Time-Dependent Route Scheduling on Road Networks

Lei Li*, Jiwon Kim*, Jiajie Xu#, Xiaofang Zhou*#

* School of Information Technology and Electrical Engineering, The University of Queensland
# School of Computer Science and Technology, Soochow University
{l.li3, jiwon.kim, zxf}@uq.edu.au

Abstract

Navigation has been an important tool for human civilization for thousands of years, and the latest technologies like online map services and GPS satellites have brought it up to a new level. Now people can easily identify where we are on earth, find any places they want to go, and retrieve best routes to get there. Although there are plenty of tools that are convenient and fast enough for basic uses, it is still far from optimal. For example, most systems only consider various type of distances as the optimization goals, while the traveling time, which needs to consider traffic conditions, is a more appropriate one. However, it is both hard to acquire the traffic condition information and to compute time-dependent fastest paths. Therefore, in this article, we present an introduction to the time-dependent route scheduling, from speed profile generation to route scheduling and query answering.

1 Introduction

With the prevalence of GPS enabled devices and mobile network, various of navigation systems have been widely adopted by public transportation systems, logistics companies, private vehicles and a broad range of location-based services. These systems first find where we are on planet earth, then compute a reasonable path to our destinations, most of which are based on shortest path algorithms [4, 9, 15]. During the trip, they provide turn-by-turn navigation using real-time map-matching and real-time path computation. Some of them even keep users' trajectories, like O2O taxi service providers *Uber* and *DiDi*. In fact, they are becoming more and more popular around the world and have obtained a tremendous amount of trajectories generated by their taxi drivers every day. However, although these trajectories can reveal the traffic conditions of different parts of a city at different time periods, they are mostly used for behavior analysis and customer support.

In spite of their popularity, there are still some untreated shortcomings. The most obvious one among them is the lack of considering the traffic condition. The reason for it is two-fold. Firstly, it is hard to describe traffic condition because unlike the distance, it changes over time and is not an inherent property of the road. Secondly, it is hard to compute the fastest path as the query is actually more complicated. Therefore, in this article, we present some techniques to solve these problems.

A road network is usually modeled as a time-dependent graph, where each edge is associated with a function that returns the time cost of traveling from one vertex to another at a given departure time. The set of these functions is called a speed profile. Obviously, with the help of traffic sensors and cameras, we can always get an accurate speed profile. But they are expensive and unrealistic to cover the entire road network. On the other hand, with the vast amount of historical and real-time trajectory data at hand, we are able to derive a speed profile at a much larger scale with little cost [12]. However, it is not straightforward to achieve this. We first need to attach the GPS points of the trajectory data to the roads by *map matching* [16]. After that, these attached

Table 1: Time-Dependent Path Problems

$Weight$ is a static function, $Time$ is the time-dependent function and $\beta(v_i)$ is the departure time from v_i. v_s is the starting vertex and v_d is the destination vertex.

Graph Type	**Path Problem**		**Objective**	**Waiting**
Static Graph	Shortest Path		$\Sigma_{i=s}^{d} Weight(v_i, v_{i+1})$	No
Time-Dependent Graph	*SSFP*	General	$\Sigma_{i=s}^{d} Time(v_i, v_{i+1})$	No
		Earliest Arrival	$Min(Arrival(v_d))$	
		Latest Departure	$Max(Departure(v_s))$	
	ISFP		$Min(Arrival(v_d) - Departure(v_s))$	Starting Vertex
	MORT		$Min(\Sigma_{i=s}^{d} Time(v_i, v_{i+1}, \beta(v_i)))$	A set of vertices

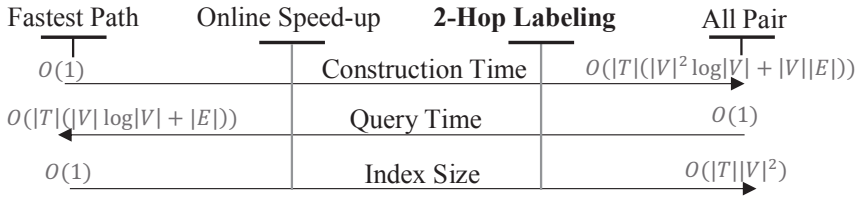

Figure 1: Time-Dependent Path Index Categories

GPS points are converted to speed data by collecting them into different time slots. Because many roads and time slots may not have any speed data at all, various techniques are proposed to estimate the missing values. Finally, compression is appreciated due to the large size of a raw speed profile.

With the speed profile at hand, the shortest path problem can be extended to the time-dependent scenario and categorized according to the weight function type and defining whether waiting on the vertices or not. The shortest path problem is at one extreme that the weight functions are static and no waiting is allowed. If the weight functions are time-dependent, it is the *Single Starting-Time Fastest Path (SSFP)* problem. It can be solved in $O(|V| \log |V| + |E|)$ time if *first-in first-out property* holds [6]. *Earliest Arrival Path* and *Latest Departure Path* are two variations of it. If waiting is allowed on the starting vertex, it is the *Interval Starting-Time Fastest Path (ISFP)* [10, 5]. It can be viewed as finding an optimal departure time during a given time period that has the shortest total travel time. The problem complexity is $\Omega(T(|V| \log |V| + |E|))$ [8] and T is number of turning points in the destination vertex's travel cost function. If waiting is further allowed on a set of vertices, the problem is generalized to the *Minimal On-Road Time (MORT)* problem [11, 13]. In this case, the total travel time is decomposed into waiting time and on-road time, and waiting on some vertices can reduce the on-road time, like avoiding a traffic jam. Logistics companies use it to reduce their fuel consumption and tourists use it to plan their trips. The categories are shown in Table 1.

