SCALABLE PUBLIC TRANSIT ROUTE PLANNING ON THE WEB BASED ON RAPTOR

Tibo Vanheule
Supervisors: prof. dr. ir. Ruben Verborgh, prof. dr. Pieter Colpaert, Julian Andres Rojas Melendez

ABSTRACT

The goal is to shift towards a shared responsibility strategy, allowing the client to calculate the ideal routes while the server manages the data. This involves determining the relevant data to send to the client, experimenting with fragmentation, and using an ontology to support multiple datasets/sources.

INTRODUCTION

As a first-year student at the University of Ghent in Belgium, you probably live in a dorm and may not be familiar with the area. You might wonder about the best restaurants and leisure activities, as well as how to get to your classes, exams, or home. Public transportation is a cost-effective and eco-friendly option for students without a car.

To plan your routes effectively, you might use route planners like `nmbs.be` or `maps.google.com`, which rely on algorithms to calculate the best routes based on various criteria. However, public transportation poses unique challenges compared to road networks, such as considering factors like transfer numbers and cost of different transportation modes.

Most route planners use a centralized data strategy, collecting and integrating datasets for different transit modes. However, this approach limits the usability of third-party datasets.

The goal is to shift towards a shared responsibility strategy, allowing the client to calculate the ideal routes while the server manages the data. The challenge lies in determining the relevant data to send to the client to minimize bandwidth usage and processing load. Experimenting with fragmentation to send only required data and using an ontology to support multiple datasets/sources are also part of the goals.

In summary, the aim is to create a shared responsibility between the client and server for route calculation, optimize data transmission to the client, and support multiple datasets/sources using an ontology.

RELATED WORK

Route planning algorithms

The history of routing algorithms for transport planning saw significant milestones, such as the development of the Dijkstra algorithm in 1958 and its subsequent improvement in 1987. However, progress in route planning for public transport was limited until the 9th Dimacs challenge [1] in 2005, which provided a substantial road network dataset for researchers to work with.

In 2009, the paper "Car or Public Transport: Two Worlds"[2] identified unique challenges in public transport routing, using five common "tricks of the trade" for fast routing on transportation designed explicitly for road networks. These techniques included bidirectional search, exploiting hierarchy, graph contraction, goal direction, and distance tables. Regardless, these tricks are difficult to apply to public transport routing due to the absence of hierarchy and computational inefficiencies in local searches.

A significant breakthrough came with the introduction of transfer patterns[3], the first algorithm capable of solving queries in milliseconds on a poorly structured transport network. Transfer patterns uses two key ideas: (1) Of the shortest paths, many share the same transfer pattern (2) Direct connections that do not require a change of vehicle can be looked up rather quickly. Improvements to this algorithm are Frequency-Based Search [4] and Scalable transfer patterns [5].

The Connection Scanning Algorithm [6] organizes data as a single array of connections. It is not graph-based like RAPTOR [7] and is centred around elementary connections. The algorithm uses a straightforward approach, assembling timetable connections into an array sorted by departure time and scanning connections.

Trip-based public transit routing [8] uses trips and transfers as fundamental building blocks, with precomputed transfers to speed up queries. Additionally, the authors updated the trip-based model using techniques like transfer patterns and hub-labeling.

Ontologies

A small study to identify a fitting ontology observed that many of the studied transport ontologies are tailored to specific use cases, such as urban freight and lack a broad domain.

"Ontologies of Wayfinding: A Traveler's Perspective"[17] describes two ontologies of "wayfinding" for multiple transportation modes. However, it focuses solely on concepts and does not define any properties, relations, or axioms. Additionally, it is only intended for urban areas.

"An ontology and algorithm for a server-side Public Transport Query System (PTQS)" [10] to provide timely information services for travellers. This ontology identifies four concepts: vehicle, route, station, and organization. While it was a good start in 2005, it does not capture the complete functional requirements to solve queries in an acceptable time.

