Shortest Paths in Time-Dependent FIFO Networks Using Edge Load Forecasts [*]

Frank Dehne
School of Computer Science
Carleton University
Ottawa - Canada
frank@dehne.net

Masoud T. Omran
School of Computer Science
Carleton University
Ottawa - Canada
mtomran@scs.carleton.ca

Jörg-Rüdiger Sack
School of Computer Science
Carleton University
Ottawa - Canada
sack@scs.carleton.ca

ABSTRACT

We study the problem of finding shortest paths in time-dependent networks with edge load forecasts where the behavior of each edge is modeled as a time-dependent arrival function with FIFO property. Here, we present a new algorithm that computes for a given start node s and destination node d, the shortest paths and earliest arrival times for all possible starting times. Our algorithm runs in time $O((F_d + \lambda)(|E| + |V| \log |V|))$ where F_d is the output size (number of linear pieces needed to represent the earliest arrival time function) and λ is the input size (number of linear pieces needed to represent the local earliest arrival time functions for all edges in the network). Our method improves significantly on the best previously known algorithm which requires time $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between the start node s to any node in the network. It has been conjectured that there are cases where F_{max} is of super-polynomial size; however, even in such cases, F_d might still be of linear size. In such cases, our algorithm would take polynomial time to find the solution, while other methods require super-polynomial time. Both of the above methods are not useful in practice for graphs where F_d is of super-polynomial size. For such graphs, we present the first approximation method to compute for all possible starting times at s, the earliest arrival times at d within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s.

1. INTRODUCTION

Finding shortest paths in networks is one of the basic operations in Transportation Science, Computer Networks, Robotics, VLSI Design and many other applications. Although well-studied conventional static shortest path algorithms play a fundamental role in applications with non-changing nature, many real-world applications are changing over time [1, 2, 6, 4]. For example, in a road network, the shortest path from a given start node to a destination node during rush hour is not the same as during low traffic periods. Here, we study dynamically changing applications in which network properties are changing over time in a predictable manner and are given as edge load forecasts. For example, in many road networks the traffic load on each link changes predictably during the day. We are interested in finding the shortest path between two nodes of the network for any given time during the day. More precisely, a time-dependent network with edge load forecasts is modeled as a directed graph $G = (V, E)$ where each edge (v, w) is assigned an arrival time function $a_{vw}(t)$ which represents the arrival time at node w for departure time t at node v. Typically, piece-wise linear arrival time functions are used to approximate more complex functions. The notion of arrival time function is extensible to any path $p = \langle v_1, v_2, \ldots, v_k\rangle$ of the network. Starting from v_1 at time t implies an arrival at v_k at time $a_p(t) = a_{v_{k-1}v_k}(\ldots(a_{v_2v_3}(a_{v_1v_2}(t))))$. We note that if the $a_{v_{i-1}v_i}(t)$ are piece-wise linear then $a_p(t)$ is piece-wise linear as well. Given a start node s and a destination node d, our goal is to find the earliest arrival time function $A_{sd}(t)$ from s to d for all $t \in [0, T]$. $A_{sd}(t)$ is the minimum over all $a_p(t)$ for all possible paths p from s to d. As discussed by Orda and Rom in [15], this problem is NP-hard in its general form but there are variations of the problem which are not. For example, in earlier work [10], we considered a version of the problem, where the slope of each linear piece is either 0 or 1. This can be viewed as a network in which edges are available during given intervals and the travel time in each interval is of fixed value. In such a network, if an edge is not available for some arrival time, then one can wait until the next interval becomes available. For such networks, we proposed an $O(\kappa(|E| + |V|Log|V|))$ time algorithm. where κ denotes the total number of availability intervals in the entire graph.

