTP-caching nog steeds mogelijk. Hierdoor blijft de server low-cost.

Er werden twee client-side algoritmes ontwikkeld die van NLCF's gebruik maken: een heuristische methode en een speedup. Deze algoritmes lossen het *Earliest Arrival Time*-probleem op. De eerste maakt enkel gebruik van NLCF's. Met een heuristiek krijgt elke bereikbare stop een score op basis van de belangrijkheid van het station, de tijd om die te bereiken ... Wanneer het resultaat nog niet gevonden is, kiest de client de stop met de hoogste score om het volgende NLCF te bepalen. De speedup-methode gebruikt een combinatie van de twee soorten fragmen-

ten om de optimale oplossing te kunnen garanderen. Bij elke vertrekstop hoort één NLCF die gepagineerd wordt met behulp van hypermedia. Op deze manier blijven pagina's beperkt in grootte en moet de client enkel data downloaden die noodzakelijk is.

Om de methodes onderling te vergelijken werd een webapplicatie geprogrammeerd. De query-snelheid van de heuristiek-methode is dubbel zo snel als de originele methode, maar bevat slechts voor 78% van de geteste queries de optimale oplossing. Bij de speedup-methode werd een versnelling van 37% waargenomen terwijl 38% minder data werd verbruikt ten opzichte van de originele methode.

Figuur 6.1 geeft een overzicht van de Linked Data Fragments-as met daarop de verschillende manieren om routes te plannen. Een basic LCF is het meest low-cost voor de server, maar vergt meer moeite van de client. De twee optimalisaties maken gebruik van extra filtering waardoor meer inspanning van de server geëist wordt en zich dus rechts van basic LCF bevinden. De speedup-methode maakt gebruik van drie HTTP-parameters waardoor deze minder cachebaar is. Bij overbelasting van de server kan de grootte van de pagina's en fragmenten desnoods dynamisch aangepast worden.



Figuur 6.1: Linked Data Fragmenten-as met de heuristiek- en speedup-methodes aan toegevoegd.

Naast het opzetten van een convertor die een GTFS feed omzet naar connecties is er ook een merger geïmplementeerd. Deze zorgt ervoor dat connecties van verschillende Linked Connections servers kunnen opgehaald worden door de client. Er kunnen dynamische connectiestromen toegevoegd of verwijderd worden.

Deze masterproef heeft clients-side routeplanning met 37% sneller gemaakt met behulp van de speedup-methode. Hoewel dit duidelijk merkbaar is bij de client is hiermee niet voldaan aan de

hypothese die stelt dat de snelheid meer dan verdubbelt met behulp van extra filtering. Ondanks een extra voorbewerkingsfase weegt het extra voordeel van een databesparing van 38% op voor de client om deze acceleratiemethode te implementeren in Linked Connections.

## 6.2 Future work

In deze laatste sectie bespreken we enkele mogelijke uitbreidingen van Linked Connections. Hierbij ligt de nadruk op intermodaliteit en het gebruiken van realtime informatie.

### Intermodaliteit

Een Linked Connections server is verantwoordelijk voor het aanbieden van connecties van een (of meerdere) provider(s). De client moet zelf kunnen beslissen van welke providers data nuttig kan zijn. Een belangrijk aspect hierbij is metadata over stops. Momenteel zijn er geen afspraken over het gebruiken van vaste identifiers voor stops. Enkel hiermee kan er interoperabiliteit ontstaan tussen datasets. Een client moet weten of de identifier van de NMBS voor Antwerpen-Centraal dezelfde is als die van de Nederlandse Spoorwegen. Ook moet er geweten zijn welke operatoren en welke soort vervoersmiddelen actief zijn in bepaalde stations of met *footpaths* bereikbaar zijn.

