Voorwoord

Hier komt wat tekst.

Brecht Van de Vyvere, januari 2016

Toelating tot bruikleen

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: dr. ir. Ruben Verborgh, Pieter Colpaert

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

Routeplanning is niet meer weg te denken uit ons dagelijks leven. Is het nu voor de trein naar het werk te nemen of het vliegtuig naar je vakantiebestemming, de mogelijkheden zijn onbeperkt. Sinds enkele jaren wordt transportdata gepubliceerd volgens General Transit Feed Specification (GTFS). Dankzij deze uniforme structuur kunnen er algoritmes bedacht worden om data te combineren en intermodaliteit toe te laten. Bestaande oplossingen maken gebruik van een webservice die zoveel mogelijk vragen van de gebruiker probeert te beantwoorden, maar deze manier is moeilijk uitbreidbaar. *Linked Connections* biedt hier een antwoord op door routeplanning mogelijk te maken op basis van gepubliceerde data. Doordat de server enkel verantwoordelijk is voor het publiceren van connecties is deze makkelijk uitbreidbaar via hypermedia. De cliënt berekent zelf welke data nodig is om een om een bepaalde route te kunnen plannen rekening houdend met de eisen van de gebruiker. De huidige implementatie van Linked Connections laat enkel filtering in de tijd toe waardoor de snelheid van het algoritme aan banden ligt. Deze masterproef introduceert een optimalisatie voor routeplanning met Linked Connections.

Trefwoorden

Linked Connections, routeplanning, optimalisatie, GTFS

Inhoudsopgave

1 Inleiding **1**

1.1 Probleemstelling 2

1.2 Onderzoeksvraag 4

1.3 Hypotheses 4

1.4 Oriëntatie 5

2 Literatuurstudie **6**

2.1 Dienstregeling openbaar vervoer 6

2.1.1 GTFS 6

2.1.2 GTFS-RT 8

2.2 Semantisch web 8

2.2.1 RDF 8

2.2.2 Bronidentificatie 9

2.2.3 Vocabularium 9

2.2.4 JSON-LD 10

2.2.5 Benoemde grafen 11

2.2.6 SPARQL 12

2.2.7 Linked Data Fragments 13

2.2.8 Triple Pattern Fragments 14

2.2.9 Analogie met routeplanning 15

2.3 REST 16

2.3.1 Beperkingen 16

2.4 Routeplanning algoritmen 17

2.4.1 Dijkstra 18

2.4.2 Connection Scan Algorithm 19

3 Linked Connections **21**

3.1 Principe . . . 21

3.1.1 Technologieën . . . 22

3.1.2 Basic Linked Connections Fragments . . . 23

3.2 Convertor naar connecties . . . 24

3.2.1 Connectiescript . . . 24

3.2.2 JSON-LD stream . . . 27

3.3 Cliënt . . . 28

3.3.1 Merger . . . 29

3.4 Conclusie . . . 31

4 Optimalisatie **32**

4.1 Geofiltering . . . 32

4.1.1 Geoparameter . . . 32

4.1.2 Criteria . . . 33

4.1.3 Direct bereikbare stops . . . 34

4.1.4 Buren . . . 34

4.2 Neighbour Linked Connections Fragment . . . 35

4.3 Heuristische techniek . . . 38

4.3.1 Parameters . . . 38

4.3.2 Formule . . . 39

4.3.3 Voordeel . . . 40

4.3.4 Nadelen . . . 40

4.4 Speed-up techniek . . . 40

4.4.1 Hypermedia . . . 41

5 Resultaten **43**

5.1 Opstelling . . . 43

5.1.1 Lokale server versus productieserver . . . 43

5.1.2 Fragmentgrootte . . . 44

5.2 Hypotheses . . . 46

5.2.1 Snelheid . . . 47

5.2.2 Efficiëntie . . . 49

5.2.3 Dataverbruik 49
5.2.4 Serverbelasting 50
5.3 Conclusie 50

6 Conclusies en perspectieven **51**
6.1 Conclusie 51

7 Future work **54**
7.1 Preprocessing 54
7.2 Multicriteria queries 54
7.3 Intermodaliteit 55
7.4 Grote netwerken 56
7.4.1 Realtime informatie 56

Woordenlijst **59**

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 de 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 twee probleem hierbij is dat deze data geen Open Data is. Sinds augustus 2015 is de Belgische overheid öpen-by-default". Dit houdt in dat alle gegevens van de overheid publiek moeten zijn, tenzij explicitiet verklaard wordt waarom deze niet open kan, bijvoorbeeld wegens privacy-schending. Sommige 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 technologiën 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 minder valide mensen.

Open Data is sinds kort in een sterke opmars. Zo is er geschat [1] 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 cliënt over alle data lokaal. Zo kunnen alle behoeften van de cliënt voldaan worden. Dit is kostelijk voor de cliënt, 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 1.1 kunnen er meerdere parameters meegegeven worden: waar is het startpunt, wanneer wil je vertrekken, waar wil je aankomen...

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

Listing 1.1: Klassieke webservice interface

Er zijn tiental 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:

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

Er zijn ook enkele nadelen hieraan verbonden:

  - Personalisatie is zeer moeilijk.

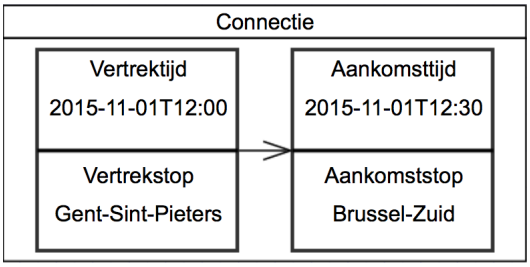
[1] http://www.decroo.belgium.be/nl/groen-licht-voor-federale-open-data-strategie-overheidsdata-voortaan-vrij-beschikbaar

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

In figuur 1.3 zie je deze mogelijkheden weergegeven op een Linked Data Fragments-as (LDF). Dit is een conceptueel framework om de balans tussen cliënt en server weer te geven. Later (2.2.7) zal dit beter uitgelegd worden.

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

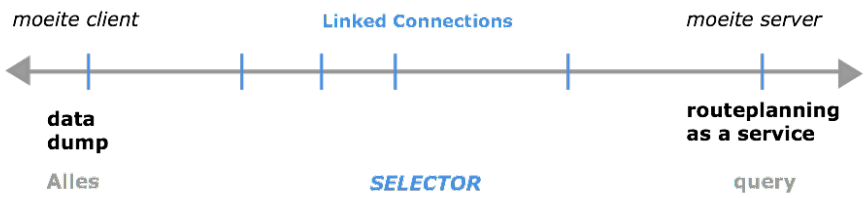
Linked Connections is een manier om transportdata te publiceren zodanig dat het mogelijk is om een route te berekenen hieruit. De basiseenheid is een connectie (zie 1.2). Een connectie is de verbinding tussen een vertrek- en eindstop zonder onderbreking, met respectievelijk een vertrek- en aankomsttijd. 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.



Figuur 1.2: Voorbeeld van een connectie. Deze bestaat uit een vertrek- en aankomstplaats, resp. met vertrek- en aankomsttijd

Met Linked Connections is het mogelijk om client-side een route te berekenen terwijl server-side connecties ter beschikking stelt. Dit introduceert nieuwe trade-offs die onderzocht kunnen worden. Deze worden weergegeven op de LDF-as **??**

De bestaande Linked Connections implementatie werkt momenteel enkel voor een server die connecties ter beschikking stelt. Een probleem hierbij is dat connecties enkel opvraagbaar zijn



Figuur 1.3: Linked Connections opent nieuwe trade-offs tussen cliënt en server aan.

binnen een tijdsinterval. De cliënt wordt overstelpt met connecties die niet nuttig zijn voor het berekenen van de route.

Het doel van deze masterproef is te onderzoeken hoe meerdere datastromen van connecties samengebundeld kunnen worden. Vervolgens wordt onderzocht wat de performantie met de huidige implementatie is en hoe deze verbeterd kan worden.

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 snel 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 cliënt?
3. Kunnen we garanderen dat de snelste route gevonden wordt bij extra filtering?
4. Wat is het effect van caching op de berekeningstijd?

1.3 Hypotheses

Volgende hypotheses zijn waargenomen:

1. De huidige implementatie met tijdsfilter werkt te traag voor routes over lange afstand.
2. De snelheid van het algoritme hangt af van het aantal connecties die gescand moeten worden. Een request meer sturen is minder erg dan meer connecties per request.
3. Het toevoegen van een extra filter om enkel nuttige connecties op te vragen zal client-side routeplanning minstens dubbel zo snel maken.

Deze hypotheses zullen in hoofdstuk **??** en 6 besproken worden.

1.4 Oriëntatie

In volgend hoofdstuk komt een uitgebreide literatuurstudie over het semantisch web en technologiën die een belangrijke hebben bij routeplanning. In hoofdstuk 3 worden Linked Connections onder de loep genomen. Hierna wordt een optimalisatietechniek voorgesteld in hoofdstuk 4. Als voorlaatste hoofstuk wordt de performantie getest van de oorspronkelijke implementatie en de optimalisatie. Ten slotte, in hoofdstuk **??** wordt een antwoord geformuleerd op de vraag hoe we client-side routeplanner sneller kunnen maken en welke aspecten toekomstig onderzoek vergen.

Hoofdstuk 2

Literatuurstudie

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.

2.1.1 GTFS

In 2005 introduceerde Google een specificatie om transportdata uniformiteit te verzekeren, genaamd General Transit Feed Specification (GTFS) [6]. GTFS is een set van regels die om statische tijdstabellen in op te stellen. Deze specificeert welke bestanden noodzakelijk en optioneel zijn en welke informatie hierin verplicht en optioneel is. Deze bestanden zijn opgesteld volgens het Comma Seperated Values (CSV) formaat. Niet alle bestanden zijn noodzakelijk voor deze masterproef. Zie [6] voor de volledige documentatie. Hieronder volgt een uitleg van de meest gebruikte termen en bestanden.

Terminologie

Een *stop* is een plaats waar een voertuig stopt, dit kan gaan om een perron, bushalte etc. Elke ov-bedrijf bestaat uit een aantal *routes*. Dit zijn vastgelegde trajecten, bijvoorbeeld het traject tussen Gent Flanders Expo - 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.

Zoals je kan opmerken is routeplanning een verwarrend 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 (hoofdstuk **??**).

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

2.1.2 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 onvoorziene 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. Een 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.

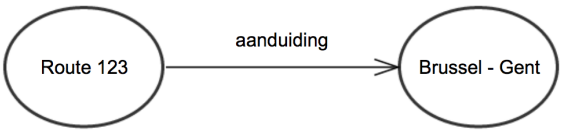
2.2 Semantisch web

Het semantisch web is een verzameling technologiëen (URI, RDF, SPARQL, ontologiën...) 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.

