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. Erik Mannens, Prof. Rik Van de Walle
Scriptiebegeleider: Dr. Ir. Ruben Verborgh, Ing. 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

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. Zoals je ziet, moet routeplanning met meerdere factoren rekening kunnen houden. Om dit allemaal te berekenen wordt er voor 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.

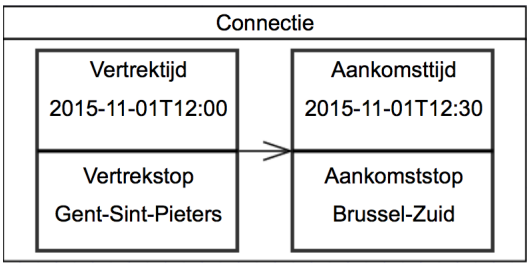
[1] http://www.decroo.belgium.be/nl/groen-licht-voor-federale-open-data-strategie-overheidsdata-voortaan-vrij-beschikbaar

– 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.

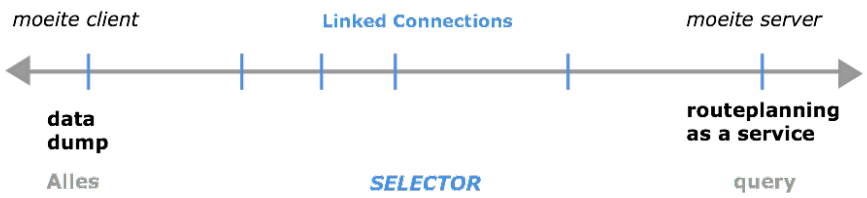
Linked Connections is een manier om transportdata te publiceren zodanig dat het mogelijk is om een route te berekenen hieruit. De basiseenheid is een connectie (zie 1.2). Een connectie is de verbinding tussen een vertrek- en eindstop zonder onderbreking, met respectievelijk een vertrek- en aankomsttijd.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



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

connecties ter beschikking stelt. Een probleem hierbij is dat connecties enkel opvraagbaar zijn 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 en/of meerdere datastromen.

2. De snelheid van het algoritme hangt af van het aantal connecties die gescand moet 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 5 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 6 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) [2]. 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 [2] 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 5).

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.

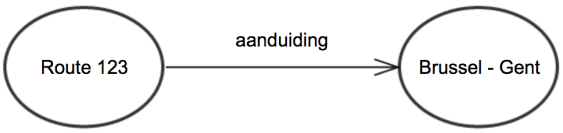
2.2 Semantisch web

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)[**?**] 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.

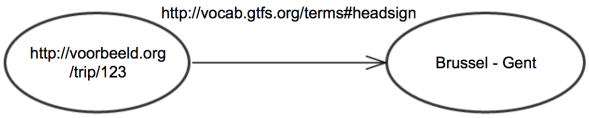
Net zoals je bij object georinteerd programmeren worden er klasses met bepaalde attributen gemaakt om concepten uit de echte wereld zo reëel mogelijk weer te geven. In de wereld van het semantisch web wordt hiervoor gebruik gemaakt van vocabularia.

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"
```

[2] vocab.gtfs.org/terms