Een volgende stap zou het combineren zijn van bussen (De Lijn) en treinen (NMBS) die actief zijn binnen dezelfde regio. Er moet gemodelleerd worden wanneer het nuttig is om bepaalde modaliteiten te gebruiken. Het is bijvoorbeeld logisch om bij grote afstanden eerst de trein te nemen en daarna de bus, desnoods met een bus naar het vertrekstation van de trein. Hieruit blijkt dat de vraag naar metadata steeds groter zal zijn. Enkele voorbeelden zijn:

- faciliteiten zoals rolstoeltoegankelijkheid,
- locatie van perrons,
- mogelijke wandelafstanden,
- gebied waar de publieke vervoersmaatschappij actief is, bijvoorbeeld in GeoJSON-formaat,
- welke modi beschikbaar zijn.

Op basis van deze metadata kan een client slimme beslissingen nemen om connecties op te halen van de juiste providers. Met behulp van de *merger* die in subsectie 3.3.1 besproken werd, kunnen verschillende connectiestromen makkelijk gesorteerd samengevoegd worden voor CSA.

### Dynamisch fragmenten

Momenteel wordt het tijdsinterval van basic LCF's statisch bepaald door vaste tijdsintervallen. Dit gaf bij het testen van sommige query's problemen (Sectie 5.2). Er moet een systeem komen die bepaalt hoe fragmenten het best verdeeld worden om lege te vermijden. Dit kan opgelost worden door deze intervallen vooraf te berekenen.

### Realtime-informatie

Momenteel werkt Linked Connections enkel met statische data. GTFS-RT stelt realtime-informatie afzonderlijk van de GTFS data zelf beschikbaar. Updates betekenen in feite een herordening van connecties. Een mogelijkheid is om aan de statische connecties data toe te voegen over de realtime-status. Dit kan zowel door de client als de server gedaan worden. CSA zou dan moeten uitgebreid worden waardoor zowel de statische als de realtime snelste route kan berekend worden.

## Bijlage A

## Code

Alle broncode is geplaatst op Github als *open source*. Sommige implementaties bevinden zich op aparte *branches*. Er kan veranderd worden van branch door links naast de groene knop ‘new pull request’ te klikken. De broncode kan gedownload worden als ZIP-bestand door op de knop ‘Download ZIP’ te klikken.

Volgende repositories (met als URL-prefix: <https://github.com>) werden gebruikt:

- **/brechtvdv/gtfs2connections** bevat de scripts die een GTFS-feed in een MySQL-databank laadt en hier vervolgens connecties uit genereert.
- **/brechtvdv/server.js** bevat code voor een Linked Connections server. De NLCF voor de heuristiek is geïmplementeerd op de ‘feature-optimization’-branch. De speedup-techniek is geïmplementeerd op de ‘speed-up’-branch.
- **/brechtvdv/client.js** bevat de code voor een Linked Connections client met gelijknamige branches zoals de servercode.
- **/brechtvdv/gtfs-connectionstops** bevat code om queries te genereren, een mapper die voor interoperabiliteit tussen stops uit verschillende datasets zorgt en het script die buren voor elke stop berekent.
- **/brechtvdv/thesis** bevat naast de L<sup>A</sup>T<sub>E</sub>X-bestanden van de masterproef en het extended abstract ook de demo-webapplicatie en resultaten in *OpenDocument Spreadsheet*-formaat.
- **/brechtvdv/dockerize-lc** bevat code om een Linked Connections server te laten lopen in een Docker-container. Dit werd uiteindelijk niet gebruikt voor deze masterproef, maar kan geïnteresseerden helpen in het opzetten van verschillende LC servers op één systeem.

- **/linkedconnections/csa.js** bevat naast code voor het Connection Scan Algorithm ook de merger die verschillende connectiestromen samenvoegt.

