Merging a Connection Scan Algorithm with a guided search in an industrial setting

Arthur Finkelstein[1] , **Jean-Charles Régin**[2]
[1]Instant System
[2]Université Côte d'Azur
arthur.finkelstein@instant-system.com, jean-charles.regin@univ-cotedazur.fr

Abstract

We present GDCSA an algorithm improving on the Connection Scan Algorithm using goal directed techniques for journey planning with multicriteria range queries in public transit networks. We show how GDCSA both answers the needs of the user with a run time of less than a second on most networks and the needs of an industrial setting being an easy, maintanable and adaptable algorithm.

1 Introduction

With the advent of smart-phones, millions of passengers use computer-based journey planning systems to obtain public transport directions. Those directions need to be given in a reasonable amount of time and users do not want only the shortest path from A to B but may want a journey with the lowest walking distance, the least transfers, something in between or any number of combinations of criteria.

Multiple variants of the public transport routing problem exist but one with a high practical relevance is the Pareto range query problem (PRQP) because travelers do not want to arrive significantly later than the earliest arrival time and may have specific preferences on the number of transfers, the walk distance or any other criteria. But while being of practical relevance, this variant is difficult to solve quickly which is problematic because no user will wait more than a second in front of a smart-phone.

The goal of this article is to show how we developed an algorithm meeting the needs of real users while being integrable in an industrial product.

Answering the needs of the users

Nowadays users want a system that responds perfectly to their wants and needs. And tomorrow, the users will want even more. This means fast and flexible algorithms that can solve the PRQP.

Algorithms that yield fast query times for public transit routing in large metropolitan networks are numerous, with search based algorithms like the Connection Scan Algorithm (CSA) [Dibbelt *et al.*, 2018] or RAPTOR [Delling *et al.*, 2015], preprocessing based approaches with Transfer Patterns [Bast *et al.*, 2010; Bast and Storandt, 2014] or Trip-Based Public Transit Routing [Witt, 2015].

Our users require a flexible algorithm as such preprocessing based methods are off the table. Thus, we focus our interest on search based methods because they are the fastest and most flexible among the remaining methods. The two main being the CSA and the RAPTOR.

Among search based methods, CSA based algorithms are simple, short, easy to implement, expendable and have good performances which make them often used in journey planning systems (e.g. Instant System on the Paris metropolitan network, TrainLine on the European train network, ...). In addition, the PRVCSA, the Pareto range query variant of the CSA, seems to be one of the faster algorithm to solve the PRQP as mentionned in multiple well-known articles [Bast *et al.*, 2016; Dibbelt *et al.*, 2018]. Therefore with regards to the needs of the users, CSA based algorithms are the solution.

Answering the needs of an industrial setting

We can identify multiple problems when integrating an algorithm in an industrial setting : the accuracy of the data, the maintainability and adaptability of the algorithm and the integration into an existing system.

The problem when using public transport journey planning systems on metropolitan data is that the data is never 100% accurate, we can have multiple problems ranging from lines that have not been updated with the new stops and departure times and any number of other problems. As such an algorithm that is robust to inaccurate data is essential.

With inaccurate data and real life usage, we have a lot of bug fixing and in a small company like ours with 30 employees, any one of the R&D engineers should be able to debug the core of the journey planning because a debug task is assigned independently of who wrote the code. This means that an algorithm that is easy to code and understand is vital.

An easily adaptable algorithm is important because each client has different needs, depending on the size of the public transport network, the wish of the community and other factors. This leads to an algorithm that has to be easily modifiable to allow the output, input or core to be tailor-made for each client. For example certain clients want to use the GPS position of the user as departure or arrival instead of stops, or combining free floating bikes or kick-scooters with public transport which means journeys to or from other modes will have specific constraints.

A complete app containing other components, such as tick-

eting, next departure times, favorite notification, guiding, transport on demand and more, involves more complex data structures than the ones described in the literature. For example simple identifiers are used for benchmarking whereas they are more complex in an app to be human readable. So access times to the data structures are longer and therefore the run time is slower. We also have the problem of persistence and the need for databases which limit access times, and when connecting to real time providers the correspondence between the line and stop ids is never automatic.

Our solution

Our solution to those constraints is the Goal-Directed CSA (GDCSA), that combines goal-directed techniques with a PRVCSA to solve the PRQP. Other algorithms can't meet all those needs, RAPTOR is a mildly more complex algorithm, with specific data structures and slower response times for the PRQP, and CSAccel is a complex algorithms with no variant for the PRQP.

2 Goal-Directed Connection Scan Algorithm

The GDCSA shares the same goals as the CSA, a simple and efficient code. We want an algorithm that is easy to implement and to understand, while improving run time for the PRQP on dense metropolitan networks.

The main idea is to partition the graph of stops into areas, in a roughly geographical manner, to avoid scanning connections that will only take us away from the target. By using upper and lower bounds on the duration between stops and more specifically between areas, we only use a sub-set of the areas to solve the PRQP.

