On the Complexity of Time-Dependent Shortest Paths

Luca Foschini* John Hershberger† Subhash Suri‡

Abstract

We investigate the complexity of shortest paths in time-dependent graphs, in which the costs of edges vary as a function of time, and as a result the shortest path between two nodes s and d can change over time. Our main result is that when the edge cost functions are (polynomial-size) piecewise linear, the shortest path from s to d can change $n^{\Theta(\log n)}$ times, settling a several-year-old conjecture of Dean [Technical Reports, 1999, 2004]. We also show that the complexity is polynomial if the slopes of the linear function come from a restricted class, present an output-sensitive algorithm for the general case, and describe a scheme for a $(1+\epsilon)$-approximation of the travel time function in near-quadratic space. Finally, despite the fact that the arrival time function may have superpolynomial complexity, we show that a *minimum delay* path for any departure time interval can be computed in polynomial time.

1 Introduction

Time-dependent networks are used to model situations in which the cost of traversing an edge varies with time. While the general framework has many applications, the everyday problem of route planning on road networks is easily the most compelling. Due to varying congestion on roads during the day, both the time to travel from a source node s to a destination d and the optimal path between them can change over time. In fact, traffic conditions often create temporary bottlenecks that cause one to reach the destination at more or less the same time despite leaving much later, at a significant reduction in travel time. By plotting the arrival time as a function of the departure time, one can plan optimal-*time* paths, instead of optimal-*length* paths, as is the current practice.

Besides their natural applications in the management and planning of complex networks such as highways and communication networks, time-dependent shortest paths are also a fundamental problem with non-trivial complexity issues and subtle implications of model parameters. The problem has been studied since 1969 when Dreyfus [13] observed that Dijkstra's algorithm can be used to find a time-dependent shortest path, given a starting time at the source node. Dreyfus implicitly assumed an unrestricted *waiting policy* at intermediate nodes to ensure that if waiting at a node results in an earlier arrival time (due to reduced congestion), then optimal wait times are utilized at nodes. It turns out, however, that the choice of waiting policies and the type of the edge cost functions have non-trivial implications on time-dependent shortest paths, which were explored more fully in the 1980's and 1990's by Orda and Rom [28]. Their findings include, among other results, that if waiting is *forbidden* anywhere in the network and the network is not FIFO (first-in-first-out), then the shortest path may be non-simple and nonconcatenated, violating the subpath optimality property. There may also be paths that traverse an infinite number of edges but still have finite delay. Thus, in these cases neither Dijkstra's algorithm nor Bellman-Ford can find time-dependent shortest paths, and variants of these problems are known to be NP-hard [30].

In time-dependent networks, a non-FIFO edge is equivalent to a FIFO edge if waiting is allowed: one can always wait at the tail of an edge to optimize the arrival time at the head of the edge. In this paper, therefore, we focus on the FIFO model with the understanding that waiting at nodes is permitted to deal with any non-FIFO effects. The problem of shortest paths still exhibits surprising complexity, even when the edges are FIFO and their cost functions are piecewise linear and continuous. In particular, the complexity of the following natural problem, which is the focus of our work, has been open for some time: *Given a source node s, compute the* arrival time function *at another node d for all starting times at s.*

Orda and Rom [28] and many others have con-

*Department of Computer Science, University of California, Santa Barbara, CA 93106. This work was partially supported by National Science Foundation grants CNS-1035917 and CCF-0514738.

†Mentor Graphics Corp, 8005 SW Boeckman Rd., Wilsonville, OR 97070

‡Department of Computer Science, University of California, Santa Barbara, CA 93106. This work was partially supported by National Science Foundation grants CNS-1035917 and CCF-0514738.

sidered this problem due to its special significance for planning. Rather than asking for the best path starting at a particular time, which is easily computed by Dijkstra's algorithm, the goal here is to construct a *travel planning map* that gives, for any desired arrival time, the optimal time to depart so as to minimize the travel time. We should point out that in our problem time is considered a *continuous* variable, and so the problem has a distinctly different character from the *discrete time* version in which only a finite set of departure times is permitted [3, 5, 24]. In the discrete version, the complexity of the arrival time function is bounded by the discreteness of the departure times, but the arrival time function may suffer from severe and unnatural discontinuities.[1]

Orda and Rom [28] extended the shortest path finding paradigm of Bellman and Ford [15] to work with linear functions as edge costs, instead of scalar costs as in the non-time-dependent case. They left the *computational complexity* of the problem unresolved. Specifically, they presented the running time of their algorithm in terms of a *functional complexity* that "hides" the actual cost of composing and minimizing the arrival time functions. If the edge cost functions are piecewise linear, then their compositions and minimizations at intermediate nodes are also piecewise linear, but the *number of breakpoints* in these functions can grow, and this complexity is omitted from the analysis [28]. It is possible that Orda and Rom may have assumed that this complexity remains polynomially bounded, but no provable bound is known. In fact, in several subsequent papers, both in theoretical works as well as applied papers, the authors have simply presented the performance of their algorithms as if the *size of the arrival time function* is not much worse than linear [12, 18].

The most systematic study of the arrival time function's complexity was performed by Dean [7, 8], who after an initial erroneous claim that the complexity is linear in the number of the edge function breakpoints [7] *conjectured* that this complexity may be *superpolynomial* [8]:

> *In a FIFO network with piecewise linear arc arrival functions, the arrival function at a destination node can have a superpolynomial number of pieces.*

Our Results A resolution of this conjecture is our main result. In particular, we show that in an n-node graph with (polynomial-size) piecewise linear edge cost functions, the arrival time function has complexity $n^{\Theta(\log n)}$ in the worst case. More specifically, even with linear edge costs, there exists an n-node graph in which the shortest path from a source s to a destination d changes $n^{\Omega(\log n)}$ times, or equivalently the arrival time function has $n^{\Omega(\log n)}$ breakpoints. On the other hand, for any n-node graph whose piecewise linear edge cost functions have at most K pieces in total, the complexity of the arrival time function is at most $Kn^{O(\log n)}$. Both of these bounds assume that either the edges respect FIFO transit, or allow arbitrary waiting at nodes to deal with non-FIFO behavior of edges.

We analyze the complexity of the arrival time function by formulating connections to the *parametric* shortest paths problem [4, 17, 23]. The edge costs in both the parametric and the time-dependent shortest path problems are functions of the underlying parameter (γ for the former, and time t for the latter), but they interact in different ways. In the former, the cost of a path is the sum of edge costs with the *same value of* γ throughout the network, while in the latter, the time parameter varies over the path through compositions: the arrival time at the second edge depends on the transit time over the first edge, and is different from the initial departure time, and so on. In order to use the known lower bound on parametric shortest paths, we show how to transform an instance of the parametric shortest path problem into a time-dependent one while retaining all the breakpoints. The proof of the upper bound first reduces the problem to one with linear cost functions, and then adapts an inductive argument of Gusfield [17].

We also address the algorithmic question of computing the arrival time function. First, we propose an output-sensitive algorithm, so that when the arrival time function is well behaved, we get an efficient scheme. Second, we propose a $(1+\epsilon)$-*approximation* scheme that computes a representation of the arrival time function so that the travel time estimated using this approximation is at most $(1+\epsilon)$ times the true value. If the maximum and minimum travel times are $D_{\max}$ and $D_{\min}$, the approximation has size $O(K\frac{1}{\epsilon}\log(D_{\max}/D_{\min}))$.