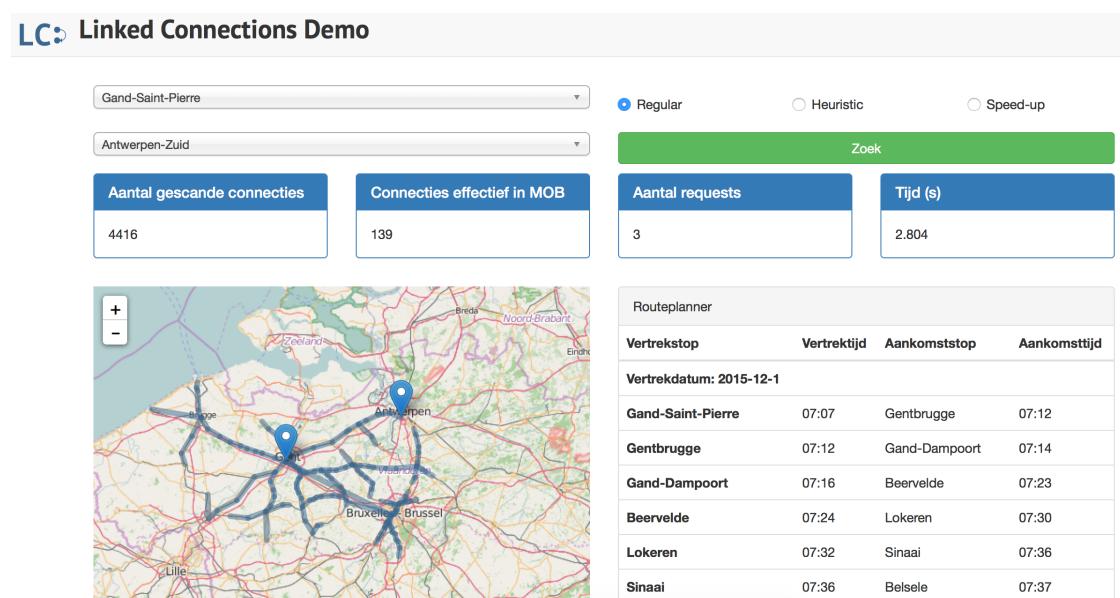
## Bijlage B

# Demo

Figuur B.1 toont hoe de webapplicatie eruit ziet na het berekenen van de *Earliest Arrival Time*-query tussen ‘Gent-Sint-Pieters’ en ‘Antwerpen-Zuid’ op 1 december 2015 om 7u ’s ochtends. Naast enkele statistieken is er ook een visualisatie van de minimaal opspannende boom van connecties op een kaart en een routeplanningsadvies. Tabel B.1 toont een overzicht van de weergegeven statistieken voor elke techniek die deze query oplost.

Methode	Scan	MOB	Requests	Tijd (s)
BLCF’s	4416	139	3	2.8
Heuristiek	953	135	1	2.3
Speedup	953	135	2	1.4

Tabel B.1: Enkele statistieken voor het uitvoeren van de Earliest Arrival Time-query tussen ‘Gent-Sint-Pieters’ en ‘Antwerpen-Zuid’ op 1 december 2015 om 7u ’s ochtends.



Figuur B.1: Screenshot van de webapplicatie.

# Bibliografie

- [1] Hannah Bast, Matthias Hertel, and Sabine Storandt. *Scalable Transfer Patterns*, chapter 1, pages 15–29.
- [2] Pieter Colpaert, Alejandro Llaves, Ruben Verborgh, Oscar Corcho, Erik Mannens, and Rik Van de Walle. Intermodal public transit routing using Linked Connections. In *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, October 2015.
- [3] Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public transit labeling. *CoRR*, abs/1505.01446, 2015.
- [4] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015.
- [5] Google Developers. *GTFS referentie*. Google, 2015.
- [6] Ben Strasser and Dorothea Wagner. Connection scan accelerated. In *Proceedings of the Meeting on Algorithm Engineering & Experiments*, pages 125–137, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.
- [7] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, April 2014.

