International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

# Review Paper on Online Shortest Path Computation

**Pratik P.Watane[1], Prof. Prachi V. Kale[2]**

[1]ME (CSE) Scholar, Department of CSE, P R Patil College of Engg. & Tech., Amravati-444602, India
[2]AssitantantProfessor, Department of CSE, P R Patil College of Engg. & Tech., Amravati-444602, India

**Abstract:** ***The online shortest path problem aims at computing the shortest path based on live traffic circumstances. The problem of point-to-point fastest path computation in static spatial networks is extensively studied with many pre computation techniques proposed to speed up the computation. Most of the existing approaches make the simplifying assumption that travel times of the network edges are constant. However, the real world spatial networks the edge travel times are time dependent on the arrival time to an edge determines the actual travel time on the edge. we have study online computation of fastest path in time-dependent spatial networks and present a technique which speeds-up the path computation. We show that our fastest path computation based on a bidirectional time-dependent A* search significantly improves the computation time and storage capacity. With extensive experiments using real data-sets (including a variety of large spatial networks with real traffic data) we demonstrate the efficacy of our proposed techniques for online fastest path computation.***

**Keywords:** Shortest path, time dependent spatial network, fastest path computation

## 1. Introduction

Shortest path computation is an important function in modern car navigation systems. The always growing popularity of online map applications and their wide deployment in mobile devices and car navigation systems, increase number of client search for point to point fastest paths and the corresponding travel times. On static road networks where edge costs are constant, this problem has been extensively studied and many efficient speedup techniques have been developed to compute the fastest path in a matter of milliseconds [1, 2, 3,4]. The static fastest path approaches make the simplify assumption that the travel time for each edge of the road network is constant. However, in real world the actual travel time on a road segment heavily depends on the traffic congestion and, therefore, there is a function of time i.e., time dependent. For example, Figure 1 shows the variation of travel time for a Particular road segment of I-10 freeway in Los Angeles as a function of arrival-time to the segment. As shown, the travel-time changes with time and the change in travel-time is significant. For instance, from 8AM to 9AM the travel-time of the segment changes from 32 minutes to 18 minutes. By induction, one can observe that the time dependent edge travel times yield a considerable change in the actual fastest path between any pair of nodes throughout the day. Specifically, the fastest between a source and a destination node varies depending on the departure time from the source. Unfortunately, all those techniques that assume constant edge weights fail to address the fastest path computation in real world time dependent spatial networks. The time dependent fastest path problem was first shown by Dreyfus [5] to be solvable super-polynomial in FIFO networks by a trivial modification to Dijkstra algorithm where, analogous to shortest path distances, the arrival time to the nodes is used as the labels that form the basis of the greedy algorithm.

The FIFO property which typically holds for many networks including road networks, suggests that moving objects exit from an edge in the same order they entered the edge1.

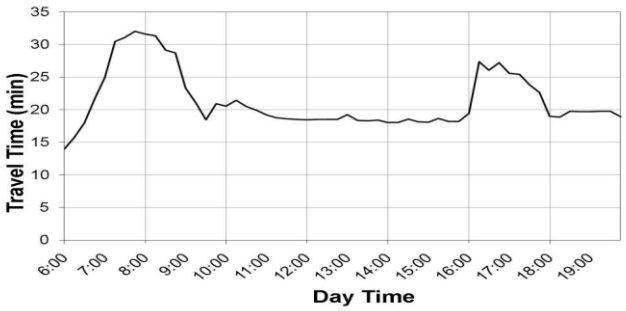



**Figure 1**

However, the modified Dijkstra algorithm [5] is far too slow for online map applications which are usually deployed on very large networks and require almost instant response times. On the other side, there are many efficient pre computation approaches that answer fastest path queries in near real time (e.g., [1]) in static road networks. However, it is infeasible to extend these approaches to time dependent networks. This is because the input size (i.e., the number of fastest paths) increases drastically in time dependent networks. Specifically, since the length of a*s-d* path changes depending on the departure time from *s*, the fastest path is not unique for any pair of nodes in time dependent networks. It has been conjectured in [6] and settled in [7] that the number of fastest paths between any pair of nodes in time dependent road networks can be super-polynomial. Hence, an algorithm which considers the every possible path (corresponding to every possible departure-time from the source) for any pair of nodes in large time-dependent networks would suffer from exponential time and prohibitively large storage requirements. For example, the time dependent extension of Contraction Hierarchies (CH) [8] and SHARC [9] speed-up techniques suffer from the impractical precomputation times and intolerable storage complexity.

