Fastest-Path Computation

DONGHUI ZHANG
College of Computer & Information Science
Northeastern University

Synonyms

fastest route; driving direction

Definition

In the United states, only 9.3% of the households do not have cars. Driving is part of people's daily life. GIS systems like MapQuest and MapPoint are heavily relied on to provide driving directions. However, surprisingly enough, existing systems either ignore the driving speed on road networks, or assume the speed remains constant on the road segments. In both cases the users' preferred leaving time does not affect the query result. For instance, MapQuest does not ask the users to input the day and time of driving. However, during rush hours, inbound highways to big cities have much lower speed than usual. So a fastest path computed during non-rush hours, which may consists of some inbound highway segments, may not remain the fastest path during rush hours.

Consider a road network modelled as a graph, where each node is a road intersection and each edge is a road segment. Let each edge store a speed pattern, e.g. a piecewise-constant function. For instance, in a working day, during rush hour (say from 7am to 9am) the speed is 0.3 miles per minute (mpm), and at other times of the day the speed is 1mpm.

The ***Time Interval All Fastest Paths (allFP) Query*** is defined as follows. Given a source node s, an end node e, and a leaving time interval at s, the *allFP* query asks to enumerate all fastest paths, each corresponding to a disjoint sub-interval of leaving time. The union of all sub-intervals should cover the entire query time interval.

An allFP query example is: *I may leave for work any time between 7am and 9am; please suggest all fastest paths, e.g. take route A if the leaving time is between 7 and 7:45, and take route B otherwise*.

It is also interesting to solve the allFP problem with an arrival time interval. For instance: *I need to drive from my home to New York International Airport. Please suggest all fastest paths if the anticipated arrival time is between 5pm and 6pm*.

There are two characteristics that distinguish the allFP problem from other shortest/fastest path problems.

- The query is associated with a leaving/arrival time INTERVAL, not a time instant. In fact if the leaving time were fixed as a time instant, many existing algorithms could be applied, such as the Dijkstra's shortest path computation and the A* algorithm.
- Time is continuous. If time were distinct, e.g. one can only leave precisely at the top of the hours, one could run existing time-instant algorithms multiple times.

Historical Background

Most existing work on path computation has been focused on the shortest-path problem. Several extensions of the Dijkstra algorithm have been proposed, mainly focusing on the maintenance of the priority queue. The A* algorithm [8] finds a path from a given start node to a given end node by employing a heuristic estimate. Each node is ranked by an estimate of the best route that goes through that node. A* visits the nodes in order of this heuristic estimate. A survey on shortest-path computation appeared in [11].

Performance analysis and experimental results regarding the secondary-memory adaptation of shortest path algorithms can be found in [4, 13]. The work in [3] contributes on finding the shortest path that satisfies some spatial constraints. A graph index that can be used to prune the search space was proposed in [15].

One promising idea to deal with large-scale networks is to partition a network into fragments. The boundary nodes, which are nodes having direct links to other fragments, construct the nodes of a high-level, smaller graph. This idea of hierarchical path-finding has been explored in the context of computer networks [6] and in the context of transportation systems [5]. In [12], the materialization trade-off in hierarchical shortest path algorithms is examined.

In terms of fastest-path computations, there exists work assuming the discrete model [1, 9], the flow speed model [14], or theoretical models beyond road network [10]. The discrete model [1, 9] assumes discrete time. The flow speed model work [14] addresses the fastest path query for a given leaving time instant, not a time interval. The theoretical model work [10] suggests operations on travel functions that are necessary without investigating how this operations can be supported.

To compute fastest paths on a road network with a leaving/arrival time interval and with continuous time, one can utilize a novel extension of the A* algorithm. More details are given below.

Scientific Fundamentals

A simple extension to the A* algorithm can NOT be used to solve the allFP query. Let n_0 be the node to be expanded next and let n_0 have three neighbor nodes, n_1, n_2 and n_3. A* picks the neighbor node n_i ($i \in [1..3]$) to continue expanding if the travel time from s to n_i plus the estimated travel time from n_i to e is the smallest. The problem is that since the leaving time is not a fixed time instant, depending on the leaving time instant different neighbors should be picked.

