Historical Traffic-Tolerant Paths in Road Networks

Pui Hang Li Man Lung Yiu
The Hong Kong Polytechnic University
Department of Computing
{csphli,csmlyiu}@comp.polyu.edu.hk

Kyriakos Mouratidis
Singapore Management University
School of Information Systems
kyriakos@smu.edu.sg

ABSTRACT

Historical traffic information is valuable for transportation analysis and planning, as well as for route search services. In view of these applications, we propose the k traffic-tolerant paths problem (TTP) on road networks, which takes a source-destination pair and historical traffic information as input, and returns k paths that minimize the aggregate (historical) travel time. Unlike the shortest path problem, the TTP problem has a combinatorial search space that renders the optimal solution expensive to compute. We propose an exact algorithm and a heuristic algorithm for this problem. Experiments on real traffic data demonstrate the effectiveness of TTP paths and the efficiency of our proposed algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Performance

Keywords

road networks, road traffic

1. INTRODUCTION

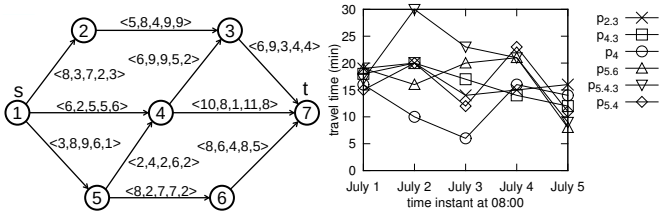
In this paper, we propose a novel problem called k traffic-tolerant paths (TTP), which finds application in transportation analysis, planning, and route search services. The idea is to extract a set of k paths $P^k_{s,t}$ from a road network such that it best approximates the shortest travel times for a given source-destination (SD) pair (v_s, v_t) at any time. Specifically, this problem requires a road network with historical travel time $G(V, E, W_m)$, where W_m maps an edge to its travel times at m time instants. Given an integer k and a SD pair (v_s, v_t), the TTP problem finds a set of k paths from v_s to v_t, denoted by $P^k_{s,t}$, that minimizes the following error:

$$\xi(P^k_{s,t}) = \frac{1}{m} \cdot \left(\sum_{j=1}^{m} \min_{p \in P^k_{s,t}} \{\tau_j(p) - \tau_j(sp_j)\} \right) \quad (1)$$

where $\tau_j(p)$ is the travel time of path p at time instant j and sp_j denotes the shortest path from v_s to v_t at time instant j. Observe that the minimization of $\xi(P^k_{s,t})$ is equivalent to the minimization of the following measure:

$$\Psi(P^k_{s,t}) = \sum_{j=1}^{m} \min_{p \in P^k_{s,t}} \tau_j(p) \quad (2)$$

because sp_j is independent of $P^k_{s,t}$ and the summation function is distributive.



(a) a road network (b) travel times of paths

Figure 1: Running Example

We illustrate TTP by the sample road network shown in Figure 1(a). Each edge is labeled with $m = 5$ weights that represent the travel times of edges at 5 time instants (e.g., 8:00am on July 1–5). Suppose that the SD pair (v_s, v_t) is (v_1, v_7). There are 6 possible paths from v_s and v_t, and their travel times at different time instants are shown in Figure 1(b). For clarity, we indicate the intermediate nodes in a path in the subscript, e.g., the path $p_{5.4.3}$ passes through v_5, v_4, v_3. Given that $k = 2$, the optimal path set is $P^k_{opt} = \{p_4, p_{5.6}\}$ because it has the minimum value $\Psi(P^k_{opt}) = 56$. We elaborate the applications of TTP below.

Application 1: transportation planning analysis. Transportation planners evaluate the reliability of transportation systems by analyzing the reliability of routes for representative SD pairs, which are chosen by their expertise. For example, representative SD pairs could be: city center to airport, ports to the industrial area, etc. Their current practice [4] is to select one route per SD pair and calculate the travel time reliability of each route. Our proposed TTP can provide k paths instead of a single path per SD pair to transport planners for reliability analysis. Hence, they can obtain a more comprehensive insight of the reliability of transportation systems.

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666483.

Application 2: reducing query cost in online route services. Online route services [8] have access to real-time traffic information[1], which are more accurate than modeling the travel times of road segments by using only historical traffic data [5,7]. Thus, they provide query users with fastest path(s) according to up-to-date traffic. Although these services may use shortest path indices [10] to answer queries efficiently, they incur substantial maintenance time and may not catch up with frequent traffic updates (e.g., every 30 seconds [1]).