The algorithm used to partition the graph is the Inertial Flow [Schild and Sommer, 2015], a recursive algorithm that cuts a set of nodes in two subsets by computing the min-cut, with a flow algorithm, and then the same method is called for each subset. The stopping conditions are either a maximum depth or a minimum size of nodes in a subset.

Then a lower bound is computed between every pair of areas, it is the journey with the minimum duration over all the journeys of the day that connects the boundaries of both areas. The lower bounds can be computed once and for all in a preprocessing which can be parallelized.

The GDCSA works in four phases :

The first computes the upper bound using an earliest arrival CSA, using the maximum arrival time τ_t as defined in [Dibbelt *et al.*, 2018]. The upper bound we use is the time span to satisfy a journey request : $\overline{d_{PT}}(s,t,\tau_s) = \tau_t - \tau_s = \tau_s + 2 \cdot (x - \tau_s) - \tau_s = 2 \cdot (x - \tau_s)$.

The second iterates over each area to only keep the ones that will be useful to the journey planning by using a simple inequality between upper and lower bounds : $\underline{d_{PT}}(s,a) + \underline{d_{PT}}(a,t) \leq \overline{d_{PT}}(s,t,\tau_s)$ where a is an area.

The third will merge all the connections from the chosen areas and sort them.

The last will launch a PRVCSA using only the sorted connections of the opened areas.

More details on the algorithm can be found in [Finkelstein and Régin, 2020].

3 Experiments

We experimentally evaluate the GDCSA and compare it to the PRVCSA with 4 criteria: maximizing the departure time, minimizing the arrival time, minimizing the number of transfers and minimizing the walked distance.

Our test instances are based on the data of the public transit network of 3 cities (Paris, Berlin and Stockholm) and 2 country wide train networks (Germany and Switzerland), the data is openly available via a GTFS feed (https://transitfeeds.com/) which has been downloaded in October 2019.

The number of connections and stops for each network is seen in the table below. We can see that the city wide network range from big with Paris, to smaller with Stockholm, the goal of those public transit network is to show the performances of the GDCSA on dense networks. Whereas the country wide train networks show the performances on sparse networks.

The footpaths were given in the GTFS feed but the graph was not transitively closed, we then programmatically generated the missing ones.

Our experiments were conducted on a Intel Core i7-7700HQ processor with 16 GB of RAM.

In our evaluation, we ran for each variant of the algorithm the same set of 1000 queries generated randomly. The source and target stops are chosen uniformly at random. The departure time is picked uniformly at random within the day.

The GDCSA uses an inertial flow partitioning with a maximum depth of 12.

Instance	# Conn.	# Stops	Algorithm	Time [ms]
Paris	3209401	44534	PRVCSA	7858
			GDCSA	2981
Berlin	1379755	28651	PRVCSA	1383
			GDCSA	338
Stockholm	703326	14258	PRVCSA	847
			GDCSA	89
Germany	3601420	74398	PRVCSA	2587
			GDCSA	529
Switzerland	2599675	29844	PRVCSA	1289
			GDCSA	147

We can see that the run time of the GDCSA is 2.5 to 9 times faster than the PRVCSA and is mostly under the one second limit dictated by the users. The run times for the smaller networks have a greater gain from the GDCSA, whereas the bigger network the lower the gains are.

4 Conclusion

The idea is promising with an easy to code algorithm, using as a center piece a PRVCSA, while boasting good performances on dense metropolitan networks as well as sparse country wide networks.

The results on the Paris metropolitan network are still a bit underwhelming but we are positive that using better machines, optimizing the code and maybe parallelizing some tasks could help us reach our goal.

References

[Bast and Storandt, 2014] Hannah Bast and Sabine Storandt. Frequency-based search for public transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22, 2014.

[Bast *et al.*, 2010] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast routing in very large public transportation networks using transfer patterns. In *European Symposium on Algorithms*, pages 290–301. Springer, 2010.

[Bast *et al.*, 2016] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

[Delling *et al.*, 2015] Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015.

[Dibbelt *et al.*, 2018] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *Journal of Experimental Algorithmics (JEA)*, 23:1–56, 2018.

[Finkelstein and Régin, 2020] Arthur Finkelstein and Jean-Charles Régin. Goal directed techniques meet connection scan algorithm. https://drive.google.com/file/d/10vQd7AP8oipNJazpWsADinNKSq7QDavn/view?usp=sharing, 2020.

[Schild and Sommer, 2015] Aaron Schild and Christian Sommer. On balanced separators in road networks. In *International Symposium on Experimental Algorithms*, pages 286–297. Springer, 2015.

[Witt, 2015] Sascha Witt. Trip-based public transit routing. In *Algorithms-ESA 2015*, pages 1025–1036. Springer, 2015.