Similarly, "A public transportation ontology to support user travel planning" [13] is a domain ontology with limited multi-modal support, but lacks multi-operator support. Although it could potentially support RAPTOR, it adds complexity by defining six different types of journey patterns.

On the other hand, "Introducing the Public Transport to the Web of Data" [9] acknowledges the existence of Transmodel[18] and presents an ontology derived from multiple data models. The ontology is not directly available.

The "TRANSIT" [19] vocabulary for describing transit systems and routes is based on GTFS but has not been updated since 2011 and has been superseded by linked-GTFS. The linked-GTFS ontology [14] maps GTFS in CSV reference to RDF and includes important classes such as Feed, Agency, Stop, Station, Route, Trip, StopTime, and Service.

The Transmodel ontology [15] is aligned with CEN standards and European directives and supports journey patterns. However, it may contain too many concepts, which makes it difficult to find relevant information. On the other hand, the OSLO Mobility ontology [16], developed by OSLO, is based on the EPIP profile and is less broad than the Transmodel ontology. It describes network topology and timetables for public transportation, detailing route planning, cartography, and stop lookup. The ontology provides information on lines, service points, planned stop points, and timetables, as well as service journey day types.

An overview of all ontologies can be found in Table 1.

APPROACH

The implementation was divided into two parts, including the implementation of Raptor for the browser and the conversion of the GTFS data. The implementation of Raptor involved analyzing the data used by the existing implementation [20].

The most important classes identified for refactoring in later steps included RaptorAlgorithmFactory, QueueFactory, RouteScanner, RaptorAlgorithm, and ScanResults. Additionally, a new class called DataLoader was created to handle all data in one class and process new fragments to populate the needed data structures.

Adapt for the browser

The browser can run compiled Typescript or JavaScript natively but now also supports modules [21]. But to further simplify the import of our Node.js application in the browser, we bundle the script files into one using Esbuild[22], an extremely fast bundler and TypeScript compiler. Other features include minify, target, drop:console, and source maps. The target option ensures that all the Node.js-specific APIs are converted to code the browser supports.

Data parser

In our project, we developed a program to convert GTFS data to the Oslo ontology, which is based on an EPIP profile and is compatible with Transmodel. We simplified the overview (Figure 1) by removing unnecessary entities related to dead runs and shape information. We desire to provide essential data for route calculation only, starting with the scheduled stop point as a start point, which contains passing times and service links.

Important entities:

- **Line and route**: A representation of a line or route used to conceptualise a path in a network.
- **Service journey pattern**: Describes the order of points in a route. For each point in this class, a passenger can get off or on.
- **Stoppoint in service journey pattern**: Describes a visit of a service to a scheduled stop point. Give more information about if passengers can be dropped off or picked up.
- **Schedueled Stop Point**: Conceptualizes a point where passengers visit. It also links with a physical stop point, defined in another vocabulary.
- **Service journey**: is a vehicle journey allowing passengers to get on or off.
- **Passing time**: Describes the visit in time of the vehicle to a stopping point.
- **Service Calendar**: Describes on which days a trip is

We encountered issues with the Oslo JSON-LD context, which contained empty types. We corrected this and stored the updated version in MongoDB[1]. Overall,

[1] `https://mx2.tibovanheule.space/ontology/context`

Ontology	Journey patterns?	Multi-modal?	Multi-operator?	Designed for use with routeplanners	directly available for reuse?	Citation
Intoducing the public transport domain to the web of data	yes	yes	yes	yes	no	[9]
An Ontology-based public transport query system	no	no	yes	yes, query systems	no	[10]
iCity Ontology	yes	no	no	no, urban systems	GitHub	[11]
Genclon	yes	no	no	no, city logistics	no	[12]
Transportation ontology definition and application for the content personalization of user interfaces	yes	yes	yes	yes	no	[13]
linked-gtfs	yes	yes	yes	yes	yes	[14]
Transmodel ontology	yes	yes	yes	yes	GitHub	[15]
OSLO Mobility: Timetables and Planning	yes	yes	yes	yes	yes	[16]