```
}
```

Listing 2.1: Voorbeeld trip in JSON.

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 [**?**] 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:

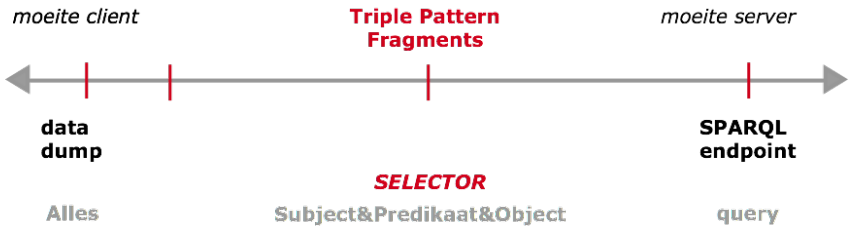
- Hypermedia controls. Dit zijn URL's die meer informatie bevatten over bepaalde entiteiten in het fragment, maw informatie over gerelateerde Linked Data Fragmenten. Voor datadumps en het resultaat van een SPARQL query komt dit neer op het derefereren van de URL die een bepaald object voorstelt.

- 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

2.4.1 Dijkstra

2.4.2 Heuristieken

2.4.3 CSA

Alternatieve wegen

Realtime informatie

2.5 Bestaande routeplanners

2.5.1 Transitland

OneStop-ID's

2.5.2 Navitia

Hoofdstuk 3

Linked Connections

Linked Connections [1] is een framework die *client-side* routeplanning mogelijk maakt. Volgende sectie verduidelijkt hoe de huidige implementatie werkt. Vervolgens wordt uitgelegd hoe connecties worden gegenereerd uit een GTFS feed. Ten slotte wordt verduidelijkt hoe intermodaal routeplannen werkt.

3.1 Principe

Stel je voor dat je de tram moet nemen naar je werk: je stapt ergens op, wacht vijf stophaltes en aan de zesde stophalte stap je af. Dit is een route die uit zes connecties bestaat. Een connectie is de verbinding tussen een vertrek- en eindstop zonder onderbreking. Bij zo'n connectie horen respectievelijk ook een vertrek- en aankomsttijd. Een route kan berekend worden met het CSA algoritme (zie 2.4.3) die een query en een gesorteerde lijst op vertrektijd als inputwaarde vereist.

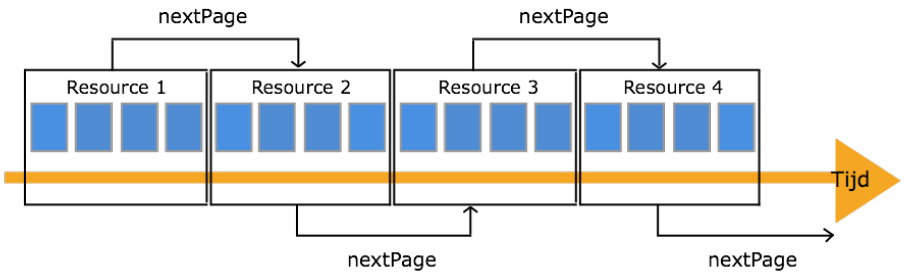
Een Linked Connections server presenteert deze gesorteerde connecties in de vorm van Linked Data Fragments (2.2.7) aan de cliënt. Momenteel is het enkel mogelijk om de vertrektijd van een query mee te geven als parameter. Met behulp van texttithydra:nextPage links kan de cliënt makkelijk opeenvolgende fragmenten ophalen. (zie figuur **??**).

```
http://voorbeeld.org/connecties?vertrekTijd=2015-10-21T11\%3A30
```

Listing 3.1: HTTP URL van fragment met Linked Connections.

Het publiceren van de data gebeurt met behulp van drie technologiëen:

- REST zorgt ervoor dat resources cachebaar zijn. Hier zijn fragmenten de resources. Het aantal mogelijke URI's is afhankelijk van het tijdsinterval van de fragmenten. Als T het



Figuur 3.1: Connecties zijn gesorteerd onderverdeeld in fragmenten die verbonden zijn met hydra:nextPage links.

tijdsinterval in minuten is van de fragmenten, dan kan het aantal mogelijke URI's berekend worden met formule 3.1.

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

Als T = 10 min, dan zijn er 144 verschillende fragmenten. De meeste ov-bedrijven 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 ook voor het makkelijk uitbreiden van connecties. Dit kan gaan van een koppeling met geonames tot koffiebars in de buurt.

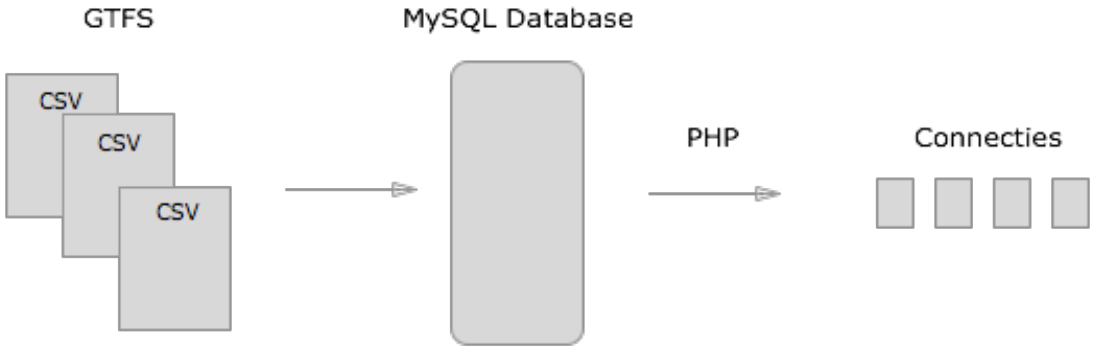
- Het semantisch web zorgt voor de semantische interoperabiliteit van de connecties en de API. Dit laat toe om generische clients te bouwen.

Op de LDF-as (zie 3.2) staat huidige implementatie aan de linkerkant. 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. Later (5.1) zal de exacte performantie verduidelijkt worden.

Figuur 3.2: Linked Data Fragmenten-as met Linked Connections

3.2 Voorbewerking

Gelinkte connecties worden berekend uit een GTFS *feed*. De convertor in deze masterproef maakt gebruik van een MySQL-databank om queries op uit te voeren. [1] Het is belangrijk op te merken dat connecties niet gesorteerd moeten zijn bij het voorbewerken. Deze worden later in de databank van de Linked Connections server ingeladen die zelf sorteert. Figuur 3.3 toont een overzicht van de verschillende stappen om connecties te genereren.



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 hieruit.

Connecties worden dag per dag berekend. Ofwel worden start- en einddatum meegegeven als parameters, ofwel worden start- en einddatum van de GTFS feed zelf genomen. Vervolgens worden alle services uit *calendar* en *calendar_dates* van een bepaalde dag opgehaald. Bij elke service hoort een bepaalde *route* en een verzameling *trips* die dan worden opgehaald uit *trips*. Een laatste stap is het overlopen van *stoptimes.txt*. Uit de combinatie van twee opeenvolgende stoptimes kan een connectie berekend worden. Lijst 3.2 bevat een voorbeeld van twee opeenvolgende stoptijden van een bepaalde trip.