## 2. Related Work

In the last decade, numerous efficient fastest path algorithms with pre computation methods have been proposed. However, there are limited numbers of studies that focus on efficient computation of time dependent

fastest path (TDFP) problem. Cooke and Halsey [10] first studied TDFP computation where they solved the problem using Dynamic Programming in discrete time. Another discrete time solution to TDFP problem is to use time expanded networks [11]. The time expanded network (TEN) and discrete time approaches assume that the edge weight functions are defined over a finite discrete window of time $t \in t_0, t_1, .., t_n$, where $t_n$is determined by the total duration of time interval under the consideration. Therefore, the problem is reduced to the problem of computing minimum weight paths over a static network per time window. Hence, we can apply any static fastest path algorithms to compute TDFP. Although these algorithms are easy to design and implement, they have numerous shortcomings. First, TEN models create a separate instance of network for each time instance hence yielding a substantial amount of storage overhead. Second, such approaches can only provide approximate results because the model misses the state of the network between any two discrete time instants. Moreover, the difference between the shortest path obtained using TEN approaches and the optimal shortest path is unbounded. This is because the query time can be always between any two of the intervals which are not captured by the model, and hence the error is accumulated on eac edge along the path. In [12], George and Shekhar proposed a time aggregated graph approach where they aggregate the travel times of each edge over the time instants into a time series.

Their model requires less space than that of the TEN and the results are still approximate with no bounds. In [5], Dreyfus showed that TDFP problem can be solved by a generalization of Dijkstra's method as efficiently as for static fastest path problems. However, Halpern [13] proved that the generalization of Dijkstra's algorithm is only true for FIFO networks. If the FIFO property does not hold in a time dependent network, then the problem is NP-Hard. In [14], Orda and Rom introduced Bellman-Ford based algorithm where they determine the path toward destination by refining the arrival time functions on each node in the whole time interval T. In [15], Kanoulas et al. proposed Time Interval All Fastest Path (allFP) approach in which they maintain a priority queue of all paths to be expanded instead of sorting the priority queue by scalar values.

They enumerate all the paths from source to a destination node which incurs exponential running time in the worst case. In [16], Ding et al. used a variation of Dijkstra's algorithm to solve the TDFP problem. With their TDFP algorithm, using Dijkstra like expansion, they decouple the path selection and time refinement (computing earliest arrival-time functions for nodes) for a given starting time interval T. Their algorithm is also shown to run in exponential time for special cases [17]. The focus of both [15] and [16] is to find the fastest path in time dependent road networks for a given start time interval. The ALT algorithm [18] was originally proposed to accelerate fastest path computation in static road networks. With ALT, a set of nodes called landmarks are chosen and then the shortest distances between all the nodes in the network and all the landmarks are computed and stored. ALT employs triangle inequality based on distances to the landmarks to obtain a heuristic function to be used in A* search. The time dependent variant of this technique is studied in [19] (unidirectional) and [20] (bidirectional A* search) where heuristic function is computed w.r.t lower-bound graph. However, the landmark selection is very difficult and the size of the search space is severely affected by the choice of landmarks. So far no optimal strategy with respect to landmark selection and random queries has been found. Specifically, landmark selection is NP-hard [21] and ALT does not guarantee to yield the smallest search spaces with respect to fastest path computations where source and destination nodes are chosen at random. Our experiments with real world time dependent travel-times show that our approach consumes much less storage as compared to ALT based approaches and yields faster response times.