Finally, we show that the *minimum delay* path over a given departure time interval can be computed in polynomial time, without having to compute the arrival time function explicitly. In particular, it is possible to find the minimum delay path whose departure time lies in a specified interval in $O(K(n\log n + m))$ time.

[1] One can show that if the true travel time functions for edges are linear but are approximated discretely using fixed-size buckets, then the error in the travel time estimate can grow exponentially, depending on the *slope* of the linear function.

Related Work The study of shortest paths is a cornerstone of combinatorial algorithms. The classical algorithms, such as those by Dijkstra and Bellman-Ford [6, 15], compute shortest paths in a *static* network with fixed edge costs. Many applications, however, deal with a dynamic network in which the edge costs are subject to change. (Changes in the network topology, such as inclusion or exclusion of an edge, can also be modeled through edge costs.) Time-dependent networks are one way to model such dynamics, but there are others as well, which we briefly discuss below.

The *dynamic shortest paths* problem deals with insertion and deletion of edges in a graph, with a focus on maintaining a data structure for efficiently updating the shortest path tree from a single source or supporting fast shortest path queries between two nodes in response to such a change [11, 20, 29, 31]. The *stochastic shortest paths* problem attempts to capture uncertainty in edge costs by modeling them as random variables and computing paths with minimum expected costs [2, 26, 27]. The *weighted region shortest paths* [21, 22] problem models the dynamics of the environment in a continuous space: the space is divided into regions and each region has a different *speed* of travel, however, the speed is not a function of time.

The problem that bears most similarity to time-dependent paths is that of *parametric shortest paths*. In this problem, the cost of each edge e in the graph varies as a linear function of a parameter γ, i.e., $c(e) = a\gamma + b$. The shortest path from a source node s to a destination node d depends on the value of the parameter γ, and the goal in the parametric shortest path problem is to compute the shortest paths for all values of γ. A result by Carstensen [4] shows that in the worst case the shortest path from s to d can change $n^{\Omega(\log n)}$ times as γ varies; a simpler and more direct proof of this lower bound is given by Mulmuley and Shah [23]. An upper bound of $n^{O(\log n)}$ is given by Gusfield [17].

A more restricted form of the parametric function is used by Karp and Orlin [19] and Young, Tarjan, and Orlin [32]. In their model the edge costs are defined as $c(e) = c_e - \gamma$. This simpler form is far more tractable and generally leads to only a polynomial number of distinct shortest paths. Recently, Erickson [14] also used this form to give an efficient algorithm for parametric maximum flow in planar graphs. However, it is the more general linear form studied by Carstensen and Gusfield that is relevant to our time-dependent shortest paths.

Time-dependent shortest paths have also been studied with an *applied* perspective, with emphasis on empirical validation and applications [10, 12, 24, 25]. Our result shows that none of these algorithms can be expected to perform well in the worst case.

2 Definitions and Preliminaries

We consider a directed graph G with n vertices and m edges, where the cost of each edge is a piecewise linear function of time, representing the time-dependent delay along it. The edge costs respect first-in-first-out (FIFO), meaning that each linear segment in the delay function has slope at least -1. Instead of reasoning with delays, we find it more convenient to work with *arrival time* functions, as did Orda and Rom [28]. In particular, if P is a set of paths with common first and last nodes u and v, then $A[P]$ is the *earliest arrival time* function for P. $A[P]$ is a function of the start time t at node u, written in full as $A[P](t)$, minimizing the arrival time at v over all paths in P, for each value of t. (We use square brackets around the path set argument of $A[P]$ to emphasize that it is discrete; we use parentheses around the continuous t argument.) We define simplified versions of the notation for three common cases:

$A[e]$ is the arrival time function for an edge $e = (u, v)$, that is, $A[e](t)$ gives the time of arrival at vertex v for travel on e departing from vertex u at time t. The function $D[e](t) \equiv (A[e](t) - t)$ is the *delay* or *travel time* along the edge e. Since travel times are nonnegative, $A[e](t) \geq t$ for all t.

$A[p](t)$ is the path arrival function for the path $p = (v_1, v_2, \ldots, v_q)$ defined by functional composition as $A[p](t) = A[(v_{q-1}, v_q)] \cdot A[(v_{q-2}, v_{q-1})] \cdots A[(v_1, v_2)](t)$.

$A[s, d](t) \equiv A[\mathcal{P}_{s,d}]$ is the earliest arrival time function for source s and destination d, minimizing over the set $\mathcal{P}_{s,d}$ of all paths in G from s to d, $A[s, d](t) = \min(A[p](t) \mid p \in \mathcal{P}_{s,d})$.

We denote by K the total number of linear segments in all the edge arrival time functions $A[e]$. By a *breakpoint*, we mean a value of t at which the slope of the arrival time function changes. In a piecewise linear function, the number of breakpoints differs from the number of linear segments by only one, so for the sake of notational brevity, we use the two counts interchangeably in our analysis, depending on the context. The following easy lemma characterizes the composition of $A[e]$ functions.

LEMMA 2.1. *Suppose that $x(t)$ and $y(t)$ are monotone, piecewise linear functions. Then their composition $z(t) = y \cdot x(t) = y(x(t))$ is also monotone and piecewise*

linear. If $z(t)$ has a breakpoint at $t = t_0$, then either x has a breakpoint at t_0 or y has one at $x(t_0)$.

Proof: The composition of monotone functions is monotone. For every t_0 such that $t = t_0$ is not a breakpoint of x and $x(t_0)$ is not a breakpoint of y, x and y are linear functions $x(t) = at + b$ and $y(t) = ct + d$, for particular constants a, b, c, and d. Then $z(t) = y(x(t)) = c(at+b)+d$, which is linear in t. The functional form of $z(t)$ is fixed in the neighborhood of t_0, and so z cannot have a breakpoint there. Thus z is linear except at breakpoints derived from those of x and y. □

2.1 Primitive and Minimization Breakpoints

Breakpoints of edge functions $A[e]$ are called *primitive breakpoints*. The function $A[p]$ for a path p is the composition of edge functions; breakpoints of $A[p]$ that arise from the primitive breakpoints of the composed functions are called *primitive images*. A primitive image b_F in the function $F = f_q \cdot f_{q-1} \cdots f_1$ that occurs at $t = t_F$ is the image of a primitive breakpoint b in some f_i if b occurs at $t = t_i$ in $f_i(t)$ and $t_i = f_{i-1} \cdot f_{i-2} \cdots f_1(t_F)$. The primitive breakpoint b whose image is b_F is called the *preimage* of b_F.

Breakpoints in $A[s, d]$ can be of two types, primitive images and *minimization breakpoints*. A minimization breakpoint occurs in $A[s, d]$ at some time t if the arrival time of the path p_1 that was optimal at time $t - \epsilon$ becomes larger than the arrival time of another path p_2 at time $t + \epsilon$, for an infinitesimal ϵ. A minimization breakpoint b occurs because of minimization at a particular node v (the first node in the last section of $p_1 \cap p_2$), somewhere after s and at or before d; each node x in $p_1 \cap p_2$ from v to d has a minimization breakpoint at $t = b$ that is an image of the one at v. Figure 1 illustrates the two types of breakpoints. Informally, minimization breakpoints are created by the pointwise minimum operation on the arrival time functions among all the paths $p \in \mathcal{P}_{s,d}$, while primitive images are the breakpoints that each path arrival function has as a consequence of being the functional composition of piecewise linear edge functions.