```
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: Vereenvoudigde stoptimes in CSV.

[1] Ondertussen is een veel snellere convertor gemaakt die volgens een ander principe werkt, zonder databank. Zie github.com/linkedconnections/gtfs2lc

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.

Codevoorbeeld 3.3 toont hoe een connectie eruit ziet bij output. Merk op dat een '@context' niet is toegevoegd en ook niet in een graaf zit. In subsectie 3.2.1 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": "2015-10-21T06
      :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: Connectie voor een bepaalde trip met bijkomstige route in JSON-LD.

De tijdszone van vertrek- een aankomsttijd staat in Coordinated Universal Time (UTC). Er moet altijd rekening gehouden worden met de tijdszone van de GTFS feed. In dit voorbeeld 3.2 werd een feed uit België genomen. Om de Belgische tijd (UTC+1) om te zetten naar UTC moet er een uur afgetrokken worden van de tijd die vermeld staat in GTFS stoptijden.

Een van de moeilijkheden voor het genereren van connecties was het ophalen van trips die voor middernacht vertrekken, en dus geclassificeerd zijn onder die dag, maar na middernacht pas stoppen. GTFS lost dit op door de tijd na middernacht door te tellen, 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. Om dit op te lossen worden er twee extra vlaggen toegevoegd in de databank of de vertrek- en aankomsttijd van een stoptijd voor of na middernacht plaatsvinden.

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 (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.

	Tijd (min)	Connecties
NMBS	0.21	55479
NS		
De Lijn	14.97	1049186

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

3.2.1 JSON-LD stream

Om de connecties als Linked Data te publiceren werd er voor gekozen om JSON-LD te gebruiken. JSON wordt beschouwt als het *de facto* standaardformaat 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 [2]. Deze geeft aan dat de context van een object met '@context' van toepassing is op alle andere objecten van het document. Zo moet er maar eenmalig een context opgegeven worden en wordt impliciet verondersteld dat de andere objecten deze context gebruiken. De voorbewerker kan nu connecties als Linked Data wegschrijven zonder extra dataverlies en gestroomlijnd. Dit stroomlijnen is belangrijk om de data later lijn per lijn te kunnen inladen in een databank.