Table 1: An overview of the ontologies for

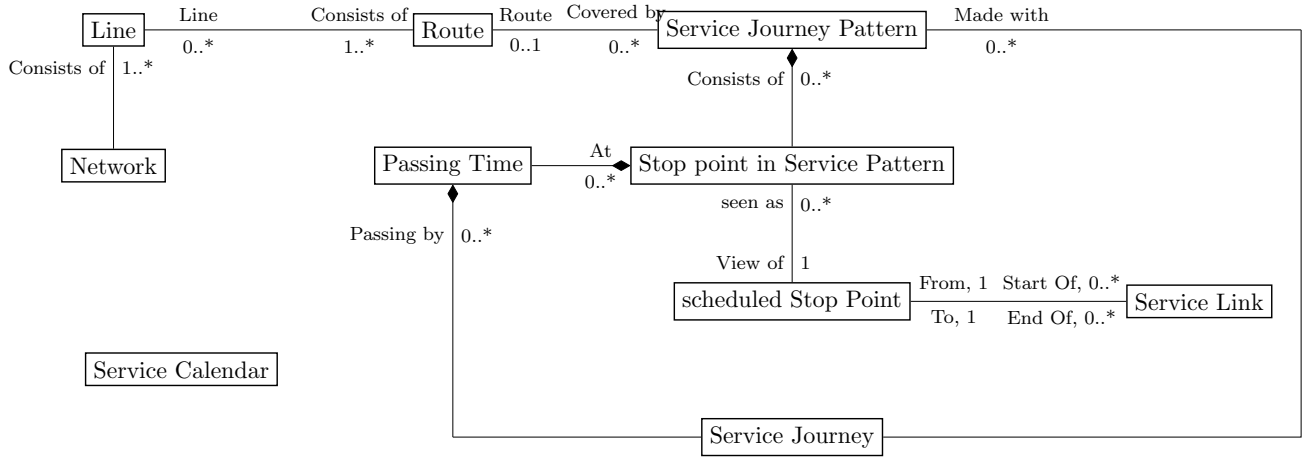

Figure 1: Simplified overview of the OSLO ontology

our program facilitates the conversion of GTFS data to the Oslo ontology, paving the way for enhanced data utilization.

To process GTFS data, so we developed our own GTFS streamer using NodeJS streams [23],node-stream-zip [24] and csv-parse[25] modules. This allowed us to extract files from the GTFS zip as needed. The parsed data was then efficiently written to MongoDB using bulkWrite operations, overcoming memory issues. MongoDB's flexibility and scalability made it a great choice for storing our JSON-LD fragments.

We used this process to create a merged view for our entry point, with connected entities embedded in scheduled stop points for quick lookups using Raptor. We added the rest of the information to separate collections in MongoDB, allowing us to retrieve either the entity alone or with referenced entities as a graph. Indexing for embedded documents is using a lot of storage, but it's necessary to speed up creation time.

Fragmenting

To start fragmenting, we can use MongoDB's spatial capabilities, including near and within operations. However, MongoDB requires storing coordinates as GeoJSON, and creating a new index on the GeoJSON field, which affects storage and insert speed. We can use the $*near* operator for selecting stop points within a certain distance, but this approach has drawbacks, such as not testing reachability and overlapping fragments. Despite quick query response times with a "2dsphere" index, the response time is still influenced by the chosen distance.

One strategy is to work with both source and target stations to create a Linestring, buffer it to create a Polygon, and select scheduled stop points within this shape using the MongoDB $*within* operator. However, this approach also requires GeoJSON data, a 2dsphere index, and may return stop points that shouldn't be sent to the client. Adjusting the buffer size or adapting the approach can mitigate these issues.

