Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Casey, Bradley, Bhaskar, Ashish, Guo, Hao, & Chung, Edward
(2014)
Critical review of time-dependent shortest path algorithms: A multimodal trip planner perspective.
Transport Reviews, *34*(4), pp. 522-539.

This file was downloaded from: http://eprints.qut.edu.au/72279/

© Copyright 2014 Routledge

This is an Accepted Manuscript of an article published by Taylor & Francis Group in Transport Reviews on 2014, available online at: http://www.tandfonline.com/[Article DOI].

Notice: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

http://doi.org/10.1080/01441647.2014.921797

Critical Review of Time-Dependent Shortest Path Algorithms: A Multimodal Trip Planner Perspective

Bradley Casey*[§], Ashish Bhaskar*, Hao Guo* and Edward Chung*

**Faculty of Science and Engineering, Queensland University of Technology, Brisbane, Australia*

ABSTRACT *A multimodal trip planner that produces optimal journeys involving both public transport and private vehicle legs has to solve a number of shortest path problems, both on the road network and the public transport network. The algorithms that are used to solve these shortest path problems have been researched since the late 1950s. However, in order to provide accurate journey plans that can be trusted by the user, the variability of travel times caused by traffic congestion must be taken into consideration. This requires the use of more sophisticated time-dependent shortest path algorithms, which have only been researched in depth over the last two decades, from the mid-1990s. This paper will review and compare nine algorithms that have been proposed in the literature, discussing the advantages and disadvantages of each algorithm on the basis of five important criteria that must be considered when choosing one or more of them to implement in a multimodal trip planner.*

Keywords: multimodal; trip; planning; time-dependent; dynamic; shortest path algorithms;

1. Introduction

Research into the general shortest path problem began back in the late 1950s (Bellman 1958; Dijkstra 1959), and there have been many journal papers published since then on improvements and extensions to these basic algorithms. The algorithms used to solve the static shortest path problem has many applications; However for multimodal trip planning, the results that are produced using static data can be inaccurate when congestion on the road network is present.

Firstly, the travel times of the road leg of a multimodal trip will usually be underestimated in peak times, which can cause a user of the trip planner to mistrust any results they receive and make them frustrated. More importantly, when the user then parks and attempts to catch public transport for the rest of their journey, they could miss the specified service, and in turn, any connecting services for the rest of their journey, which will drastically affect when they reach their destination. Additionally, bus on-time reliability is often subject to road congestion, and trains and other forms of public transport can be affected by congestion within their network, which will cause delays from their scheduled timetables, which can cause transfer connection issues for commuters, as well as increase their frustration with using public transport.

Therefore, due to the time-dependent nature of travel times leading up to, during and clearing of congestion, a more sophisticated approach to finding the quickest journey must be used. For more realistic and accurate results, algorithms that solve the time-dependent (dynamic) shortest path problem are required. This paper will investigate and critically review the algorithms that have been proposed to solve the dynamic shortest path problem, with a perspective of applying these algorithms within a multimodal trip planner.

[§]Corresponding Author. Email: bv.casey@qut.edu.au

This paper will have the following structure: First, a detailed literature review will be presented in two parts, general dynamic shortest path research, and more specific papers that introduced the algorithms that are to be compared. The second section will introduce the requirements of a multimodal trip planner with respect to utilisation of shortest path algorithms, and the implications for incorporating dynamic travel time data and associated shortest path algorithms into this type of application. The criteria for the comparison of the dynamic shortest path algorithms will be outlined, and then a detailed critical review of each algorithm will be presented using these criteria. Finally, conclusions of the review will be made.

2. Literature Review

2.1. *General Dynamic Shortest Path Information*

The problem of finding the shortest path using time-dependent travel times was first defined by Cooke and Halsey (1966). They extended on the work published by Bellman (1958) to establish Equation (1) to represent the time-dependent problem:

$$f_i(t) = \min_{j \neq i}\left(g_{ij}(t) + f_j\left(t + g_{ij}(t)\right)\right), \quad i = 1,2,\cdots,N-1$$
$$f_N(t) = 0 \tag{1}$$

They considered a set of N cities, with every pair of cities linked by road, and the Nth city is the destination. $g_{ij}(t)$ is defined as the time required to travel from city i to city j, and varies as a function of the starting time t at i. Given a matrix function $G(t) = g_{ij}(t)$, which may or may not be symmetrical, then the problem is to find the path that minimises the travel time from i to N when the entire journey begins at $t = t_0$. $g_{ij}(t)$ can vary with time in any way required, and if a pair of cities are not connected, then $g_{ij}(t)$ is equal to a very large number or to infinity. They assumed a discrete time scale $t_0, t_0 + 1, t_0 + 2, \cdots$, and that all $g_{ij}(t)$ are defined with positive integer values for $t \in S = \{t_0, t_0 + 1, t_0 + 2, \cdots\}$. $E_i(t)$ denotes the set of all paths that leave i at $t \in S$ and reach N in a finite time after a finite number of steps. Since the time for each path in $E_i(t)$ is positive, there is a minimum among these times. In order to solve the equations in (1), they proposed a modified form of Bellman's iteration scheme (Bellman 1958), to converge to the shortest path between any two vertices in a finite number of iterations.

Chabini (1997) categorised various types of dynamic shortest path problems based on 6 criteria:

(1) Fastest vs. minimum cost (or shortest) path problems

For fastest path problems, the link travel time that varies over time is the link cost, whereas minimum cost path problems have a more general link cost that varies with time.

(2) Discrete vs. continuous representation of time

Discrete representations model time as a set of integers, whereas for continuous representations, time is treated as real numbers. Alternatively, a discrete dynamic network

can be represented as a static network by using a time-space expansion; however the explicit use of this representation may not be the best approach to solve this type of problem.