2.2.1 RDF

Resource Description Framework (RDF)[1] is een conceptueel model om bronnen op het web 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 object georiënteerde omgeving van klassen en attributen. Een feit [1] 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.

[1] Deze drieledige structuur wordt ook *triple* of *tuple* genoemd.



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.

2.2.2 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) 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 IBCN nummer 123 uit een bibliotheek, deze kan geïdentificeerd worden via *https://bibliotheek.org/books/123*. Als andere bronnen, zoals de auteur, informatie bevatten over dit boek, dan kan er zonder verwarring verwezen worden hiernaar.

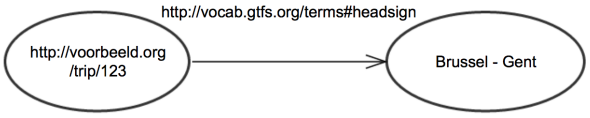
Net zoals je bij object georinteerd programmeren worden er klasses met bepaalde attributen gemaakt om concepten uit de echte wereld zo reëel mogelijk weer te geven. In de wereld van het semantisch web wordt hiervoor gebruik gemaakt van vocabularia.

2.2.3 Vocabularium

Een vocabularium/ontologie is een verzameling klasses en eigenschappen die binnen een bepaalde context samenhoren. Zo is RDF zelf ook een vocabularium 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 gentroduceerd waarmee er gespecificeerd kan worden of een bron een klasse, eigenschap, waarde of datatype voorstelt.

Zo werd er voor GTFS data ook een vocabularium[2] 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 meest bekende voor 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.

2.2.4 JSON-LD

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-LD om deze informatie aan zoekmachines duidelijk te maken. JSON-LinkedData is dus een serialisatie formaat voor gelinkte data. Het grote voordeel van JSON-LD is de compabiliteit met bestaande tools die met JSON werken. Zo kan een JSON-LD document als apart script bestand toegevoegd worden aan een website om semantische informatie weer te geven. Een JSON-LD document bestaat uit twee delen: een context die bepaalt hoe de data geïnterpreteerd moet worden en de data zelf. Listing2.1 bevat informatie over het route voorbeeld in JSON.

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

Listing 2.1: Voorbeeld trip in JSON.

[2] vocab.gtfs.org/terms

Om deze informatie semantisch te beschrijven, wordt er context toegevoegd. In listing 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"
}
```

Listing 2.2: Voorbeeld 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*).

2.2.5 Benoemde grafen

Triples kunnen onderverdeeld worden in grafen. Dit is handig om eigenschappen, metadata... toe te kennen aan een groep triples. In hoofdstuk 2.3.1 zal blijken dat dit voor hypermedia API's een belangrijke rol speelt. De oorspronkelijke triple is nu een quad geworden met <*graaf*><*subject*><*predikaat* ><*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. Listing 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"
    }, ...
  ]
}
```

Listing 2.3: Voorbeeld 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.

2.2.6 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 SPARQL-query die de URI's van alle verschillende luchthavens in Italië opvraagt.

```
"SELECT DISTINCT ?entity
WHERE {
  ?entity a dbpedia-owl:Airport;
       dbpprop:cityServed dbpedia:Italy.
}"
```

Listing 2.4: Voorbeeld van een SPARQL query.

Dit is nu nog een relatief simpele vraag voor een SPARQL-*endpoint*. Het grote probleem [8] 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 reëele toepassingen is dit ontoelaatbaar dat wanneer veel cliënts 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 query'en.

2.2.7 Linked Data Fragments

Een Linked Data dataset is een collectie triples die door een 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 hieraan voldoet. Door een URL te derefereren 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 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 booleaanse 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:

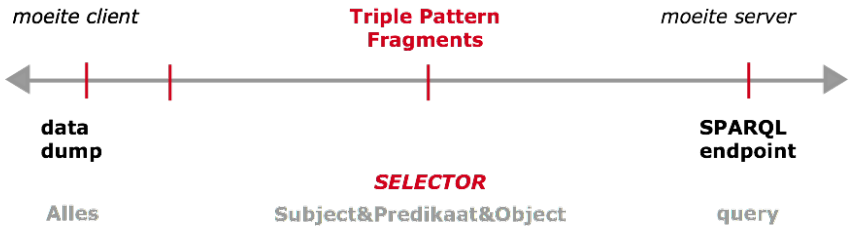
- Hypermedia controls. Dit zijn URL's die meer informatie bevatten over bepaalde entiteiten in het fragment, maw informatie over gerelateerde Linked Data Fragmenten. Voor datadumps en het resultaat van een SPARQL query komt dit neer op het derefereren van de URL die een bepaald object voorstelt.

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

Het interessante aan deze as is dat verschillende interfaces met elkaar vergeleken kunnen worden. LDF's is dus geen technologie, maar een visie hoe de balans tussen cliënt en server kan afgewogen worden. Met data, controls en metadata in het achterhoofd werd in 2014 een oplossing bedacht om Linked Data query'en schaalbaar te maken op het web 2.2.8.

2.2.8 Triple Pattern Fragments

Bij SPARQL-endpoints (2.2.6) is er een probleem dat endpoints niet 100% online blijven doordat er geen beperking is in de grootte en complexiteit van queries. Een Triple Pattern Fragments (TPF) is een nieuwe Linked Data Fragment's visie die de trade-off tussen cliënt en server afweegt (2.4) door een gulden middenweg te vinden.



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

Deze gulden middenweg houdt twee zaken in:

- servers bieden een simpele, *low-cost* interface aan.
- cliënts 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, predikaat en object waarvan minstens een element een waarde heeft. Een voorbeeld van 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[3]: hydra:totalItems, hydra:itemsPerPage, hydra:firstPage, hydra : nextPage. Bij 2.2.8 zal het totaal aantal items z'n nut bewijzen.

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

Listing 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

[3]http://www.w3.org/ns/hydra/core#

het mogelijk om veel cliënts 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 de 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 2.4. De WHERE-clausule bestaat uit triple patronen: de eerste stelt LDF voor van alle luchthavens en het tweede patroon stelt alle entiteiten die ten dienste van Italië staan. De eerste stap van het algoritme haalt voor beide patronen het eerste fragment op. Dankzij metadata weet de cliënt welk patroon het minste triples bevat. Over het resultaat van dit patroon wordt er geïterereerd en worden de variabelen hiervan ingevuld bij de andere WHERE-clausules. Nu wordt hetzelfde algoritme recursief uitgevoerd op de ingevulde, overgebleven WHERE-clausules. 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. Vervolgens wordt elke triple van dit patroon *gejoint* met het tweede patroon. Als de *count* metadata van deze join een is, weten we dat dit een goed antwoord is.

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

2.2.9 Analogie met routeplanning

Een SPARQL-endpoint is vergelijkbaar met een routeplanner API: een server staat in voor het berekenen van een oplossing op een complex vraag. Een cliënt 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 routeplanningserver complexe queries probeert op te lossen via een interface met HTTP parameters. Beiden problemen kunnen opgelost worden door de intelligentie te verschuiven van de server naar de cliënt en ervoor te zorgen dat de server een simpele, cachebare interface aanbiedt. Waar Triple Pattern Fragments (TPF) een oplossing biedt voor het query'en van RDF triples op het web, zal later (hoofdstuk 3) aangetoond worden

hoe dit succesverhaal toegepast kan worden voor routeplannen via Linked Connections (LC).

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

2.3 REST

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*. Hieronder volgt een overzicht van deze beperkingen.

2.3.1 Beperkingen

- Client-server: een client en server moeten losgekoppeld zijn. De server biedt voor een uniforme interface aan dat door verschillende soorten cliënts (app's, websites...) gebruikt kan worden.

- Staatloos: de server houdt geen staat bij van de client. Als twee requests R1 en R2 dezelfde informatie opvragen, moet R2 hetzelfde antwoord krijgen als R1. De informatie die de server van de cliënt krijgt zou voldoende moeten zijn om een antwoord terug te kunnen geven.

- Cachebaar: antwoorden van de server moeten gecachet worden. Als twee dezelfde aanvragen worden verstuurd, mag enkel de eerste effectief berekend worden[4]. De tweede 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 enkel elke bron moet identificeerbaar (zie ook 2.2.2) zijn met een onveranderlijke URI, maar ook elke collectie. Een URI van een bron bestaat uit de URI van de

[4]Rekening houdend met de ingestelde cacheregels die bijhouden hoelang bepaalde document gecachet mogen worden.

collectie waartoe het behoort + '/' + de identificatie van de bron zelf.

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

Listing 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.

- 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, 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. Listing 2.7 toont een voorbeeld van zo'n complexe actie.

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

Listing 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 cliënt welke acties allemaal mogelijk zijn en dan is het aan de cliënt 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 cliënt hierbij geherprogrammeerd moet worden.

2.4 Routeplanning algoritmen

Routeplannnig heeft als doel om verschillende routes te berekenen die bepaalde eisen vervullen. Dit zijn zogenaamde Pareto-optimale oplossingen. Er zijn verschillende soorten queries. De

vroegste aankomsttijd (Engels: *Earliest Arrival Time* (EAT)) query berekent de snelste route vanaf een bepaald moment en plaats om een bepaalde bestemming te bereiken. Een *multicriteria* query houdt rekening met meerdere criteria zoals het maximaal aantal overstappen, rolstoeltoegankelijkheid etc. Een andere veelgebruikte term binnen publieke-transportrouteplanning is een profielquery (Engels: *profile query*). Deze bepaalt alle Pareto-optimale oplossingen voor een vertrekstop binnen een bepaald tijdsinterval. We zullen nu de twee basisalgoritmen bespreken om aan routeplanning te doen.

2.4.1 Dijkstra

Het algoritme van Dijkstra was tot voor kort de de facto standaard voor het berekenen van het kortste pad tussen twee knopen. 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. We zullen enkele optimalisaties van routeplanning met Dijkstra bespreken, omdat deze tot interessante inzichten kunnen leiden.

Het idee is om een geöptimaliseerde graaf vooraf te berekenen. Hierop kan er dan met Dijkstra zeer snel gequery't worden. Enkele technieken die dit mogelijk maken, zijn:

- Vooraf berekenen van *hubs*. Een hub is een subset van belangrijke stops. Deze stops komen vaak voor in Pareto-optimale routes. Bij continentaal routeplannen kunnen metropolen bijvoorbeeld als een hub worden beschouwd.

- Het berekenen van connecties wordt in verschillende ronden beschouwd. Eerst worden connecties beschouwd tussen stations die rechtstreeks, dat wil zeggen zonder overstappen, bereikbaar zijn. Daarna wordt er met maximaal een overstap rekening gehouden enzovoort. Dit principe wordt gebruikt in Round-Based Public Transit Routing (RAPTOR) [5] .

