Client-side route planning: preprocessing the OpenStreetMap road network for Routable Tiles

Harm Delva[1,*], Julián Andrés Rojas Meléndez[1], Ben Abelshausen[2], Pieter Colpaert[1] and Ruben Verborgh[1]

[1] IDLab, Ghent University – imec, Ghent, Belgium; harm.delva@ugent.be, julianandres.rojasmelendez@ugent.be, pieter.colpaert@ugent.be, ruben.verborgh@ugent.be
[2] Open Knowledge Belgium, Brussels, Belgium; ben@openknowledge.be

* Author to whom correspondence should be addressed.
This abstract was accepted to the Academic Track of the State of the Map 2019 Conference in Heidelberg after peer-review.

Travelers have higher expectations than current route planning providers can fulfill, yet new solutions struggle to break through. Matching user experience from existing applications is already challenging without the large-scale infrastructure most of them have at their disposal; additionally integrating datasets such as the road network, public transportation schedules, or even real time air quality data is an even more laborious endeavour.

The World Wide Web Consortium (W3C) and the Open Geospatial Consortium (OGC) mention the usage of Linked Open Data as a best practice for publishing interoperable geospatial datasets. Instead of relying on proprietary data formats or monolithic CSV files, Linked Open Data uses the Resource Description Framework (RDF) data model as a framework for existing domain models. Every data element, and even the relations between them, receives a Uniform Resource Identifier (URI). Data publishers can reuse these identifiers to unambiguously refer to resources on the Web, thus making individual data sets more interoperable. The ultimate goal being automated integration, giving even clients the power to execute the queries. Client-side querying differs from traditional approaches but provides some advantages: (i) it takes the load off the service provider, (ii) the data can be cached for subsequent queries, (iii) the user leaks less personal data.

The OpenStreetMap road network has recently been published as routable Linked Open Data, following a similar approach to vector tiles [1]. However, executing route planning queries on the client is still an unsolved problem. Long-distance queries require large amounts of data and downloading all the data takes a long time. State-of-the-art route planning algorithms achieve better query execution times by using auxiliary data that has been computed in a preprocessing phase [2, 3]. The biggest bottleneck in client-side querying is the network; downloading more data to improve query times will ultimately make querying even slower. Client-side route planning requires a different approach to match the quality of service of existing services.

We explore several ways of preprocessing the routable tile data to improve the *user-perceived performance* of query evaluation. Each tile can be processed independently from the others, resulting in a pleasingly parallel workload. As a first step, we compute how to efficiently traverse pedestrian areas. Only the boundary edges of these areas are defined in OpenStreetMap which means that most routing engines route along these edges, yielding suboptimal paths. This step materializes the information that's already present in the base data to make it easier for route planners to ingest.

Secondly, we identify which nodes and ways are actually needed to cross a specific tile, filtering out the elements that are only used for local traffic. Queries only need the full tiles around the departure and arrival locations. This process becomes more effective at higher zoom levels: at the lowest zoom level it decreases the file sizes by roughly 50% and this increases to 86% at higher zoom levels. Finally, we discard all untagged nodes that aren't used to connect ways – replacing them with precomputed distances between the remaining nodes. This reduces the file size by another 50%, regardless of the zoom level.

Each step yields a different view of the tile data and the results are published as Linked Open Data, in accordance with the W3C and OGC best practices. Similar preprocessing steps as the ones described in this paper are often done while setting up a route planning service, with a significant amount of duplicate work as a result. This process could be greatly simplified if the resulting data is freely accessible.

We integrated the resulting datasets into a route planner for public transportation that uses Dijkstra's algorithm for the road network pathfinding. All data are downloaded dynamically as they are needed. We found that short-distance one-to-many queries such as finding the closest nearby bus stops that initially took around 400 ms to complete only take around 260 ms with the preprocessed data, and the first results are presented after 140 ms. The difference becomes bigger over long distances; computing a 50 km and a 100 km journey used to take respectively 7 minutes and 56 minutes. Using the preprocessed data lowers the query times to respectively 10 s and 21 s. We see similar improvements in the amount of downloaded data; these journeys initially required 30 MB and 170 MB of data, which is reduced to 3 MB and 9 MB.

Our contributions in this paper are two-fold; we have developed preprocessing techniques driven by our need to make a serverless route planner more efficient and in doing so discovered that the challenges we faced are not unique – and that we as a community could benefit from sharing more data with each other.

References

[1] Colpaert, P., Abelshausen, B., Rojas Meléndez, J., Delva, H., & Verborgh, R. (2019). Republishing OpenStreetMap's roads as Linked Routable Tiles. In *Proceedings of the 16th ESWC: Posters and Demos*.

[2] Barron, C., Neis, P., & Zipf, A. (2014). A comprehensive framework for intrinsic OpenStreetMap quality analysis. *Transactions in GIS*, *18*(6), 877-895.

[3] Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the Seventh International Conference on Experimental and Efficient Algorithms*, 319-333.