(3) First-In-First-Out (FIFO) networks vs. non-FIFO networks

It was initially proposed by Dreyfus (1969) that the problem defined by Cooke and Halsey (1966), described in equation (1), could be solved by a generalisation of Dijkstra's Algorithm (Dijkstra 1959) as efficiently as for static shortest path problems. However, Kaufman and Smith (1993) proved that this was only the case if the network adheres to the FIFO condition, that is, $t + g_{ij}(t) \leq (t+1) + g_{ij}(t+1)$ must be true for all times t and links (i,j). In other words, departing from i at an earlier time t must not arrive later at j than departing from i at a later time, otherwise it would be better to wait till the later time and then depart. However, this FIFO condition only applies to the fastest paths problem, and is limited to forward-searching labelling algorithms only.

(4) Waiting is allowed vs. waiting is not allowed at nodes

Road networks are usually modelled with no waiting allowed at the nodes (which in this case represent intersections), whereas for public transport networks in which the nodes represent physical stops, waiting at nodes is allowed. However, public transport networks can also be represented using a time-space expansion (Schulz 2005), in which the waiting at a stop is represented as a link between two consecutive event nodes.

(5) The quantity of origin/destination nodes and departure times in question

Depending on what question a user may ask, different shortest-path searches may have to be performed. For instance, if the user specifies a certain origin, and wants to know when they will reach one or more destinations, a one-to-all (one origin, all other nodes are destinations) search may be required, either for one specific departure time from the origin, or for all possible departure times. Otherwise, if the user needs to arrive at a particular destination before a specified time, an all-to-one search for all departure times would be required.

(6) Integer vs. real-valued link travel times and link travel costs.

Similar to (2), the link travel times and costs may also be represented as integers or as real numbers.

Dean (2004) shows how to reduce the number of FIFO time-dependent shortest path query variants down to two fundamental queries, and then provides properties and solution algorithms for these two queries. Dean (2004) uses arc arrival time functions as they usually lead to simpler and more natural formulations compared to previous papers published on time-dependent shortest path algorithms. $a_{ij}(t)$ is an arrival time function for every arc $(i,j) \in A$, and gives the time of arrival at j if one departs i at time t. The function $a_{ij}(t) - t$ gives the travel time along arc (i,j) if one departs at time t.

Inverse arc arrival time functions, $a_{ij}^{-1}(t)$ are then defined, which gives the latest time one may depart i in order to arrive at j by time t by travelling along the arc (i,j). The path arrival time function of a path $p = i_1 \to i_2 \to \cdots \to i_k$ is given by the composition of the arc arrival time functions along p: $a_p(t) = a_{i_{k-1}i_k}\left(a_{i_{k-2}i_{k-1}}\left(\cdots a_{i_{k-2}i_{k-1}}(t)\right)\right)$. The inverse path arrival

time function of a path p, $a_p^{-1}(t)$ is then defined as the inverse of the corresponding path arrival time function, or equivalently, the composition of the inverse arc arrival time functions along the path. $a_p^{-1}(t)$ gives the latest time one may depart along p in order to reach the end by time t.

From these definitions, Dean (2004) then let $P(s,d)$ denote the set of paths between a source node s and destination node d. The function $EA_{sd}(t) = \min\{a_p(t) : p \in P(s,d)\}$ gives the earliest arrival time at node d if one leaves node s at time t. It is also noted that $EA_{sd}(t) - t$ provides the travel time of the time-dependent shortest path from s to d if one departs at time t. A similar function $LD_{sd}(t) = \min\{a_p^{-1}(t): p \in P(s,d)\}$ gives the latest departure time one may leave node s in order to arrive at node d by time t. It is stated that $LD_{sd}(t)$ and $EA_{sd}(t)$ are inverses of each other.

From these two functions, Dean (2004) provides a table of the different query variants, and how they can be reduced down to the two fundamental queries, $EA_{s*}(t)$ (one to all for a specific departure time) and $EA_{s*}(*)$ (one to all for all departure times), and these can be seen in Table 1. The time-reversal transformation is similar to how the static one-to-all and all-to-one shortest paths are made equivalent, by reversing all the arcs in the network, but also reversing the direction of time. That is, the following changes need to be made:

- Reverse the roles of all sources and destinations
- Reverse the roles of departure and arrival times, negating all of these times in the process
- Reverse the direction of all arcs
- Replace each arc arrival time function $a_{ij}(t)$ with $-a_{ij}^{-1}(-t)$

[Table 1]

Dean (2004) then states and proves four lemmas, and using these definitions and proofs, describes the algorithms for solving the two fundamental time-dependent shortest path problems. For the $EA_{s*}(t)$ problem, both a label-correcting algorithm and a label-setting algorithm are provided, both of which are analogous to their static algorithm counterparts. For the $EA_{s*}(*)$ problem, Dean (Dean 2004) states that while the label-correcting algorithm has an advantage in terms of simplicity, it is not the most efficient algorithm for this problem. Two algorithms are then described using a label-setting method, one for discrete time, and one for piece-wise linear continuous time

2.2. *Specific Dynamic Shortest Path Algorithm Approaches*

The more recent papers on time-dependent shortest path algorithms have focused on how the more sophisticated static algorithms, such as hierarchical methods that require pre-processing, can be extended to work on dynamic shortest path problems.

Chabini (1998) studies 3 types of shortest path problems in discrete dynamic networks: the one-to-all fastest path problem departing from an origin at a given time interval, the all-to-one fastest paths problem for all departure time intervals, and the all-to-one minimum cost paths problem for all departure time intervals. For the all-to-one problem, they consider both waiting and no-waiting policies, and showed that the waiting allowed variant is a special case of the no-waiting variant. For the all-to-one fastest paths problem, they present a Decreasing Order of Time (DOT) algorithm. They denoted Equation (2) as the fastest travel time to destination q departing from node i at time t.

$$\pi_i(t) = \begin{cases} \min_{j \in A(i)} \left(d_{ij}(t) + \pi_j \left(t + d_{ij}(t) \right) \right) & i \neq q \\ 0 & i = q \end{cases} \quad (2)$$

$A(i)$ is the set of nodes that have an incoming arc starting from node i, and $d_{ij}(t)$ is the time-dependent travel time. Chabini (1998) proposes that labels $\pi_i(t)$ can be set in a decreasing order of time intervals, and proves this by stating that since all arc travel times are positive integers, labels corresponding to time step t never update labels corresponding to time steps greater than t, which implicitly reflects the acyclic property along the time dimension of the time-space expansion of a discrete dynamic network.