- Voor dat 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 [7] wordt bijvoorbeeld de graaf met alle stations (338 000) van Noord-Amerika geëxpandeerd naar een graaf van 100 miljoen knopen.

- De Public Transit Labeling techniek [4] maakt gebruik van etikettering. Zo krijgt elke verbinding twee etikettes, 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. Dit gebruikt het idee dat publieke vervoernetwerken ingedeeld kunnen worden in afgescheiden kleinere netwerken. Vooraf wordt er voor elke paar stops berekend wat de Pareto-optimale oplossingen zijn. Deze worden voorgesteld als overstappatronen. Zo'n overstap patroon is de optimale route tussen twee stations met daarbij een aanduiding waar overgestapt moet worden. Dankzij deze voorstelling kan er een minimale graaf opgebouwd worden waardoor query'en zeer snel wordt. Het probleem hierbij was dat het ten koste van ofwel het geheugengebruik, ofwel de berekeningstijd is. In de paper [7] is een verbetering van deze techniek voorgesteld die niet alleen een snelle voorbewerkingstijd heeft, maar ook weinig geheugen inpalmt.

2.4.2 Connection Scan Algorithm

Een andere recent ontwikkelde algoritme is het Connection Scan Algorithm (CSA). Routeplanning wordt in plaats van een algoritmeprobleem, een dataprobleem. Als bouwsteen wordt een connectie genomen. Dit is een verbinding tussen twee stops zonder dat er gestopt wordt tussenin. 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 Earliest Arrival Time in lineaire tijd beantwoorden door connecties te scannen. Doordat de connecties gesorteerd zijn volgens vertrektijd moet er pas gescand worden vanaf de ingestelde vertrektijd. 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 en het een stop kan bereiken die nog niet toegevoegd is of de aankomsttijd sneller kan bereiken. 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.

Een ander probleem is de modellering van wandelafstanden (Engels: *foot paths*). Een bepaalde tijd is nodig om bijvoorbeeld te veranderen van perron of vervoersmiddel. Vervoersmaatschappijen hebben hun eigen manieren om overstappen in te calculeren. Het is ook een

profielafhankelijk probleem. Een marathonloper zal bijvoorbeeld minder tijd nodig hebben dan een rolstoelgebruiker. Wegens de complexiteit van *footpaths* wordt er in deze masterproef verondersteld dat overstappen onmiddellijk kan gebeuren.

CSA werkt zeer snel door de lokaliteit van connecties. Deze kunnen makkelijk in het geheugen geladen en overlopen worden. Nog een voordeel van CSA is dat routeplanning een dataprobleem geworden is. Het overstapprobleem moet opgelost worden in data. Ook de interoperabiliteit tussen verschillende operatoren is een dataprobeem: identifiers van stops, routes etc. moeten persistent voorgesteld kunnen worden. Linked Data speelt hier een prominente rol mee.

Het grootste nadeel bij CSA is schaalbaarheid. Doordat enkel in de tijd wordt gescand, worden connecties van elke stop bekeken. Om dit probleem op te lossen werd een uitbreiding gemaakt, genaamd *accelerated Connection Scan Algorithm* (ACSA) . Dit heeft als doel om profielqueries, multicriteria... sneller te kunnen berekenen op netwerken die over meerdere landen kunnen gaan. [2] 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 leiden. Ook wordt er een set van connecties bijgehouden voor lange afstanden. Dit zijn routes die buiten de cel liggen. Het idee is om een subset van nuttige connecties te bekomen door binair te zoeken. Over deze subset kan dan CSA toegepast worden.

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

Hoofdstuk 3

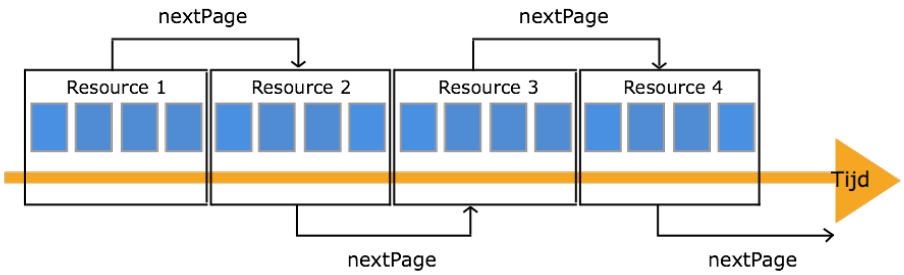
Linked Connections

Linked Connections [3] is een framework dat *client-side* routeplanning mogelijk maakt. Alle algoritmen uit de literatuurstudie hebben als doel om een bepaalde server te lopen waarboven een webservice gebouwd kan worden. CSA biedt een alternatieve oplossing op Dijkstra om routeplanning meer een datakarakter te geven. Linked Connections vervolledigt het dataprobleem door de intelligentie door te schuiven naar de cliënt. Volgende sectie de cliënt de snelste route kan berekenen.

3.1 Principe

Een Linked Connections server is verantwoordelijk voor het publiceren van connecties (zie 1.1). Deze connecties worden door een Linked Connections cliënt opgevraagd om dan met CSA (zie 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 met intelligente cliënts te gebruiken. Een eenvoudige selector is de tijdsfunctie die connecties met een vertrektijd binnen een bepaald tijdsinterval beschouwt als een fragment. Zo'n fragment noemen we een *basic Linked Connections fragment* (basis LCF). Met behulp van hypermedia-links (zie ook 2.3.1) 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 functionaliteit machine-verstaanbaar te maken. Figuur 3.1 toont een overzicht hoe de connecties gepresenteerd worden door de server.

De cliënt kan met behulp van textithydra:nextPage links makkelijk het volgende fragment



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

ophalen. De connecties van een fragment worden als invoer doorgegeven aan CSA (2.4.2). Deze bouwt een minimale overspannende boom met de snelste tijd om stops te bereiken. Wanneer het resultaat gevonden is, kan de cliënt 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 2.3) worden gevolgd om ervoor te zorgen dat de fragmenten, de REST resources, cachebaar zijn. Er is namelijk een eindige verzameling URI's, afhankelijk van het ingestelde tijdsinterval. Listing 3.1 toont een template van deze REST interface.

```
http://voorbeeld.org/connecties/?vertrekTijd=?
```

Listing 3.1: URI template van basis Linked Connections fragment.

Als T het tijdsinterval in minuten is van de fragmenten, dan kan het aantal mogelijke URI's berekend worden met formule 3.1. Is T=10min, dan zijn er 144 verschillende fragmenten.

$$24 * 60/T \tag{3.1}$$

De meeste publieke vervoersmaatschappijen rijden niet 24/24 dus in praktijk zijn er nog minder fragmenten. 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 vorm van de 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 [1] en Linked Connections[2] ontologiën. De API functionaliteit maakt gebruik van de Hydra ontologie. Deze beschrijft dat de cliënt kan zoeken door de vertrektijd parameter 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 gequery't worden over het web met bijvoorbeeld *Triple Pattern Fragments* (TPF) (zie ook 2.2.8) Er bestaan al talrijke servers online die mogelijke uitbreidingen kunnen zijn. Met behulp van GeoNames [3] kan er meer informatie gevonden worden over bepaalde geolocaties. Ook voor *Points of Interest* is een Linked Data portaal te vinden. [4].

3.1.2 Basic Linked Connections Fragments

Basic LCF zijn fragmenten met connecties die onderverdeeld zijn met de vertrektijd tussen een bepaald tijdsinterval. Het kost de server weinig moeite om data te publiceren door de hoge cachebaarheid. Een cliënt moet daarentegen veel connecties scannen om tot een route te komen. In hoofdstuk **??** zal de exacte performantie van routeplanning met deze fragmenten verduidelijkt worden.

Figuur 3.2: Linked Data Fragmenten-as met Linked Connections

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

[1] http://lov.okfn.org/dataset/lov/vocabs/gtfs

[2] Linked Connections ontologie: http://semweb.mmlab.be/ns/linkedconnections

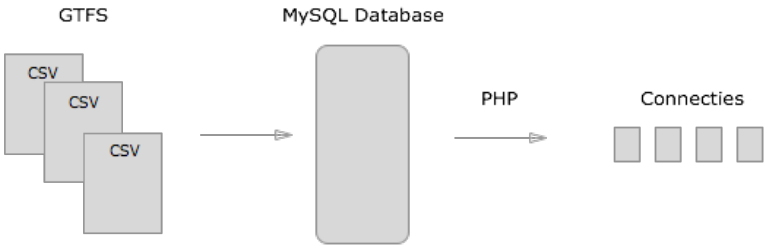
[3] Momenteel is er al een TPF-server met gelinkte GeoNames beschikbaar: http://data.linkeddatafragments.org/geonames

[4] Ook hiervoor is een Linked Data portaal aanwezig die door intelligente cliënts gebrowset kunnen worden: http://data.linkeddatafragments.org/linkedgeodata

3.2 Convertor naar connecties

Door het wereldwijde gebruik van GTFS hebben we in 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 query'en.

Figuur 3.3 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 3.3: 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.

Een connectie vereist informatie uit verschillende tabellen. Deze moeten per dag d stapsgewijs 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. Voor elke service komt overeen met een route en meerdere trips. Een route is niet essentieel voor een connectie,

Figuur 3.4: Per dag worden de corresponderende service ID's, trip ID's en route ID's berekend. De vertrek-/aankomststopplaats met respectievelijk vertrek-/aankomsttijd worden berekend uit een verzameling stoptimes.

maar bevat wel interessante informatie zoals de routebeschrijving en type voertuig.

3. De laatste stap is het overlopen van *stoptimes.txt*. 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
```

Listing 3.2: Voorbeeldstoptijden uit GTFS *stop_times.txt*.

Hiermee zijn alle stappen doorlopen en kan een connectie weggeschreven worden. Codevoorbeeld 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"
}
```

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

Als je tijd van de stoptijden in 3.2 vergelijkt met de aankomst- en vertrektijd in de connectie 3.3 zie je dat de tijd in de connectie een uur vroeger is. Dit komt doordat de tijd van connecties in Coordinated Universal Time (UTC) staat. Dit is herkenbaar aan de 'Z' die vanachter staat. Er moet altijd rekening gehouden worden met de tijdszone van waar de GTFS feed zich afspeelt. Het voorbeeld van 3.2 stelt een feed uit België voor in wintertijd. België volgt in de wintertijd UTC+1 en in de zomertijd UTC+2. Er moet dus een uur afgetrokken in deze periode 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, bijvoorbeeld 2u 's nachts staat weergegeven als 26u. Volgens GTFS rijden deze stoptimes 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 vertrek- en aankomsttijd van een stoptijd voor of na middernacht plaatsvinden. Met aparte queries kunnen de stoptijden na middernacht opgevraagd worden.