LEMMA 2.2. *If $x(t)$ and $y(t)$ are monotone, piecewise linear functions, then so is* $\min(x(t), y(t))$.

It follows from Lemmas 2.1 and 2.2 that the function $A[s, d]$ is nondecreasing and piecewise linear.

OBSERVATION 2.3. *$A[s, d]$ can be partitioned into compact intervals (with respect to the parameter t) in which one combinatorial, simple path of G is optimal. Intervals are separated by minimization breakpoints in the piecewise linear function representing $A[s, d]$. (See Figure 1.)*

The number of breakpoints of arrival functions is expressed as $B(\cdot)$. For instance, $B(A[s, d])$ is the number of breakpoints of the earliest arrival function between nodes s and d. We give separate bounds on the cardinality of the sets of primitive images and minimization breakpoints. We first focus on primitive images.

LEMMA 2.4. *Two distinct breakpoints b, b' in the monotone piecewise linear function $f_1 \cdot f_2 \cdots f_q$ are images of two distinct primitive breakpoints in the set of piecewise linear functions $\{f_1, f_2, \ldots, f_q\}$.*

Proof: If b and b' are images of breakpoints from different functions f_i and f_j, they are clearly distinct. If they are images of breakpoints from the same function f_i, then by Lemma 2.1 and associativity of functional composition their preimages occur at different t values in the domain of f_i, and hence are distinct. □

The following key lemma shows that primitive breakpoints do not create too many images. In particular, while an exponential number of different paths may use an edge, a primitive breakpoint creates at most one image in the arrival time function at any node, and in particular in $A[s, d]$.

LEMMA 2.5. *Any two distinct primitive images b, b' in $A[s, d]$ are images of two distinct primitive breakpoints among the functions $A[e]$, for $e \in G$.*

Proof: If b and b' are images of breakpoints from different edge functions, then their preimages must be distinct. Therefore, we need to consider only pairs of breakpoints whose preimages belong to $A[e]$ for the same edge $e \in G$. Recall that $A[s, d] = \min(A[p] \mid p \in \mathcal{P}_{s,d})$ where $\mathcal{P}_{s,d}$ is the set of all paths in G from s to d. Any breakpoints whose preimages come from $A[e]$ must belong to $A[p]$ for some $p \in \mathcal{P}_{s,d\,|\,e}$, where $\mathcal{P}_{s,d\,|\,e}$ is the subset of paths in $\mathcal{P}_{s,d}$ that contain the edge e. If $e = (u, v)$, then by distributivity of function composition over the min operation we can rewrite $A[\mathcal{P}_{s,d\,|\,e}] = \min(A[p] \mid p \in \mathcal{P}_{s,d\,|\,e})$ as

$$\min(A[p] \mid p \in \mathcal{P}_{vd}) \cdot A[e] \cdot \min(A[p] \mid p \in \mathcal{P}_{su}),$$

corresponding to the concatenation of the shortest path from s to u with edge $e = (u, v)$, followed by the shortest path from v to d. But the first and last terms

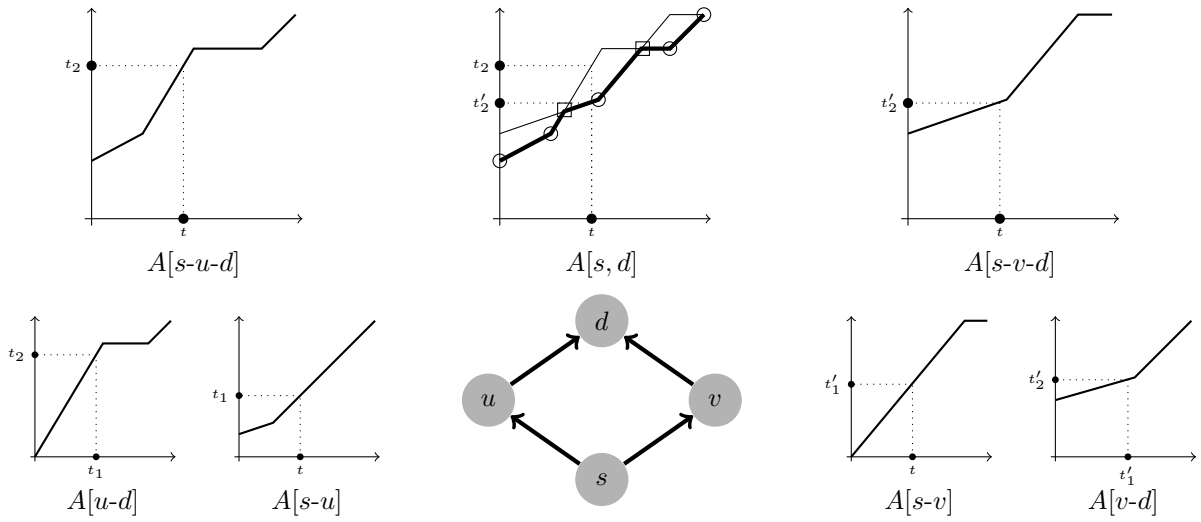

Figure 1: Illustration of primitive and minimization breakpoints. The four subfigures at the bottom show the arrival time functions for the four edges in the graph. The figures on the top show arrival time functions for the path (s, u, d) on the left, the path (s, v, d) on the right, and the result of their pointwise minimization at d, namely, $A[s, d]$. The figures also show the progression of time: when following the path (s, u, d), leaving at time t we reach u at time t_1, and reach d at time t_2. When following the path (s, v, d), the arrival times are t_1' at v and t_2' at d. Of these two paths, the latter is quicker for departure time t, as shown in $A[s, d]$. The subfigure for $A[s, d]$ also highlights primitive images (drawn as circles) and minimization breakpoints (drawn as squares).

in the composition are just $A[s, u]$ and $A[v, d]$, so we have

$$A[\mathcal{P}_{s,d\,|\,e}] = A[v, d] \cdot A[e] \cdot A[s, u].$$

All three of the composed functions are monotone and piecewise linear, so Lemma 2.4 applies: if two distinct breakpoints in $A[\mathcal{P}_{s,d\,|\,e}]$ are images of primitive breakpoints in $A[e]$, they must be images of distinct breakpoints in $A[e]$. □

Corollary 2.6. *In every $A[s, d]$ there are at most K primitive images.*

The preceding lemma bounds only the number of primitive breakpoints. The minimization breakpoints are harder to analyze, and they occupy much of the rest of the paper. An important aid in their analysis is the idea of a layered graph, which we now describe.