The pseudo-code for the DOT algorithm is as follows:

Initialisation

$\pi_i(t) = \infty, \forall (i \neq q)$

$\pi_q(t) = 0, \forall (t < M - 1)$

$\pi_i(M-1) = StaticShortestPaths\big(d_{ij}(M-1), q\big), \forall (i)$

Note: $\pi_i(t) = \pi_i(M-1), \forall (t \geq M-1, i)$

Main Loop

For $t = M - 2$ down to 0, do:

For $(i, j) \in A$

$$\pi_i(t) = \min \left(\pi_i(t), d_{ij}(t) + \pi_j \left(t + d_{ij}(t) \right) \right)$$

where M is the number of time intervals.

For departure time intervals greater than or equal to $M - 1$, the computation of the fastest paths is equivalent to a static shortest paths problem, and the algorithm above assumes that a StaticShortestPaths procedure (for example, static Dijkstra's algorithm) is given, with optimal running time SSP. Chabini (1998) then shows that Algorithm DOT solves the all-to-one fastest paths problem with a serial computation time $\theta(SSP + nM + mM)$, where n is the number of nodes, and m is the number of arcs, and proves that this is optimal in worst-case-run-time complexity for the all-to-one fastest paths problem.

Chabini and Lan (2002) extends the A* heuristic (Hart, Nilsson et al. 1968) to shortest path problems in dynamic networks that satisfy the FIFO property. They present efficient adaptions of the A* heuristic to compute fastest paths from one origin to one destination, as well as for multiple departure times from the origin node. The algorithms are implemented and their performance is compared to a dynamic adaption of Dijkstra's algorithm which is stopped once the destination node is reached. Using a network of 3000 nodes, 10000 links and 100 time intervals, the dynamic adaptions of A* led to a savings ratio of 11 in terms of the number of nodes selected, and a savings ratio of five with respect to computation time. They also show that the computational savings increase with the size of the network. For the one-to-one problem for one departure time, they propose the use of static lower bounds for the computation of the estimate of travel time from the current node to the destination. For all $(i, j) \in A$, let $d_{ij}^{min} = \min_{t=0,\cdots,M-1} \{ d_{ij}(t) \}$, which finds the minimum travel time to traverse each link in A. From this, a virtual static network is constructed with d_{ij}^{min} as the link travel time. An all-to-one static shortest path algorithm is applied to this network with destination node q, which provides the lower bound on the estimate for travelling from any node to the destination node.

Delling and Wagner (2007) adapted the goal directed search based on landmarks (ALT) to work in a time-dependent scenario. The static algorithm uses pre-processing to calculate the shortest path from each point to a number of landmarks. It then uses this information in a

similar way to A* to provide a heuristic value of the remaining cost to the destination. They show that the ALT algorithm still produces correct results as long as the edge weights do not drop below their initial value, so therefore as long as the changes in the network are moderate, no updates on the pre-processing are required.

Nannicini et al. (2012) proposed a novel approach based on the ALT algorithm to apply a bidirectional search, where the backward search is used to bound the set of nodes that have to be explored by the forward search. Their testing results showed that the approach is effective in practice, if a small approximation factor is accepted, and results in a speed-up of several times over Dijkstra while finding slightly suboptimal solutions.

Delling and Nannicini (2012) extend the bidirectional time-dependent search algorithm to allow for core routing, which consists of carrying out most of the search on a subset of the original node set. They also tackle the dynamic scenario where the piecewise linear arc cost functions are not fixed, but can have their coefficients updated. They provide computational results to show their approach is a significant speed-up with respect to the original algorithm, and is able to deal with the dynamic scenario with only a small computational effort to update the cost functions and related data structures.

Delling (2011) extends the SHARC (Shortcuts and arc-flags) algorithm presented in Bauer and Delling (2009) by adapting it such that the correctness can still be guaranteed in a time-dependent scenario. Brunel et al. (2010) improved upon the time-dependent SHARC algorithm by reducing the space overhead by a factor of up to 11.5 with a loss in query performance by a factor of 4. This is done by identifying unimportant parts of the pre-processing and removing them while still ensuring that correctness is guaranteed.

Batz et al. (2009) explain how to extend contraction hierarchies to networks with time-dependent edge weights, or time-dependent contraction hierarchies (TCH). The main weakness of this approach is the high memory requirements for the pre-processed data. Batz et al. (2013) presents an improved TCH algorithm, by storing only approximated time-dependent edge weights in the shortcuts, which contains a lot of redundancy, while original edges store exact travel times. Searches on the approximated TCH (ATCH) yield small subgraphs that contain shortcuts, and then after unpacking the shortcuts, an exact time-dependent search can then be applied. This reduces the memory usage significantly while only moderately affecting query time.

Chen et al. (2013) studies time-dependent reliable shortest path problems (TD-RSPP). They consider two variants of TD-RSPP. The first one is to determine the earliest arrival time and associated reliable shortest path for a given departure time, which they refer to as the “forward” TD-RSPP. The second problem is to determine the latest departure time and associated reliable shortest path for a given preferred arrival time, which is referred to as the “backward” TD-RSPP. They show in the paper that the TD-RSPP is not reversible, that is, the backward version cannot be solved by the algorithms designed for the forward problem using the reverse search from destination to origin. They then propose two efficient solution algorithms to solve the forward and backward TDRSPP, and prove their optimality.

3. Multimodal Trip Planner Requirements and Implications for Dynamic Data Utilisation

The main issue when developing a multimodal trip planner that combines automobile and transit legs into an integrated journey plan is how to deal with this integration of the two networks. This because they are different, both in a conceptual sense of what the nodes and arcs represent, as well as the practical implementation of the data structures that store these

networks. There are two methods that are available to manage this integration; they are the Multi-level Network approach and the Multi-Step Algorithm approach.