The reachability strategy uses MongoDB's $*graphLookup* operator for finding reachable neighbors. It involves a complex implementation with some drawbacks, such as the BSON limit issue. To overcome this, an own version of the $*graphLookup* operator was made in JS, but was slow. A precomputed reachable fragmentation strategy is suggested for faster response times.

EVALUATION

The Oslo organization has defined constraints for conforming to an application profile, including cardinality and vocabulary usage. They also provide a SHACL file for ontology constraints which we tested by transforming a JSON-LD document to N-Quads and validating it using rdf-validate-shacl[26]. We encountered some violations due to miswriting a field.

Fragmentation evaluation

We conducted a fragmentation evaluation using Python to measure response time and request size for different fragment strategies. The reachable strategy proved to be the fastest, and the precomputed version saved time compared to the JavaScript version. The geospatial strategies, where slower than expected (index).

Raptor evaluation

Then we conducted an evaluation of the RAPTOR algorithm by randomly selecting origin-destination station pairs and running RAPTOR on them 50 times. The evaluation showed varying performance for different fragmentation strategies.

In Figure 2, there is a notable increase in total and raptor time from $K = 1$ to $K = 2$, rising from 12 seconds to an average of 50 seconds. This is expected as direct trips at $k = 1$ require less data fetching and exploration compared to $k = 2$. The difference between higher k values is less significant, indicating a less steep increase.

Geospatial strategies, particularly the Within strategy with a 25 km buffer, exhibit high response time and size issues. The Reachable strategy performs well, while the combination of Within and Reachable Neighbours with a depth of three performs the best.

CONCLUSION AND FUTURE WORK

The execution time of a query is related to the number and size of fragments requested. Comparing fragmentation evaluation to performance results, we see correlations. For instance, the worst performer in the raptor evaluation "within" also performs poorly in fragmentation evaluation. When the size of fragments is too small, such as Near with 5 km, quickly perform worse for queries requiring higher K values due to Raptor algorithm requesting more fragments, leading to worsened performance.

Choosing the perfect fragmentation strategy is challenging and depends on response time, size, and overlap. For $k = 1$, smaller options like a 5 km buffer or a depth of one for reachable neighbors are suitable. For higher K values, integrating precomputed reachable with "within" provides a balanced solution. It is important to note that different strategies may be applicable to alternate transportation networks, such as urban bus networks due to differing network densities.

Future work

To further improve the response size, let the client store a list of IDs that have been gotten. It could be interesting to use such a list for each fragmentation request to filter out stations. This would essentially remove all overlap in each fragment and further decrease the size.

Further combinations of fragmentation could be analysed—for example, use Within only for the initial request. When a trip leads to a fragment outside the fragment, we could use one of the reachable strategies. This would also decrease a lot of overlap.

The server load, especially memory, is relatively high; a suspicion is the MongoDB driver of Node.js. Although it makes sense to write the client in js, it makes less sense for the server. We could, for example, rewrite it into cpp.

References

[1] "9th DIMACS Implementation Challenge: Shortest Paths," Jun. 2017. [Online]. Available: https://web.archive.org/web/20170606011114/http://www.dis.uniroma1.it/challenge9/format.shtml

[2] H. Bast, "Car or Public Transport—Two Worlds," in *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, ser. Lecture Notes in Computer Science, S. Albers, H. Alt, and S. Näher, Eds. Berlin, Heidelberg: Springer, 2009, pp. 355–367. [Online]. Available: https://doi.org/10.1007/978-3-642-03456-5_24

[3] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger, "Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns," in *Algorithms – ESA 2010*, ser. Lecture Notes in Computer Science, M. de Berg and U. Meyer, Eds. Berlin, Heidelberg: Springer, 2010, pp. 290–301.