In fact, the majority of queries are *recurrent queries* issued by daily users, e.g., finding the fastest route from home to office at 8:00am every day. Such regular patterns appear in human trajectories, as revealed in current scientific studies [6, 9]. A promising method is to precompute k candidate paths per recurrent query (in an offline phase) and then update their travel times by live traffic information (during online operations). This approach eliminates the index maintenance cost and bounds the online computation cost by the number of candidate paths. It scales better than existing methods based on shortest path indices. Although Malviya et al. [8] have proposed some heuristics for finding k candidate paths, those paths do not necessarily minimize the historical travel time error (in our Equation 1). In contrast, in our TTP problem, we aim to compute k candidate paths that minimize the historical travel time error.

The rest of this paper is organized as follows. We first present an exact algorithm for TTP in Section 2, and then develop a fast heuristics method in Section 3. Then, in Section 4, we evaluate the effectiveness of TTP paths and the efficiency of our algorithms on real traffic data. Finally, we conclude our work in Section 5.

2. EXACT METHOD

Our exact method for TTP consists of two phases: **(Phase I)** generating a set $\mathcal{C}$ of candidate paths and **(Phase II)** finding the optimal combination of k paths from the set $\mathcal{C}$.

A simple implementation (for Phase I) is to enumerate all possible size-k combinations of paths. As an example, we assume $k = 3$ and consider the SD pair (v_1, v_7) in the road network in Figure 1. Since there are $|\mathcal{C}| = 6$ possible paths (from v_1 to v_7) in the road network, we would enumerate $\binom{|\mathcal{C}|}{k} = \binom{6}{3} = 20$ size-3 combinations in total. However, this implementation does not scale well with a large road network.

We optimize the algorithms for both phases to reduce the search space $\binom{|\mathcal{C}|}{k}$. For Phase I, we develop pruning rules to eliminate unpromising paths that cannot contribute to the optimal solution (i.e., reducing the value $|\mathcal{C}|$). For Phase II, we adopt the branch-and-bound paradigm and design pruning rules to discard partial combinations that cannot lead to the optimal solution.

2.1 Phase I: Generating Candidates

We face two challenges in generating candidate paths. First, the number of all possible paths from source to destination is incredibly large on a sizable road network. Exploring all of them is impractical. Second, many paths lead to long travel times, so those paths should not be included in the optimal solution.

To overcome both challenges, we prune unpromising paths by leveraging the dominance property. Since every edge in the road network has a cost vector $w(e)$ with size m, all possible paths p connecting a SD pair has a m-dimensional cost vector $\overrightarrow{p}$ also. As p represents a vector, we can define and exploit dominance of paths.

DEFINITION 1 (VECTOR AND PATH DOMINANCE).
Let $\overrightarrow{v}$ and $\overrightarrow{u}$ be two m-dimensional vectors. $\overrightarrow{v}$ is said to dominate $\overrightarrow{u}$ if and only if $\forall 1 \leq j \leq m, \overrightarrow{v}.j \leq \overrightarrow{u}.j$. We denote this as $\overrightarrow{v} \preceq \overrightarrow{u}$.

Let p and p' be two paths from v_s to v_t. p' is said to dominate p if and only if $\overrightarrow{p'} \preceq \overrightarrow{p}$.

Recall that TTP is to minimize the aggregate value derived by the paths in a k-combination. Thus, we aim to retain those paths with short travel times over m time slots. By the concept of path dominance, if a path p' dominates another path p, this implies p' has smaller travel times than p at all time instants, so p should be pruned in Phase I. This dominance concept leads to the following pruning rule.

PRUNING RULE 1 (PATH DOMINANCE PRUNING).
Given a path p, if there is a path p' such that $\overrightarrow{p'} \preceq \overrightarrow{p}$, then p can be pruned.

This pruning rule is applicable only when all possible paths between v_s and v_t are known. However, enumerating all possible paths is expensive even on a medium-sized road network. We need to avoid exploring the entire search space during enumeration.