The multi-level network approach constructs a single graph data structure, where each different mode of transport is on a separate level. Connections are made between nodes in different levels where transitions between the modes occur, such as at Park'n'Ride locations between automobiles and public transport. A dynamic version of this approach is used by Bosquet et al. (2009) to compute a one-way multimodal shortest path, and they also propose a strategy for solving the two-way approach.

On the other hand, Khani et al. (2012) propose an intermodal path algorithm, which performs a sequential application of transit and automobile shortest paths, which results in the optimal intermodal path and indicates which park-and-ride location is optimal for transferring between private and public modes. This is similar to the Multi-Step Algorithm approach that is proposed and implemented in Casey et al. (2012) and Casey et al. (Casey, Guo et al. 2013), however the algorithm steps differ in the order they are performed. The Multi-Step Algorithm approach keeps the two graph data structures separate, and generates an overall journey by performing the following algorithm:

(1) Find the park'n'ride locations within a specified distance of the origin (based on the cost of driving from the origin to destination by automobile only)
    (a) Calculate the shortest path to drive by automobile from the origin to each of the park'n'ride locations.
(2) For each of the park'n'ride locations that satisfy (1), find the stops within a specified walking or cycling distance of these locations.
    (a) Calculate the shortest path to walk from the park'n'ride location to each of its neighbouring stops.
(3) Find the stops within the specified walking distance of the destination
    (a) Calculate the shortest path to walk from each of the neighbouring stops to the destination.
(4) Using the stops surrounding the park'n'ride locations as origin stops, and the stops surrounding the destinations as destination stops, calculate the shortest paths via public transport from the set of origin stops to the set of destination stops.

This paper focuses on comparing the dynamic algorithms for use in the Multi-Step Algorithm approach, mainly because it allows for the use of any implementation of a shortest path algorithm in each of the steps without much modification of the algorithm itself, as long as the inputs and outputs remain the same. Integration of dynamic shortest path algorithms into the multi-layered network approach requires modification of these algorithms to account for the differences in the road network and transit network; primarily that the dynamic road network assumes no waiting at nodes, whereas the transit network is principally based on waiting to make connections between services.

There are three main types of shortest path problems that are being solved in the Multi-Step algorithm described above: car journeys along the road network, walking paths between stops and park'n'ride and the destination, and the public transport journeys. Walking journeys are not time-dependent to a significant degree (apart from time spent waiting at traffic signals), and therefore can use a static shortest path algorithm. In fact, the walking paths between each park'n'ride and the surrounding stops can be pre-computed and stored in memory, to further reduce on query computation time. The public transport shortest path search in the static multimodal trip planner is already based on time-dependent data, although that data does not change once the network graph data structure has been constructed. In order to integrate dynamic data into the public transport network, additional "dynamic" nodes and arcs can be added and deleted when required, and the search algorithm does not require

significant modification, apart from giving preference to the dynamic nodes and arcs over their static counterparts.

Therefore, the calculation of shortest paths for car legs is the primary consideration for the comparison of the dynamic shortest path algorithms introduced in the literature review section. For this comparison, it is assumed that the time-dependent road network satisfies the FIFO and no waiting at nodes conditions.

There are two types of car legs that may need to be solved during the multi-step algorithm approach. Firstly, an initial shortest path search is calculated between the origin and destination, which is used to reduce the number of park'n'rides that are included in the overall search. This is a single one-to-one search, and may be either a forward search through time or a reverse search back in time, each of which may require a different dynamic algorithm.

Secondly, the car journeys to each of the park'n'rides that have been chosen must be calculated. When a search is required from a specific departure time, i.e. a forward search, this is a one-to-many search from the origin. The arrival time at each of the park'n'rides can then be added to the walking time to each of the surrounding stops, which is then used as input into the transit shortest path search. Once the transit paths have been found, it may be necessary to recalculate the car journey shortest path, with a reverse one-to-one search back in time, to minimise the waiting time spent at the first transit stop. This reverse search does not affect the earliest arrival time at the destination however, and would only be useful in minimising the total travel time of the journey.

When a search is required from a specified arrival time, i.e. a reverse search, the transit shortest path search requires a travel time or arrival time at the origin stops (which would require the car travel time to be known); However, the departure times at the initial origin stops has an effect on when the traveller must start driving due to the time-dependent travel times on the road network. Therefore, in order to provide a lower bound on the travel time to the origin stops, a static shortest path search is required using the minimum (free-flow) speeds along each road in the network as link costs. These travel times can then be used in the transit shortest path algorithm, which in return will provide the latest departure times at each of the origin stops. These times, once walking has also been taken into account, can then be used to perform a one-to-one search back in time from each of the park'n'ride locations to the origin.

4. Critical Review

4.1. *Criteria for Comparison*

The criteria categories by which the time-dependent shortest path algorithms will be compared against each other are as follows:

- Operational (Query) computational performance (amount of time required to compute a shortest path for a single user request)
- Pre-processing computational requirements (amount of time required to pre-process the network in order to improve the operational performance)
- Memory requirements (amount of memory required to store pre-processed information, etc.)
- Ease of implementation
- Applicability to multimodal trip planner

4.2. *Comparison of Time-dependent Shortest Path Algorithms*

This section will provide a comparison of the time-dependent shortest path algorithms presented in the literature review. They include:

- Dynamic Adaptation of Dijkstra's algorithm
- Decreasing Order of Time (DOT) algorithm
- Dynamic A* algorithm
- Dynamic Uni-directional ALT algorithm (uni-ALT)
- Dynamic Bi-directional ALT algorithm (TDALT)
- Dynamic Bi-directional core-based routing algorithm (TDCALT)
- Dynamic extension of SHARC algorithm
- Time-dependent contraction hierarchies (TCH)
- Time-dependent reliable shortest path problem algorithms.

Algorithm: Dynamic Adaption of Dijkstra's Algorithm

Operational Query Performance: Slowest (however it may be competitive where one-to-many searches can be utilized)

Pre-processing Computational Requirements: None

Memory Requirements: None for preprocessing, just the mandatory memory requirement for storing the basic network graph data structure.