In two different studies, The Contraction Hierarchies (CH) and SHARC methods (also developed for static networks) were augmented to time-dependent road networks in [8] and [9], respectively. The main idea of these techniques is to remove unimportant nodes from the graph without changing the fastest path distances between the remaining (more important) nodes. However, unlike the static networks, the importance of a node can change throughout the time under consideration in time dependent networks, hence the importance of the nodes are time varying. Considering the super-polynomial input size, and hence the super-polynomial number of important nodes with time-dependent networks, the main shortcomings of these approaches are impractical pre-processing times and extensive space consumption. For example, the precomputation time for SHARC in time dependent road networks takes more than 11 hours for relatively small road networks (e.g. LA with 304,162 nodes) [9]. Moreover, due to the significant use of arc flags [9], SHARC does not work in a dynamic scenario: whenever an edge cost function changes, arc flags should be recomputed, even though the graph partition need not be updated. While CH also suffers from slow pre-processing times, the space consumption for CH is at least 1000 bytes per node for less varied edge-weights where the storage cost increases with real-world time-dependent edge weights. Therefore, it may not be feasible to apply SHARC and CH to continental size road networks which can consist of more than 45 million road segments (e.g., North America road network) with possibly large varied edge weights.

## 3. Time Dependent Path Planning

We explain the difference between fastest computation in time dependent and static spatial networks. We also discuss the importance and the feasibility of time dependent route planning. To illustrate why classic fastest path computations in static road networks may return non-optimal results, we show a simple example in Figure 2 where a spatial network is modelled as a time dependent graph and edge travel times are function of time. Consider the snapshot of the network with edge weights corresponding to travel-time values at $t$=0. With classic fastest path computation approaches that disregard time dependent edge travel-times, the fastest path from *s* to *d*

goes through $v_1$, $v_2$, $v_4$ with a cost of 13 time units. However, by the time when $v_2$is reached (i.e., at $t$=5), the cost of edge $e(v_2, v_4)$ changes from 8 to 12 time units, and hence reaching $d$ through $v_2$ takes 17 time units instead of 13 as it was anticipated at $t$=0. In contrast, if the time-dependency of edge travel times are considered and hence the path going through $v$3 was taken, the total travel cost would have been 15 units which is the actual optimal fastest path. We call this shortcoming of the classic fastest path computation techniques as no look ahead problem. Unfortunately, most of the existing state of the art path planning applications suffers from the no look ahead shortcoming and, hence, their fastest path recommendation remains the same throughout the day regardless of the departure time from the source. Although some of these applications provide alternative paths under traffic conditions (which may seem similar to time dependent planning at first), we observe that the recommended alternative paths and their corresponding travel times still remain unique during the day, and hence no time dependent planning. To the best of knowledge, these applications compute *top-k* fastest paths (i.e., *k* alternative paths) and their corresponding travel times with and without taking into account the traffic conditions. The travel times which take into account the traffic conditions are simply computed by considering increased edge weights (that corresponds to traffic congestion) for each path. However, our time dependent path planning results in different optimum paths for different departure times from the source. For example, consider Figure 3(a) where Google Maps offer two alternative paths (and their travel times under no-traffic and traffic conditions) for an origin and destination pair in Los Angeles road network. Note that the path recommendation and the travel times remain the same regardless of when the user submits the query. On the other hand, Figure 3(b) depicts the time dependent path recommendations (in different colours for different departure times) for the same origin and destination pair where we computed the time-dependent fastest paths for 38 consecutive departure times between 8AM and 5:30PM, spaced 15 minutes apart2. As shown, the optimal paths change frequently during the course of the day.