Here, we consider a general class of time-dependent shortest path problems in which the FIFO property holds. The FIFO property is very common in many networks, including road networks, and is defined as having non-decreasing arrival time functions on all edges of the network. This means that for every edge (v, w), a later start at v implies a later arrival at w which is typically the case for predictable dynamic networks. A naive algorithm for this case which computes the earliest arrival time function for every possible path from s to d and then calculates the lower envelope would need ex-

[*]Research supported by NSERC, SUN Microsystems of Canada and HPCVL.

ponential time in many cases because many networks would have an exponential number of possible paths between s and d. As shown by Orda and Rom in [15], the time-dependent shortest paths problem for a time-dependent network with FIFO property can be solved in time $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between s and any node in the network.

In this paper, we present a new algorithm for solving the time-dependent shortest paths problem for a time-dependent network with FIFO property. The algorithm runs in time $O((F_d + \lambda)(|E| + |V| \log |V|))$, where F_d is the output size (number of linear pieces needed to represent the earliest arrival time function) and λ is the input size (number of linear pieces needed to represent the local earliest arrival time functions for all edges in the network). Our method improves significantly upon the best previously known method by Orda and Rom [15]. It has been conjectured [9] that there are cases, where F_{max} is of super-polynomial size. Since $F_{max} \geq F_d$, even in such cases, F_d might still be of linear size. In such cases, our algorithm would take polynomial time to find the solution, while other methods require super-polynomial time.

We also study the case where the output size F_d might be [9] super-polynomial. All previously known methods (including our method outlined above) are not useful in practice for such graphs. In this paper, we present the first approximation method for such instances of the time-dependent shortest path problem. Our method computes for all possible starting times at s the earliest arrival times at d within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E|+|V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s.

The remainder of this paper is organized as follows. In the following Section 2 we discuss previous work and related relevant results from similar problem settings. In Section 3 we discuss some structural properties of the time-dependent shortest path problem. Our new algorithm which solves the time-dependent shortest paths problem for a time-dependent network with FIFO property in time $O((F_d + \lambda)(|E| + |V| \log |V|))$ is presented in Section 4. Our approximation algorithm for time-dependent shortest path instances with possibly super-polynomial size output is presented in Section 5. Section 6 concludes the paper.

2. PREVIOUS ALGORITHMS AND RESULTS

The problem of finding a time-dependents shortest path was first proposed in 1966 by Cooke and Halsey [7]. They considered time to have discrete values. In real-world applications, arrival time functions are usually continuous time functions and approximated by piece-wise linear functions. For the remainder, we assume the FIFO property to hold since this is the case for most practical networks and makes the problem polynomial time solvable. In the following paragraphs, we review previous results for this problem setting.

2.1 Lower Envelope Algorithms

By definition, the earliest arrival time function from s to d is the minimum over all arrival time functions for every path from s to d. This leads to a simple algorithm: compute the arrival time functions of all paths from s to d and compute the lower envelope. For more information on lower envelope algorithms see e.g. [16]. Although this gives the correct solution, such an algorithm is not efficient in that there could be an exponential number of paths from s to d leading to exponential time complexity.

2.2 Label Correcting Algorithms

A slightly modified version of standard label-correcting algorithms (e.g., Bellman-Ford [3]) solves the time-dependent shortest path problem. Here, instead of computing labels for a specific time, one can do this simultaneously for all values of t. In this case, instead of working with scalar arrival times at each node, we consider earliest arrival time functions over all values of time. Orda and Rom [15] proposed such an algorithm which on a FIFO network with piece-wise linear functions has time complexity $O(F_{max}|V||E|)$ where $F_{max} \geq F_d$ is the maximum number of linear pieces needed to represent the earliest arrival time function between s and any node in the network. This has been the best known approach since 1990 when it was presented. Our algorithm is a significant improvement of this method as well as of the methods outlined in the following paragraph.

2.3 Label-Setting Algorithms

In a label-setting algorithm the goal is to compute, in small pieces, actual correct values of output functions rather than iteratively revising these functions. This approach is similar to Dijkstra's algorithm [11] for the static shortest path problem. In contrast to label-correcting algorithms, it is not possible to simply replace scalar label values by functions to solve the problem because a minimum element (i.e., one function which is minimum over the entire domain) may not exist. The main idea of the algorithm is to determine the latest time ϕ, for each node, so that the current earliest arrival time function for any time less than ϕ gives the actual earliest arrival time to the node. For FIFO networks with piece-wise linear arrival time functions, Dean [9] suggested a label-setting algorithm that performs a single chronological scan through time to establish output functions. The algorithm employs the same approach used for solving parametric shortest path problems [5]. In the worst-case, this algorithm has a running time of $O(F^*|E| \log |V|)$ where F^* is the total number of pieces over all output functions in the network.