Observe that many paths share a common prefix among the possible paths from v_s to v_t. During path enumeration, we can safely disqualify a prefix (before it reaches v_t) by computing the minimum possible travel time of the paths originating from a common prefix at each time instant j, denoted by $LB_j(\hat{p})$. It is a sum of two terms: (i) the exact travel time of a prefix at time instant j, i.e., $\tau_j(\hat{p})$ and, (ii) the *minimum* travel time from the end node of the prefix to v_t. Therefore, we define a prefix path $\hat{p}$ and $LB_j(\hat{p})$ as follows.

DEFINITION 2 (PREFIX PATH).
Given a path $p = \langle v_{a_1}, v_{a_2}, ..., v_{a_n} \rangle$, $\hat{p} = \langle v_{\hat{a}_1}, v_{\hat{a}_2}, ..., v_{\hat{a}_m} \rangle$ is a prefix path of p if and only if $m \leq n$ and for $1 \leq i \leq m, v_{a_i} = v_{\hat{a}_i}$.

Given a prefix path $\hat{p}$, for $1 \leq j \leq m$, its minimum possible travel time at time instant j, denoted by $LB_j(\hat{p})$, is calculated as $LB_j(\hat{p}) = \tau_j(\hat{p}) + \tau_j(sp_j^{last})$, where sp_j^{last} denotes the shortest path from the last node of $\hat{p}$ to v_t at time instant j.

We define the lower-bound cost vector of $\hat{p}$ as $\overrightarrow{\hat{p}} = \langle LB_1(\hat{p}), LB_2(\hat{p}), ..., LB_m(\hat{p}) \rangle$.

For example, let us consider $\hat{p} = \langle v_1, v_5 \rangle$ and $LB_2(\hat{p})$ in our sample network. We have $\tau_2(\hat{p}) = 8$ and $\tau_2(sp_2^{last}) = 8$, where $sp_2^{last} = \langle v_5, v_6, v_7 \rangle$. Hence, $LB_2(\hat{p}) = 8 + 8 = 16$. By computing $LB_j(\hat{p})$ for each time instant $j \in [1..m]$, we obtain a cost vector $\overrightarrow{\hat{p}}$ that lower bounds the cost vector of any path sharing the common prefix $\hat{p}$. Similarly, we can apply the dominance concept and the pruning rule for $\overrightarrow{\hat{p}}$.

PRUNING RULE 2 (PREFIX PATH PRUNING).
Given a set of paths $\mathcal{D}$ and a prefix path $\hat{p}$, if there is a path $p' \in \mathcal{D}$ such that $\overrightarrow{p'} \preceq \overrightarrow{\hat{p}}$, then every path p with the prefix $\hat{p}$ can be pruned.

We propose to implement Phase I as follows. First, we apply a heuristic to find a set of pruning paths $\mathcal{D}$, which will be used to support our pruning rules. We will discuss how to select the set $\mathcal{D}$ shortly. Next, we initialize an empty prefix path and an empty candidate set $\mathcal{C}$. Then, we apply a DFS-like procedure to find all qualified candidates. The procedure recursively constructs different prefixes stemming from v_s and compares them against $\mathcal{D}$. If a prefix is dominated by a path in $\mathcal{D}$, the algorithm discards and stops expanding the prefix. Otherwise, the recursion keeps expanding the prefix by visiting the neighbors of its end node until reaching v_t. Each expanded path (from v_s to v_t) becomes a candidate for the next phase.

[1]Collected by road-side sensors, crowdsourcing, or purchased from traffic information providers.

2.1.1 How to Select the Pruning Path Set $\mathcal{D}$?

The selection of $\mathcal{D}$ is critical for pruning efficiency. We have empirically found that $\mathcal{D}$ yields the best performance in most cases if it includes the $k+1$ paths described below.

Initially, we compute the shortest path sp_{avg} (from v_s to v_t) in the road network, by setting the weight of each edge e to $w_{avg}(e) = \sum_{j=1}^{m} w_j(e)/m$. We then insert sp_{avg} into $\mathcal{D}$. The intuition is that such a path is not bad at all time instants.

Next, in the road network, we set the weight of each edge e to $w_s(e) = \min_{j=1}^{m} w_j(e)$. We then execute the following steps for k iterations. In each iteration, we perform a shortest path search from v_s to v_t, obtain a shortest path sp_i, and insert it into $\mathcal{D}$. Then, we update $w_s(e) = \max_{j=1}^{m} w_j(e)$ for each edge e on sp_i. We hope that this would force the shortest path search in subsequent iterations to find other paths that are significantly different from sp_i.