Ease of Implementation: Easy. The static Dijkstra algorithm is well-known and code is widely available for it. The dynamic adaption is relatively simple, it requires modification of the code to calculate the cost from the current node to its neighbour nodes, to take into consideration the time that travel will start and use the time-dependent travel time data appropriately. Some modification will also need to be made to the data structure to allow for the additional time-dependent travel time data. However this modification to the data structure will also need to be made for all other algorithms as well.

Applicability to Trip Planner: Applicable in theory to all of the time-dependent shortest path searches that are required in the multimodal trip planner. In particular, the one-to-many search capability of Dijkstra is useful for the forward search from the origin to all parking areas, since the search would only need to be executed once, as opposed to a separate search per parking area. However, in practice the operational query performance may not be satisfactory for this type of application, where a result is expected within a second or two. It would still be useful for testing purposes however, as it is guaranteed to produce the optimal solution, and can be used as a baseline to compare the other algorithms in terms of computational performance.

Algorithm: Decreasing Order of Time (DOT) algorithm

Operational Query Performance: Slow for one-to-one searches, however it is proven that for all-to-one for all departure time problems, this algorithm has the optimal run-time complexity.

Pre-processing Computational Requirements: Very low, it requires a static shortest paths computation for the latest time-step, which could be pre-computed ahead of time.

Memory Requirements: Low, storing the static shortest path values for every node to the destination node.

Ease of Implementation: Easy.

Applicability to Trip Planner: As this algorithm is specifically designed for all-to-one for all departure times problems, it is not directly applicable to most of the dynamic shortest path searches that are required in the multimodal trip planner, as described in the previous

section. However, it could be applied in an alternative approach, where multiple worker threads execute the DOT algorithm from each of the parking areas to all other nodes, and store the data computed for each time-step and node. This can then be quickly accessed by query threads, which would greatly reduce the computational time required for these searches. However, these worker threads would need to recompute the values before each new time-step, to provide a moving time-window as new time-dependent data is provided. Also, this algorithm is designed to operate on a time-dependent network that uses discrete time-steps as opposed to a continuous one, and therefore some adaption to the data may be required.

Algorithm: Dynamic A* Algorithm

Operational Query Performance: Fast for one-to-one searches, however this is dependent on whether the all-to-one static shortest path search is performed as a pre-processing operation.

Pre-processing Computational Requirements: Low. An all-to-one static shortest path search is required to compute the lower bounds that are used for the heuristic estimate. In the case of the trip planner, this static search would need to be computed and travel time values stored for each park'n'ride.

Memory Requirements: low, dependent on the number of park'n'rides. Only the travel time lower bound values are required to be stored, no path predecessor information is required. Therefore memory requirements are proportional to Pn where P is the number of park'n'rides and n is the number of nodes in the network.

Ease of Implementation: Easy to Medium. Similar to the Dynamic Adaption to Dijkstra's Algorithm, this adaption can make use of code from a static A* algorithm, with modifications to how the total cost is calculated from the current node to its neighbours. In this case, the same modification would need to be made to make use of the time-dependent travel time data in calculating the known portion of the total cost, while the heuristic portion of the cost would refer to the lower bounds that were calculated in pre-processing. A additional loop of code would need to be written for the pre-processing step, to calculate the all-to-one static shortest path searches for each of the park'n'ride locations using a standard Dijkstra's algorithm, and to store these travel times lower bounds in memory.

Applicability to Trip Planner: Due to the pre-processing step required to compute the lower bounds for the heuristic estimate, this algorithm may not provide an improvement in computational speed for random origin-destination requests. However, for the multimodal portion of the trip planner, the park'n'ride locations provide a finite set of destination nodes upon which the pre-processing searches can be performed.

Algorithm: Dynamic Uni-directional ALT algorithm (uni-ALT)

Operational Query Performance: Fast. The results from the experiments that Delling and Wagner (2007) presented in their paper that when searching on a European network of approximately 18 million nodes and 42.6 million edges, the Dynamic Uni-directional ALT algorithm provide a speed-up of approximately factor 3 – 5 over a Dynamic Dijkstra algorithm, although this is dependent on how much the level of traffic has perturbed the travel times from free-flow travel times. The query performance is also dependent on how many landmarks are available from pre-processing. Therefore tests would need to be performed once this algorithm is implemented to find the optimal number of landmarks, and the particular positions of these landmarks, that results in the fastest and most reliable query performance.

Pre-processing Computational Requirements: Medium, increases if the number of landmarks chosen is increased, and is also dependent on the heuristic used to select which

nodes as the appropriate landmarks. For the European network, the amount of time required for preprocessing varied between 26 minutes to 85 minutes.

Memory Requirements: medium to high, proportional to the size of the network and the number of landmarks used. For the European network, 8 landmarks required 1100 MB of memory, while increasing that number to 64 landmarks required 8800 MB of memory.

Ease of Implementation: medium, dependent on whether a static ALT algorithm has already been implemented. While the ALT algorithm uses a heuristic estimate in a similar manner to A*, there are a number of options for heuristics to choose landmarks, and these would need to be implemented and tested to find the optimal one to use for a given network. Modifying the static algorithm to operate on time-dependent data would require about the same amount of work as a Dijkstra or A* adaption, along with some additional work to implement code that recalculates the preprocessed values should they change significantly from the current values. However, in the case of a traffic network, if free-flow speed is used as the baseline for the pre-processing stage, these values should not have to be adjusted very often, if at all.

Applicability to Trip Planner: The Dynamic Uni-directional ALT algorithm is designed specifically for fast 1-to-1 queries, and should be applicable to all dynamic shortest path searches that are required in the multimodal trip planner. However, computational testing would need to be performed for the steps where shortest paths to park'n'rides are calculated, to compare its performance in computing a search per park'n'ride versus a single Dijkstra search to all park'n'rides.