Recently, Ding et al. [12] presented a simpler label-setting algorithm for the time-dependents shortest path problem for FIFO networks with piece-wise linear functions. The algorithm scans a sequence of time steps the size of which depends on the values of the arrival time functions. Careful analysis of this algorithm shows that this approach yields a solution with time complexity $O(\gamma(|E| + |V| \log |V|))$ which contains a factor γ that is possibly unbounded because it depends on the relative *values* of arrival time functions. An example instance showing that the number of scanned time steps can be unbounded and independent on $|E|, |V|$, and λ is depicted in Figure 1.

3. STRUCTURAL PROPERTIES

Our new algorithm makes extensive use of some structural properties of the problem discussed in this section.

3.1 Solution Function Structure

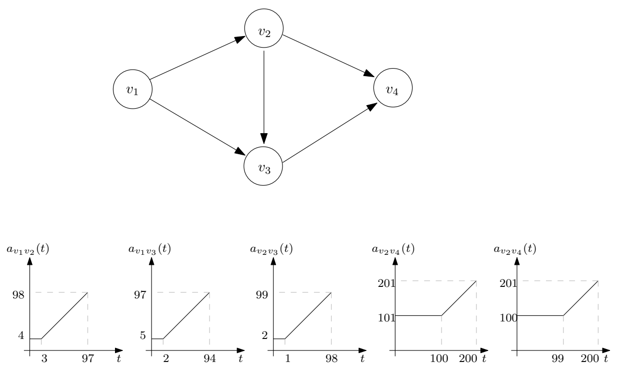

Figure 1: An example instance showing that the number of scanned time steps for the Ding et al. [12] algorithm, can be unbounded.

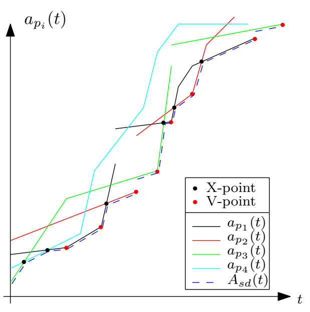

Figure 2: X and V-points

The earliest arrival time function from s to d, $A_{sd}(t)$, is a piece-wise linear function since all input arrival time functions are assumed to be piece-wise linear functions and the function operators used (*function inverse, linear combination, function compound, min, max*) do not change the linearity of the result. We are interested in the points on the curve $A_{sd}(t)$ that connect its different linear pieces, and will refer to them as *change points*. We differentiate between two types of change points. First, a change point may represent the intersection between two pieces of arrival time functions on different paths. Second, a change point may represent a change point on one of the input arrival time functions for a path from s to d. We refer to the first type as X-point and to the second type as V-point. Figure 2 depicts an arrangement of arrival time functions and identifies X and V-points. Every V-point corresponds to a change point on the arrival time function, $a_p(t)$, on some path p from s to d. Each change point on the $a_p(t)$ function is the result of a boundary point between two linear pieces of arrival time functions on an edge of p introduced because of a compound operation for computing $a_p(t)$. In the following lemma, we will show that every boundary point of an edge arrival time function can create at most one V-point on $A_{sd}(t)$.