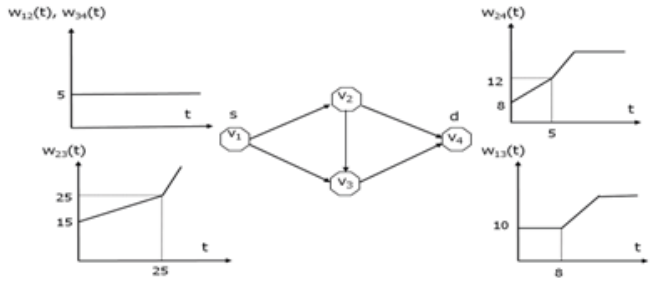
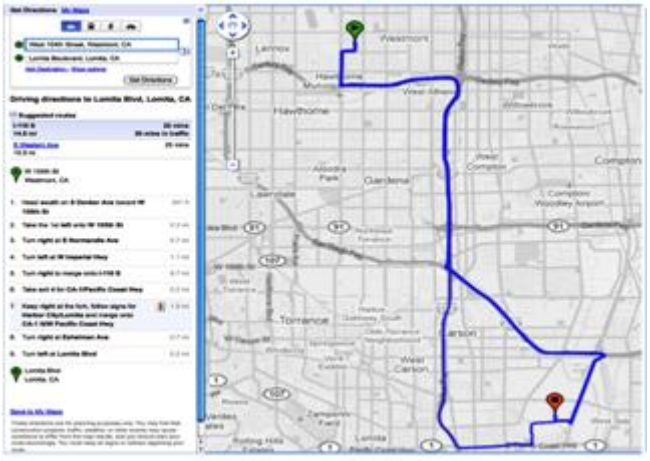
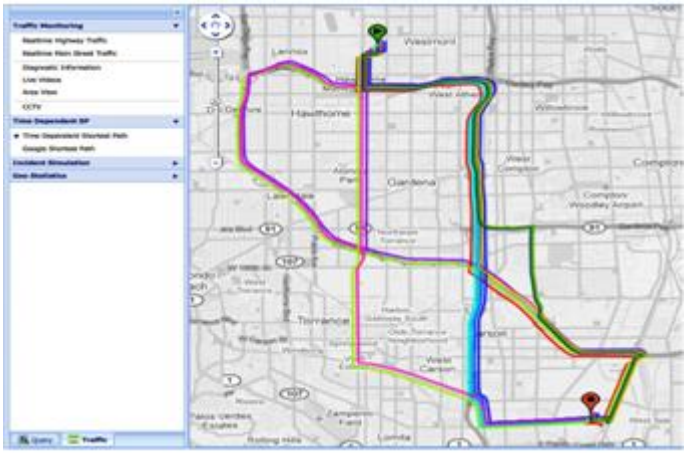



**Figure 2:** Time-dependent graph

One may argue against the feasibility of time dependent path planning algorithms due to a unavailability of the time-dependent edge travel-times, or b) negligible gain of time dependent path planning (i.e., how much time-dependent planning can improve the travel-time) over static path planning. To address the first argument, note that recent advances in sensor networks enabled instrumentation of road networks in major cities for collecting real time traffic data, and hence it is now feasible to accurately model the time dependent travel times based on the vast amounts of historical data. For instance, at our research centre. Meanwhile, we also witness that the leading navigation service providers started releasing their time dependent travel time data for road networks at high temporal resolution. With regards to the second argument, several recent studies showed the importance of time dependent path planning in road networks where real-world traffic datasets have been used for the assessment. For example, in [23] we report that the fastest path computation that considers time-dependent edge travel-times in Los Angeles road network decreases the travel-time by as much as 68% over the fastest path computation that assumes constant edge travel-times. We made the similar observation in another study [24] under IBM's Smart Traffic Project where the time-dependent fastest path computation in Stockholm road network can improve the travel time accuracy up to 62%. Considering the availability of high resolution time dependent travel time data for road networks, and the importance of time dependency for accurate and useful path planning, the need for efficient algorithms to enable next-generation time dependent path planning applications becomes apparent and immediate.



(a) Static path planning



(b) Time-dependent path planning

**Figure 3:** Static vs. Time-dependent path planning

## 4. Proposed Work

We propose a bidirectional time dependent fastest path algorithm (BTDFP)based on A* search [25]. There are two main challenges to employ bidirectional A* search in time dependent networks. First, finding an admissible heuristic function (i.e., lower bound distance) between an intermediate $v_i$ node and the destination $d$ is challenging as the distance between $v_i$and $d$ changes based on the departure time from $v_i$. Second, it is not possible to implement a backward search without knowing the arrival time at the destination. We address the former challenge by partitioning the road network to non-overlapping partitions (an off-line operation) and precompute the intra (node-to-border) and inter (border-to-border) partition distance labels with respect to Lower-bound Graph G which is generated by substituting the edge travel times in *G*with minimum possible travel-times. We use the combination of intra and inter distance labels as a heuristic function in the online computation. To address the latter challenge, we run the backward search on the lower-bound graph (*G*) which enables us to filter-in the set of the nodes that needs to be explored by the forward search.