Een databank gebruiken heeft als grote nadeel dat de data ingeladen moet worden vooraleer berekeningen kunnen plaatsvinden. In 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 (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 rapper. In 3.2 zie je dat voor kleine datasets (zoals NMBS en NS) het minder dan minuut duurt om de connecties voor een dag te berekenen. De Lijn scoort opnieuw zeer slecht door de grootte van de dataset.

	Connecties	Aantal stops	Tijd (min)	Tijd met nieuwe convertor (min)
NMBS	55479	597	0.21	
NS	41796	2531	1.30	
De Lijn	1049186	35275	14.97	

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

Ondertussen is er een veel snellere convertor [5] gemaakt die connecties zonder databank kan genereren. In de laatste kolom van 3.2 staat de tijd met de nieuwe convertor vermeld. Deze werkt veel sneller en heeft geen bottleneck meer.

Connecties worden een per een weggeschreven naar een apart bestand. Hiervoor wordt er gebruik gemaakt van een nieuwe specificatie, genaamd *JSON-LD stream*.

3.2.2 JSON-LD stream

Om connecties als Linked Data te publiceren werd er voor gekozen om JSON-LD te gebruiken. JSON wordt beschouwt als het *de facto* 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 steken (zie 2.3). Deze methode vereist dat de graaf in het geheugen geladen moet worden vooraleer verdere operaties mogelijk zijn.
- elk object een context geven (zie 2.2). Met deze methode kan object per object gestreamt worden, maar zorgt voor grote overhead door de context die telkens mee gepubliceerd wordt.

Een oplossing hiervoor is de JSON-LD streamspecificatie [6]. 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

[5] Zie: github.com/linkedconnections/gtfs2lc
[6] Zie: https://github.com/pietercolpaert/jsonld-stream

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 cliënt aangemaakt kan worden.

3.3 Cliënt

De cliënt is verantwoordelijk voor het berekenen van de snelste route met CSA. Deze kan met een paar lijntjes code aangemaakt worden (zie 3.4).

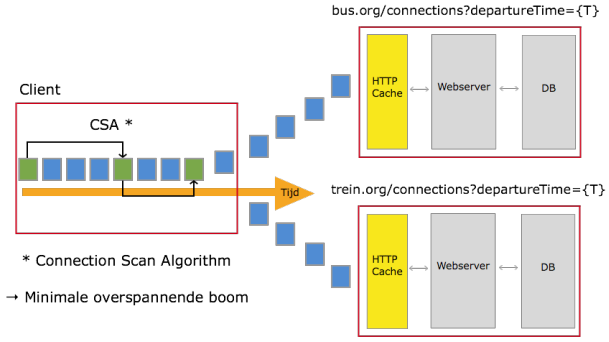
```
var planner = new window.lc.Client({"entrypoints" : ["http://
    example.linkedconnections.org/"]});
planner.query({
      "departureStop": "Brussel-Zuid",
      "arrivalStop": "Gent-Sint-Pieters",
      "departureTime": new Date("2015-11-05T10:00")
      }, function (stream) {
        stream.on('result', function (pad) {
          // pad bevat verzameling connecties die snelste route voorstellen
        });
        stream.on('data', function (connectie) {
          // connectie is gebruikt geweest voor minimale overspannende boom
        });
});
```

Listing 3.4: Code om een LC cliënt op te zetten in JavaScript.

Er zijn twee fases vooraleer routes effectief berekend kunnen worden:

- Configuratiefase: de LC client moet weten welke servers er bereikt moeten worden door de webadressen toe te voegen in de *entrypoints*-rij. Ook moet de query ingesteld worden. Momenteel wordt enkel het startpunt, aankomstpunt en vertrektijd ondersteund.
- Nadat de planner geïnitialiseerd is, zal deze connecties opvragen en doorgeven aan CSA. CSA stuurt zelf notificaties wanneer een bepaalde actie ondernomen is. Zo wordt de *data*-notificatie uitgestuurd wanneer een connectie gebruikt is voor de minimale overspannende boom.

Momenteel is er enkel ondersteuning om een connectiestroom van een *entrypoint* op te vragen. Andere implementaties die van CSA gebruik maken, houden alle connecties bij in een server. De bedoeling van Linked Connections is om schaalbaar verschillende servers te kunnen opzetten voor elke modi en/of land. Figuur **??** toont een overzicht van een opstelling met meerdere servers. Rechts staan twee servers, de ene verantwoordelijk voor busdata, de andere voor treindata.



Figuur 3.5: 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.

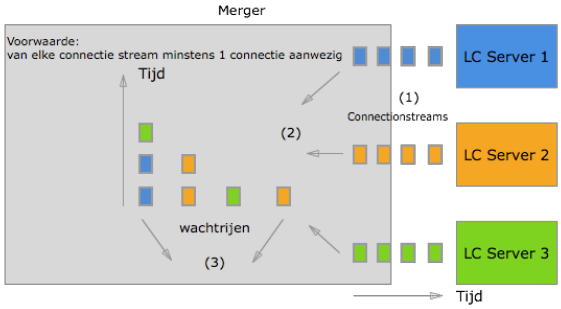
3.3.1 Merger

Een merger zorgt ervoor dat een 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 cliënt 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 notificatieluisteraars. Dit werkt op een asynchrone manier waardoor het niet vanzelfsprekend is om aan de sorteringsvoorwaarde te voldoen. De volgorde waarin deze notificaties worden opgeworpen is onvoorspelbaar waardoor een een mechanisme ontworpen moet worden om te verzekeren dat elke stroom aan bod is gekomen.

Figuur 3.6 toont een overzicht van een merger. De merger luistert (1) naar de verschillende onderling gesorteerde datastromen. Wanneer er data vrijgeven wordt, wordt dit toegevoegd aan

een van de wachtrijen (2). Deze wachtrijen hebben als voorwaarde dat ze zelf gesorteerd zijn.



Figuur 3.6: Overzicht merger

Om te kunnen garanderen dat de totale connectiestroom (3) gesorteerd is, wordt er vereist 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. 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 cliënt 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 het toevoegen zijn van busdata wanneer het doel binnen een bepaalde straal bereikbaar is.

Listing 3.5 toont hoe een merger geïnitialiseerd kan worden. Merk op dat bij elke connectiestroom een naam toegevoegd is. Zo kan de merger controlleren dat er voor elke stroomminstens een connectie in de wachtrijen aanwezig is. De cliënt kan hiermee ook gemakkelijk acties ondernemen. Zo kan vanaf een bepaald moment beslist worden om een connectiestroom toe te voegen aan de merger.

```
"var connectionsStreams = [
    [ 'stream1', connectionsReadStream1 ],
    [ 'stream2', connectionsReadStream2 ],
    ...
];

var connectionsReadStream = new MergeStream(connectionsStreams,
    query.departureTime);

connectionsReadStream.on("data", function (connection) {
    connectionsReadStream.addConnectionsStream('busStroom',
```

```
        newConnectionsReadStream);
});"
```

Listing 3.5: Voorbeeldcode van een merger. Deze voegt meerdere stromen van connecties samen.

3.4 Conclusie

We hebben gezien hoe connecties gegenereerd kunnen worden uit een GTFS feed en hoe een Linked Connections server deze als *basic LC fragments* aanbiedt aan de cliënt. 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 snel deze opstelling werkt voor een of meerdere datasets.

Hoofdstuk 4

Optimalisatie

De bedoeling van de optimalisatie is het vergroten van het percentage nuttige connecties. Met basis LCF's werd er enkel in de tijd gefilterd waardoor ook connecties van ontoegankelijke stopplaatsen werden teruggeven. De optimalisatietechnieken die in dit hoofdstuk besproken worden, zullen dit probleem proberen oplossen met behulp van geofiltering.

4.1 Geofiltering

Het CSA-algoritme bouwt met de binnenkomende connecties een minimale overspannende boom. Dit kan visueel voorgesteld worden als een cirkel rondom het vertrekpunt van de query. Het idee van geofiltering bij Linked Connections is om enkel connecties gesorteerd in de tijd n afstand van het vertrekpunt terug te geven. Een extra filter betekent extra HTTP parameter(s).

4.1.1 Geoparameter

Een eerste inspiratie was om connecties onder te verdelen zoals kaarten als Open Street Maps werken. Hierbij wordt er met *tiles* gewerkt. Een *tile* is een rechthoekig 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, wordt er geopteerd om het vertrekpunt van een query toe te voegen als extra HTTP parameter. Listing **??**) toont deze nieuwe interface.

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://example.org?departureTime={...}&departureStop={...}
\label(label:nieuwe-interface)
```

Listing 4.1: Interface LDF met optimalisatie

4.1.2 Criteria

In eerste instantie moeten we een "straal-definitie bepalen. Hiervoor zijn er twee mogelijkheden:

- De straal tussen de vertrekstop en een stop s is het minimaal aantal connecties nodig om deze te bereiken. Een straal van grootte een stemt overeen met stops die met een connectie bereikt kunnen worden. Een straal van grootte twee is een stop waarvoor twee connecties nodig zijn enzovoort. Connecties worden als het ware *gejoint* aan elkaar.

- Het minimum aantal overstappen (Engels: *transfers*) wordt als straal van een stop genomen. Dit geeft extra interessante mogelijkheden zoals Pareto-optimale routes bepalen op basis van snelste aankomsttijd n maximaal aantal overstappen. Dit idee wordt ook bij RAPTOR (2.4.1) gebruikt. Voortaan zullen we de letter K gebruiken voor het minimum aantal overstappen.

Deze masterproef maakt gebruik van het aantal overstappen.
Twee nieuwe termen zullen regelmatig gebruikt worden:

- *direct bereikbare stop*: dit is een stop dat zonder overstappen bereikbaar is vanuit de vertrekstop. In GTFS termen betekent dit dat de snelste route slecht uit een trip kan bestaan.

- *indirect bereikbare stop*: dit is een stop die minstens K overstappen vereist vanuit de vertrekstop.

- *buur*: dit is een stop die direct of indirect bereikbaar is.

Een andere belangrijke criteria om geoptimaliseerd connecties terug te geven is dat er geen connecties teruggeven mogen worden van buren die theoretisch gezien nog niet bereikbaar zijn. Voor elke buur is een bepaalde *minimale tijd* nodig om deze te bereiken.

Kortom, voor elke stop moet zijn buren bepaald worden samen met informatie over het minimaal aantal overstappen en minimale tijd om deze te bereiken. Hiervoor wordt er een extra voorbewerkingsfase toegevoegd en wordt er een extra *feature* aan het connectiegenerator script toegevoegd.

4.1.3 Direct bereikbare stops

Connecties worden gegenereerd uit een reeks stoptijden van dezelfde trip. Dit is informatie die gebruikt kan worden om direct bereikbare stops te berekenen. Zo'n reeks stelt een aantal paren stops voor met een bepaalde tijd die nodig is in die trip. Tijdens het genereren van connecties wordt voor elke stop een minimale overspannende boom bijgehouden met als knopen rechtstreeks bereikbare stops en als verbindingen de minimale tijd. Nadat de connecties gegenereerd zijn wordt deze informatie weggeschreven naar een CSV-bestand zodat we dit kunnen hergebruiken om buren mee te berekenen.

4.1.4 Buren

Het berekenen van buren houdt het vinden van niet-direct bereikbare stops voor elk stop in. Een indirect bereikbare stop van s met minimum aantal overstappen $K = 1$ is een direct bereikbare stop van een direct bereikbare stop van s die niet direct bereikbaar is bij s zelf. Als er n stops zijn in een dataset, moeten er $n * n$ afstanden berekend worden. Dit zou gemakkelijk met Floyd-Warshall kunnen, maar bij grote netwerken zoals bij De Lijn is er een probleem met geheugen. De Lijn heeft namelijk afgerond 35000 stopplaatsen waardoor een matrix van 35000 x 350000 in het geheugen moet gehouden worden. Om dit toch te kunnen berekenen, maken we gebruik van een algoritme die gebaseerd is op Dijkstra. Pseudocode hiervan is te vinden in 1. Er kan ingesteld worden tot hoeveel overstappen er verder gezocht moet worden. Het algoritme maakt gebruik van een wachtrij van stops die nog bekeken moeten worden. Elke stap probeert de minimale tijdsafstand en/of aantal overstappen te minimaliseren van een buur van een stop. Het object dat bekomen wordt voor een bepaalde stop staat afgebeeld in listing 4.2. Dit wordt weggeschreven naar een aparte buren-tabel op een LC server.