[4] H. Bast and S. Storandt, "Frequency-based search for public transit," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 13–22. [Online]. Available: https://doi.org/10.1145/2666310.2666405

[5] H. Bast, M. Hertel, and S. Storandt, "Scalable Transfer Patterns," in *2016 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, ser. Proceedings. Society for Industrial and Applied Mathematics, Dec. 2015,

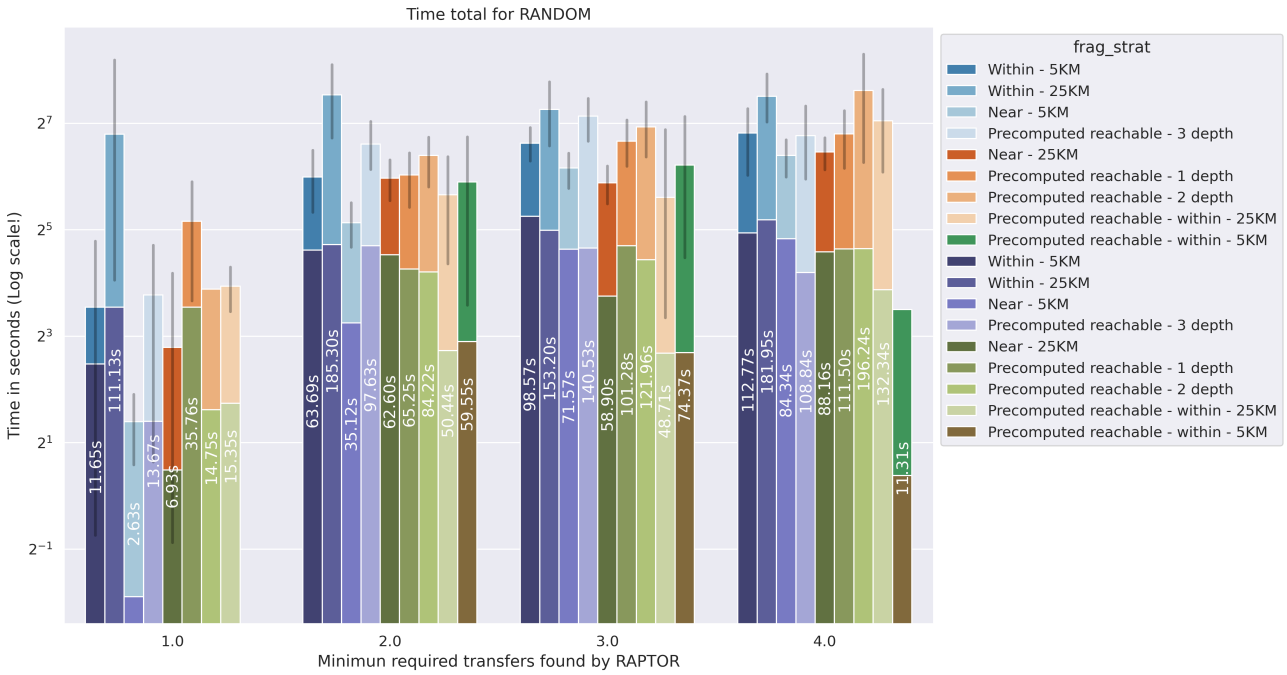

Figure 2: Avarage time needed to solve a query. We included the total needed time and the time for RAPTOR to execute. The difference between the two is due to data fetching and processing. Note that the y-axis is logarithmic. The black bars are error bars. the value in the bars is the total time.

pp. 15–29. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611974317.2

[6] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Intriguingly Simple and Fast Transit Routing," in *Experimental Algorithms*, ser. Lecture Notes in Computer Science, V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, Eds. Berlin, Heidelberg: Springer, 2013, pp. 43–54.

