Scalable Algorithms for Bicriterion Trip-Based Transit Routing

A. Prateek Agarwal[a,*], B. Tarun Rambha[b]

[a] Ph.D. Student, Indian Institute of Science (IISc) Bangalore, Karnataka, India
prateeka@iisc.ac.in
[*] Corresponding author
[b] Assistant Professor, Department of Civil Engineering
Center for Infrastructure, Sustainable Transportation, and Urban Planning (C*i*STUP)
Indian Institute of Science (IISc) Bangalore, India
tarunrambha@iisc.ac.in

Extended abstract submitted for presentation at the 11th Triennial Symposium on Transportation Analysis conference (TRISTAN XI)
June 19-25, 2022, Mauritius Island

March 31, 2022

Keywords: transit routing; shortest paths; multi-criteria optimization, hypergraph partitioning

1 INTRODUCTION

Public transit systems are an indispensable part of any metropolitan city. Its success depends on many factors, chief among which is the ease users can query for optimal journeys using mobile apps. Public Transit Routing (PTR) is typically aided by mobile apps and backend algorithms. Ideally, these algorithms must be fast and should provide details on efficient routes between origins and destinations of passengers. Conventional approaches model the transit network as a time-expanded or time-dependent graph and run a variant of the Dijkstra's algorithm. However, this method turns out to be too slow for large networks. Furthermore, while planning a journey using public transit, the number of transfers is equally important besides travel time. Popular techniques developed for PTR in the past decade include—Transfer Patterns algorithm (Bast *et al.*, 2010), Connection Scan Algorithm (CSA) (Dibbelt *et al.*, 2013), Round based Public Transit Routing algorithm (RAPTOR) (Delling *et al.*, 2015), and Trip-Based public Transit Routing (TBTR) algorithm (Witt, 2015). The main goal of our work is to propose routing algorithms that are efficient and practical. Specifically, we address the following types of routing problems. The improvements summarized above are based on country- and city-level transit datasets.

- HypTBTR: Shortest path algorithms are usually made more efficient by partitioning the underlying graph. We propose "HypTBTR" by combining TBTR with a partitioning-based speed-up. For one-to-one shortest path queries, HypTBTR was found to be 30–40% faster than the TBTR algorithm. Additional analysis studying the effect of different weighting schemes and hypergraph partitioning tools (KaHyPar, hMETIS, Integer Program) is also presented.

- One-To-Many rTBTR: Solving profile queries is one of the major bottlenecks in modern PTR approaches such as Scalable Transfer Patterns (Bast *et al.*, 2016), HypRAPTOR (Delling *et al.*, 2017), and HypTBTR. To address this problem, we extend the popular rTBTR algorithm to its One-To-Many variant. This algorithm not only reduces the preprocessing times significantly but is also helpful from a practical standpoint as users often query the shortest path between two locations instead of two transit stops (a location can have multiple stops near it). Compared to the existing approaches, our implementations were 40–50% faster.

- Multilevel extensions for HypTBTR and HypRAPTOR: The reduced query times in HypTBTR and HypRAPTOR come at the cost of increased preprocessing. We solve this issue using multilevel partitioning. Compared to standard partitioning implementations, preprocessing times were reduced by approximately 20–50%. Apart from CSA, none of the PTR algorithms have been extended to incorporate multilevel partitioning.

2 METHODOLOGY

HypTBTR: We introduce an additional preprocessing step to TBTR which partitions routes into p disjoint sets $R_1, R_2, \ldots, R_p$, also known as *route cells*. The central idea in HypTBTR is to construct a hypergraph $\mathcal{G}$ in which every node represents a route, and a hyperedge between a subset of nodes that represents *intersecting routes* (two routes are called intersecting if they have at least one stop in common). Footpaths or walking connections are also treated as routes with two stops. A partitioning algorithm (based on a min-cut approach) is then used to generate route cells. By definition, route cells are mutually exclusive and exhaustive (i.e., $R_i \cap R_j = \emptyset$ for all i and j, and $\bigcup_{i=1}^{p} R_i = R$). Consider the example in Figure 1. Subfigure (b) shows the corresponding hypergraph with each route as a node (a hyperedge is added between the red, orange, and grey nodes). The black node represents the footpath. Assuming $p = 3$, a partitioning algorithm creates three route cells namely R_1 (dark blue), R_2 (cyan), and R_3 (purple), as shown in Subfigure (c). Subfigure (d) shows the cutstops derived from the route cells in (c).