```
{
  stop_id : 123,
  aantal_direct_bereikbare_stops: X,
  // andere metadata over de stop
  buren: [
```

```
    {
      stop_id: 253,
      minimaleTijdsAfstand: 1800, // seconden
      K: 2
    }, {
    ...
    }
  ]
}
```

Listing 4.2: Data over stop serverside

Zoals je kan zien in 1 wordt er ook het aantal rechtstreeks bereikbare stops bijgehouden voor elke buur. Volgende sectie zal hierover meer uitleg geven. Tabel 4.1 toont hoelang het duurt om de buren te berekenen volgens de grootte van het netwerk van de NMBS en NS.

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 gebruik gemaakt kan worden van deze informatie om het routeplannen sneller te maken.

4.2 Neighbour Linked Connections Fragment

Wanneer de cliënt een request met vertrektijd n 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). Op deze manier kan de cliënt veel informatie in een request bekomen. De query die hierbij gebruikt wordt op de server staat vermeld in listing 4.3.

```
SELECT *
FROM connecties
WHERE (vertrekStop = queryStop && vertrekTijd > queryTijd && vertrekTijd <
    queryTijd + interval)
```

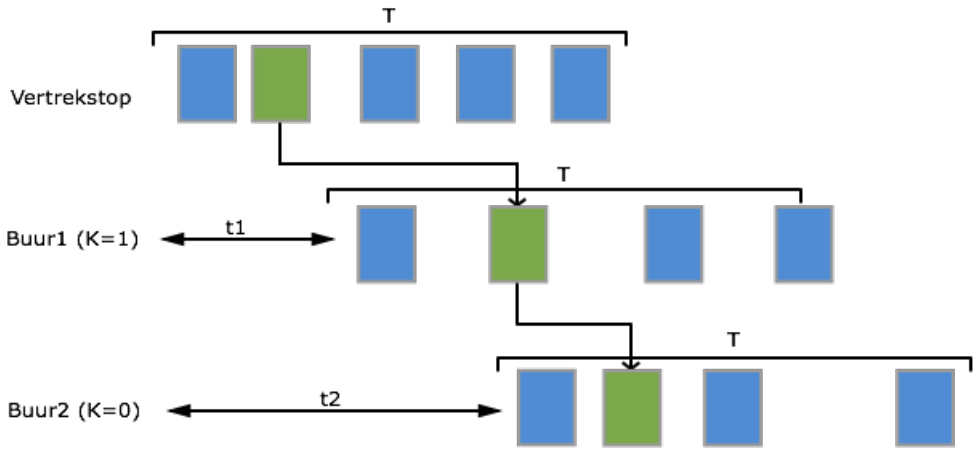
Algorithm 1 Berekenen van Minimale Overspannende Boom van burenstops voor een GTFS stops.txt. T is het maximaal aantal overstappen waarmee rekening gehouden wordt.

```
K ← 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
        mob[b.stop_id] = { K : 0, tijdsafstand : b.tijdsafstand, aantal_rechtstreekse_buren:
start.buren.length };
        if (T >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.tijdsafstand +
b.tijdsafstand, aantal_rechtstreeks_buren: start.buren.length };
                if (b.K <T) then
                    wachtrij.push(b); // nog verder te onderzoeken
                end if
            else
                if (start.K + 1 <mob[b.stop_id].K) then mob[b.stop_id].K = start.K + 1;
                end if
                if (start.tijdsafstand + b.tijdsafstand <mob[b.stop_id].tijdsafstandl) then
mob[b.stop_id].tijdsafstand = start.tijdsafstand + b.tijdsafstand;
                end if
            end if
        end for
    end while
end for
```