Algorithm: Dynamic Bidirectional ALT Algorithm (TDALT)

Operational Query Performance: Fast to very fast, but improvement in computation performance is balanced by an increase in the divergence from an optimal solution. For experiments performed on a European network with approximately 18 million nodes and 42.6 million edges, for an approximation factor of 1 (no approximation i.e. optimal solution), the Dynamic Bi-directional ALT algorithm was twice as quick as the Dynamic Dijkstra's algorithm, and with an approximation factor of 2, it was 55 times as quick, however almost 50% of the paths computed were suboptimal and while the average relative error was small, the worst case produced a solution that had a cost 50% larger than the optimal value. They suggest the best trade-off between performance and quality of the solution is an approximation factor of about 1.15, and if the approximation factor is reduced to below 1.05, it is slower than the uni-directional ALT algorithm, which always produces an optimal result.

Pre-processing Computational Requirements: Medium, it uses the same pre-processing approach as the Dynamic ALT algorithm

Memory Requirements: Medium to high, similar to Dynamic ALT Algorithm

Ease of Implementation: medium, depending on whether a static ALT algorithm has already been implemented. The main difficulty would be in testing to find the optimal approximation factor that provides a good trade-off between performance and solution quality for a given network.

Applicability to Trip Planner: The Bidirectional ALT Algorithm is also designed specifically for fast 1-to-1 queries, and is applicable to all of the dynamic shortest path searches required in the multimodal trip planner. However, due to the fact that it sacrifices optimality of the solution for query performance, testing is required to find the optimal approximation factor. As with the Dynamic uni-directional ALT algorithm, testing would also need to performed to compare it to Dynamic Dijkstra's algorithm for the searches involving multiple park'n'rides.

Algorithm: Dynamic Bi-directional core-based routing algorithm (TDCALT)

Operational Query Performance: Very fast. In experiments performed on the European network with 18 million nodes and 46.2 million edges, depending on the value of the approximation constant and contraction parameters, query times varied between 188ms for conservative values, and 8ms for more aggressive approximation and contraction values. This is faster by at least one order of magnitude compared to uni-ALT and TDALT, and about 50 times as fast as Dynamic Dijkstra's algorithm.

Pre-processing Computational Requirements: Medium to high, it is dependent on the amount of contraction that is performed on the network. c denotes the maximum expansion of a bypassed node and h is the hop-limit of added shortcuts. On the European network, $c = 0.5$ and $h = 10$ requires 15 minutes of pre-processing time, while $c = 5.0$ and $h = 90$ requires 134 minutes of pre-processing time. This is compared with a pre-processing time of 28 minutes for TDALT on the same network.

Memory Requirements: Medium. For the European network, the additional memory space required for pre-processed data is about half that required for TDALT, and interestingly, decreases initially as contraction is increased, but it increases after a certain point. It is suggested that this is due to the core shrinking, so landmark distances are only needed for a small number of nodes, but interpolation points for shortcuts increases as contraction increases.

Ease of Implementation: Medium to hard. This algorithm uses techniques from both ALT algorithms and shortcut-based algorithms, and therefore both the pre-processing and query algorithms are relatively complicated compared to Dijkstra or A* algorithms. The algorithm that is suggested also allows for updates to the cost functions, and therefore would require additional code to perform this update function.

Applicability to Trip Planner: As with uni-ALT and TDALT, TDCALT is primarily designed for performing very fast 1-to-1 queries, and is applicable to all of the dynamic shortest path searches required in the multimodal trip planner. Again, it does sacrifice optimality of solutions for improved performance, so tests would be required to find the optimal parameters for this trade-off. In terms of its relative performance for searches involving multiple park'n'rides, it performs significantly better than Dijkstra's algorithm, so unless a query needs to include a large number of park'n'rides, TDCALT should perform the overall search in a faster time.

Algorithm: Dynamic extension of SHARC algorithm

Operational Query Performance: Extremely Fast. The European network used in experiments for the algorithms already discussed was used for comparison of this algorithm. It was shown that for the most conservative version of the Dynamic SHARC algorithm (the one with the least pre-processing), it provided a speed up of between 70 and 150 over the Dynamic Dijkstra's algorithm, on the order of about 1 – 2 ms. The space-efficient version of the Dynamic SHARC algorithm sacrificed some query performance for the reduced memory usage, by a factor of 4. With a high amount of compression, it was still able to complete a query within 3 ms on a German road network with approximately 4.7 million nodes and 10.8 million arcs.

Pre-processing Computational Requirements: Very high. For the European network, the most conservative version of Dynamic SHARC required just less than 7 hours of processing, while for a heuristic-based version that provides much faster query performance, about 22 hours of pre-processing was required. The space-efficient version required just under 4 hours of preprocessing on the German network, however this network is about $\frac{1}{4}$ the size of the European network.

Memory Requirements: Low to High. For the European network, the additional memory space required per node for pre-processed data was about the same as uni-ALT and TDALT

up to about 50% more, depending on the particular approach taken. The space-efficient version significantly reduced this memory requirement however, by a factor of almost 11.5 under the highest level of compression tested on the German network, from about 156 bytes per node with no compression down to about 13 bytes per node.

Ease of Implementation: Hard, depending on whether a static SHARC algorithm has been implemented. There are multiple steps involved in the pre-processing; including initialisation, iteration and finalisation, and each of these steps consist of multiple sub-steps. The query algorithm is also significantly modified from the Dijkstra's algorithm. On top of these implementation requirements, there are four variants of time-dependent SHARC, which differ in how the arc-flags are computed in preprocessing, and they include economical, generous, aggressive and heuristic versions. The compression used in the space-efficient algorithm requires additional implementation work as well.

Applicability to Trip Planner: As with uni-ALT, TDALT and TDCALT, Dynamic SHARC is designed specifically for very fast 1-to-1 queries, and is applicable to all of the dynamic shortest path searches required in the multimodal trip planner. As the query performance is very fast, it is very unlikely that Dijkstra's algorithm will outperform this algorithm for the searches involving multiple park'n'rides. The main concern to applying this algorithm to a multimodal trip planner is the high pre-processing requirements; both in time and memory usage, and this may need to be considered if these resources are limited.