2.2 A Layered Graph Representation

Our proofs use an acyclic *layered graph* representation of arbitrary graphs, which increases the number of vertices from n to $O(n^2)$ but has the advantage of simpler exposition. (Because our bounds are superpolynomial, the increase from n to n^2 does not affect them asymptotically.) Without loss of generality, suppose that the vertices of G are indexed from 1 to n, with $s = v_1$ and $d = v_n$. Figure 2 shows an example. We create n layers numbered from 1 to n, with each vertex $v_i \notin \{s, d\}$ replicated in each of the middle $n - 2$ layers. Layer 1 consists of the single vertex s and layer n consists of the vertex d. All edges of the layered graph connect consecutive layers and are directed from the lower indexed layer to the higher indexed one. Vertex s is joined to all vertices of layer 2 that correspond to neighbors of s in G; similarly, all vertices of G that are neighbors of d have edges to d from their copies in layer $n - 1$. For each edge $(v_i, v_j) \in G$ and each pair of adjacent layers, we put edges between the copies of v_i and v_j. Each of these edges has the same arrival time function as the original edge in G. Finally, we add edges from v_i to v_i between all adjacent layers—these edges have the arrival time function $A[v_i^j, v_i^{j+1}](t) = t$ (namely, zero delay edges), where v_i^j is the copy of v_i in layer j. It is easy to see that any shortest path from s to d in the original graph maps to a set of equivalent shortest paths in the layered graph, except the latter paths are *acyclic* and every s–d path in the layered graph has precisely $n-1$ edges. (The transformation of the parametric shortest path problem to a layered graph setting is analogous, with the cost of the edges (v_i^j, v_i^{j+1}) set to zero.)

There are two reasons why this layered graph representation is convenient. First, for both time-

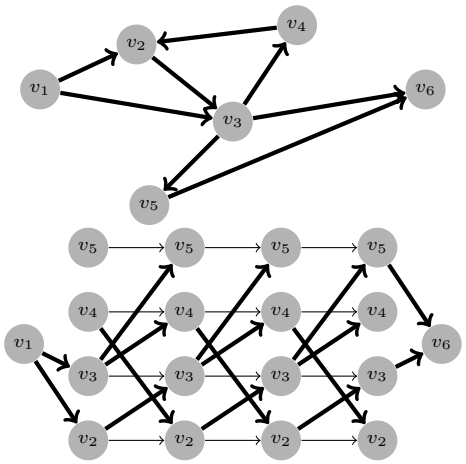

Figure 2: The layered graph transformation (on the bottom) of the graph on the top. The layered graph never contains cycles, even if the original graph does.

dependent and parametric shortest paths, the analyses focus on *linear* cost functions as an important case. Despite its apparent simplicity, the linear case is both a bit artificial and tricky to frame correctly. For example, in the case of time-dependent shortest paths, every linear function $A[e]$ with slope not equal to 1 must intersect the line $A = t$, meaning that $A[e](t) - t < 0$ for either $t \to \infty$ or $t \to -\infty$. That is, the delay is negative at some point, violating one of the key assumptions. Furthermore, a graph with negative edge costs may also contain negative cost cycles, making the function $A[s, d]$ meaningless for some values of t. The latter problem also arises in the parametric shortest path problem. The conversion to the layered graph avoids this problem. Second, in the parametric shortest path problem the addition of a constant weight to all edges between two adjacent layers preserves the relative order of all s–d paths, because each such path uses precisely one edge between two adjacent layers. This will be an important property for our lower bound construction.

3 Lower Bound

In this section we prove a superpolynomial lower bound on the number of breakpoints in the arrival time function $A[s, d]$. Our construction uses only linear cost functions $A[e]$ for all the edges, and our analysis is based on a lower bound for parametric shortest paths.

We remind the reader that, in the parametric shortest path problem, edges of a directed graph G have weights that vary linearly with a parameter γ: for $e_i \in E$, $c(e_i) \equiv c_i(\gamma) = a_i\gamma + b_i$. The length of the shortest path from s to d, which we denote by $L[s, d]$ to parallel our notation for time-dependent paths, is a function of γ. Specifically, $L[s, d] = \min(L[p] \mid p \in \mathcal{P}_{s,d})$, the pointwise minimum of $|\mathcal{P}_{s,d}|$ linear functions $L[p]$. Let $B(L[s, d])$ denote the number of breakpoints in this pointwise minimum $L[s, d]$, which is the number of times the shortest path from s to d changes over the domain of γ. The following result is due to Carstensen [4] and Mulmuley and Shah [23].

THEOREM 3.1. ([4, 23]) *There exists a graph G with n nodes, linear parametric edge weights, and two nodes s and d such that the number of breakpoints of $L[s, d]$ is $B(L[s, d]) = n^{\Omega(\log n)}$.*

The main idea in our lower bound is to transform an instance of the parametric shortest path problem into an instance of time-dependent shortest paths, by treating the parameter γ as the time parameter t. A key difference between the two models is that while the cost of a parametric shortest path is simply the sum of its edge costs for a fixed value of γ, a time-dependent shortest path involves a varying (monotonically growing) time parameter t. In order to map a fixed γ into a range of time parameters without distorting the breakpoints too much, we scale the edge delays down so that γ $(= t)$ does not change too much during the time it takes to travel from s to d. In order to calculate the scaling correctly, we need all the edge weights to be nonnegative, but, as noted earlier, linear parametric weights are negative for some values of γ. Therefore, given a worst-case graph G from Theorem 3.1, we first convert it into a layered graph as described in the previous section, and then transform that into a time-dependent shortest path instance.

The Reduction

Let the set of breakpoints in the worst-case parametric shortest path instance correspond to the parameter values $\gamma_1 < \gamma_2 < \cdots < \gamma_N$, where $N = B(L[s, d])$. We compute the minimum edge weight in the interval $[\gamma_1, \gamma_N]$, namely $c_{\min} = \min_i \min(c_i(\gamma_1), c_i(\gamma_N))$, and then modify the weight of every edge by adding $W = \max(0, -c_{\min})$ to it. (Recall that in the layered graph, edges connect vertices of adjacent layers only.) Every s–d path p in the original graph with at most $n - 1$ edges and length $L[p]$ corresponds one-to-many with a set of equivalent s–d paths in the layered graph, each with exactly $n - 1$ edges and length $L[p] + (n - 1)W$. Breakpoints of the $L[s, d]$ function are identical in the original and layered graphs. In the rest of this section we assume that the parametric costs $c_i(\gamma)$ have been translated up by W and the graph is layered, so that

$c_i(\gamma) \geq 0$ for $\gamma \in [\gamma_1, \gamma_N]$ and $L[s,d]$ is the path length in the layered graph.

Let γ_j and γ_{j+1} be two successive breakpoints of $L[s,d](\gamma)$. Let p_j be the shortest path from s to d during the interval $[\gamma_j, \gamma_{j+1}]$. (Our construction works even if p_j is not unique, i.e., multiple paths have the same length function as p_j, but for simplicity assume p_j is unique.) Because the length of any path in G is a linear function of γ, it must be the case that for $\bar{\gamma}_j = (\gamma_j + \gamma_{j+1})/2$, every path in G not equal to p_j is strictly longer than p_j. Define $\bar{L}_j = L[s,d](\bar{\gamma}_j)$ and let $\bar{L}'_j$ be the length of the second-shortest path at $\gamma = \bar{\gamma}_j$. Define $\Delta_j = \bar{L}'_j - \bar{L}_j$, and $\Delta_{\min} = \min_j \Delta_j$.