```
OR (vertrekStop = buur1 && vertrekTijd > vroegste_vertrektijd_buur1 &&
    vertrekTijd < vroegste_vertrektijd_buur1 + interval)
OR (vertrekStop = buur2 && vertrekTijd > vroegste_vertrektijd_buur2 &&
    vertrekTijd < vroegste_vertrektijd_buur2 + interval)
OR ...
```

Listing 4.3: SQL query template van optimalisatie.

Vooraf moet een bepaald tijdsinterval ingesteld worden waarin de snelste route zich zou moeten bevinden.



Figuur 4.1: Neighbouring Linked Connections Fragment bevat connecties van vertrekstop en buren binnen een bepaald tijdsinterval.

Het basisidee is dat een NLCF meer nuttige informatie bevat dan een basic LCF. Hierbij zijn er twee trade-offs:

- Het maximaal aantal overstappen K. Stops die verder liggen hebben een grotere kans op meer overstappen. De vraag is of het nuttig is om moeilijk bereikbare stops in rekening te brengen.
- Het tijdsinterval T waarin connecties van een stop worden teruggegeven.

Een Linked Connections server is nu verantwoordelijk voor beide soorten fragmenten. Volgende sectie bespreekt een eerste mogelijkheid dat de cliënt kan opgaan.

4.3 Heuristische techniek

De heuristische techniek maakt enkel gebruik van neighbouring LC Fragments. Als het antwoord niet gevonden is met een request, 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.

4.3.1 Parameters

De heuristiek in deze implementatie maakt gebruik van volgende parameters:

- De snelheid v van de connectie. De afstand d tussen de stop en de eindbestemming, met andere woorden hoe dichter bij onze bestemming hoe beter. Een stop die op de rechtstreekse weg tussen vertrekpunt en eindpunt ligt, krijgt zo een hogere score.

- De cosinus-gelijkaardigheid. De cosinus van twee vectoren bepaalt hoe sterk de vectoren in dezelfde richting liggen. De ene vector bestaat uit het start- en eindpunt van de connectie en de andere uit start- en eindbestemming van de route.

- De belangrijkheid van een station. Als maatstaf kan het aantal rechtstreeks bereikbare stops vanuit een bepaalde stop genomen worden. 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 Lichtervelde, maar vaak is het dat de snelste route via Brugge gaat. Brugge ligt niet op de rechtstreekse weg tussen Gent en Lichtervelde dus moet er metadata zijn die de belangrijkheid van bepaalde stations aanduidt. Er moet een soort van knoophiërarchie van stops komen zoals we besproken hebben in de literatuurstudie. In tabel 4.2 zie je een overzicht van enkele stations met hun aantal rechtstreekse stops en drukte [1]. Brussel is het meest centrale punt in België. Zijn rechtstreekse stops zijn dan ook het meest van alle stations en in heeft een gelijke proportie met het gemiddeld aantal stoptijden. Gent en Antwerpen zijn gelijkwaardig in grootte en drukte. Antwerpen heeft met minder rechtstreekse stopplaatsen toch een grotere frequentie. De laatste drie stations bestaan uit twee stations waarlangs verschillende routes samenkomen en een station waar niet overgestapt kan worden. Dit weerspiegelt zich zowel in de eerste als tweede waarde.

[1] De drukte van een station is bepaald met de open dataset van stations: github.com/irail/stations

We kunnen concluderen dat het aantal rechtstreekse stops een goede parameter voor de metaheuristiek 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.

4.3.2 Formule

De metaheuristiek bestaat uit volgende formule:

$$prioriteit = \alpha * afstand/tijd + \beta * aantal\_rechtstreekse\_stops + \gamma * \cos(theta)$$

- De snelheid (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 duidt de belangrijkheid van het station aan.
- De cosinusgelijkheid theta: de cosinus van de hoek tussen de connectievector en de vector van rechtstreekse weg bepaalt de mate of de connectie ons veel dichter bij het doel brengt.

Deze parameters moeten eerst genormaliseerd worden zodat elke parameter een score heeft van 0 tot en met 1. Wanneer de cliënt een fragment opgehaald heeft, wordt er een bepaald aantal connecties eerst geanalyseerd voor de normalisatie. Voor elke parameter wordt er de maximale waarde bepaald. Daarna wordt deze genormaliseerde waarde vermenigvuldigd met een bepaalde gewichtsfactor alpha, beta.... De sommatie bepaalt de prioriteit van het station. Deze wordt in een prioriteitswachtrij gestopt om in $O(1)$ tijd het belangrijkste station te kunnen opvragen.

Het toepassen van de heuristiek en toevoegen aan de prioriteitswachtrij wordt enkel op nuttige connecties toegepast. Dit wil zeggen dat CSA deze connectie heeft toegevoegd aan

de minimale overspannende boom. De module die verantwoordelijk is voor het ophalen van fragment wacht dus tot alle connecties gescant zijn om vervolgens het volgende beste fragment te bepalen.

4.3.3 Voordeel

- Om niet van

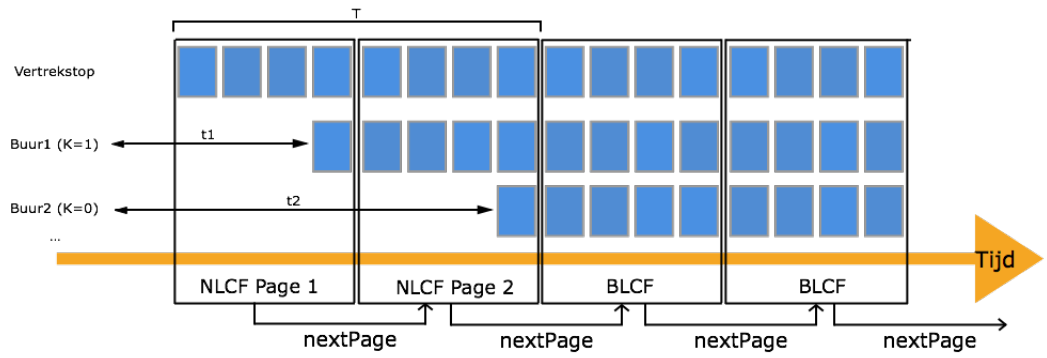
4.3.4 Nadelen

- Er wordt minder efficiënt omgegaan met connecties dan gewenst. Een NLCF geeft voor elke buurstop ($K <= maxK$) connecties terug binnen een bepaald tijdsinterval. Dit moet allemaal in een fragment waardoor het tijdsinterval klein moet blijven. Hierdoor daalt de kans dat verre buren bereikt kunnen worden. Wanneer een nieuwe vertrekstop gekozen wordt, kunnen er connecties voor de tweede, derde... keer verzonden worden. Een groot deel van de connecties van een nieuwe vertrekstop zijn ook onnuttig, namelijk deze die geografisch liggen aan de kant van waar vertrokken werd.

- Het werken met een heuristiek zorgt voor een suboptimale oplossing. Dit wil zeggen dat er een route gevonden wordt, maar niet gegarandeerd de snelste. Dit ligt aan het feit dat een heuristiek niet voldoende is om de snelste route te bepalen. Zo zou er voor elk station een maximale tijdswaarde bepaald moeten worden die ervoor zorgt dat de connectie met de snelste route hierin ligt. Een mogelijke uitbreiding zou zijn om T en K te maximaliseren en dit fragment dan te pagineren. Een grotere T en K zorgt ervoor dat meer queries in een fragment opgelost kunnen worden. Wegens tijdsgebrek om te onderzoeken of een maximale tijdsinterval per stop mogelijk is, gaan we hier niet dieper op in. Er is wel geweten of de snelste route binnen het tijdsinterval T ligt van het eerste NLCF. De volgende besproken techniek bouwt verder op dit idee.

4.4 Speed-up techniek

De *speed-up* techniek maakt enkel in het begin gebruik van een NLCF. Indien het resultaat nog niet gevonden is, wordt verdergezocht met basic LCF's. Een NCLF heeft een veel grotere efficiëntie (zie 5.2.3. Door verder te zoeken met basic LCF's kan de snelste aankomsttijd gegarandeerd worden.



Figuur 4.2: Overzicht van resources van speed-up optimalisatietechniek.

4.4.1 Hypermedia

De originele implementatie maakt gebruik van hydra:nextPage links om volgende fragmenten te vinden. Dit principe kunnen we toepassen op een NLCF (zie ook 4.2). Op deze manier kunnen we een grotere T waarde instellen zonder dat kleinere routes onnodig veel connecties downloaden. K moet maximaal zijn om te kunnen garanderen dat de optimale oplossingen ertussen zit. Uit de resultaten van basic Linked Connections Fragments (zie 5.1.2) blijkt dat queries gemiddeld het snelst opgelost zijn als de fragmenten een grootte hebben van rond de duizend connecties. Het initiële NLCF wordt per duizend connecties onderverdeeld. Listing 4.4.1 toont de nieuwe interface. Het NLCF fragment bevat naast tijdsinterval T ook een pagina-grootte. Hiervoor kan hetzelfde tijdsinterval genomen worden als basic LCF's.

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

Listing 4.4: Interface optimalisatie met hypermedia links

Met deze techniek worden de twee nadelen (4.3.4 van vorige techniek weggewerkt:

- Er is een hogere efficiëntie, omdat een groter tijdsinterval gekozen kan worden. Daarbij worden connecties slechts een keer teruggeven, omdat er slechts een NLCF gebruikt wordt.
- De optimale oplossing wordt gevonden. De NLCF geeft een gefilterde lijst van connecties voor alle mogelijke buren (dankzij max K) 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 query nog niet gevonden is, wordt simpelweg verdergezocht met basic LCF's met vertrektijd na T. Deze

garandeert ook een optimale oplossing.

Hoofdstuk 5

Resultaten

In dit hoofdstuk bespreken we de testen die een antwoord geven op de onderzoeksvraag en hypotheses. In sectie 5.1 worden enkele praktische zaken besproken. Daarna worden de drie besproken technieken uit hoofdstukken 3.1.2 en **??** onderling vergeleken.

5.1 Opstelling

Deze masterproef maakte gebruik van de officiële GTFS dataset van de Belgische spoorwegenmaatschappij NMBS.

Zowel de cliënt 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 cliënt is een webapplicatie waarin queries werden uitgevoerd. Als browser werd Firefox gebruikt.

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[1]. 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 een productieserver.

[1]Linked Connections server te vinden op: http://belgianrail.linkedconnections.org/connections/

Figuur 5.1 toont aan dat er een verschil in snelheid is tussen een lokale en een productieserver. Dezelfde queries werden berekend met een 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.

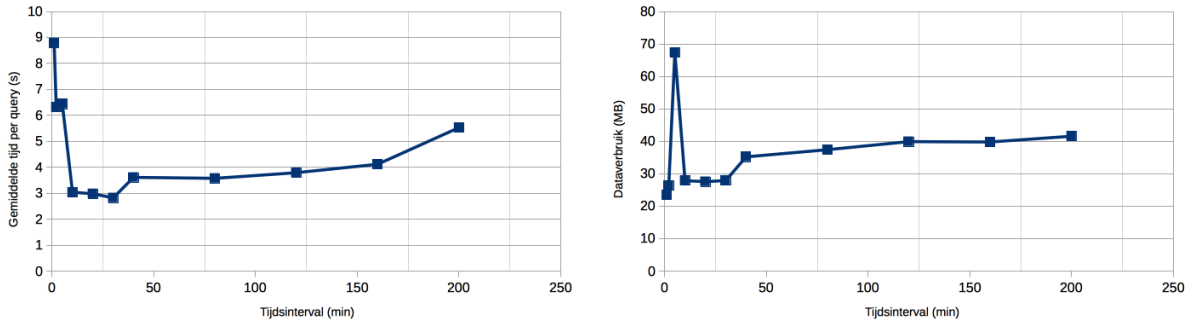
5.1.2 Fragmentgrootte

Basic Linked Connections Fragment's (BLCF) hebben een vast tijdsinterval waarin connecties worden teruggeven. Dit komt met een gemiddeld aantal connecties. Tabel 5.2 toont een overzicht van verschillende fragmentgroottes die mogelijk zijn met de hoeveelheid data die er gemiddeld aan verbonden is.