3.3 Client

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

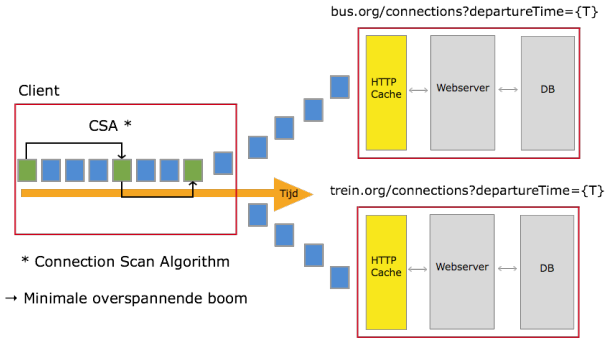
```
var planner = new window.lc.Client({"entrypoints" : ["http://
    example.linkedconnections.org/"]});
planner.query({
      "departureStop": "Brussel-Zuid",
      "arrivalStop": "Gent-Sint-Pieters",
      "departureTime": new Date("2015-11-05T10:00")
      }, function (stream) {
        stream.on('result', 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 client op te zetten in JavaScript.

Linked Connections van verschillende ov-bedrijven worden gedistribueerd opgesteld. De cliënt is verantwoordelijk voor het samenvoegen van connecties. In figuur 3.3 staat een overzicht van een client - meerdere servers opstelling. Rechts staan twee Linked Connection servers, de ene verantwoordelijk voor de connecties van een vervoersmaatschappij van bussen, de andere van

[2] https://github.com/pietercolpaert/jsonld-stream

treinen. Links staat een client die de fragmenten ophaalt van beide servers. Voor het scannen zelf kan plaatsvinden, moeten deze samengevoegd worden. Daarna kan de snelste route met Connection Scan Algorithm (CSA) gepland worden.



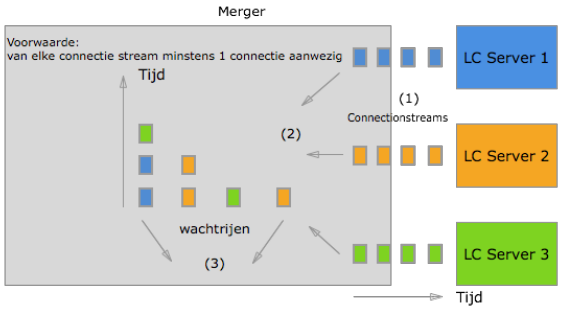
Figuur 3.5: Opstelling van een client en twee Linked Connections servers.

3.3.1 Merger

Een merger voegt meerdere stromen van gesorteerde connecties samen tot een stroom. Dit is noodzakelijk voor de cliënt om CSA te kunnen toepassen.

Connecties komen onder de vorm van een *stream* binnen. Zo'n stream werkt asynchroon met events. Deze worden opgeworpen wanneer bijvoorbeeld data beschikbaar is. Figuur 3.6 toont een overzicht van een merger. Deze luistert (1) naar de verschillende datastromen tot deze data vrijgeven. De connecties worden toegevoegd in wachtrijen. Wachtrijen (2) hebben als voorwaarde dat ze zelf gesorteerd zijn. Datastromen worden telkens gepauzeerd na het opvangen van data. Dit is noodzakelijk om de cliënt beslissingstijd te geven. Zo kan er beslist worden om een bepaalde connectiestroom uit te schakelen of toe te voegen. Een andere reden waarom de merger de connectiestromen pauzeert, is het feit dat de wachtrijen minstens een connectie moet bevatten van elke connectiestroom. Wanneer de merger connecties teruggeeft, worden alle connecties van elke wachtrij met dezelfde lokaal minimale vertrektijd teruggegeven. Zo blijven alle wachtrijen synchroon.