The total number of edges in any s–d path is exactly $n-1$ in the layered graph. Therefore, if we arbitrarily perturb the weights of edges at $\gamma = \bar{\gamma}_j$ by at most $\Delta_{\min}/(2n) \leq \Delta_j/(2n)$, the path p_j will continue to be the shortest s–d path.

We now compute an ϵ-interval around $\bar{\gamma}_j$ such that variation of γ within the interval keeps the weight perturbation small. Every cost function $c_i(\gamma) = a_i\gamma + b_i$ has finite slope a_i. If $a_i \neq 0$, we can choose a parameter $\epsilon_i \equiv \Delta_{\min}/(2n|a_i|)$ so that $|c_i(\bar{\gamma}_j) - c_i(\bar{\gamma}_j + \epsilon)| < \Delta_{\min}/(2n)$ for any ϵ with $|\epsilon| < \epsilon_i$. If $a_i = 0$, $|c_i(\bar{\gamma}_j) - c_i(\bar{\gamma}_j + \epsilon)| = 0 < \Delta_{\min}/(2n)$ for every ϵ, so ϵ_i can be set to infinity. Define $\epsilon_{\min} = \min_i \epsilon_i$. This is a constant that depends only on G and the cost functions $c_i(\gamma)$.

Observation 3.2. *Given a directed graph with real edge weights, multiplying all edge weights by a constant κ does not change the combinatorial type of any point-to-point shortest path.*

Lemma 3.3. *Given a directed graph G with parametric edge weights, multiplying every edge weight function by a constant κ does not change the values of ϵ_i and $\epsilon_{\min}$ defined above.*

Proof: If edge weights are multiplied by κ, Δ_j and $\Delta_{\min}$ are also multiplied by κ. The constant that multiplies γ in the weight function $c_i(\gamma)$ changes from a_i to $\kappa \cdot a_i$. The value of ϵ_i is a ratio with $\Delta_{\min}$ in the numerator and a_i in the denominator, so the two κ's cancel, leaving ϵ_i unchanged. It follows that $\epsilon_{\min}$ is also unchanged. □

Let $L_{\max}$ be the maximum value of $L[s,d](\gamma)$ over the finite interval of γ bounded by breakpoints, i.e., $\gamma \in [\gamma_1, \gamma_N]$. We set the constant $\kappa = \frac{1}{2}\epsilon_{\min}/(L_{\max} + \Delta_{\min})$. By Observation 3.2, if we multiply each input cost function by κ, the resulting parametric shortest path problem has the same breakpoints as the original one, and by Lemma 3.3 the values of ϵ_i and $\epsilon_{\min}$ are unchanged.

We convert the parametric shortest path problem to a time-dependent one by setting the time-dependent delay function $D[e_i] \equiv D_i(t) = \kappa \cdot c_i(t)$, that is, we scale the costs by κ and equate the time t with the γ parameter. (Recall that $A[e_i] = D[e_i] + t$.)

The Analysis

We now show that the shortest s–d path that leaves from s at time $t = \bar{\gamma}_j$ is equal to p_j. We first compute the arrival time $A[s,k](\bar{\gamma}_j)$ for each node k on the path p_j. Without loss of generality assume that the nodes of p_j are numbered sequentially from $s = 1$ to $d = n$. (Equivalently, the nodes are numbered by layers in the layered graph.) To relate the time-dependent shortest path to the parametric shortest path, we introduce the shorthand $\sigma_k = \sum_{i<k} D_i(\bar{\gamma}_j) = \kappa \sum_{i<k} c_i(\bar{\gamma}_j)$, which is κ times the length of the $(k-1)$-edge prefix of the parametric shortest path for $\gamma = \bar{\gamma}_j$.

Lemma 3.4. *The arrival time at the k^{th} node of the path p_j, $A[s,k](\bar{\gamma}_j)$, is at most $\bar{\gamma}_j + \epsilon_{\min}\frac{k-1}{2n} + \sigma_k$.*

Proof: The proof is by induction. The claim is trivial for $k = s = 1$. Assuming that the claim holds for some $k \geq 1$, we prove it for $k+1$. Because $D_k(t)$ is a linear function, the delay for the edge $e_k = (k, k+1)$ is between $D_k(\bar{\gamma}_j)$ and $D_k(\bar{\gamma}_j + \epsilon_{\min}\frac{k-1}{2n} + \sigma_k)$. Note that $0 \leq \sigma_k \leq \sigma_d \leq \kappa \cdot L_{\max} < \epsilon_{\min}/2$. Because p_j has $n-1$ edges, $k < n$ and hence $\epsilon_{\min}\frac{k-1}{2n} < \epsilon_{\min}/2$. This means that the delay for e_k is between $D_k(\bar{\gamma}_j)$ and $D_k(\bar{\gamma}_j + \epsilon_{\min})$, i.e., the delay differs from $D_k(\bar{\gamma}_j) = \kappa \cdot c_k(\bar{\gamma}_j)$ by at most $\kappa \cdot \Delta_{\min}/(2n) < \epsilon_{\min}/(4n)$. For convenience define $A_k \equiv A[s,k](\bar{\gamma}_j)$. Because $D_k(t)$ is nonnegative for all $\gamma_1 < t < \gamma_N$, $A_k \leq A_{k+1}$ for all $1 \leq k < n$. Then the arrival time at node $k+1$ is

$$\begin{aligned} A_{k+1} &= A_k + D_k(A_k) \\ &\leq \bar{\gamma}_j + \epsilon_{\min}\frac{k-1}{2n} + \sigma_k + D_k(\bar{\gamma}_j) + \frac{1}{4n}\epsilon_{\min} \\ &\leq \bar{\gamma}_j + \frac{k}{2n}\epsilon_{\min} + \sigma_{k+1}. \end{aligned}$$

□

It follows from the lemma that the arrival time at d is $A[s,d](\bar{\gamma}_j) \leq \bar{\gamma}_j + \epsilon_{\min}/2 + \sigma_d$. As noted in the proof, $\sigma_d < \epsilon_{\min}/2$, and so $A[s,d](\bar{\gamma}_j) < \bar{\gamma}_j + \epsilon_{\min}$. The cost (delay function) of each edge in G varies by at most $\kappa \cdot \Delta_{\min}/(2n)$ during the interval $t \in [\bar{\gamma}_j, \bar{\gamma}_j + \epsilon_{\min})$, which is the time interval needed to traverse p_j starting at $t = \bar{\gamma}_j$. This means that, as argued above in the context of perturbing parametric shortest paths, p_j is the shortest s–d path for starting time $t = \bar{\gamma}_j$. This is true for each j between 1 and $N-1$, and so there are at

least $N-2$ combinatorial changes to the s–d shortest path in the time-dependent graph between $t=\gamma_1$ and $t=\gamma_N$. We have established the following theorem:

THEOREM 3.5. *There exists a graph G with n nodes and linear edge arrival functions $A[e]$ that contains a pair of nodes s,d such that $B(A[s,d]) = n^{\Omega(\log n)}$.*

4 Upper Bound

We now show that the number of minimization breakpoints in any arrival time function in an n-node graph is $K \times n^{O(\log n)}$, where K is the total number of linear pieces among all the edges of G. A key to the proof is the following crucial observation.