	NMBS	
t (min)	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.

Om te bepalen wat de optimale fragmentgrootte is, werden een honderd queries uitgevoerd met telkens een andere fragmentgrootte. De resultaten in 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 een lineair verband tussen het aantal gedownloade fragmenten en het ingestelde tijdsinterval. Bij zeer kleine fragmenten kunnen er uitschieters ontstaan.



(a) De gemiddelde tijd in seconden om queries uit te voeren in functie van het tijdsinterval van basic LCF.

(b) De hoeveelheid opgevraagde data in megabyte in functie van het tijdsinterval van basic LCF.

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

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

	Kilobyte	Megabyte
Grootte connectie	0,26158	0,00026158

Tabel 5.3: Grootte van een connectie.

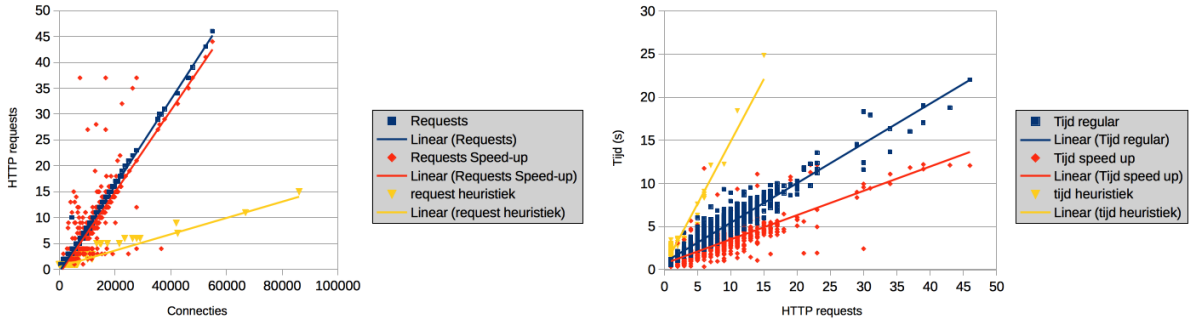
Uit 5.1 kan ook afgeleid worden dat de snelheid van routeplanning afhankelijk is van de combinatie van het aantal connecties en het aantal requests. Voor korte routes zijn kleinere fragmentgroottes beter. Lange routes zijn veel sneller met minder requests en een groot tijdsinterval. Om niet te groot 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 gescant moeten worden, zijn te vermijden.

Voortaan zullen we als referentie pagina's van ongeveer 1000 connecties nemen. In tabel **??** staat een overzicht van de verschillende paginagroottes voor de drie technieken die getest zijn. Voor NLCF is een verschillende grootte voor de gecachete snelheidstest en niet-gecachte. Het aantal mogelijke overstappen K is gemaximaliseerd zodat elke stop bereikbaar is vanuit een bepaalde vertrekstop. Voor de NMBS is $maxK = 5$.

De reden waarom het tijdsinterval van NLCF's beperkt blijft tot 100-120 minuten is omdat

Techniek	BLCF (min)	NLCF cache (min)	NLCF zonder cache	Fragmentgrootte (min)
Basic LCF	20	-	-	-
Speed-up	20	30	40	120
Heuristiek	20	100	100	100

Tabel 5.4: Tijdsintervallen voor de verschillende technieken.



(a) HTTP requests in functie van aantal connecties.

(b) Lineair verband tussen aantal HTTP requests en tijd om route te berekenen zowel met als zonder caching.

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 al bereikbaar waardoor deze soort fragmenten niet meer opwegen in snelheid.

5.2 Hypotheses

Volgende twee hypotheses worden aangekaard:

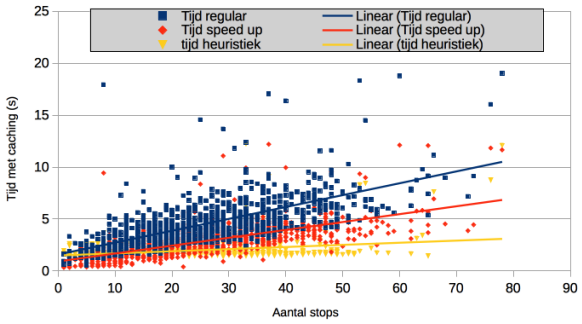
- *Hoe meer connecties, hoe meer bandbreedte. Bijgevolg meer tijd.*

Bovenstaande grafiek (5.2a toont aan dat het aantal requests lineair afhankelijk is met het aantal connecties. Ook grafiek 5.2b) toont een lineair verband tussen de tijd en het aantal requests. Enkel requests met connecties zijn in rekening gebracht. Voordat connecties effectief opgevraagd worden, stuurt de cliënt een request om de juiste context op te vragen. Door de kleine grootte (1495 kB) en het feit dat dit bij alle technieken wordt gedaan, worden deze requests verwaarloosd. Beide figuren (5.2a en 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. In volgende subsectie wordt de snelheid getest in functie van de afstand.

5.2.1 Snelheid

Om de snelheid te meten van de verschillende technieken werden er 900 random routes gegenereerd. Volgende paragrafen tonen aan of er een groot verschil in performantie is afhankelijk van client-side caching.

Met caching



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

Op grafiek 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 speed-up techniek en oorspronkelijke techniek met basic LCF's. Bij de heuristische techniek zijn de queries waarvoor geen optimale route gevonden werden, weggelaten. Dit zijn queries waarvoor meer dan 20 requests gestuurd zijn zonder enig resultaat. Meestal zijn dit ook moeilijkere queries, bijvoorbeeld buitenlandse stations zijn moeilijker bereikbaar. Deze moeilijkere queries brengen de grootste tijd met zich mee dus de werkelijke snelheid ligt gemiddeld wat hoger. 78 % van de queries zijn gelukt met de heuristische methode. De huidige implementatie die enkel basic LCF's gebruikt is overduidelijk de traagste. De speed-up techniek is zoals verwacht iets sneller. Bij uitschieters is er een groot verschil van enkele seconden tijd te merken tussen basic LCF's en de speed-up techniek. In tabel 5.5 staat een overzicht van de gemiddelde snelheid per query. Zoals je kan zien kan de speed-up en heuristische methode ongeveer dubbel zoveel queries oplossen in dezelfde tijd als de basis methode met basic LCF's.

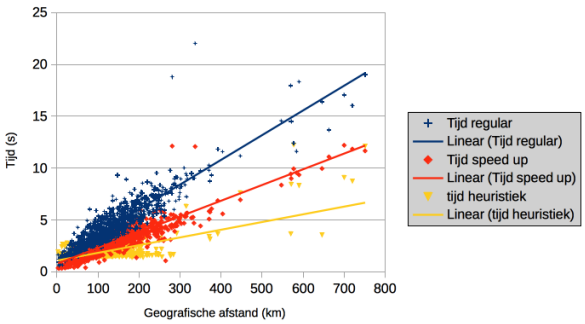
De snelheid van de heuristiek is een stuk groter en constanter dan de andere twee technieken. Om dit te kunnen benaderen met de speed-up techniek, hebben we de paginagrootte voor de volgende test vergroot van 30 naar 40 minuten. De fragmentgrootte blijft weliswaar even groot.

Figuur 5.3 toont aan dat het aantal stops overeenkomt met de afstand tussen begin- en

	Basis methode	Speed-up	Heuristiek
Gemiddelde snelheid (s/query)	4.54	2.87	1.96

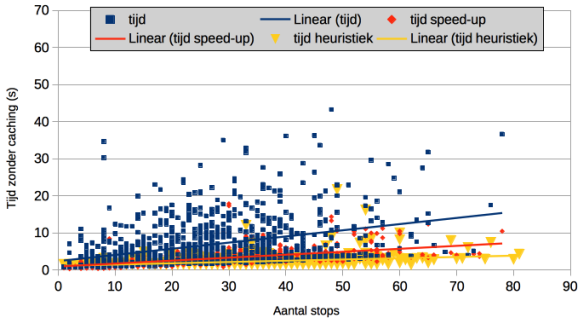
Tabel 5.5: Percentage nuttige connecties ten op zichte van alle gescande connecties voor elke techniek.

eindpunt.



Figuur 5.3: Verband tussen afstand begin-en eindpunt en tijd om te berekenen

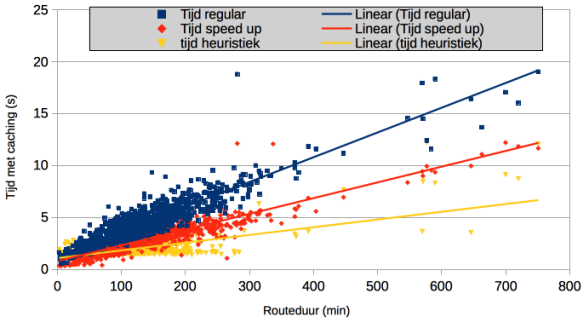
Zonder caching



Figuur 5.4: tijdzondercachingvolgensaantalstops

Figuur **??** toont de snelheid zonder client-side caching aan. In vergelijking met de vorige grafiek (5.2) is er een groter verschil tussen basic LCF's en de speed-up techniek. Dit is te wijten aan de grotere fragmentgrootte in vergelijking met vorige test. In figuur 5.5 zie je dat de meeste routes tussen de 0 en 200 minuten liggen. Een grotere fragmentgrootte zorgt ervoor dat in minder requests het resultaat bekomen kan worden.

De snelheid bij client-side routeplanning hangt niet alleen af van het aantal connecties, maar



Figuur 5.5: De tijd om routes te berekenen in functie van de routeduur.

ook het aantal requests. Door een grotere fragmentgrootte te kiezen, moeten er minder requests gestuurd worden en kan het resultaat sneller berekend worden.

5.2.2 Efficiëntie

CSA gebruikt maar een beperkt aantal van de gescande connecties om een minimale overspannende boom te berekenen. De gegevens uit 5.6 tonen aan dat de speed-up 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 speed-up techniek.

	Basis methode	Speed-up	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.

5.2.3 Dataverbruik

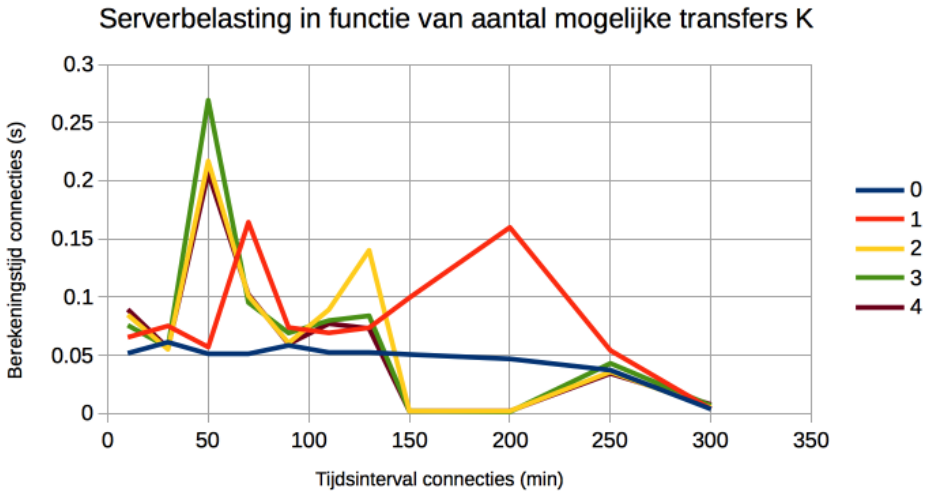
Naast efficiëntie is ook dataverbruik een belangrijk aspect bij routeplanning. Figuur **??** geeft een overzicht hoeveel data er gemiddeld verbruikt wordt met hun onderlinge percentages.

	Basis methode	Speed-up	Heuristiek
Dataverbruik (MB)	2.69	1.68	1.97
Percentage	1	0.62	0.73

Tabel 5.7: Percentage dataverbruik voor elke techniek.

5.2.4 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 K, het maximaal aantal overstappen, en verschillende tijdsintervallen de gemiddelde tijd berekend om connecties op te halen. Zoals je kan zien in 5.6 heeft de server minder last als er meer connecties moeten teruggegeven worden. Hoe groter de straal en hoe groter het tijdsinterval, hoe sneller antwoord kan teruggegeven worden.



Figuur 5.6

5.3 Conclusie

Methode	Snelheid (query/s)	Dataverbruik (MB)	Optimaal
Basis	15.74	2.69	ja
Speed-up	28.70	1.68	ja
Heuristiek	32.89	1.97	nee

Tabel 5.8: Overzicht van de verschillende metrieken per techniek.

Hoofdstuk 6

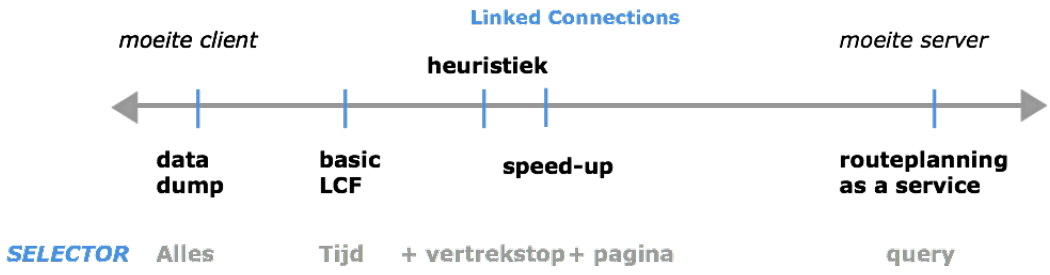
Conclusies en perspectieven

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

In deze masterproef werd een client-side routeplanningssysteem opgezet die gebruik maakt van het Linked Connections framework. Er werd een convertor opgezet die een GTFS feed omzet naar connecties. Met deze data werd de basis implementatie van Linked Connections met enkel een tijdsfilter getest. Dit gaf een gemiddelde snelheid van 4.54 seconden per query (5.5). . De onderzoeksvraag van deze masterproef stelt de vraag of dit sneller kan.

Om een snellere querytijd te bekomen, werden twee andere technieken ontwikkeld. Figuur 6.1 geeft een overzicht van de Linked Data Fragments as met verschillende manieren om routes te plannen. Hierop is te zien hoe de *speed-up* en *heuristische* technieken meer complexiteit van de server eisen om de cliënt belasting in te perken.



Figuur 6.1: Linked Data Fragmenten-as met optimalisatietechnieken

Er werden twee nieuwe termen voor fragmenten geïntroduceerd: basic Linked Connections

Fragments en Neighbouring Linked Connections Fragments (NLCF). Dit laatste fragment maakt gebruik van de tijdsafstand tussen twee stops om connecties te filteren. Zowel de speed-up techniek als de heuristische techniek maken gebruik van deze NLCF's. Uit de resultaten 5.5 en 5.2 kan afgeleid worden dat dit het routeplannen hoogstens dubbel zo snel maakt.

De heuristische methode maakt enkel gebruik van NLCF. De cliënt is verantwoordelijk voor het kiezen van een volgende vertrekstop als het resultaat nog niet gevonden is. Om een goede afschatting te maken werd de belangrijkheid van een station gequoteerd aan de hand van het aantal rechtstreeks bereikbare stops. Dit kan makkelijk berekend worden tijdens het genereren van connecties. Bij deze techniek kan echter niet gegarandeerd worden dat de optimale oplossing gevonden wordt. Slechts 78% van de queries werden optimaal opgelost. Dit kan als een future-work beschouwd worden (zie 7.1).