2.2 Phase II: Finding Optimal Combination

In this section, we present the implementation for Phase II, i.e., enumerating k-combinations of paths from the candidate set $\mathcal{C}$ in order to find the optimal k-combination. Since we have obtained the cost vectors of candidate paths in Phase I, the road network is no longer required in this phase.

The number of k-combinations of paths is $\binom{|\mathcal{C}|}{k}$, so it is expensive to generate them, especially when $|\mathcal{C}|$ is large. Thus, we develop pruning rules to prevent exploring unnecessary k-combinations. The key idea of the pruning rules is to early stop extending any *partial combination* $\hat{P}$ (which contain fewer than k candidates) by computing its lower bound aggregate value $\Psi_{lb}(\hat{P})$.

Table 1 shows the set of candidates $\mathcal{C}$ and their historical travel times after Phase I in our sample network. Suppose that $k = 3$ and $P_{best} = \{p_1, p_2, p_3\}$ is the best combination found so far and $\Psi(P)$ is 58. Among the remaining k-combinations yet to be examined, some of them share common paths. For example, $\{p_1, p_2, p_4\}$, $\{p_1, p_2, p_5\}$ and $\{p_1, p_2, p_6\}$ have $\{p_1, p_2\}$ as common paths. We proceed to derive the lower bound aggregate value of the partial combination $\hat{P} = \{p_1, p_2, p_*\}$, denoted by $\Psi_{lb}(\hat{P})$, where p_* is either p_4, p_5 or p_6. $\Psi_{lb}(\hat{P})$ is calculated as $\sum_{j=1}^{m} \min\{\min_{p\in\{p_1,p_2\}} \tau_j(p), \min_{p_*\in\{p_4,p_5,p_6\}} \tau_j(p_*)\} = 15 + 16 + 12 + 14 + 8 = 65$. Since $\Psi_{lb}(\hat{P}) > \Psi(P_{best})$, we can safely discard the above three combinations.

Table 1: All Possible Paths from v_1 to v_7, with historical travel times

Path	τ_1	τ_2	τ_3	τ_4	τ_5
$p_1\langle v_1, v_2, v_3, v_7\rangle$	19	20	14	15	16
$p_2\langle v_1, v_4, v_3, v_7\rangle$	18	20	17	**14**	12
$p_3\langle v_1, v_4, v_7\rangle$	16	**10**	**6**	16	14
$p_4\langle v_1, v_5, v_6, v_7\rangle$	19	16	20	21	**8**
$p_5\langle v_1, v_5, v_4, v_3, v_7\rangle$	17	30	23	21	9
$p_6\langle v_1, v_5, v_4, v_7\rangle$	**15**	20	12	23	11

With the above lower bound aggregate value, we can prune an unpromising partial combination as follows:

PRUNING RULE 3 (PARTIAL COMBINATION PRUNING). *Let P_{best} be a known k-combination. Given a partial combination $\hat{P}$, if $\Psi_{lb}(\hat{P}) > \Psi(P_{best})$, then $\hat{P}$ can be pruned.*

The effectiveness of partial combination pruning depends on how early we can obtain a complete k-combination with a high pruning power, i.e., close to the optimal solution. In order to discover a good combination early, we adopt a branch-and-bound paradigm to expand a partial combination. We insert a candidate $p \in \mathcal{C}$ to a partial combination in ascending order of $\tau_{min}(p) = \min_{j=1}^{m} \tau_j(p)$. We also maintain a data structure to reduce the overhead of computing $\Psi_{lb}(\hat{P})$.

To sum up, we enumerate the optimal combination by the following procedure. We initialize an empty partial combination $\hat{P}$ and recursively insert a candidate $p \in \mathcal{C}$ into $\hat{P}$ in order of $\tau_{min}(p)$. If $\hat{P}$ contains fewer than k paths, then we compare its lower bound cost with that of the best complete combination P_{best} found so far. If $\hat{P}$ has a smaller cost, then we further expand it. When $\hat{P}$ contains k paths, we can compute its exact cost and update P_{best} to $\hat{P}$ if $\hat{P}$ is better.