However, due to the inherent complexity of $\Omega(T(|V| \log |V| + |E|))$, the fastest path is slow to compute. Hence, various indexes on paths are extended to time-dependent scenario to speed up query answering. Figure 1 illustrates the categories. On one extreme is the fastest path directly with no index, and on the other extreme is the *All Pair* pre-computation. Most of the existing approaches fall into the *online speed-up* category, which adds various precomputed structures like shortcuts [1] and landmarks [3] to prune the searching space. Nevertheless, an expensive searching is still required. Another approach is to extend the 2-Hop labeling [2], which is widely used in the static graph, to the time-dependent environment. However, although it is much faster than the *online search* approaches, its label size is already big on the static graph, it would be much larger on the time-dependent road network.

The rest of this article is organized as follow: we first present how to derive a speed profile from the trajectory date in Section 2. Then we introduce a general form a fast path problem *MORT* in Section 3. In Section 4, we present the time-dependent 2-hop labeling for fast query answering. Finally, we conclude in Section 5.

2 Speed Profile Generation

2.1 Speed Data Collection

A trajectory is a series of GPS points. After matching it on the map, each point is attached to a road. With the timestamp and road length information, we can compute the road speeds at different time points. Next we convert it into a speed profile. The most straightforward method is to use these speed data directly, which would result in a set of linear piecewise speed functions. However, it is not practical because it would be too zigzag and hard to use. Thus, we use a histogram-based approach to collect the speed data. Specifically, we divide one day's time into $\mathbb{T}$ slots with the same length. Then the speed data that fall into the same slot will be added up together to get an average speed. Thus, the influence of the outliers is reduced dramatically. $\mathbb{T} = 5$ minutes has the best performance according to our experiment.

2.2 Missing Value Estimation

Even though the GPS-based trajectory data has a higher coverage of the road network than other approaches, it is still hardly possible to cover every edge. So it also faces the sparsity problem. There are several approaches to solve this problem. Firstly, the speed profile of each road can be viewed as a vector. Then by finding the most similar ones to fill each other, we can estimate the missing values. Moreover, the speed profile of the whole road network is essentially a matrix. Thus, we can take advantage of matrix-factorization based collaborative-filtering, together with other road features to fill the voids [14]. Last but not the least, hidden Markov can also be used to estimate the missing speeds [14].

2.3 Compression

The histogram-based speed profile can be viewed as a type of *Time Series Data* [7], and compressing a speed profile falls into the category of *Time Series Segmentation*. *Sliding window*, *top-down* and *bottom-up* are three popular approaches to compress it. The *Sliding window* algorithm has a linear running time, but it has a bad compression rate, while the *bottom-up* algorithm takes the longest time to compute but has the best compression rate. After compression, the histogram-based speed profile is changed into a set of linear piecewise functions.

3 Minimal On-Road Time Route Scheduling

3.1 *MORT* Problem

As described in Section 1, the *MORT* is the general form of all the other path problems. It is more complicated because it is hard to know if we should wait on a vertex or not, and how long should we wait on it. The problem is solve by computing a *minimum cost function* $C_i(t)$, which records the minimum on-road travel time from v_s to v_i that arrives v_i on time t, for each vertex v_i. When the destination's $C_d(t)$ is computed, the result is found. The minimum cost function is from the perspective of arrival time rather than the departure time for the following reasons: if we depart at the same time, we can have different waiting schedules and different arrival time, which would further result in different on-road time. Therefore, the departure time perspective cannot handle the waiting. If we use the arrival time, we can ignore when we departed, where and how long we waited, because there is always a minimal value at each arrival time. There are two ways to construct the minimum cost functions: we can build the minimum cost functions over the whole query time interval iteratively in a *Dijkstra* way, or constructs it sub-time-interval by sub-time-interval until the end of the time interval. The key observation of this problem is that the minimum cost function on the waiting vertices is non-increasing because

waiting on a vertex would not increase the on-road time, and we can wait on it until the traffic condition becomes better. By configuring the size of waiting vertex set and departure time interval, we can use it solve any kind of path problem.