LEMMA 1. *Suppose P_e is the set of all paths that include edge $e = (v, w) \in E$ and $a_e(t)$ is the arrival time function on e which has λ_e linear pieces. Then, all arrival time functions $a_{p_e}(t), p_e \in P_e$, create in total at most λ_e V-points on $A_{sd}(t)$.*

Proof: Let

$$a_e(t) = \begin{cases} \alpha_e^1 t + \beta_e^1 & 0 \le t \le T_e^1 \\ \alpha_e^2 t + \beta_e^2 & T_e^1 < t \le T_e^2 \\ \vdots & \vdots \\ \alpha_e^{\lambda_e} t + \beta_e^{\lambda_e} & T_e^{\lambda_e - 1} < t \le T_e^{\lambda_e} \\ \infty & T_e^{\lambda_e} < t \end{cases}$$

be the arrival time function on e. For every boundary point $T_e^i, i = 1 \dots \lambda_e$, consider path p_e^i to be the concatenation of a path with the latest starting time (LST) from s which arrives at v at time T_e^i and a path with earliest arrival time (EAT) to d which starts from v at time T_e^i. Because of the definition of p_e^i, for any path $p_e \in P_e$ other than p_e^i, T_e^i will create a change point either at (LST, EAT) or to the left and above this point. Since (LST, EAT) is on $a_{p_e^i}(t)$ and FIFO property holds, any points that fall to the left and above (LST, EAT) are not on $A_{sd}(t)$, thereby other paths can not create new change points on $A_{sd}(t)$. This proves that all arrival time functions $a_{p_e}(t), p_e \in P_e$, create in total at most λ_e V-points on $A_{sd}(t)$. $\square$

Let $\lambda = \sum_{e \in E} \lambda_e$ be the total number of linear pieces on edge arrival time functions in the entire network. Since every V-point comes from a boundary point on some edge arrival time function, Lemma 1 implies that there can not be more than $O(\lambda)$ V-points on $A_{sd}(t)$.

3.2 Possibly Super-polynomial Output Size

In [9], the author conjectured that in a FIFO network with piece-wise linear arrival time functions on edges, the earliest arrival time function $A_{sd}(t)$ from a source node s to a destination node d may have more than a polynomial number of linear pieces. This means that there possibly exist network structures that result in super-polynomial complexity for earliest arrival time functions to some nodes of the network.

The *super-polynomial structure* could appear as a subgraph of the actual input network, resulting in earliest arrival time functions with super-polynomial number of pieces for destination nodes whose shortest path from s passes through the super-polynomial structures. However, the earliest arrival time function from s to d could easily be of linear size since the earliest arrival time path may not intersect the super-polynomial structure at all. In this case, F_{max} would be of super-polynomial size and F_d would be of linear size.

4. A NEW ALGORITHM FOR INSTANCES WITH POLYNOMIAL SIZE OUTPUT

Our new algorithm is based on the idea that instead of building all earliest arrival time functions for all nodes in the network, we find the earliest arrival time function to destination node d directly. The main problem here is to find all starting times for which the earliest arrival time function from s to d, $A_{sd}(t)$, changes from one linear piece to another as well as all linear functions between these change points. In Section 3, we introduced two types of change points in $A_{sd}(t)$: V-points and X-points. In Section 3, we also showed that at most $O(\lambda)$ V-points exist on $A_{sd}(t)$, where λ is the total number of pieces in all input arrival time functions. Moreover, using Dijkstra's static shortest path algorithm for every change point of all arrival time functions of the input, both forward to d and backward to

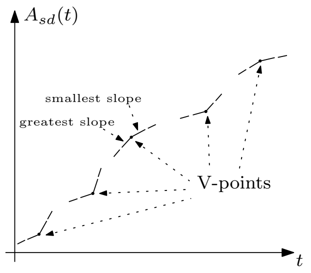

Figure 3: A sample $A_{sd}(t)$ function with all V-points and their adjacent linear functions.

s, we capture all V-points that can potentially be on $A_{sd}(t)$. (For reversibility of the time-dependent shortest path problem see [8].) To construct the entire function $A_{sd}(t)$, we also compute the linear pieces to the left and to the right of each V-point. These are pieces with the earliest arrival time and the smallest (greatest) slope on the right (left) vicinity of each V-point. Figure 3 shows a sample $A_{sd}(t)$ function once all V-points have been detected. Note that, given a time t_0, the smallest (greatest) slope piece can be computed while computing the earliest arrival time to d for starting time t_0. This is accomplished by keeping the product of slopes for each node in the shortest path tree as a secondary key when Dijkstra's algorithm finds two or more entries of the heap that have the same arrival time value. In this case, selecting the entry with smallest (greatest) slope leads to the smallest (greatest) slope linear piece. To show the correctness of this approach consider any two paths from s to d starting at time t_0 with equal arrival times but different slopes on their arrival time functions. The first time where they have equal arrival time values is either at d or at some earlier node d'. In the latter case, they will share the same "postfix" path from d' to d. In either case, selecting the smallest (greatest) slope product from the heap, among equal arrival time values, maintains the smallest (greatest) slope.