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



Figuur 3.6: Overzicht merger

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

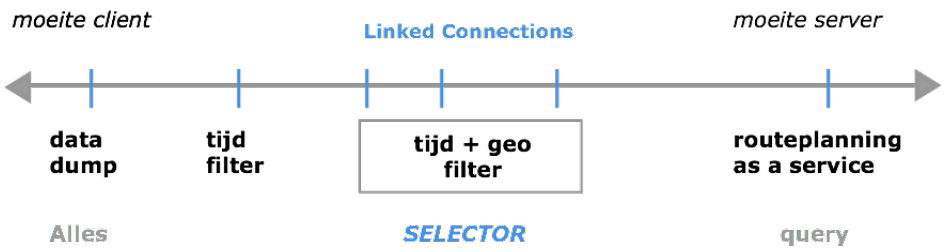
Listing 3.5: Voorbeeldcode van een merger. Deze voegt meerdere stromen van connecties samen.

3.4 Voor- en nadelen

- Routeplanning is een dataprobleem geworden. Datapubliceerders zijn verantwoordelijk voor het publiceren van connecties en bijhorende data. Zo kan er makkelijk data toegevoegd worden via hypermedia: *point of interests*, rolstoelvriendelijkheid etc. De cliënt bepaalt zelf welke informatie deze wil gebruiken.
- Connecties zijn makkelijk distribueerbaar over meerdere servers.

Hoofdstuk 4

Optimalisatie



Figuur 4.1: Linked Data Fragmenten-as met Linked Connections optimalisatietechniek

4.1 Geofiltering

4.2 Server

4.2.1 Cachebaarheid

Aantal mogelijke URI's voor 1 dag met enkel tijdsfilter en fragmenten om de 10 minuten: 24u * 60 min / 10 min = 144 URI's

4.2.2 Grootte fragmenten

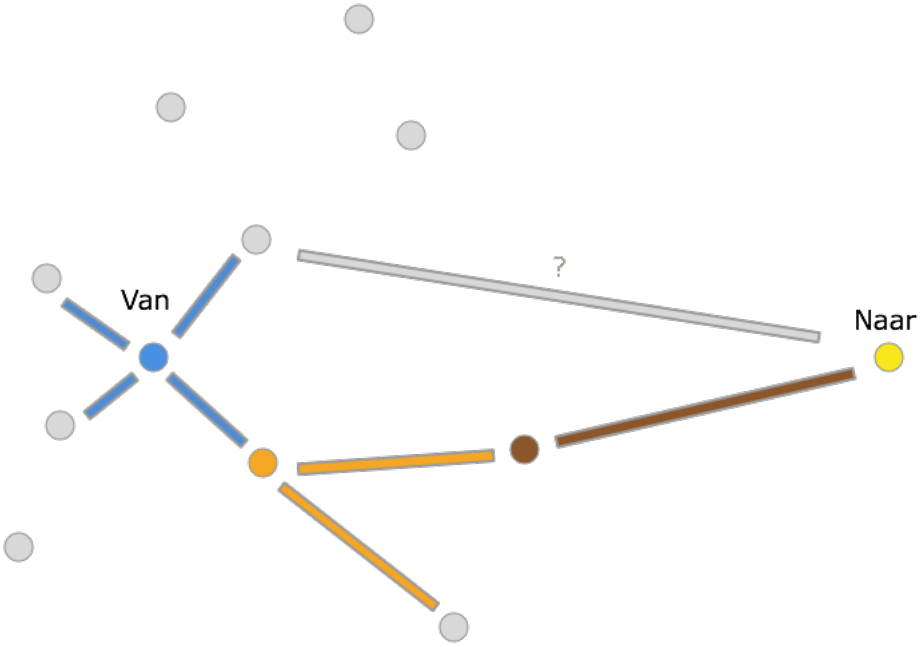
Hoeveel connecties per fragment best? Hangt samen met snelheid Snelheid requests ¡ snelheid verwerken connecties ? Is er bottleneck?

Verhouding aantal connecties/request -¿ tijdsinterval aanpassen

4.2.3 Burenconnecties

4.3 Client

4.3.1 Heuristiek



Figuur 4.2: Heuristiek client voor volgende vertrekstop

4.4 Voor- en nadelen

Hoofdstuk 5

Resultaten

- Gebruikte datasets - Wandelwegen - Queries

5.1 Origineel

5.2 Optimalisatie

Hoofdstuk 6

Conclusies en perspectieven

6.1 Conclusie

Hoofdstuk 7

Future work

7.1 Overstappen

Doordat identifiers bij GTFS niet persistent zijn, is het een uitdaging om stops van verschillende datasets te koppelen met elkaar.

7.2 metadata

- Vehicle type Om client slimmer te maken moet deze weten wat voor voertuigen een bepaalde Linked Connections server aanbiedt. Deze informatie kan uit GTFS gehaald worden en meegegeven als metadata aan de client.

- Gebied van datafeed Welk GeoJSON gebied overspant een GTFS feed.

7.3 Client heuristiek

Bibliografie

[1] 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.

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

[3] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Low-cost queryable Linked Data through triple pattern fragments. In *Proceedings of the 13th International Semantic Web Conference: Posters and Demos*, volume 1272, pages 13–16, October 2014.

Lijst van figuren

Lijst van tabellen