We explain our bidirectional time dependent fastest path approach that we generalize bidirectional A* algorithm proposed for static spatial networks [26] to time dependent road networks. Our proposed solution involves two phases. At the precomputation phase, we partition the road network into non-overlapping partitions and precompute lower-bound distance labels within and across the partitions with respect to G(V,E). Successively, at the online phase, we use the pre computed distance labels as a heuristic function in our bidirectional time dependent A* search that performs simultaneous searches from source and destination. As showed in [5], the time dependent fastest path problem can be solved by modifying Dijkstra algorithm. We refer to modified Dijkstra algorithm as time dependent Dijkstra (TD-Dijkstra). TD-Dijkstra visits all network nodes reachable from *s* in every direction until destination node *d* is reached. On the other side, a time dependent A* algorithm can significantly reduce the number of nodes that have to be traversed in TD-Dijkstra algorithm by employing a heuristic function h(v) that directs the search towards destination. To guarantee optimal results, h(v) must be admissible and consistent (a.k.a, monotonic). The admissibility implies that h(v) must be less than or equal to the actual distance between v and d. With static road networks where the length of an edge is constant, Euclidian distance between *v* and *d* is used as h(v). However, this simple heuristic function cannot be directly applied to time-dependent road networks, because, the optimal travel-time between v and d changes based on the departure-time $t_v$ from v. Therefore, in time-dependent road networks, we need to use an estimator that never overestimates the travel-time between v and d for any possible tv. One simple lower-bound estimator is deuc(v, d)/max(speed), i.e., the Euclidean distance between v and d divided by the maximum speed among the edges in the entire network. Although this estimator is guaranteed to be a lower-bound, it is a very loose bound, and hence yields insignificant pruning. With our approach, we obtain a much tighter bound by utilizing the pre computed distance labels. Assuming that an online time dependent fastest path query requests a path from source *s* in partition *Si* to destination *d* in partition S. The fastest path must pass through from one border node $b_i$ in $S_i$ and another border node $b_j$ in $S_j$. We know that the time-dependent fastest path distance passing from $b_i$ and $b_j$ is greater than or equal to the pre computed lower-bound border-to-border (e.g., $LTT$ ($b_l$, $b_t$)) distance for $S_i$ and $S_j$ pair. We also know that a time-dependent fastest path distance from s to $b_i$ is always greater than or equal to the precomputed lower-bound fastest path distance of s to its nearest border node $b_s$. Analogously, same is true from the border node $b_d$(i.e., nearest border node) to din $S_j$. Thus, we can compute a lower-bound estimator of s by $h(s) = LTT(s, b_s) + LTT(b_l, b_t) + LTT(b_d, d)$.

**Lemma 1.**Given an intermediate node vi in $S_i$ and destination node d in $S_j$, the estimator $h(v_i)$ is admissible, i.e., a lower bound of time-dependent fastest path distance from $v_i$ to d passing from border nodes bi and $b_j$ in $S_i$ and $S_j$ ,respectively.

*Proof.*Assume $LTT(b_l, b_t)$ is the minimum border-to-border distance between *Si* and $S_j$, and $b'_l$,$b'_j$ are the nearest border nodes to *vi* and *d* in *G*, respectively. By definition of $G(V,E)$, $LTT(v_i, b'_i) \leq TDFP(v_i, b_i, t_{vi})$, $LTT(b_l, b_t) \leq TDFP(b_i, b_j, t_{bi})$, and $LTT(b'_j, d) \leq TDFP(b_j, d, t_{bj})$ Then, we have $h(v_i) = LTT(v_i, b'_i) + LTT(b_l, b_t) + LTT(b'_j, d) \leq TDFP(v_i, b_i, t_{vi}) + TDFP(b_i, b_j, t_{bi}) + TDFP(b_j, d, t_{bj})$