Thus far, we have determined all V-points and the slope of $A_{sd}(t)$ in their vicinity. We build the remaining part of $A_{sd}(t)$ by adding the missing piece between every pair of consecutive V-points on $A_{sd}(t)$. Due to the linearity of the input arrival functions, the X-points between two consecutive V-points are in concave position (seen from below). Consider two consecutive V-points, V_l and V_r, found in the previous step together with the linear pieces in their vicinity. Two cases arise for the linear pieces to the right of V_l and to the left of V_r. They either overlap, or they intersect in some point $I = (x_I, y_I)$. If they overlap, then the piece connecting the two V-points is the solution (Figure 4-a). In case of an intersection, two cases are possible. First, if calculating a static shortest path at time x_I returns the same arrival time y_I as for the intersection point then the pieces, v_l to I and I to v_r are the solution (Figure 4-b). Second, if calculating a static shortest path at time x_I returns a value less than y_I, then we found a new linear piece that is hiding the intersection point $I = (x_I, y_I)$ (Figure 4-c). The linear pieces to the right of V_l and to the left of V_r intersects the new piece, and we recurse. See Theorem 1 for further details.

Algorithm 1 below shows the entire TDSP algorithm. It determines both V-points and X-points in separate sections. After initializing $A_{sd}(t)$ at the beginning of the algorithm (Line 2), in Lines 3 through 11 we capture all V-points along

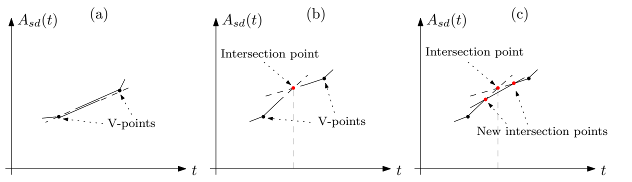

Figure 4: (a) Overlapping pieces (b) Intersection point on $A_{sd}(t)$ (c) Intersection point hidden by another linear piece

with their adjacent linear pieces to their left and right. In Lines 12 through 36 we detect all X-points on $A_{sd}(t)$. In the first phase, for every edge $e = (v, w)$ in the network and for every boundary point T_e^i corresponding to an edge e, the algorithm finds the latest starting time (LST) from s to arrive at v at time T_e^i by calculating a static shortest path algorithm (Dijkstra) backwards from v to s at time T_e^i (line 5). We also execute a forward static shortest path to obtain the earliest arrival time (EAT) at d starting from v at time T_e^i (Line 6). This provides the rightmost and lowest V-point that could be found on $A_{sd}(t)$ as a result of boundary point T_e^i. As shown earlier, for all other paths that include e, V-points for T_e^i will be hidden by some other linear pieces. In order to make sure that (LST, EAT) is on the final solution we calculate a static shortest path from s to d starting at time LST (Line 7). If the arrival time is the same as EAT, then (LST, EAT) is indeed a V-point on $A_{sd}(t)$. In this case, to find the linear pieces near V-points on $A_{sd}(t)$, we find the linear pieces with the smallest slope and the largest slope adjacent to the left and right of each V-point, respectively. Finally, we add each V-point found along with its adjacent linear pieces to a list for use in the next step (Lines 8 through 11).