LEMMA 4.1. *Between any two adjacent primitive images, $A[s,d](t)$ forms a concave chain.*

Proof: If a minimization breakpoint of $A[s,d]$ occurs at some t_0 where there is no primitive image, then $A[s,d] = A[p_1]$ for $t = t_0 - \epsilon$ and $A[s,d] = A[p_2]$ for $t = t_0+\epsilon$, where $\epsilon > 0$ is infinitesimal and $p_1, p_2 \in \mathcal{P}_{s,d}$. That is, $A[s,d] = A[p_1 \cup p_2]$ in the neighborhood $|t - t_0| \le \epsilon$. Both $A[p_1]$ and $A[p_2]$ are linear in that neighborhood, because neither contains a primitive image, and the two intersect at $t = t_0$. The minimum of two linear functions forms a concave corner at their intersection. Since every corner of $A[s,d]$ between two consecutive primitive images is concave, the lemma is proved. □

A crude estimate that follows from Lemma 4.1 is that there can be as many minimization breakpoints between two adjacent primitive images as the number of different slopes in all possible $A[p]$, for $p \in \mathcal{P}_{s,d}$. Unfortunately, this observation does not yield useful upper bounds, since the number of different slopes among all the possible paths from s to d can be of the same order as the number of paths from s to d. As an example, consider the series-parallel graph shown in Figure 3 in which each of the $\Theta(n)$ edges e_i can be chosen independently to belong to an s–d path. If $A[e_i]$ is linear with slope equal to the i^{th} prime number, and all other edges e have $A[e](t) = t$, then the number of distinct slopes in all $A[p]$, for $p \in \mathcal{P}_{s,d}$, is $2^{\Theta(n)}$.

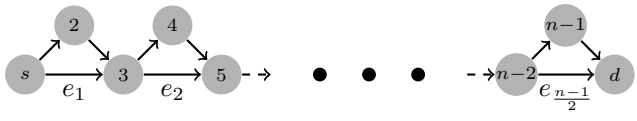

Figure 3: Each e_i can be chosen independently for membership in a s–d path.

To focus on the complexity of $A[s,d]$ between primitive images, we consider the case in which $A[s,d]$ contains no primitive images at all, namely when each $A[e]$ is linear. As noted in Section 2.2, in this case we have to convert G to a layered graph to avoid negative-time cycles. Let us define $A_{\text{lin}}[s,d]$ to be the minimum arrival time function over all paths $p \in \mathcal{P}_{s,d}$, assuming that every $A[e]$ is linear. Then $B(A_{\text{lin}}[s,d])$ is the maximum number of minimization breakpoints in the s–d arrival time function for linear edge functions in the layered graph.

LEMMA 4.2. *The number of breakpoints in the arrival time function for s–d paths is at most K times the maximum number for the same function assuming linear edge functions:*

$$B(A[s,d]) \quad \le \quad K \times B(A_{\text{lin}}[s,d]).$$

Proof: Consider an edge $e = (u,v) \in E$. If $e \notin p$ for any $p \in \mathcal{P}_{s,d}$, we can delete it from E: its arrival time function $A[e]$ cannot contribute to $A[s,d]$. Otherwise, consider the restricted path set $\mathcal{P}_{s,d\,|\,e}$, which is the subset of paths in $\mathcal{P}_{s,d}$ that contain edge e, and the corresponding function $A[s,d\,|\,e] = A[v,d]\cdot A[e]\cdot A[s,u]$. If b is a primitive breakpoint of $A[e]$, let b' be its image in $A[s,d\,|\,e]$, that is, $b = A[s,u](b')$. (It is possible that discontinuities may hide a breakpoint, i.e., if $\lim_{t\uparrow b'} A[s,u](t) < b$ and $\lim_{t\downarrow b'} A[s,u](t) > b$, but this does not affect the breakpoint location in $A[s,d]$.)

Let PI be the union, over all edges e, of the images in $A[s,d\,|\,e]$ of the primitive breakpoints of $A[e]$. Then PI is a set of values of t, in the frame of reference of s, the start of all s–d paths. Let us break the function $A[s,d]$ at all $t \in PI$. Some of these values of t are primitive images in $A[s,d]$ and some are not, but all primitive images in $A[s,d]$ belong to PI. Therefore the function $A[s,d]$ is concave between two consecutive elements of PI, by Lemma 4.1. Between any two consecutive $b, b' \in PI$, a fixed set of paths $P \subseteq \mathcal{P}_{s,d}$ contributes to $A[s,d]$, i.e., $A[s,d] = \min(A[p] \mid p \in P)$ between b and b'. Let $E_P \subseteq E$ be the set of all edges that belong to paths in P. No edge $e = (u,v) \in E_P$ has a primitive breakpoint in $A[e]$ during the interval $(A[s,u](b), A[s,u](b'))$, i.e., (b,b') mapped to u's frame of reference. Thus we can replace the piecewise linear function $A[e]$ by the linear function that is active during the interval $(A[s,u](b), A[s,u](b'))$ without affecting the value of $A[s,d]$ during the interval $t \in (b,b')$. Likewise we can set $A[e] = \infty$ for all $e \notin E_P$ without affecting the value of $A[s,d]$ during $t \in (b,b')$. This gives a set of linear edge functions such that $A_{\text{lin}}[s,d]$ is equal to the original $A[s,d]$ during $t \in (b,b')$. But $|PI| \le K$, and so $B(A[s,d]) \le K \times B(A_{\text{lin}}[s,d])$. □

To bound $B(A_{\text{lin}}[s,d])$, the number of breakpoints in $A[s,d]$ if all $A[e]$ are linear, we adapt an argument due to Gusfield [17].

The proof uses the layered graph representation to bound the complexity $B(A[s,d])$ of the arrival time function. In particular, let us consider a generic layered graph $G_{n,c}$, which is a graph on $nc+2$ nodes consisting of two distinguished nodes s and d and n columns of c nodes each—see Figure 4 for illustration. Each edge of $G_{n,c}$ either joins s to a vertex in column 1, or joins a vertex in column i to a vertex in column $i+1$, for $1 \le i \le n$, or joins some vertex in column n to d. We prove the following lemma.

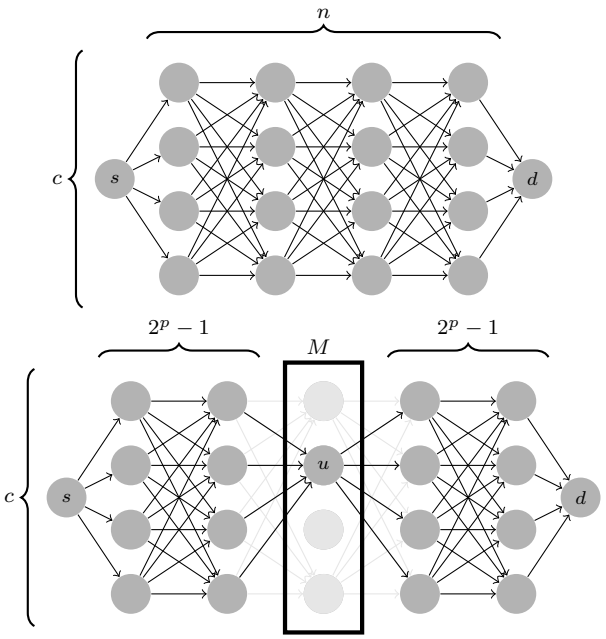

Figure 4: $G_{n,c}$ (top) and two copies of $G_{2^p-1,c}$ concatenated through the vertex u with the middle column M highlighted (bottom).