We can use our $h(v)$ heuristic with unidirectional time-dependent A* search in road networks. The time-dependent A* algorithm is a best-first search algorithm which scans nodes based on their time-dependent cost label (maintained in a priority queue) to source similar to [5]. The only difference to [5] is that the label within the priority queue is not determined only by the time-dependent distance to source but also by a lower-bound of the distance to *d*, i.e., $h(v)$ introduced above. To further speed-up the computation, we propose a bidirectional search that simultaneously searches forward from the source and backwards from the destination until the search frontiers meet. However, bidirectional search is challenging in time-dependent road networks for two following reasons. First, it is essential to start the backward search from the arrival-time at the destination $t_d$ and exact $t_d$ cannot be evaluated in advance at the query time (recall that arrival-time to destination depends on the departure time from the source in time-dependent road networks). We address this problem by running a backward A* search that is based on the reverse lower-bound graph *G* (the lower-bound graph with every edge reversed). The main idea with running backward search in*G* is to determine the set of nodes that will be explored by the forward A* search. Second, it is not straightforward to satisfy the consistency (the second optimality condition of A* search) of $h(v)$ as the forward and reverse searches use different distance functions. Next, we explain bidirectional time-dependent A* search algorithm (Algorithm 1) and how we satisfy the consistency. Given $G = (V, E, T)$, *s* and *d*, and departure-time *ts* from *s*, let $Q_f$ and $Q_b$ represent the two priority queues that maintain the

labels of nodes to be processed with forward and backward A* search, respectively. Let $F$ represent the set of nodes scanned by the forward search and $N_f$ is the corresponding set of labelled vertices (those in its priority queue). We denote the label of a node in $N_f$ by $d_{fv}$. Analogously, we define $B$, $N_b$, and $d_{fv}$ for the backward search. Note that during the bidirectional search $F$ and $B$ are disjoint but $N_f$ and $Nb$ may intersect. We simultaneously run the forward and backward A* searches on $G(V, E, T)$ and $G$, respectively (Line 4 in Algorithm 1). We keep all the nodes visited by backward search in a set $H$ (Line 5). When the search frontiers meet, i.e., as soon as $N_f$ and $N_b$ have a node $u$ in common (Line 6), the cost of the time-dependent fastest path ($TDFP(s, u, t_s)$) from $s$ to $u$ is determined.