One possible solution is to expand all such neighbors simultaneously. However, expanding all picked neighbors may result in an exponential number of paths being expanded regardless of the size of the answer set.

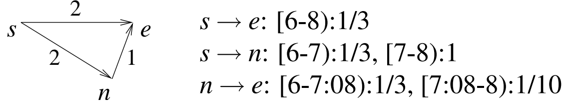
Instead, [7] proposed a new algorithm called **IntAllFastestPaths**. The main idea of the algorithm is summarized below:

1. Maintain a priority queue of expanded paths, each of which starts with s. For each path $s \Rightarrow n_i$, maintain $T(l, s \Rightarrow n_i) + T_{est}(n_i \Rightarrow e)$ as a piecewise-linear function of $l \in$ leaving time interval I. Here, $T(l, s \Rightarrow n_i)$ is the travel time from s to n_i, measured as a function of leaving time l. $T_{est}(n_i \Rightarrow e)$ is a lower bound estimation function of the travel time from n_i to the end node e. A straightforward lower-bound estimator is $d_{euc}(n_i, e)/v_{max}$, which is the Euclidean distance between n_i and e, divided by the max speed in the network. A better estimator called the *boundary node estimator* which is described later.
2. Similar to the A* Algorithm, in each iteration pick a path from the priority queue to expand. Pick the path, whose maintained function's minimum value during I is the minimum among all paths.
3. Maintain a special travel-time function called the *lower border function*. It is the lower border of travel time functions for all identified paths (i.e. paths already picked from the priority queue) that end to e. In other words, for any time instant $l \in I$, the lower border function has a value equal to the minimum value of all travel time functions of identified paths from s to e. This function consists of multiple travel time functions, each corresponding to some path from s to e and some subinterval of I during which this path is the fastest.
4. Stop either when there is no more path left in the priority queue, or if the path picked to be expand next has a minimum value no less than the maximum value of the lower border function. Report the lower border function as the answer to the allFP query.

Below is a running example. The example involves a simple road network given in Figure 1. The goal is to find the fastest path from s to e at some time during $I =$[6:50-7:05].

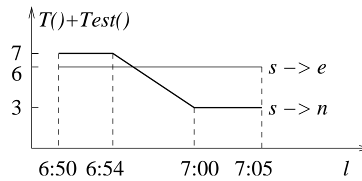
Initially, the priority queue contains only one entry, which corresponds to the unexpanded node s. It has two neighbors e and n. It can be derived that, $T(l \in \texttt{[6:50-7:05)}, s \rightarrow e) = 6min$ and
$T(l \in \texttt{[6:50-7:05)}, s \rightarrow n) =$

$$\begin{cases} 6, & l \in \texttt{[6:50-6:54)} \\ \frac{2}{3}(\texttt{7:00} - l) + 2, & l \in \texttt{[6:54-7:00)} \\ 2, & l \in \texttt{[7:00-7:05]} \end{cases}$$



Fastest-Path Computation. Figure 1 A simple road network. Distances are given on the edges. Speed patterns (#mpm) are given at the right of the network.

As expressed in step 1 of Algorithm IntAllFastestPaths, in the priority queue, the paths are ordered not by $T()$, but by $T() + T_{est}()$. The functions of the two paths are compared in Figure 2. Here, $T_{est}(n \Rightarrow e) = 1$min, since $d_{euc}(n, e) = 1$ mile and $v_{max} = 1$mpm.



Fastest-Path Computation. Figure 2 Comparison of the functions $T() + T_{est}()$ associated with paths $s \rightarrow e$ and $s \rightarrow n$.

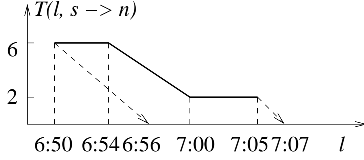
According to step 2, the path $s \rightarrow n$ to be expanded next, since its minimum value, 3, is smaller than the minimum value, 6, of the path $s \Rightarrow e$.