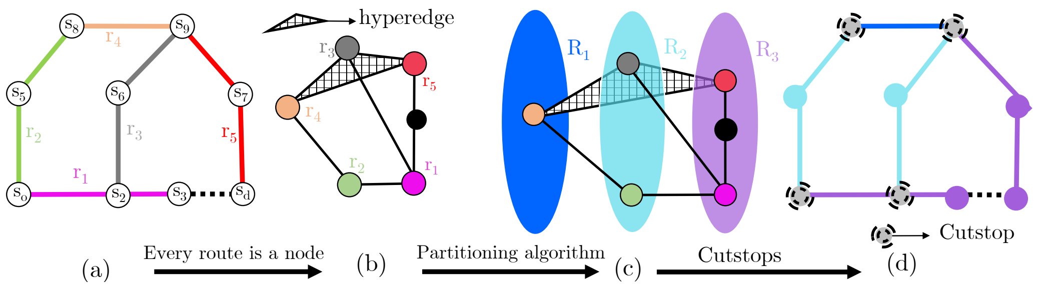

Figure 1 – *HypRAPTOR: Route partitioning using a hypergraph representation*

Similar to route cells, *stops cells* $S_0, S_1, \ldots, S_p$ is a partition of stops. To construct a stop cell S_i, we start by including all stops belonging to the routes in the corresponding route cell R_i. However, the resulting stop cells are not mutually exclusive due to cutstops. Thus, define S_0 to contain all the cutstops and update stop cell S_i as $S_i \leftarrow S_i \setminus S_0$. For example, in Figure 1, the stop cells are $S_0 = \{s_0,\ s_2,\ s_8,\ s_9\}$, $S_1 = \emptyset$, $S_2 = \{s_5,\ s_6\}$, and $S_3 = \{s_3,\ s_d,\ s_7\}$. The next step is to find and store optimal routes between these cutstops, referred to as *fill-in*. To this end, a profile query (using our version of One-To-Many rTBTR) is made for all possible pairs of cutstops in S_0. If a route belongs to an optimal journey between a pair of cutstops, it is added to the fill-in set.

In the query phase, given the source s_o and destination s_d, the first step is to identify S_o and S_d, i.e., the stop cell to which s_o and s_d belong to. Let the corresponding route cells be R_o and R_d. If the source stop is a cutstop, then S_o and R_o are set to S_0 and $\emptyset$ respectively (the same convention is used for the destination stop). A trip is scanned only if it is a part of fill-in or if all its stops belong to the source or destination cells.

Multilevel Extensions Consider the example in Figure 2. The base network is an abstraction of a transit network before partitioning. Nodes in white indicate stops, and those in blue show the source and destination. In Standard Partitioning, the network is partitioned into six parts. Red nodes represent cutstops. Labels show the stop cell ID for each partition ($S_1, S_2, \ldots, S_6$) and S_0 is assumed to contain all the red cutstops. Thus, using standard partitioning, HypTBTR's fill-in trip set will require finding optimal journeys between $\binom{7}{2}2! = 42$ source-destination permutations of the cutstops.

In Multilevel Partitioning, we use two levels. In Level 1, the network is partitioned into three parts (S_1, S_2, S_3). These are referred to as *parent partitions*. Green nodes show the cutstops of Level 1. Next, in Level 2, each parent partition is divided into two subparts (i.e., *child partitions*). E.g., S_1 is divided into S_{11} and S_{12}. Cutstops are shown using pink nodes here. Finding the fill-in set $\mathcal{F}$ can then be divided into parts. First, Compute the trips required to travel between cutstops at the topmost level, i.e., $\binom{4}{2}2! = 12$ source-destination permutations of the four green cutstops of Level 1. Second, For each parent partition, determine the trips required to travel between its cutstops and the cutstops of its children. E.g., in Figure 2, arrows between Levels 1 and 2 indicate 4 source-destination permutations for S_1 and its children S_{11} and S_{12}. Similarly for S_2 and its children S_{21} and S_{22}, we have 8 permutations. Thus, we get a total of $4+8+4 = 16$ permutations between the two levels.

The overall number of source-destination pairs in the multilevel scheme is $12 + 16 = 28$ compared to 42 in the standard scheme (a 33% benefit).

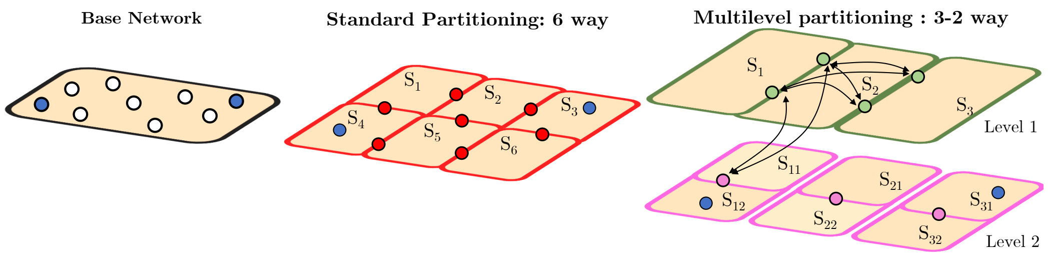

Figure 2 – *Example to illustrate the advantages of multilevel partitioning*

One-To-Many rTBTR: As stated above, given a set of cutstops, partitioning-based speed-up methods find and store the optimal paths between them. To do this, the existing approach is to apply a One-To-One algorithm (e.g., rRAPTOR, rTBTR) for all possible cutstop combinations. This step is the major bottleneck during preprocessing. We propose to solve this by extending existing rTBTR and rRAPTOR algorithms to their One-To-Many variants. For a given destination stop list, these algorithms find all optimal journeys departing within a specified time range. Our framework aggressively prunes the destination list to achieve faster runtimes. Practical applications of the proposed techniques include point-of-interest queries (find the closes five restaurants), dynamic traffic assignments, and building distance tables in isochrones.