Algorithm: Time-dependent extension of contraction hierarchies

Operational Query Performance: Extremely fast. For the European network, the query processing time is about 3 ms on average for their experiments, which provides a speed up of about 1592 over the Dynamic Dijkstra's algorithm. For the improved version of TCH, ATCH, query performance is about double that of TCH, however this is still within about 4-5 ms.

Pre-processing Computational Requirements: High. For the European network, the pre-processing time required about 3 ½ hours, which is not as high as Dynamic SHARC, but greater than what is required for the ALT-based algorithms on the same network. The ATCH requires some additional pre-processing time, slightly less than 5 hours on the European network with a high amount of traffic.

Memory Requirements: Medium to Very High. It is mentioned in the paper that this is the main weakness of this algorithm, as it requires up to a kilobyte per node. They do suggest that this can be reduced if the requirement of exact queries is dropped. The improved version of the TCH, ATCH did reduce the memory overhead caused by the preprocessing by at least half to a quarter of the memory required by TCH.

Ease of Implementation: Hard.

Applicability to Trip Planner: As with many of the algorithms mentioned in this analysis, this approach is designed for very fast 1-to-1 queries, and is applicable to all of the dynamic shortest path searches required in the multimodal trip planner. Similar to Dynamic SHARC, it is unlikely that Dijkstra's algorithm will outperform this algorithm for searches involving multiple park'n'rides. The main issue with applying this approach to a multimodal trip planner is the high memory requirements, which must be considered when deciding on appropriate server hardware.

Algorithm: Time-dependent reliable shortest path problem algorithms

Operational Query Performance: Medium. Chen et al. (2013) shows that the TD-RSPP is not reversible, therefore query performance must be considered for both the forward and backward searches separately. They performed experiments on two real-world road networks, Hong Kong (1367 nodes and 3655 links) and Chicago Regional (12982 nodes and 39018

links) networks, as well as two grid networks G1 (2000 nodes and 7820 links) and G2 (5000 nodes and 19700 links). They compared the forward TDRSP-A* method that they proposed with a label-correcting version, F-TDRSP-LC, as well as the results for the backward TDRSP. For the Hong Kong network, the F-TDRSP-A* calculated a result in 0.15 seconds on average, compared to 1.49 seconds by the F-TDRSP-LC, while B-TDRSP computed a result in an average time of 0.61 seconds. For the larger Chicago network, F-TDRSP-A* calculated a result in 2.67 seconds which is considerably faster than the F-TDRSP-LC computation time of 151.96 seconds, while the B-TDRSP computed a result in an average time of 12.73 seconds. Compared to many of the algorithms that have been analysed in this paper, these networks are considerably smaller in the number of nodes and links, while the computational speed is slower in most cases. However the problem that is being solved by the TD-RSPP algorithms is more difficult than a typical TD-SPP algorithm, so these times are not necessarily directly comparable.

Pre-processing Computational Requirements: None

Memory Requirements: None for pre-processing purposes, however extra memory would be required for storing the additional information that is used to generate the travel time distributions that are used in this approach.

Ease of Implementation: Hard. Due to the fact that the TD-RSPP is not reversible, separate algorithms are required for forward searches and backward searches, both of which are non-trivial to implement.

Applicability to Trip Planner: These algorithms could be applied to all of the time-dependent shortest path searches required in the multimodal trip planner; however additional data would need to be incorporated into the algorithms to compute the travel time distributions. Extending these algorithms for use in the transit network would also need to be considered, as public transport vehicles are also subject to variability in their on-time performance. The computational performance is also of some concern, depending on the size of the network used, as users of the trip planner would be expecting a result within a second or two of sending their request.

Table 2 summarises the comparisons that have been given in this section. Please note that the entries for each algorithm and criteria are relative to the other algorithms in the review. The absolute values of these entries depend on a number of other factors as well, such as the size of the network being searched, and the various parameters that are used for pre-processing. The papers for each of these algorithms provide more details on these absolute values for specific networks and chosen parameter values.

[Table 2]

4.3. *Open Source Software Availability*

At the time of publication of this paper, a compilation of open-source code for the algorithms reviewed in section 4.2 was not available. There are several routing engines, both online and offline, which are available to work with OpenStreetMaps, although not all are open-source. A list can be accessed from http://wiki.openstreetmap.org/wiki/Routing. Unfortunately, at this time, none of them provide real-time traffic routing functionality.

5. Conclusion

This paper has reviewed and compared nine different algorithms that can be used to solve time-dependent shortest path problems. They have been compared based on five criteria that are important to consider when choosing which algorithms to implement for a multimodal trip planner. The main issue that must be considered is balancing the amount of pre-processing time and memory required with the improvements with query performance that is provided with this extra work. A secondary issue is how easy the algorithms are to implement, as the algorithms that require more pre-processing typically require more complex programming. Finally, not all of the algorithms are applicable to all of the searches that are performed in a multimodal trip planner, for a number of different reasons. Therefore multiple algorithms may need to be implemented and used for the shortest path searches to which they are best suited to perform.

References

Batz, G. V., D. Delling, et al. (2009). Time-dependent contraction hierarchies. Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09).

Batz, G. V., R. Geisberger, et al. (2013). "Minimum time-dependent travel times with contraction hierarchies." J. Exp. Algorithmics **18**: 1.1-1.43.

Bauer, R. and D. Delling (2009). "SHARC: Fast and Robust Unidirectional Routing." ACM Journal of Experimental Algorithmics.

Bellman, R. (1958). "On a routing problem." Q. Appl. Math. **16**: 87-90.

Bousquet, A., S. Constans, et al. (2009). On the adaptation of a label-setting shortest path algorithm for one-way and two-way routing in multimodal urban transport networks. International Network Optimization Conference.

Brunel, E., D. Delling, et al. (2010). Space-Efficient SHARC-Routing. Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings, Springer.