In general, to expand a path $s \Rightarrow n$, first all the required information for n and its adjacent nodes needs to be retrieved, Then, for each neighbor n_j of n the following steps need to be followed:

- Given the travel time function for the path $s \Rightarrow n$ and the leaving time interval I from s, determine the time interval during which the travel time function for the road segment $n \rightarrow n_j$ is needed.
- Determine the time instants $t_1, t_2, \ldots \in I$ at which the resulting function, i.e. the travel time function for the path $s \Rightarrow n_j$, $T(l \in I,\ s \Rightarrow n_j)$, changes from one linear function to another.
- For each time interval $[t_1, t_2), \ldots$, determine the corresponding linear function of the resulting function $T(l \in I,\ s \Rightarrow n_j)$.

In this example, the time interval for $n \rightarrow e$ is determined to be [6:56, 7:07] as shown in Figure 3. At time 6:50 (start of I), the travel time along the path $s \rightarrow n$ is 6 minutes. Therefore, the start of the leaving time interval for $n \rightarrow e$, i.e. the start of arrival time interval to n, is 6:50+6min = 6:56. Similarly, the end of the leaving time interval is 7:05+2min = 7:07.

During the time interval [6:56-7:07], the travel time

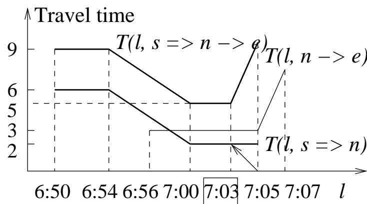


Fastest-Path Computation. Figure 3 The time interval, [6:56-7:07], during which the speed on $n \rightarrow e$ is needed.

on $n \rightarrow e$, $T(l \in [\texttt{6:56-7:07}], n \rightarrow e)$ is

$$\begin{cases} 3, & \text{if } l \in [\texttt{6:56-7:05}) \\ 10 - \frac{7}{3}(\texttt{7:08} - l), & \text{if } l \in [\texttt{7:05-7:07}] \end{cases}$$

There are two cases that trigger the resulting travel time function $T(l, s \Rightarrow n \rightarrow e)$ to change from one linear function to another. In the first, simple case the function $T(l, s \Rightarrow n)$ changes. The time instants at which the resulting function changes are the ones at which $T(l, s \Rightarrow n)$ changes. In Figure 4, these correspond to time instants 6:50, 6:54 and 7:00. In the second, trickier case, the changes of the resulting function are triggered by the changes of $T(l, n \rightarrow e)$, e.g. at time 7:05. In this example, one can determine that at time 7:03, $T(l, s \Rightarrow n \rightarrow e)$ changes. The reason is that if one leaves s at 7:03, since the travel time on $s \Rightarrow n$ is 2 minutes, one will arrive at n at 7:05. At that time the travel time function of $n \rightarrow e$ changes. To find the time instant 7:03, compute the intersection of the function $T(l, s \Rightarrow n)$ with a 135° line passing through the point (7:05, 0). The time instant 7:03 is the leaving time corresponding to that intersection point.



Fastest-Path Computation. Figure 4 The time instants at which $T(l, s \Rightarrow n \rightarrow e)$ changes to another linear function, and the $T(l, s \Rightarrow n \rightarrow e)$ function.

Now that all the four time instants 6:50, 6:54, 7:00, and 7:03 have been determined, the 4-piece function $T(l \in I, s \Rightarrow n \rightarrow e)$ can be derived by combining $T(l, s \Rightarrow n)$ and $T(n \rightarrow e)$.

For each l, $T(l, s \Rightarrow n \rightarrow e)$ is equal to $T(l, s \Rightarrow n)$ plus $T(l', n \rightarrow e)$, where l' is the time at which node n is reached. That is, $l' = l + T(l, s \Rightarrow n)$. The following algorithm should be used to expand a path, for every identified time instant $t \in \{t_1, t_2, \dots\}$ (e.g. 6:50):

- Retrieve the linear function of $T(l, s \Rightarrow n)$ at time t. Let it be $\alpha * l + \beta$.
- Retrieve the linear function of $T(l', n \rightarrow e)$ at time $t' = t + (\alpha * t + \beta)$. Let it be $\gamma * l' + \delta$.
- Compute a new linear function $(\alpha * l + \beta) + (\gamma * (l + \alpha * l + \beta) + \delta))$, which can be re-written as $(\alpha * \gamma + \alpha + \gamma) * l + (\beta * \gamma + \beta + \delta)$. This is the linear function as part of $T(l, s \Rightarrow n \rightarrow e)$, for the time interval from t to the next identified time instant.