In the second part of the algorithm, we first sort all V-points by ascending order of LST value (Line 12). Then, starting from the first two V-points, we read pair of consecutive points from the list and build the $A_{sd}(t)$ function between them as we scan these points (Lines 13, 15 and 35). Two cases are possible: either the linear function to the right of the first V-point, RF_1, and the linear function to the left of the second V-point, LF_2, overlap or they intersect. In case of overlap, the linear piece between the two points must be a piece of $A_{sd}(t)$ (Lines 16 and 17). If RF_1 and LF_2 intersect, we add the intersection point to a stack (Lines 19-21). Here, either there is a linear piece below the intersection point that prevents it from being on the final solution or the intersection point is on $A_{sd}(t)$. In the first case, we find the linear piece with the greatest slope and add two new intersection points to the stack (Lines 29-33). If the intersection point is on the final solution (Lines 24-27) we add the linear piece on RF_1 between the first V-point and the intersection point to $A_{sd}(t)$. Line 34 adds the linear piece to the left of the right V-point. Finally, in Line 36, we add an unbounded linear piece if the last linear function is unbounded.

THEOREM 1. *Given a source node s and destination node d, the TDSP algorithm outlined above correctly determines $A_{sd}(t)$ for all $t \in [0, \infty)$.*

Proof: The algorithm first finds all V-points on $A_{sd}(t)$ along with linear pieces to the left and right of each V-point.

Algorithm 1: $TDSP(G, V, E, s, d)$

```
begin
    $A_{sd}(t) \leftarrow NULL$
    for every edge $e = (v, w) \in E$ do
        for $i = 0$ to $\lambda_e$ do
            $LST \leftarrow SPbackward(v, s, T_e^i)$
            $EAT \leftarrow SPforward(v, d, T_e^i)$
            $TMP \leftarrow SPforward(s, d, LST)$
            if $EAT = TMP$ then
                $f_l \leftarrow GSEATfunction(s, d, LST)$
                $f_r \leftarrow SSEATfunction(s, d, LST)$
                $InsertToList(L, \{LST, EAT, f_l, f_r\})$
    $Sort(L)$
    $\{LST_1, EAT_1, LF_1, RF_1\} \leftarrow RemoveItem(L)$
    while $NotEmpty(L)$ do
        $\{LST_2, EAT_2, LF_2, RF_2\} \leftarrow RemoveItem(L)$
        if $Overlap(RF_1, LF_2)$ then
            $AddLinearPiece(A_{sd}(t), RF_1, LST_1, LST_2)$
        else
            $(PX_1, PY_1) \leftarrow (LST_1, EAT_1)$
            $(PX_2, PY_2) \leftarrow IntersectionPoint(RF_1, LF_2)$
            $Push(S, (PX_2, PY_2, RF_1, LF_2))$
            while $NotEmpty(S)$ do
                $(PX_2, PY_2, f_l, f_r) \leftarrow Pop(S)$
                $TMP \leftarrow SPforward(s, d, PX_2)$
                if $TMP = PY_2$ then
                    $AddLinearPiece(A_{sd}(t), f_l, PX_1, PX_2)$
                    $PX_1 \leftarrow PX_2$
                else
                    $f_m \leftarrow GSEATfunction(s, d, PX_2)$
                    $(IX_1, IY_1) \leftarrow IntersectionPoint(f_l, f_m)$
                    $(IX_2, IY_2) \leftarrow IntersectionPoint(f_m, f_r)$
                    $Push(S, (IX_2, IY_2, f_m, f_r))$
                    $Push(S, (IX_1, IY_1, f_l, f_m))$
            $AddLinearPiece(A_{sd}(t), f_r, PX_2, LST_2)$
        $\{LST_1, EAT_1, LF_1, RF_1\} \leftarrow$
        $\{LST_2, EAT_2, LF_2, RF_2\}$
    if $EAT_1 \neq \infty$ then
    $AddLinearPiece(A_{sd}(t), RF_1, LST_1, \infty)$
    return $(A_{sd}(t))$
end
```