3. HEURISTIC METHOD

In this section, we present a heuristic method called Top-Picker Algorithm (TP) to find a low-cost solution efficiently. Observe that, in our exact algorithm, the candidate set $\mathcal{C}$ is large even if we apply pruning rules in Phase I. Thus, this would lead to a huge number of path combinations in Phase II. In order to reduce the total computation cost, TP generates a bounded number of candidates (Phase I) by a heuristic and reuses the combination enumeration (Phase II) of the exact method.

The idea of TP is to limit the size of the candidate set $\mathcal{C}$, say, to at most m. It computes the shortest path sp_j (from v_s to v_t) at each time instant, and then inserts these paths into the candidate set $\mathcal{C}$.

Let us use Table 1 as an example with $k = 3$. First, we find the shortest paths at each time instants, which are p_6, p_3, p_3, p_2, p_4, respectively. Then, we insert these paths into the candidate set $\mathcal{C} = \{p_2, p_3, p_4, p_6\}$. In Phase II, we enumerate all 3-combinations of $\mathcal{C}$, such as $\{p_2, p_3, p_4\}, \{p_2, p_3, p_6\}, \{p_2, p_4, p_6\}, \{p_3, p_4, p_6\}$. Finally, we compute their costs and return $P_{opt} = \{p_2, p_3, p_4\}$.

Note that if the number of distinct shortest paths (i.e., the size of $\mathcal{C}$ in TP) is less than or equal to k, then its solution is optimal. Otherwise, its solution may not be optimal.

4. EXPERIMENTAL EVALUATION

4.1 Experimental setup

Road Network and Traffic Data: We used the road network of the United Kingdom (UK), as provided by [2]. The network has 2,321 nodes and 4,996 edges. We downloaded real and historical traffic data from [3] from January to March of 2013. The traffic data were recorded every 15 minutes and hence there are 96 traffic records per day for each road segment.

Training and testing sets of traffic data: In the introduction, we suggest two applications of our TTP problem. Transportation planning requires fast computation time, whereas online route services focuses on the travel time error of the pre-computed paths against real-time traffic. Thus, we measure the computation time and travel time error in our experiments. In order to evaluate the travel time error of the methods (travel time error measure will be discussed shortly), we divide the traffic data into *training* and *testing* sets. We take the training set for computing a path combination P and use the testing set for measuring the travel time error of P. By default, we use the historical traffic data collected during 1 - 15 March, 2013 as *training* set and the data collected during 16 - 31 March, 2013 as *testing* test. The default length of a time period is an hour (e.g., 08:00-09:00) unless specified. Therefore, the default value of m, i.e., the number of time instants, for computing P is 60.

Travel time error measures: For each method, we use the training data to compute its resulting path combination P. Then, we measure the travel time error of P by using the testing data. Specifically, we apply Equation 1, substitute P into $P^k_{s,t}$, and calculate

$\tau_j(p)$ and $\tau_j(sp_j)$ based on time instants j in the testing data.

Query: For generation of SD pairs, we pick 100 pairs uniformly at random from the road network. We report the *average travel time error* and the *average computation time* of these pairs in the subsequent sections. We choose $k = 5$ by default, which is the same as [8].

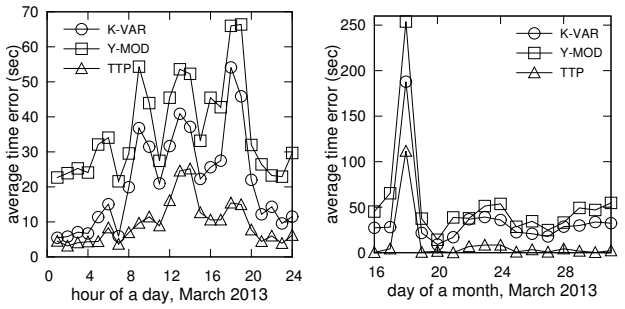
Methods: Our proposed methods are: the exact method (which is shown as TTP in figures) and Top-Picker algorithm (TP). Our competitors are two representative heuristic methods proposed in [8]: *K-variance* (**K-VAR**) and *Y-moderate* (**Y-MOD**). The additional parameters used by them are configured according to [8].

We implement all methods in C++. All the experiments were run on a PC with 3.4 GHz Intel® Core™ i7 CPU and 8 GB RAM in Linux environment.

4.2 Results

In this section, we present the travel time error and the efficiency of TTP from various perspectives including (i) different hours of a day, (ii) different days of a month and (iii) increasing the number of paths (k).