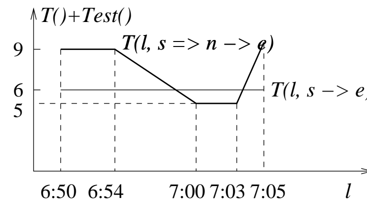
For instance, the combined function $T(l \in I, n \Rightarrow e)$, which is shown in Figure 4, is computed as follows. At t =6:50, the first linear function is a constant function 6. So $t' = t + 6$=6:56. The second linear function starting with 6:56 is another constant function 3. So the combined function is 9, which is valid until the next identified time instant.

At t =6:54, the first linear function is $\frac{2}{3}(\texttt{7:00} - l) + 2$. Therefore t' =6:54+6=7:00. The second linear function is 3. The combined function is $\frac{2}{3}(\texttt{7:00} - l) + 5$.

At t =7:00, the first function is constant 2. At t' =7:00+2=7:02, the second function is 3. So the combined function is 5.

Finally, at t =7:03, the first function is 2, and at t' =7:03+2=7:05, the second function as $10 - \frac{7}{3}(\texttt{7:08} - l')$. And thus the combined function is $2 + (10 - \frac{7}{3}(\texttt{7:08} - (l+2))) = 12 - \frac{7}{3}(\texttt{7:06} - l)$.

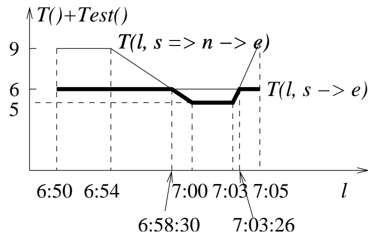
After the expansion, the priority queue contains two functions, as shown in Figure 5. Note that in both functions, the lower bound estimation part is 0, since both paths already end with e.



Fastest-Path Computation. Figure 5 The two functions in the priority queue.

The next step of Algorithm IntAllFastestPaths is to pick the path $s \Rightarrow n \rightarrow e$, as its minimum value (5min) is globally the smallest in the queue. An important question that arises here is when to stop expanding, as expanding all paths to the end node is prohibitively expensive. The algorithm terminates when the next path has a minimum value no less than the maximum value of the maintained *lower border function*.

When there is only one identified path that ends with e, the lower border function is the function of this path. In Figure 5, $T(l, s \Rightarrow n \rightarrow e)$ is the lower border function. As each new path ending with e is identified, its function is combined with the previous lower border



Fastest-Path Computation. Figure 6 The lower border and the result for Query 3.

function. E.g. in Figure 6 the new lower border function, after the function $T(l,\ s \rightarrow e)$ is removed from the priority queue, is shown as the thick polyline.

The algorithm can terminate if the next path to be expanded has a minimum value no less than the maximum value of the lower border function (in this case, 6).Since the maximum value of the lower border keeps decreasing, while the minimum travel time of paths in the priority queue keeps increasing, the algorithm IntAllFastestPaths is expected to terminate very fast. In this example, the set of all fastest paths from s to e when $l \in$[6:50-7:05] is:

$$\begin{cases} s \rightarrow e, & \texttt{if}\ l \in \texttt{[6:50-6:58:30)} \\ s \rightarrow n \rightarrow e, & \texttt{if}\ l \in \texttt{[6:58:30-7:03:26)} \\ s \rightarrow e, & \texttt{if}\ l \in \texttt{[7:03:26-7:05]} \end{cases}$$

Finally, the boundary-node estimator, which is a lower-bound travel time from n_i to e and which is used to improve the efficiency of the algorithm, is described below.