By Lemma 1, no V-points other than those considered can be on $A_{sd}(t)$. Then, the algorithm picks every two consecutive V-points to compute all X-points between them. Let $v_l = (x_l, y_l)$ and $v_r = (x_l, x_r)$ be two consecutive V-points on $A_{sd}(t)$. Also, suppose that RF and LF are the linear pieces to the right of v_l and to the left of v_r, respectively. Either RF and LF overlap or they intersect. If overlap, the linear piece on RF (or LF) from x_l to x_r is part of the solution function since no other V-points are possible between v_l and v_r. On the other hand, if the two functions intersect in some point $I = (x_I, y_I)$ and the intersection is on $A_{sd}(t)$, then the linear piece on RF from x_l to x_I is on $A_{sd}(t)$ since no other V-points are possible between v_l and v_r. If I is not on $A_{sd}(t)$, then there must be another linear piece preventing it from being on the solution. The algorithm determines such a piece with maximum slope. The extension of the linear piece must intersect both RF and LF since otherwise there must be another V-point between v_l and v_r. Let I_l and I_r be the two intersection points. We now recursively perform what we did for I, first for I_l and then I_r. Starting from v_l, we add linear pieces to the solution function once I_l is found to be on $A_{sd}(t)$. Then, we move to the next intersection. As a last step, the algorithm adds to the solution function the last piece on LF between the last intersection and x_r. Since we verify every X-point for being on $A_{sd}(t)$ and no more V-points are possible between two consecutive V-points, the algorithm finds all X-points. Since V-points and X-points are the only change points on $A_{sd}(t)$, Algorithm TDSP correctly finds all linear pieces of $A_{sd}(t)$. □

THEOREM 2. *The time complexity of the TDSP algorithm outlined above is* $O((F_d + \lambda)(|E| + |V| \log |V|))$.

Proof: First, the algorithm executes a slightly modified version of Dijkstra's shortest path algorithm both forward and backward for each edge of the graph to find all possible V-points. It then executes another modified version of Dijkstra's algorithm to find greatest and smallest slope pieces close to each V-point. Supposing that for every edge (v, w) and a given starting time at v we can compute the arrival time at w in $O(1)$ time, for a given starting time at s, the earliest arrival time at d is computed in the same time as Dijkstra's algorithm, namely $O(|E| + |V| \log |V|)$ using Fredman and Tarjan's implementation [13]. Consequently, the time-complexity of the first part is $O(\lambda(|E|+|V| \log |V|))$ where λ is the total number of linear pieces in all edge arrival time functions. Then, in the second part, the algorithm executes a modified Dijkstra's algorithm as many time as we find intersection points. At each intersection point found, we determine the linear piece with greatest slope that hides the intersection point. This guarantees that, every time we run a modified Dijkstra's algorithm at the intersection point we obtain a new linear piece which is part of the solution $A_{sd}(t)$. As a result, we will execute the modified Dijkstra's algorithm at most as many times as there are X-points on $A_{sd}(t)$. With F_d defined as the number of linear pieces on $A_{sd}(t)$, the second part runs in time $O(F_d(|E|+|V| \log |V|))$. Hence, the total time complexity of the TDSP algorithm is $O(F_d + \lambda)(|E| + |V| \log |V|))$ □