De speed-up techniek garandeert wel dat de optimale oplossing gevonden wordt. De informatie van de eerste NLCF zorgt ervoor dat er minder connecties gescand moeten worden. Deze techniek maakt gebruik van paginering met hypermedia. Deze fragmenten zijn iets kostelijker om te berekenen voor de server. Dit is echter niet erg, omdat de server dynamisch de grootte van de pagina's en fragmenten kan aanpassen afhankelijk van de serverbelasting. Door een grotere fragmentgrootte te kiezen is het mogelijk om de querytijd te verdubbelen ten opzichte van de basis methode (figuren 5.3 en **??**).

Neighbouring Linked Connections Fragments vergt extra voorbewerking. Het berekenen van de minimale tijdsafstand tussen elke stop vergt weinig extra tijd, omdat dit gebeurt tijdens het genereren van connecties. De extra fase die het minimaal aantal overstappen K berekend is optioneel als extra filter bij de heuristische methode. De heuristische methode geeft van elke stop binnen K een reeks connecties binnen tijdsinterval T terug. Bij grotere netwerken dan de NMBS kan het dus nodig zijn om toch een beperking hierop te stellen. De speed-up techniek beschouwt het aantal overstappen als maximaal om de snelste aankomsttijd te garanderen. Stations die minstens X aantal overstappen nodig hebben, bevinden zich hoogstwaarschijnlijk verder in de tijd waardoor deze connecties toch niet worden teruggeven in het eerste NLCF fragment.

Een andere optimalisatie is het dataverbruik. Door de extra tijdsafstandfilter worden minder nutteloze connecties teruggeven. Dit resulteert in 32% minder dataverbruik bij de speed-up techniek en 27% minder bij de heuristische methode. (zie 5.8). In een *real-world* omgeving zal dit waarschijnlijk nog een grotere rol spelen voor de snelheid.

Een laatste aspect die werd gerealiseerd is een *merger* van verschillende connectiestromen.

Die zorgt ervoor dat connecties van verschillende Linked Connections servers kunnen opgehaald worden door de cliënt. Om intermodaal te kunnen routeplannen moet er een systeem komen die informatie van stopplaatsen van verschillende feeds semantisch kunnen beschrijven.

We kunnen concluderen dat we geslaagd zijn in op de opzet om client-side routeplannen sneller te maken zonder de garantie te verliezen dat de snelste route gevonden is. Niet alleen snelheid, maar ook dataverbruik is een stuk efficiënter. Dit gaat ook niet ten koste van extra voorbewerkingstijd.

Hoofdstuk 7

Future work

7.1 Preprocessing

Tijdens de voorbewerkingsfase wordt voor elke stop de minimale afstand en overstappen tot elke buur bepaald. Een LC server moet connecties binnen een bepaald tijdsinterval teruggeven waarin de snelste route zich bevindt naar elke andere stop. Momenteel werd er verondersteld dat binnen een bepaald tijdsinterval T elke snelste route bepaald kan worden. Een extra feature zou een afschatting zijn van het maximale tijdsinterval waarin connecties moeten teruggegeven worden voor elke stop. Kleine stops hebben bijvoorbeeld een regelmatige verbinding naar een groter station waaruit alle andere stations bereikbaar zijn. Als er binnen het halfuur een connectie bestaat, is het enkel nuttig om connecties binnen het halfuur op te vragen. Voor verafgelegen stations, zoals Basel voor de NBMS, is een groter tijdsinterval nodig. Door deze informatie kan er met meer zekerheid en efficiënter enkel de nuttige connecties teruggeven worden.

7.2 Multicriteria queries

Momenteel werd er enkel onderzocht hoe de snelste route gevonden kan worden. Een nieuwe pareto-optimale route zou kunnen beschouwd worden die ook rekening houdt met het maximaal aantal overstappen. CSA zou moeten aangepast worden zodat elke stop meerdere mogelijke oplossingen bijhoudt zodat er niet louter naar de tijd gekeken wordt, maar ook naar de trip. Voor de optimalisatie is dit ook een interessante uitbreiding wegens de manier de data werd voorbewerkt. Elke stop houdt de minimale tijd en het minimaal aantal overstappen naar elke andere stop bij. Er kan zo een extra HTTP parameter toegevoegd worden zodat de cliënt kan aanduiden hoeveel overstappen maximaal gemaakt mag worden.

Momenteel wordt er eerst een HTTP request gestuurd om de juiste URL te bekomen van een fragment. Er zou metadata in deze request toegevoegd kunnen worden over de stops die met 0, 1... overstappen bereikbaar zijn. Zo kan de cliënt een betere afschatting kunnen doen van het maximaal aantal overstappen.

7.3 Intermodaliteit

Een Linked Connections server is verantwoordelijk voor het aanbieden van connecties van een (of meerdere) provider(s). De cliënt 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 zo kan er interoperabiliteit onstaan tussen datasets. Een cliënt 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. Hieruit blijkt dat de vraag naar metadata steeds groter zal zijn. Enkele voorbeelden:

- faciliteiten zoals rolstoeltoegankelijkheid
- locatie van perrons
- mogelijke wandelafstanden
- gebied waar de publieke vervoersmaatschappij actief is, bijvoorbeeld in GeoJSON-formaat
- welk soort voertuig

Op basis van deze metadata kan een cliënt slimme beslissingen nemen om connecties op te halen van de juiste providers. Met behulp van de *merger* die in hoofdstuk 3.6 besproken wordt, kunnen de connecties makkelijk gesorteerd samengevoegd worden voor CSA.

7.4 Grote netwerken

Het Connection Scan Algorithm (CSA) is niet schaalbaar voor zeer grote netwerken. Linked Connections werkt snel voor lokale netwerken. Een oplossing hiervoor is accelerated CSA die we besproken hebben in de literatuurstudie (2.4.2). Deze optimalisatie vereist echter preprocessing. Dit zou toepasbaar kunnen worden binnen Linked Connections door een *accelareted* Linked Connections server te ontwerpen. Wanneer de cliënt wil query'en over lange afstanden kan dit eerst hierop gedaan worden. Voor lokale queries kan er dan op de huidige implementatie overgeschakeld worden.

7.4.1 Realtime informatie

Extra transformer toevoegen die trip van connectie checkt -¿ update -¿ hersorteren wel

Bibliografie

[1]

[2] Dorothea Wagner Ben Strasser. Connection scan accelerated. 2014.

[3] 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.

[4] Thomas Pajor Renato F. Werneck Daniel Delling, Julian Dibbelt. Public transit labeling.

[5] T. Pajor Delling and R. F. F. Werneck. Round-based public transit routing. 2012.

[6] Google Developers. *GTFS referentie*. Google, 2015.

[7] Sabine Storandt Hannah Bast, Matthias Hertel. Scalable transfer patterns.

[8] 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.

Woordenlijst

GCD Greatest Common Divisor. 58

Lijst van figuren

1.1 Linked Data Fragments as: huidige oplossingen om routes te plannen. Cliënt beschikken ofwel over alle data in een dump, ofwel over een service die het route-plannen voor zich neemt. 3
1.2 Voorbeeld van een connectie. Deze bestaat uit een vertrek- en aankomstplaats, resp. met vertrek- en aankomsttijd 3
1.3 Linked Connections opent nieuwe trade-offs tussen cliënt en server aan. 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 Linked Connections fragmenten. Met hydra:nextPage links kan makkelijk het volgende fragment gevonden worden. 22
3.2 Linked Data Fragmenten-as met Linked Connections 23
3.3 Overzicht hoe connecties worden gegenereerd uit een GTFS feed. Deze wordt ingeladen in een MySQL databank. Met de scripttaal PHP worden connecties gegenereerd. 24
3.4 Per dag worden de corresponderende service ID's, trip ID's en route ID's berekend. De vertrek-/aankomststopplaats met respectievelijk vertrek-/aankomsttijd worden berekend uit een verzameling stoptimes. 25
3.5 Opstelling van een client en twee Linked Connections servers die elk verantwoordelijk zijn voor een modi. 29
3.6 Overzicht merger 30

4.1 Neighbouring Linked Connections Fragment bevat connecties van vertrekstop en buren binnen een bepaald tijdsinterval. 37
4.2 Overzicht van resources van speed-up optimalisatietechniek. 41

5.1 Snelheid en grootte hangen af van de ingestelde tijdsinterval van basic Linked Connections Fragments. 45
5.2 De tijd om een route te berekenen in functie van het aantal stops voor de NMBS. 47
5.3 Verband tussen afstand begin-en eindpunt en tijd om te berekenen 48
5.4 tijdzondercachingvolgensaantalstops 48
5.5 De tijd om routes te berekenen in functie van de routeduur. 49
5.6 50

6.1 Linked Data Fragmenten-as met optimalisatietechnieken 51

Lijst van tabellen

3.1 Tijd om een GTFS feed in te laden in een MySQL databank. 26
3.2 Tijd om connecties te genereren voor een dag. 27

4.1 Tijd om buren van alle GTFS stops te berekenen. 35
4.2 Overzicht van enkele stations van de NMBS met hun aantal rechtstreeks bereikbare stops en gemiddeld aantal stoptijden per dag. 39

5.1 Vergelijking tussen een lokale server een productieserver. 43
5.2 Grootte (MB) en aantal connecties afhankelijk van tijdsinterval van basic LCF. . 44
5.3 Grootte van een connectie. 45
5.4 Tijdsintervallen voor de verschillende technieken. 46
5.5 Percentage nuttige connecties ten op zichte van alle gescande connecties voor elke techniek. 48
5.6 Percentage nuttige connecties ten op zichte van alle gescande connecties voor elke techniek. 49
5.7 Percentage dataverbruik voor elke techniek. 49
5.8 Overzicht van de verschillende metrieken per techniek. 50