[7] D. Delling, T. Pajor, and R. F. Werneck, "Round-Based Public Transit Routing," *Transportation Science*, vol. 49, no. 3, pp. 591–604, Aug. 2015, publisher: INFORMS. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/trsc.2014.0534

[8] S. Witt, "Trip-Based Public Transit Routing," 2015, vol. 9294, pp. 1025–1036, arXiv:1504.07149 [cs]. [Online]. Available: http://arxiv.org/abs/1504.07149

[9] C. Keller, S. Brunk, and T. Schlegel, "Introducing the Public Transport Domain to the Web of Data," in *Web Information Systems Engineering – WISE 2014*, ser. Web Information Systems Engineering – WISE 2014, B. Benatallah, A. Bestavros, Y. Manolopoulos, A. Vakali, and Y. Zhang, Eds., vol. 8787. Cham: Springer International Publishing, 2014, pp. 521–530, book Title: Web Information Systems Engineering – WISE 2014 Series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-11746-1_38

[10] J. Wang, Z. Ding, and C. Jiang, "An Ontology-based Public Transport Query System," in *2005 First International Conference on Semantics, Knowledge and Grid*, Nov. 2005, pp. 62–62. [Online]. Available: https://ieeexplore.ieee.org/document/4125850

[11] "EnterpriseIntegrationLab/icity," Apr. 2023, original-date: 2017-06-27T16:53:03Z. [Online]. Available: https://github.com/EnterpriseIntegrationLab/icity

[12] N. Anand, M. Yang, J. H. R. van Duin, and L. Tavasszy, "GenCLOn: An ontology for city logistics," *Expert Systems with Applications*, vol. 39, no. 15, pp. 11 944–11 960, Nov. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741741200591X

[13] M. Houda, M. Khemaja, K. Oliveira, and M. Abed, "A public transportation ontology to support user travel planning," in *2010 Fourth International Conference on Research Challenges in Information Science (RCIS)*, 2010, pp. 127–136.

[14] "OpenTransport/linked-gtfs," Jul. 2023, original-date: 2015-02-02T10:55:09Z. [Online]. Available: https://github.com/OpenTransport/linked-gtfs

[15] E. Ruckhaus, A. Anton-Bravo, M. Scrocca, and O. Corcho, "Applying the LOT Methodology to a Public Bus Transport Ontology aligned with Transmodel: Challenges and Results," *Semantic Web*, vol. 14, no. 4, pp. 639–657,

Jan. 2023, publisher: IOS Press. [Online]. Available: https://content.iospress.com/articles/semantic-web/sw210451

[16] "OSLO Mobiliteit - Dienstregeling en Planning: Tijdstabellen (Applicatieprofiel)," Jul. 2023. [Online]. Available: https://data.vlaanderen.be/doc/applicatieprofiel/mobiliteit/dienstregeling-en-planning/tijdstabellen

[17] S. Timpf, "Ontologies of Wayfinding: a Traveler's Perspective," *Networks and Spatial Economics*, vol. 2, no. 1, pp. 9–33, Mar. 2002. [Online]. Available: https://doi.org/10.1023/A:1014563113112

[18] "Transmodel at a glance – Transmodel." [Online]. Available: https://www.transmodel-cen.eu/transmodel-at-a-glance/

[19] "TRANSIT: A vocabulary for describing transit systems and routes." [Online]. Available: https://vocab.org/transit/

[20] "Raptor implementation," Nov. 2023, publication Title: GitHub. [Online]. Available: https://github.com/planarnetwork/raptor

[21] "JavaScript modules - JavaScript | MDN," Mar. 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

[22] "esbuild - An extremely fast bundler for the web." [Online]. Available: https://esbuild.github.io/

[23] "Stream | Node.js v22.2.0 Documentation." [Online]. Available: https://nodejs.org/api/stream.html#readable-streams

[24] "node-stream-zip," Sep. 2021. [Online]. Available: https://www.npmjs.com/package/node-stream-zip

[25] "csv-parse," May 2024. [Online]. Available: https://www.npmjs.com/package/csv-parse

[26] "rdf-validate-shacl," May 2024. [Online]. Available: https://www.npmjs.com/package/rdf-validate-shacl