LEMMA 4.3. *In the graph $G_{n,c}$ with linear edge cost functions, the complexity of the arrival time function is $B(A[s,d]) \le \frac{(2n+1)^{(1+\log c)}}{2}$*

Proof: We first present the proof assuming that $n = 2^p - 1$, for an integer p, then extend it to handle other values of n. For purposes of this proof, we use base-2 logarithms: $\log \equiv \log_2$. The proof is by induction on p. We will show that $B(A[s,d]) \le \frac{(n+1)}{2}(n+1)^{\log c} = \frac{(n+1)}{2}c^{\log(n+1)}$. In the following, we count the number of segments in the arrival time function, but denote it by $B()$, the familiar breakpoint notation, since the two counts differ by only one for a piecewise linear function.

The base of the induction has $p = n = 1$, and in that case we clearly have $B(A[s,d]) \le \frac{(n+1)}{2}c^{\log(n+1)} = c$ because there are only c paths from s to d, each represented by at most one linear piece in the arrival function. Assuming that the claim holds for $n = 2^p - 1$, we now prove it for $n = 2^{p+1} - 1$. Consider $G_{n,c}$ for $n = 2^{p+1} - 1$, let M be the middle column of $G_{n,c}$, and let u be an arbitrary node in column M; see Figure 4.

We first bound the number of s–d shortest paths in $G_{n,c}$ that pass through the vertex u. This corresponds to the number of pieces in the arrival function $A[s,d \,|\, u]$, the restriction to the s–d paths passing through the node u. Since $A[s,d \,|\, u] = A[u,d] \cdot A[s,u]$, we have by Lemma 2.4 that $B(A[s,d \,|\, u]) \le B(A[u,d]) + B(A[s,u])$. By the induction hypothesis, $B(A[s,u]) \le \frac{2^p}{2}c^{\log 2^p}$, since $A[s,u]$ is the arrival function for the graph $G_{2^p-1,c}$. The same holds for $B(A[u,t])$. Therefore, we have

$$B(A[s,d \,|\, u]) \;\le\; 2\frac{2^p}{2}c^{\log 2^p} \;=\; 2^p c^{\log 2^p}$$

Since there are precisely c nodes in column M, the shortest path from s to t has c choices of the vertex u. Therefore, in the graph $G_{n,c}$, the complexity of $B(A[s,d])$ is bounded as

$$\begin{aligned} B(A[s,d]) &\le cB(A[s,d \,|\, u]) \\ &\le 2c\frac{2^p}{2}c^{\log 2^p} \\ &= \frac{n+1}{2}c^{\log(n+1)} \\ &= \frac{n+1}{2}(n+1)^{\log c}. \end{aligned}$$

This completes the proof of the lemma. When n does not have the form of $2^p - 1$, we can scale n to the next power of two, and obtain easily that $B(A[s,d]) \le \frac{(2n+1)^{1+\log c}}{2}$. □

In order to complete the proof of the theorem, we construct the layered graph $G_{n-1,n-1}$ from G, as described in Section 2.2. By the path-equivalence of this layered graph and the original graph G, we conclude that $B(A[s,d])$, the number of pieces in the arrival time function, is upper bounded by $\frac{(2n+1)^{(1+\log n)}}{2}$ in both graphs. This completes the proof of the upper bound for linear functions.

THEOREM 4.4. *For any graph G with n nodes and linear arrival time functions $A[e]$ at all edges $e \in E$, $B(A[s,d]) = n^{O(\log n)}$, for any pair of nodes s, d.*

Combining Lemma 4.2 with Theorem 4.4, we obtain the following upper bound:

THEOREM 4.5. *For any graph G with n nodes, if the arrival time functions $A[e]$ at all edges $e \in E$ are piecewise linear with a total of K segments over all e, then $B(A[s,d]) = K \times n^{O(\log n)}$, for any pair of nodes s, d. If K is polynomial in n, then $B(A[s,d]) = n^{O(\log n)}$.*

5 Algorithmic Results

The lower bound of Section 3 can be disheartening to anyone interested in computing the arrival time functions in a time-dependent graph. While one may hope that practical graphs are unlikely to exhibit such complexity, investigating the source of the complexity remains a challenging problem. In order to better understand the scope of our lower bound, we can investigate important practical subclasses of graphs or design approximation schemes with provable bounds. We study four questions along these directions. First, we show that the complexity of the arrival time function is much more well-behaved if we limit the *slopes* of the edge functions. Second, we present an *output-sensitive* algorithm for constructing the arrival time function, so that the running time of the algorithm depends not on the worst-case size, but on the actual instance size of the arrival function. Third, we present an approximation scheme that constructs a small-size approximation of the travel time function. Finally, we show that the minimum delay over a departure time query interval can be computed in polynomial time. This is somewhat surprising, since the delay function over that time interval may have superpolynomial complexity. The following four subsections present details of these algorithmic results.

5.1 Restricted Slopes

One of the simplest cases of time-dependent shortest paths arises when the slopes of the edge arrival time functions are restricted to 0, 1 and ∞: slope 1 implies constant speed travel, slope ∞ models a temporary complete stop of traffic flow (e.g., due to sudden congestion or a traffic signal), and slope 0 models release of the flow as the congestion clears. With these limited edge functions, the arrival function $A[s,d]$ does not suffer the superpolynomial complexity of the general case. In fact, the complexity of $A[s,d]$ is only $O(K)$. More generally, we show that the complexity is low (in fact, linear) if the edge functions have slopes in the set $\{0, \alpha^{-\beta}, \alpha^{-\beta+1}, \ldots, \alpha^{\beta-1}, \alpha^{\beta}\}$, for some $\alpha > 1$ and $\beta \in \mathbb{N}$. That is, each edge has a piecewise linear cost function and each piece has the form $A(t) = at + b$, with slope $a \in \{0, \alpha^{-\beta}, \alpha^{-\beta+1}, \ldots, \alpha^{\beta}\}$. (The special slope ∞ may also be included without altering the complexity.) We call such a graph an (α, β)-*slope graph.*

The proof that the complexity of the arrival time function in (α, β)-slope graphs is polynomially bounded follows from the observation that the slope of a path arrival function along a path $p \in \mathcal{P}_{s,d}$ is the product of the slopes of the edge arrival functions $A[e]$ on the edges composing p. Since a path has at most $n-1$ edges, there can be at most $2(\beta+1)(n-1)$ possible slopes appearing in any path arrival function. Between any two primitive images, $A[s,d]$ forms a concave chain (Lemma 4.1), and hence no two segments with the same slope can appear in it. Therefore, in an (α, β)-slope graph, any $A[s,d]$ has at most $2(\beta+1)nK$ breakpoints.

5.2 Output-sensitive Construction

The lower and upper bounds of Sections 3 and 4 show that the complexity of $A[s,d]$ may be as bad as $n^{\Theta(\log n)}$. In practice, however, the complexity may be much smaller, and so an algorithm that computes $A[s,d]$ in time proportional to the actual number of shortest path changes is desirable. In this section we present an *output-sensitive* algorithm whose running time depends on the number of breakpoints that occur at the nodes and edges of G for a given set of piecewise linear edge arrival time functions. The algorithm computes $A[s,v]$ for every node $v \in V$, including d.