3.2 Approximation Algorithm

The time complexity of the *MORT* algorithm is significantly affected by the number of turning points in $C_i(t)$. What is worse, it grows larger as the expansion grows, which makes the computation slower and slower. So the key to speed up is decreasing the number of turning points, especially the useless ones. However, we cannot determine if one turning point will end up with the optimal result until the final $C_d(t)$ is constructed. Therefore, we design an approximation approach that can guarantee the final result is no less than $\alpha C_d^*(t)$, $\alpha \in (0, 1]$. Suppose a route is made up of a series of consecutive edges $\hat{E} =< e_1, e_2, ..., e_n >$ and $||\hat{E}||$ is the length of $\hat{E}$. If we apply an approximation factor α_1 on e_1, α_2 on e_2 and so on, the error of the final result does not grow linearly: $||\hat{E}||' = ((((e_1\alpha_1 + e_2)\alpha_2) + e_3)\alpha_3 + ...e_n)\alpha_n = \alpha_1\alpha_2\alpha_3...\alpha_n\ e_1 + \alpha_2\alpha_3...\alpha_n e_2 + ... + \alpha_n e_n = \prod_{j=1}^{n}\alpha_j e_1 + \prod_{j=2}^{n}\alpha_j e_2 + ... + \prod_{j=n}^{n}\alpha_j e_n = \sum_{i=1}^{n}\prod_{j=i}^{n}\alpha_j e_i$. To achieve $||\hat{E}||' \geq \alpha||\hat{E}||$, we have to guarantee $\prod_{j=1}^{n}\alpha_j \geq \alpha$. In another word, we can view α as a total budget of pruning power along the route, the larger the budget assigned to a vertex, the stronger pruning power it has to reduce the turning points. Because the turning point numbers of the earlier visited vertices are much smaller than those of the latter visited ones, we concentrate the pruning power to the latter vertices by setting a global turning point number threshold ρ: Only those vertices whose turning point numbers are larger than ρ will be pruned. Such approximation approach can benefit all the related time-dependent algorithms.

4 Time-Dependent 2-Hop Labeling

Like the distance query on static graph, fastest travel time query can also use the 2-hop labeling approach. For each vertex $v \in V$, we pre-compute two sets of labels: out-labels $L_{out}(v_i) = \{(v_j, f_{v_i,v_j}(t))\}$ and in-labels $L_{in}(v_i) = \{(v_j, f_{v_j,v_i}(t)\}$, where v_j is a hop vertex and $f_{v_i,v_j}(t)$ returns the minimal cost from v_i to v_j at different departure time t. We can answer a query only using the labels: $Q_f(v_s, v_d, L)) = Min(f_{v_s,v_d,v_i}(t)) = Min(f_{v_s,v_i}(t) \oplus f_{v_i,v_d}(t)) = Min(f_{v_i,v_d}(f_{v_s,v_i}(t))) = f_{v_s,v_d}(t), t \in \mathcal{T}, \forall v_i \in H_{v_s,v_d} = L_{out}(v_s) \cap L_{in}(v_d)$, where $\oplus$ is the concatenation of two linear piecewise functions and $Min()$ takes the smaller parts of two functions. Although it is fast to answer a query, the index size is huge, which limits its power to small graphs. In order to apply it on an ordinary road network, we first partition the road network into edge-disjoint grids. Then the 2-hop labeling is constructed within each grid and also among the boundary vertices of the grids. The query can be answered by concatenating three partial results: v_s to its boundary vertices G_i^B, G_i^B to G_j^B, and G_j^B to v_d. Our time-dependent 2-hop labeling can boost the query answering time by hundreds of times.

5 Conclusions

In this article, we briefly discuss and introduce the techniques used to answer a time-dependent path query on a road network. The speed profile generation can produce the time-dependent cost functions from trajectory data through map-matching, speed data collection, missing value estimation and compression. Then we present our *MORT* algorithm, which can answer all the existing path problems. Finally, we describe the time-dependent 2-hop labeling for fast query answering.

Acknowledgement. The work reported in this article is partially supported by the National Nature Science Foundation of China (61472263, 61772356, 61572335, 61402312) and the Australia Research Council (LP130100164 , DP170101172).

References

[1] G. V. Batz, R. Geisberger, S. Neubauer, and P. Sanders. Time-dependent contraction hierarchies and approximation. In *International Symposium on Experimental Algorithms*, pages 166–177. Springer, 2010.

[2] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

[3] D. Delling. Time-dependent sharc-routing. In *European Symposium on Algorithms*, pages 332–343. Springer, 2008.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[5] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 205–216. ACM, 2008.

[6] S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[7] P. Esling and C. Agon. Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):12, 2012.

[8] L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

[9] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[10] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 10–10. IEEE, 2006.

[11] L. Li, W. Hua, X. Du, and X. Zhou. Minimal on-road time route scheduling on time-dependent graphs. *Proceedings of the VLDB Endowment*, 10(11):1274–1285, 2017.

[12] L. Li, K. Zheng, S. Wang, W. Hua, and X. Zhou. Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory. *The VLDB Journal*, pages 1–25, 2018.

[13] L. Li, X. Zhou, and K. Zheng. Finding least on-road travel time on road network. In *Australasian Database Conference*, pages 137–149. Springer, 2016.

[14] J. Shang, Y. Zheng, W. Tong, E. Chang, and Y. Yu. Inferring gas consumption and pollution emission of vehicles throughout a city. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1027–1036. ACM, 2014.

[15] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *Proceedings of the VLDB Endowment*, 5(5):406–417, 2012.

[16] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun. An interactive-voting based map matching algorithm. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 43–52. IEEE, 2010.