Different hours of a day. First, we evaluate how the average time error varies with different time periods of a day since this reveals some traffic patterns of UK. Figure 2(a) shows the average time errors of TTP and two competitors at different hours of a day. TTP achieves a smaller time error throughout the day and outperforms the others especially at the rush hours (i.e., 08:00 and 17:00) by at least 3 times. Although there is a spike at 13:00-14:00 for TTP its time error is still smaller than that of others. The figure also reveals that the traffic of UK fluctuates in three periods, namely 08:00-09:00, 13:00-14:00 and 17:00-18:00. The default time period in the subsequent experiments is 08:00-09:00.

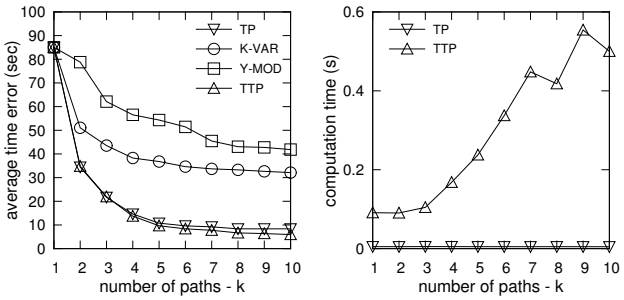


(a) varying time of a day (b) varying day of a month

Figure 2: $k = 5$, March 2013, UK

Different days of a month: Figure 2(b) shows the average time errors of three algorithms on different testing days of March, at 08:00-09:00. It can be seen that TTP consistently has smaller errors than the others. Although all of them suffer from a sudden rise in time error on 18 March, 2013, TTP can still obtain a much lower average time error of about 2 minutes while K-VAR and Y-MOD have average time errors of about 3 minutes and 4 minutes respectively.

Varying k: As shown in Figure 3(a), the average time errors of all algorithms diminish with increasing k and start to converge when $k > 5$. This is because including more paths implies it is more probable to have a path, out of k paths, with a smaller travel time error at a particular time instant. Obviously, the average time errors of TTP and TP decrease more rapidly than the others, indicating the higher marginal gain in accuracy. We also measure the computation costs of TTP and TP and present them in Figure 3(b).



(a) error (b) computation time

Figure 3: Varying k, 08:00-08:15 in March 2013, UK

The running time of TTP increases with k because the number of k-combinations enumerated also increases. In contrast, due to the small size of UK network, the number of candidates generated by TP in Phase I is limited and hence its running time remains steady.

5. CONCLUSIONS

This paper proposes and studies a novel problem called the k traffic-tolerant paths problem (TTP) in road networks, which takes a source-destination pair and historical traffic information as input, and returns k paths that minimize the aggregate historical travel time. Its applications include transportation analysis and efficient route-search services. We develop an exact algorithm and devise a heuristic for TTP. Finally, the experiments show that the exact algorithm and the heuristic achieve much higher accuracy than existing approaches.

Acknowledgments

The authors were supported by ICRG grant G-YN38 from the Hong Kong Polytechnic University.

6. REFERENCES

[1] Caltrans PeMS. http://pems.dot.ca.gov/.
[2] GB Road Traffic Counts. http://data.gov.uk/dataset/gb-road-traffic-counts/.
[3] Highways Agency Network Journey Time and Traffic Flow Data. http://data.gov.uk/dataset/dft-eng-srn-routes-journey-times/.
[4] Travel Time Reliability: Making It There On Time, All The Time. http://ops.fhwa.dot.gov/publications/tt_reliability/index.htm, 2006.
[5] U. Demiryurek, F. B. Kashani, C. Shahabi, and A. Ranganathan. Online Computation of Fastest Path in Time-dependent Spatial Networks. In *SSTD*, pages 92–111, 2011.
[6] M. C. González, C. A. Hidalgo, and A.-L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008.
[7] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding Fastest Paths on a Road Network with Speed Patterns. In *ICDE*, pages 10–10, 2006.
[8] N. Malviya, S. Madden, and A. Bhattacharya. A Continuous Query System for Dynamic Route Planning. In *ICDE*, pages 792–803, 2011.
[9] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.
[10] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest Path and Distance Queries on Road Networks: Towards Bridging Theory and Practice. In *SIGMOD*, pages 857–868, 2013.