5. AN APPROXIMATION ALGORITHM FOR INSTANCES WITH SUPER-POLYNOMIAL SIZE OUTPUT

We now present an approximation algorithm for instances where the output size F_d might be super-polynomial. Our method computes for all possible starting times $t \in [0, T]$ at s the earliest arrival times at d within error at most ϵ. The algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the earliest arrival time at d for the latest possible starting time at s. In Section 3, we showed that the number of V-points on $A_{sd}(t)$ is bounded by λ, the total number of linear pieces in all edge arrival time functions of the network (input size). Hence, for instances where the output size F_d is super-polynomial, it follows that the number of X-points on $A_{sd}(t)$ must be super-polynomial. Our approximation algorithm first computes all V-points on $A_{sd}(t)$ as outlined in the previous section. Then, for every two consecutive V-points $v_l = (x_l, y_l)$ and $v_r = (x_r, y_r)$ we compute an approximation of $A_{sd}(t)$. If $y_r - y_l \leq \epsilon$ we simply connect v_l and v_r through a linear piece. If $y_r - y_l > \epsilon$ we calculate the static shortest path backward from d to s at time $y_m = \frac{y_l + y_r}{2}$ and obtain a point $v_m = (x_m, y_m)$ on $A_{sd}(t)$. We recursively perform this splitting operation until the difference between arrival times is less than ϵ. As a final step, we connect all points obtained (V-points plus new points) through linear pieces. With $\Delta = A_{max} - A_{min}$ defined as the difference between the earliest arrival times $A_{min} = A_{sd}(t = 0)$ and $A_{max} = A_{sd}(t = T)$, the time complexity of this algorithm is $O((\frac{\Delta}{\epsilon} + \lambda)(|E| + |V| \log |V|))$. We can improve this time complexity to $O((\frac{\Delta}{\epsilon})(|E| + |V| \log |V|))$ by avoiding the calculation of the V-points altogether. We calculate the static shortest paths backwards from d to s at all times $A_{min} + i\epsilon$, $i = 1, \ldots, \lfloor \frac{\Delta}{\epsilon} \rfloor$, and then connect the points obtained by linear pieces. Here, the main complication is the possibility of discontinuities in $A_{sd}(t)$. Handling this case within the same time complexity is possible.

6. CONCLUSION

In this paper, we presented a new algorithm which solves the time-dependent shortest paths problem for a time-dependent network with FIFO property. The running time of our algorithm is $O((F_d + \lambda)(|E| + |V| \log |V|))$, where F_d is the output size and λ is the input size. Our method improves significantly on the best previously known bound by Orda and Rom [15]. We also study instances where the output size F_d is super-polynomial, for which all previously known methods (including our first method presented here) require super-polynomial time. We present the first approximation method for such instances of the time-dependent shortest path problem. Our method computes for all possible starting times the earliest arrival times within error at most ϵ. Our algorithm runs in time $O(\frac{\Delta}{\epsilon}(|E| + |V| \log |V|))$ where Δ is the difference between the earliest arrival times at d for the latest and earliest starting times at s. The methods presented in this paper are independent of the underlying static shortest path algorithm, so that more efficient shortest path algorithms than the generic Dijkstra algorithm can be used when applicable. E.g., in planar networks, applying linear time shortest path algorithms [14] will further improve our results. In many practical networks heuristics such as A^* can be applied to improve the practical performance of our methods. We are currently implementing our algorithm.

7. ACKNOWLEDGEMENT

We thank Rolf Klein and Florian Berger for insightful discussions on this topic.

8. REFERENCES

[1] Ravindra K. Ahuja, James B.Orlin, Stefano Pallottino, and Maria G.Scutella. Dynamic shortest paths minimizing travel times and costs. *Networks*, 41:205, 2001.

[2] Ravindra K. Ahuja, James B. Orlin, Stefano Pallottino, and Maria Grazia Scutellà. Minimum time and minimum cost-path problems in street networks with periodic traffic lights. *Transportation Science*, 36(3):326–336, 2002.

[3] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[4] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electr. Notes Theor. Comput. Sci.*, 92:3–15, 2004.

[5] Patricia June Carstensen. *The complexity of some problems in parametric linear and combinatorial programming.* PhD thesis, University of Michigan, 1983.

[6] Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Fates: Finding a time dependent shortest path. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 165–180, London, UK, 2003. Springer-Verlag.

[7] K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

[8] Carlos F. Daganzo. Reversibility of the time-dependent shortest path problem. *Transportation Research Part B: Methodological*, 36(7):665–668, August 2002.

[9] Brian C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical report, MIT Department of Computer Science, 2004.

[10] Frank. Dehne, Masoud T. Omran, and Jorg-R. Sack. Minimum travel time on networks with time-dependent edge availabilities. Technical report, Carleton University, Ottawa, 2009.

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

[12] Bolin Ding, Jeffrey Xu Yu, and Lu Qin. Finding time-dependent shortest paths over large graphs. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 205–216, New York, NY, USA, 2008. ACM.

[13] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[14] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[15] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990.

[16] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, New York, NY, USA, 1996.