- Partition the space into non-overlapping cells [2]. A **boundary node** [6] of a cell is a node directly connected with some other node in a different cell. That is, any path linking a node in a cell C_1 with some node in a different cell C_2 must go through at least two boundary nodes, one in C_1 and one in C_2.
- For each pair of cells, (C_1, C_2), pre-compute the fastest travel time (function) from each boundary node in C_1 to each boundary node in C_2.
- For each node inside a cell, pre-compute the fastest travel time from and to each boundary node.
- At query time, n_i and e are given. Since any path from n_i to e must go through some boundary node in the cell of n_i and through some boundary node in the cell of e, a lower-bound estimator can be derived as the summation of three parts: (1) the fastest time from n_i to its nearest boundary node, (2) the fastest time from some boundary node in n_i's cell to some boundary node in e's cell, and (3) the fastest time from e's nearest boundary node to e.

Key Applications

The key application of fastest-path computation is road navigation systems. Example systems are mapquest.com, local.live.com, and MapPoint. Such systems can produce better driving directions if integrated with traffic patterns and fastest-path computation techniques.

Future Directions

To speed up the calculation, the road network should be partitioned. At the index level, each partition is treated as a single network node and the details within each partition are omitted. Pre-computation are performed to calculate the travel time from each input edge to each output edge. The partitioning can be performed hierarchically. Another direction is that, to simplify the computed fastest paths, the algorithm should be extended to allow the users to provide a maximum number changes in road names.

Cross References

1. Dynamic Travel Time Maps
2. Modeling Road Networks
3. Query Processing in Road Network Databases
4. Routing Algorithms
5. Trip Planning Queries in Road Network Databases
6. Vehicle Routing Algorithms

Recommended Reading

[1] I. Chabini. Discrete Dynamic Shortest Path Problems in Transportation Applications. *Transportation Research Record*, 1645:170–175, 1998.

[2] V.P. Chakka, A. Everspaugh, and J.M. Patel. Indexing Large Trajectory Data Sets With SETI. In *Biennial Conf. on Innovative Data Systems Research (CIDR)*, 2003.

[3] Y. Huang, N. Jing, and E. Rundensteiner. Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations. In *VLDB*, pages 396–405, 1997.

[4] B. Jiang. I/O-Efficiency of Shortest Path Algorithms: An Analysis. In *ICDE*, pages 12–19, 1992.

[5] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *TKDE*, 10(3):409–432, 1998.

[6] F. Kamoun and L. Kleinrock. Hierarchical Routing for Large Networks: Performance Evaluation and Optimization. *Computer Networks*, 1:155–174, 1977.

[7] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding Fastest Paths on A Road Network with Speed Patterns. In *ICDE*, 2006.

[8] R.-M. Kung, E. N. Hanson, Y. E. Ioannidis, T. K. Sellis, L. D. Shapiro, and M. Stonebraker. Heuristic Search in Data Base Systems. In *Expert Database Systems Workshop (EDS)*, pages 537–548, 1984.

[9] K. Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83:154–166, 1995.

[10] A. Orda and R. Rom. Minimum Weight Paths in Time-Dependent Networks. *Networks: An International Journal*, 21, 1991.

[11] S. Pallottino and M. G. Scutellà. Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. In P. Marcotte and S. Nguyen, editors, *Equilibrium and Advanced Transportation Modelling*, pages 245–281. Kluwer Academic Publishers, 1998.

[12] S. Shekhar, A. Fetterer, and B. Goyal. Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. In *SSTD*, pages 94–111, 1997.

[13] S. Shekhar, A. Kohli, and M. Coyle. Path Computation Algorithms for Advanced Traveller Information System (ATIS). In *ICDE*, pages 31–39, 1993.

[14] K. Sung, M.G.H. Bell, M. Seong, and S. Park. Shortest paths in a network with time-dependent flow speeds. *European Journal of Operational Research*, 121(1):32–39, 2000.

[15] J. L. Zhao and A. Zaki. Spatial Data Traversal in Road Map Databases: A Graph Indexing Approach. In *Proc. of Int. Conf. on Information and Knowledge Management (CIKM)*, pages 355–362, 1994.