# Lijst van figuren

1.1	Linked Data Fragments-as: huidige oplossingen om routes te plannen. client beschikken ofwel over alle data in een dump, ofwel over een service die het route-plannen voor zich neemt. . . . .	3
1.2	Linked Connections bieden nieuwe trade-offs tussen client en server aan. . . . .	3
1.3	Voorbeeld van een connectie. Deze bestaat uit een vertrek- en aankomstplaats, resp. met vertrek- en aankomsttijd . . . . .	4
2.1	RDF representatie van een triple . . . . .	9
2.2	Triple met URI's . . . . .	10
2.3	Linked Data Fragments-as . . . . .	13
2.4	Linked Data Fragments-as met Triple Pattern Fragments. . . . .	14
3.1	Connecties zijn onderverdeeld in fragmenten volgens een bepaald tijdsinterval, zogenaamde basic Linked Connections Fragmenten (BLCF). Met hydra:nextPage links kan makkelijk het volgende fragment gevonden worden. . . . .	23
3.2	Overzicht hoe connecties worden gegenereerd uit een GTFS feed. Deze wordt ingeladen in een MySQL databank. Met de scripttaal PHP worden connecties gegenereerd. . . . .	25
3.3	Per dag worden de corresponderende service ID's, trip ID's en route ID's opgehaald. De vertrek-/aankomststopplaats met respectievelijk vertrek-/aankomsttijd van connecties worden berekend uit een verzameling stoptijden. . . . .	25
3.4	Opstelling van een client en twee Linked Connections servers die elk verantwoordelijk zijn voor een modi. . . . .	30
3.5	Overzicht merger . . . . .	31

4.1	Visualisatie van CSA. De minimaal opspannende boom kan voorgesteld worden als een groeiende cirkel rondom het vertrekpunt. . . . .	34
4.2	Neighbour Linked Connections Fragment van heuristiek-methode bevat connecties van startstop en buren binnen een bepaald tijdsinterval die rekening houdt met de minimale tijdsduur tussen startstop en buur. . . . .	40
4.3	Overzicht van resources van speedup-optimalisatietechniek. . . . .	43
5.1	Snelheid en grootte hangen af van de ingestelde tijdsinterval van basic Linked Connections Fragments. . . . .	47
5.2	De tijd om een route te berekenen in functie van het aantal stops voor de NMBS. . . . .	50
5.3	Verband tussen de geografische afstand tussen begin-en eindpunt en de tijd om de snelste route te berekenen. . . . .	51
5.4	Tijd die server nodig heeft om connecties terug te geven in functie van tijdsintervallen van fragmenten. Dit werd voor verschillende maximum aantal overstappen $K_{max}$ getest. . . . .	52
6.1	Linked Data Fragmenten-as met de heuristiek- en speedup-methodes aan toegevoegd. . . . .	55
B.1	Screenshot van de webapplicatie. . . . .	61

# Lijst van tabellen

3.1	Tijd om een GTFS feed in te laden in een MySQL databank. . . . .	27
3.2	Tijd om connecties te genereren voor één dag. . . . .	28
4.1	Tijd om buren van alle GTFS stops te berekenen. . . . .	37
4.2	Overzicht van enkele stations van de NMBS met hun aantal rechtstreeks bereikbare stops en gemiddeld aantal stoptijden per dag. . . . .	41
5.1	Vergelijking tussen een lokale server een productieserver. . . . .	46
5.2	Grootte (MB) en aantal connecties afhankelijk van tijdsinterval van basic LCF's. .	47
5.3	Grootte van één connectie. . . . .	47
5.4	Tijdsintervallen (min) voor de verschillende technieken. . . . .	48
5.5	Overzicht van de gemiddelde snelheid per query voor de methode met enkel BLCF's en de speedup en heuristiek methodes. . . . .	50
5.6	Percentage nuttige connecties ten op zichte van alle gescande connecties voor elke techniek. . . . .	51
5.7	Percentage dataverbruik voor elke techniek. . . . .	52
5.8	Overzicht van de resultaten om één query te berekenen per methode. . . . .	53
B.1	Enkele statistieken voor het uitvoeren van de Earliest Arrival Time-query tussen ‘Gent-Sint-Pieters’ en ‘Antwerpen-Zuid’ op 1 december 2015 om 7u ’s ochtends. .	60