3 RESULTS

Figure 3 shows Switzerland and Sweden when partitioned into four parts. Table 1 compares the performance One-To-Many rTBTR against rTBTR. The benefits are in the range of 90–95%. Table 2 compares the performance of HypTBTR and MHypTBTR against its base variant. For One-To-One queries, we observe benefits in the range of 23–37%. Furthermore, we notice a reduction of 5–58% in preprocessing times (not shown here) using multilevel partitioning. Results from different hypergraph weighting schemes and partitioning algorithms (like hMETIS, KaHyPar, Integer Program) will also be presented. Additional analysis illustrating the effects of the proposed techniques on city- and country-level datasets will also be shown.[1]

[1]A working paper with more details and results can be found at https://arxiv.org/abs/2111.06654

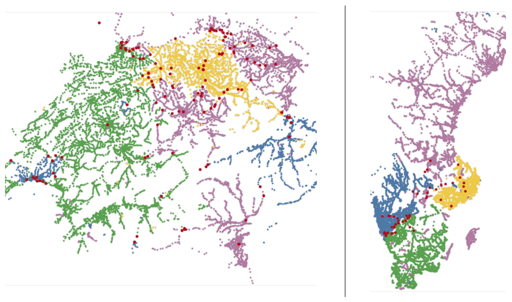

Figure 3 – *Illustration of standard 4-way partitioning for Switzerland and Sweden (Partitions in standard partitioning are indicated by blue, green, yellow, and purple. Red dots indicate cutstops)*

Table 1 – *Query times (in milliseconds) of various PTR algorithms*

Network	**RAPTOR**	**TBTR**	**rTBTR**	**One-To-Many rTBTR**
Switzerland	334.7	96.9	81 710.6	3 654.1
Sweden	103.5	6.3	1 250.9	131.8

Table 2 – *Query performance (in milliseconds) of algorithms combined with partitioning-based speed-up (values in teal indicate % gain over their base variant)*

Network	**Metric**	**Partitions**			
		4 (2-2)	**6 (3-2)**	**8 (4-2)**	**10 (5-2)**
Switzerland	HypRAPTOR	230.6 (31.1%)	240.1 (28.3%)	231.4 (30.9%)	226.7 (32.3%)
	HypTBTR	62.4 (35.6%)	67.3 (30.5%)	63.4 (34.6%)	61.4 (36.6%)
	MHypRAPTOR	230.2 (31.2%)	231.9 (30.7%)	232.4 (30.6%)	235.8 (29.5%)
	MHypTBTR	65.0 (32.9%)	65.1 (32.8%)	66.2 (31.7%)	65.4 (32.5%)
Sweden	HypRAPTOR	90.9 (12.1%)	88.7 (14%)	89.6 (13.4%)	91.1 (12%)
	HypTBTR	4.8 (23.8%)	4.5 (28.6%)	4.4 (30.2%)	4.3 (31.7%)
	MHypRAPTOR	93.5 (9.7%)	93.4 (9.7%)	85.2 (17.7%)	85.1 (17.8%)
	MHypTBTR	4.7 (25.4%)	4.3 (31.7%)	4.4 (30.2%)	4.3 (31.7%)

References

Bast, Hannah, Carlsson, Erik, Eigenwillig, Arno, Geisberger, Robert, Harrelson, Chris, Raychev, Veselin, & Viger, Fabien. 2010. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. *Pages 290–301 of: European Symposium on Algorithms*. Springer.

Bast, Hannah, Hertel, Matthias, & Storandt, Sabine. 2016. Scalable Transfer Patterns. *Pages 15–29 of: 2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM.

Delling, Daniel, Pajor, Thomas, & Werneck, Renato F. 2015. Round-Based Public Transit Routing. *Transportation Science*, **49**(3), 591–604.

Delling, Daniel, Dibbelt, Julian, Pajor, Thomas, & Zündorf, Tobias. 2017. Faster Transit Routing by Hyper Partitioning. *In: 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Dibbelt, Julian, Pajor, Thomas, Strasser, Ben, & Wagner, Dorothea. 2013. Intriguingly Simple and Fast Transit Routing. *Pages 43–54 of: International Symposium on Experimental Algorithms*. Springer.

Witt, Sascha. 2015. Trip-based Public Transit Routing. *Pages 1025–1036 of: Algorithms-ESA 2015*. Springer.