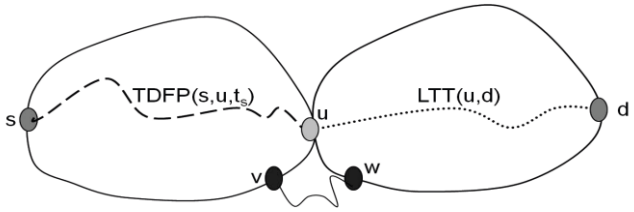



**Fig. 4.**Bidirectional search

At this point, we know that $TDFP(u, d, t_u) > LTT(u, d)$ for the path found by the backward search. Hence, the time-dependent cost of the paths (found so far) passing from $u$ is the upper-bound of the time-dependent fastest path from $s$ to $d$, i.e., $TDFP(s, u, t_s)+ TDFP(u, d, t_u) \geq TDFP(s, d, t_s)$.If we stop the searches as soon as a node $u$ is scanned by both forward and backward searches, we cannot guarantee finding the time-dependent fastest path from $u$ to $d$ within the set of nodes in $H$. This is due to inconsistent potential function used in bidirectional search that relies on two independent potential functions for two inner A* algorithms. Specifically, let $h_f(v)$ (estimated distance from node $v$ to target) and $h_b(v)$ (estimated distance from node $v$ to source) be the potential functions used in the forward and backward searches, respectively. With the backward search, each original edge $e(i, j)$ considered as $e(j, i)$ in the reverse graph where $h_b$ used as the potential function, and hence the reduced cost3 of $e(j, i)$ w.r.t. $h_b$ is computed by $c_{hb}(j, i)=c(i, j)- h_b(j)+h_b(i)$ where $c(i, j)$ is the cost in the original graph. Note that $h_f$ and $h_b$ are consistent if, for all edges $(i, j)$, $c_{hf}(i, j)$ in the original graph is equal to $c_{hb}(j, i)$ in the reverse graph. If $h_f$ and $h_b$ are not consistent, there is no guarantee that the shortest path can be found when the search frontiers meet. For instance, consider Figure 6 where the forward and backward searches meet at node $u$. As shown, if $v$ is scanned before $u$ by the forward search, then $TDFP(s, u, t_s) > TDFP(s, v, t_s)$. Similarly if $w$ is scanned before $u$ by the backward search, the $LTT(u, d)>LTT(w, d)$ and hence $TDFP(u, d, t_u) > TDFP(w, d, t_w)$. Consequently, it is possible that $TDFP(s, u,t_s) + TDFP(u, d, t_u) \geq TDFP(s, v, t_s) + TDFP(w, d, t_w)$. To address this challenge, one needs to find a) a consistent heuristic function and stop the search when the forward and backward searches meet or b) a new termination condition. In this study, we develop a new termination condition (the proof of correctness is given below) in which we continue both searches until the $Q_b$ only contains nodes whose labels exceed $TDFP(s, u, t_s) + TDFP(u, d, t_u)$ by adding all visited nodes to $H$ (Line 9-11). Recall that the label (denoted by $d_{bv}$) of node $v$ in the backward search priority queue $Qb$is computed by the time-dependent distance from the destination to $v$ plus the lower-bound distance from $v$ to $s$, i.e., $d_{bv}= TDFP(v, d, t_v) + h(v)$. Hence, we stop the search when $dbv> TDFP(s, u, t_s) + TDFP(u, d, t_u)$. As we explained, $TDFP(s, u, t_s) + TDFP(u, d, t_u)$ is the length of the fastest path seen so far (not necessarily the actual fastest path) and is updated during the search when a new common node $u\_$ found with $TDFP(s, u\_, ts)+TDFP(u\_, d, tu\_) <TDFP(s, u, ts)+TDFP(u, d, tu)$. Once both searches stop, $H$ will include all the candidate nodes that can possibly be part of the time-dependent fastest path to $d$. Finally, we continue the forward search considering only the nodes in $H$ until we reach $d$ (Line 12).

**Algorithm 1.**B-TDFP Algorithm
1. Input: $G_T$,$G$, s:source, d:destination,$ts$:departure time
2. Output: a ($s$, $d$, $ts$) fastest path
3. $FS()$:forward search, $BS()$:backward search, $Nf$/$Nb$: nodes scanned by FS()/BS(),$d_{bv}$:label of the minimum element in BS queue
4. $FS(GT)$ and $BS(G)$ //start searches simultaneously
5. $N_f \leftarrow FS(GT)$ and $N_b \leftarrow BS(G)$
6. *If* $N_f \cap N_{b\_} = \emptyset$ *then* $u \leftarrow N_f \cap N_b$
7. $M = TDFP(s, u, t_s) + TDFP(u, d, t_u)$
8. *end If*
9. *While* $d_{bv} > M$
10. $N_b \leftarrow BS(G)$
11. *EndWhile*
12. $FS(N_b)$
13. *return* ($s$, $d$, $t_s$)

**Lemma 2.**Algorithm 1 finds the correct time-dependent fastest path from source to destination for a given departure-time ts.

**Proof.** We prove Lemma 2 by contradiction. The forward search in Algorithm 1 is the same as the unidirectional A* algorithm and our heuristic function $h(v)$ is a lower bound of time-dependent distance from $u$ to $v$. Therefore, the forward search is correct. Now, let $P(s, (u), d, t_s)$ represent the path from $s$ to $d$ passing from $u$ where forward and backward searches meet and $\omega$ denotes the cost of this path. As we showed $\omega$ is the upper-bound of actual time-dependent fastest path from $s$ to $d$. Let $\varphi$ be the smallest label of the backward search in priority queue $Q_b$ when both forward and backward searches stopped. Recall that we stop searches when $\varphi > \omega$. Suppose that Algorithm 1 is not correct and yields a suboptimal path, i.e., the fastest path passes from a node outside of the corridor generated by the forward and backward searches. Let $P*$ be the fastest path from $s$ to $d$ for departure-time $ts$ and cost of this path is $\alpha$. Let $v$ be the first the backward search and $h_b(v)$ is the heuristic function for the backward search. Hence, we have $\varphi \leq h_b(v) + LTT(v, d)$, $\alpha \leq \omega < \varphi$ and $h_b(v)+ LTT(v, d) \leq LTT(s, v)+ LTT(v, d) \leq TDFP(s, v, t_s)+TDFP(v, t, t_v) = \alpha$, which is a contradiction. Hence, the fastest path will be found in the corridor of the nodes labelled by the backward search.