Casey, B., A. Bhaskar, et al. (2012). Data requirements and graph data structures for a multi-modal, multi-objective trip planner. 25th Australian Road Research Board Conference, Perth, Australia.

Casey, B., H. Guo, et al. (2013). A computational analysis of shortest path algorithms for integrated transit and automobile trip planning. 36th Australasian Transport Research Forum, Brisbane, Australia.

Chabini, I. (1997). A new algorithm for shortest paths in discrete dynamic networks.

Chabini, I. (1998). "Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time." Transportation Research Record: Journal of the Transportation Research Board **1645**(1): 170-175.

Chabini, I. and S. Lan (2002). "Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks." Intelligent Transportation Systems, IEEE Transactions on **3**(1): 60-74.

Chen, B. Y., W. H. Lam, et al. (2013). "Reliable shortest path problems in stochastic time-dependent networks." Journal of Intelligent Transportation Systems.

Cooke, K. L. and E. Halsey (1966). "The shortest route through a network with time-dependent internodal transit times." Journal of mathematical analysis and applications **14**(3): 493-498.

Dean, B. C. (2004). "Shortest paths in FIFO time-dependent networks: Theory and algorithms." Rapport technique, Massachusetts Institute of Technology.

Delling, D. (2011). "Time-Dependent SHARC-Routing." Algorithmica **60**(1): 60-94.

Delling, D. and G. Nannicini (2012). "Core routing on dynamic time-dependent road networks." INFORMS Journal on Computing **24**(2): 187-201.

Delling, D. and D. Wagner (2007). Landmark-based routing in dynamic graphs. Experimental Algorithms, Springer**:** 52-65.

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

Dreyfus, S. E. (1969). "An appraisal of some shortest-path algorithms." Operations Research **17**(3): 395-412.

Hart, P. E., N. J. Nilsson, et al. (1968). "A formal basis for the heuristic determination of minimum cost paths." IEEE Transactions on Systems Science and Cybernetics **4**(2): 100-107.

Kaufman, D. E. and R. L. Smith (1993). "FASTEST PATHS IN TIME-DEPENDENT NETWORKS FOR INTELLIGENT VEHICLE-HIGHWAY SYSTEMS APPLICATION∗." Journal of Intelligent Transportation Systems **1**(1): 1-11.

Khani, A., S. Lee, et al. (2012). Intermodal Path Algorithm for Time-Dependent Auto Network and Scheduled Transit Service. Transportation Research Record: Journal of the Transportation Research Board, Washington, D.C., Transportation Research Board of the National Academies.

Nannicini, G., D. Delling, et al. (2012). "Bidirectional A* search on time-dependent road networks." Networks **59**(2): 240-251.

Schulz, F. (2005). Timetable information and shortest paths. PhD Dissertation, Karlsruhe Univ.

Table 1. Problem variants and the method of reduction down to the fundamental problems

Desired Output	Method of Computation
$EA_{s*}(t)$, $EA_{s*}(*)$	These are the two fundamental problems, from which all other variants can be expressed
$EA_{sd}(t)$, $EA_{sd}(*)$	As with the static shortest path problem, the single-source single-destination problem seems just as difficult as problems involving either multiple sources or multiple destinations. Therefore, these can be solved as the more generic problems $EA_{s*}(t)$ and $EA_{s*}(*)$, respectively
$EA_{**}(t)$, $EA_{**}(*)$	These are solved by performing n computations of $EA_{s*}(t)$ and $EA_{s*}(*)$, respectively, for every $s \in N$
$EA_{*d}(t)$	It can be shown that this problem is no easier than computing $EA_{**}(t)$
$LD_{sd}(*)$, $LD_{s*}(*)$, $LD_{*d}(*)$, $LD_{**}(*)$,	These are computed by solving for and inverting the corresponding earliest arrival time functions. For example, $LD_{s*}(*)$ is found by solving for $EA_{s*}(*)$ and taking inverses
$LD_{s*}(t)$	It can be shown that this problem is no easier than computing $LD_{**}(t)$
$LD_{*d}(t)$	Performing a time-reversal transformation on the network transforms this into the problem of computing $EA_{s*}(t)$
$LD_{sd}(t)$	Solve the more general problem $LD_{*d}(t)$
$LD_{**}(t)$	Solve $LD_{*d}(t)$ repeatedly for every $d \in N$
$EA_{*d}(*)$	Performing a time-reversal transformation on the network transforms this into the problem of computing $LD_{s*}(*)$

Source: (Dean 2004)

Table 2. Critical Comparison of Time Dependent Shortest Path Algorithms

Algorithm	Query Performance	Pre-processing Computational Requirements	Memory Requirements	Ease of Implementation	Applicability to Multimodal Trip Planner[a]
TD-Dijkstra	Slow	None	Low	Easy	PR, T
DOT	Slow[b]	Low	Low	Easy	PR, A
TD-A*	Fast	Low	Low - Medium	Easy – Medium	PR, T
uni-ALT	Fast	Medium	Medium - High	Medium	Y, (PR – T)
TDALT	Fast[c]	Medium	Medium - High	Medium	Y, (PR – T)
TDCALT	Very Fast[c]	Medium – High	Medium	Medium – Hard	Y
TD-SHARC	Extremely Fast	Very High	Low - High[d]	Hard	Y, P
TCH	Extremely Fast	High	Medium - Very High[e]	Hard	Y, P
TD-RSPP	Medium	None	Low	Hard	D, T

[a]For applicability, the following abbreviations are used: Y – applicable to all searches, PR – applicable to park'n'ride searches, T – testing required to ensure computational performance is satisfactory, A – Applicable in an alternative approach, P – pre-processing resources must be considered for applicability, D – Additional data is required for application of algorithm

[b]DOT is slow in the stated configuration of the multimodal trip planner; however it could be applied efficiently in an alternative approach.

[c]Optimal result is not guaranteed.

[d]Memory requirement is dependent on whether the space-efficient version or standard version is used, with a relatively small trade-off in query performance.

[e]Memory requirement is dependent on whether TCH or ATCH is used, with a relatively small trade-off in query performance.