## 5. Conclusion and Future Work

We proposed a time dependent fastest path algorithm based on bidirectional A*. Unlike the most path planning studies, we assume the edge weights of the road network are time varying rather than constant. Therefore, our approach yield a much more realistic scenario, and hence, applicable to the real world road networks. We also compared our approaches with those handfuls of time dependent fastest path studies. Our experiments with real-world road network and traffic data showed that our proposed approaches outperform the competitors in storage and response time significantly. We intend to pursue this study in two different directions. First, we plan to investigate new data models for effective representation of spatiotemporal road networks. This is critical in supporting development of efficient and accurate time-dependent algorithms, while minimizing the storage and computation costs. Second, to support rapid changes of the traffic patterns (that may happen in case of accidents/events; for example), we intend to study incremental update algorithms for both of our approaches.

## References

[1] Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD (2008)
[2] Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005).
[3] Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 23–36. Springer, Heidelberg (2007)
[4] Wagner, D., Willhalm, T.: Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 776–787. Springer, Heidelberg (2003)
[5] Dreyfus, S.E.: An appraisal of some shortest-path algorithms. Operations Research 17(3) (1969).
[6] Dean, B.C.: Algorithms for min-cost paths in time-dependent networks with wait policies. Networks (2004)
[7] Foschini, L., Hershberger, J., Suri, S.: On the complexity of time-dependent shortest paths. In: SODA (2011)
[8] Batz, G.V., Delling, D., Sanders, P., Vetter, C.: Time-dependent contraction hierarchies. In: ALENEX (2009)
[9] Delling, D.: Time-dependent SHARC-routing. In: Halperin, D., Mehlhorn, K. (eds.) Esa 2008. LNCS, vol. 5193, pp. 332–343. Springer, Heidelberg (2008)
[10] Cooke, L., Halsey, E.: The shortest route through a network with timedependent intermodal transit times. Journal of Mathematical Analysis and Applications (1966)
[11] Kohler, E., Langkau, K., Skutella, M.: Time-expanded graphs for flow-dependent transit times. In: Proc. 10th Annual European Symposium on Algorithms (2002)
[12] George, B., Kim, S., Shekhar, S.: Spatio-temporal network databases and routing algorithms: A summary of results. In: Papadias, D., Zhang, D., Kollios, G. (eds.) SSTD 2007. LNCS, vol. 4605, pp. 460–477. Springer, Heidelberg (2007)
[13] Halpern, J.: Shortest route with time dependent length of edges and limited delay possibilities in nodes. Mathematical Methods of Operations Research (1969)
[14] Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time dependent edge-length. J. ACM (1990)
[15] Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. In: ICDE (2006)
[16] Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: EDBT (2008)
[17] Dehne, F., Omran, M.T., Sack, J.-R.: Shortest paths in time-dependent fifo networks using edge load forecasts. In: IWCTS (2009)
[18] Goldberg, A.V., Harellson, C.: Computing the shortest path: A* search meets graph theory. In: SODA (2005)
[19] Delling, D., Wagner, D.: Landmark-based routing in dynamic graphs. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 52–65. Springer, Heidelberg (2007)
[20] Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional a* search for time dependent fast paths. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 334–346. Springer, Heidelberg (2008)
[21] Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: CIKM (2009)
[22] PeMS, https://pems.eecs.berkeley.edu (accessed in May 2010)
[23] Demiryurek, U., Kashani, F.B., Shahabi, C.: A case for time-dependent shortest path computation in spatial networks. In: ACM SIGSPATIAL (2010)
[24] Guc, B., Ranganathan, A.: Real-time, scalable route planning using stream-processing infrastructure. In: ITS (2010)
[25] Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics (1968)
[26] Pohl, I.: Bi-directional search. In: Machine Intelligence. Edinburgh University Press, Edinburgh (1971)