The algorithm is inspired by kinetic data structures [1, 16]. In brief, the idea is to run Dijkstra's algorithm for a particular departure time t and build a set of *certificates* (predicates that are linear functions of t) that guarantee the correctness of the shortest path tree that Dijkstra's algorithm computes. We then vary the departure time t while maintaining the shortest path tree and a set of certificates that guarantee its correctness. This approach produces all the combinatorially distinct shortest path trees that exist for all values of t, producing each one as a modification of a similar, temporally adjacent neighbor.

Certificates are of two types, *primitive certificates* and *minimization certificates*. Primitive certificates ensure that the linear function corresponding to a single path remains unchanged until the failure time of the certificate; the failure of a primitive certificate corresponds to a primitive breakpoint at some edge. Minimization certificates ensure that a certain path reaches a given node first; failure of a minimization certificate corresponds to a minimization breakpoint in the graph G. Each certificate has a failure time t_{fail} that is the earliest departure time after the current time t for which the certificate fails.

The certificates are stored in a priority queue, with the one with the earliest failure time at the head. When a certificate fails, at least one node or edge in G has a breakpoint, the combinatorial shortest path from s to one or more nodes may change, and some number of certificates will need to be updated. (Real kinetic data structures spend considerable effort to make sure that few certificates need to be updated in response to any event; the structure described here does not.)

For every node $v \in V$, we create a fixed binary tree whose leaves are the incoming edges (u, v). This tree represents a tournament among the incoming edges to determine which edge carries the path that first reaches v from s for any given departure time. For simplicity, we may modify G by expanding each node v with in-degree d into a binary tree with $d - 1$ nodes, height $\lceil \log_2 d \rceil$, and all internal edges with zero delay. This gives a graph G' with $\Theta(m)$ nodes, each with in-degree at most two.

We run Dijkstra's algorithm on G', starting from node s at some time t_0. Dijkstra computes a linear function at each node v that represents $A[s, v]$ for the time interval containing t_0. Suppose that v has two predecessors u_1 and u_2, and the shortest path tree for departure time t_0 contains $e_1 = (u_1, v)$ but not $e_2 = (u_2, v)$. The correctness of Dijkstra's algorithm means that $A[s, v \mid e_1](t_0) < A[s, v \mid e_2](t_0)$, that is, the shortest path leaving s at time t_0 and reaching v via edge e_1 arrives before the shortest path with the same departure time and reaching v via e_2. We turn this into a minimization certificate by computing the time $t_{\min} > t_0$ (if any) such that the linear function $A[s, v \mid e_1]$ becomes equal to $A[s, v \mid e_2]$; the failure time of the certificate is $t_{\min}$, or ∞ if the two linear functions do not intersect after t_0. A primitive certificate for each edge $e = (u, v)$ is computed by using the inverse of the linear function $A[s, u]$ to project the next primitive breakpoint of $A[e]$ (if any) into the time domain of s; the resulting time t_e is the failure time of the certificate.

The primitive and minimization certificates of the nodes and edges of G' are placed in a priority queue ordered by the next certificate failure time. When the first certificate fails at some time t_{fail}, the linear function $A[s, v]$ changes for some set of nodes v. If a minimization certificate fails at v (the shortest path using (u_1, v) becomes more expensive than the one using (u_2, v)), then the shortest path tree changes at v. At the same time the functional form of $A[s, v]$ changes, along with $A[s, w]$ for all shortest-path-tree descendants w of v in G'. Likewise, if the primitive certificate for an edge (u, v) in the shortest path tree changes, the functional form of $A[s, *]$ for v and all its shortest-path-tree descendants changes. If the primitive certificate for an edge (u, v) *not* in the shortest path tree changes, then the minimization certificate at v needs to have its failure time updated, but no other changes are required. In all cases, if the functional form of $A[s, v \mid e]$ changes for any edge $e = (u, v)$, then the minimization certificate at v needs to have its failure time updated.

When the linear function $A[s, v]$ changes for a node v but not for its predecessor u in the shortest path tree, then we need to update certificates for all shortest-path-tree descendants of v. We traverse the subtree rooted at v, updating the functional form of $A[s, w]$ for each descendant w. We recompute the failure times of the primitive certificates for edges reachable from v and the minimization certificates at nodes reachable from v and at the heads of non-tree edges reachable from v. We update the priority queue of certificates to account for new failure times. At the end of this procedure the shortest path tree is correct for paths leaving s at time $t = t_{\text{fail}} + \epsilon$ for an infinitesimal ϵ, the functional form of $A[s, v]$ is correct for each node v and departure time $t = t_{\text{fail}} + \epsilon$, and the priority queue identifies the next time greater than t_{fail} at which a change to one or more arrival time functions will occur.

The total number of certificates is $O(m)$—at most one per node and one per edge. The time to respond to the failure of a certificate c is $O(|T_c| \log n)$, where T_c is the set of edges $e = (u, v)$ of G' whose arrival time functions $A[s, v \mid e]$ change due to the failure of c, and the logarithmic factor comes from priority queue operations. The edge set T_c consists of the shortest path subtree in G' rooted at the node or edge where c occurs, augmented with the non-tree edges directed out of that subtree. In the worst case $|T_c|$ may be $\Theta(m)$, but in cases when it is smaller, the output-sensitive algorithm exploits that fact. Expressed in terms of the original graph G, $|T_c|$ is equal to the number of outgoing edges reachable from the certificate node/edge's subtree, times a factor of at most $O(\log n)$ corresponding to the height of the tournament trees in which those edges participate in G'.

The worst-case running time of the algorithm is $O(\log^2 n)$ times the total number of breakpoints of $A[s, v \mid e]$ over all edges $e = (u, v) \in G$. One of the two logarithmic factors is unavoidable priority queue overhead, but the other is data-dependent; randomized construction of the tournament trees might reduce it to expected constant.

Note: During the preparation of this paper we learned that Dean [8] had also proposed an output-sensitive algorithm. However, that algorithm appears to be incomplete—the propagation of arrival time function changes through the shortest path tree is missing—

and the analysis seems inconsistent with the presented algorithm. Even if the missing details are included, that algorithm still may be up to a factor of $\Theta(n/\log n)$ slower than the one presented here, because it uses d-way minimization at nodes of in-degree d, instead of the binary tournaments built into our graph G'.

We also point out a flaw in the algorithm of Ding, Yu, and Qin [12], which is claimed to compute $A[s,d]$ in time $O(|A[s,d]|(n\log n + m))$. If the algorithm is executed on the simple graph shown Figure 5 in which all edges have constant delay, using a time query interval $[0, 2N]$, the running time is $\Theta(N)$. The flaw is in the proof of Lemma 5.2 in [12], in which the cost of assembling the function $A[s,i]$ out of l_i disjoint fragments is assumed to be $O(|A[s,i]|)$ instead of $O(|A[s,i]| + l_i)$. A recent paper by Dehne, Omran, and Sack [9], which we discovered while preparing the camera-ready version of this paper, also points out this flaw.

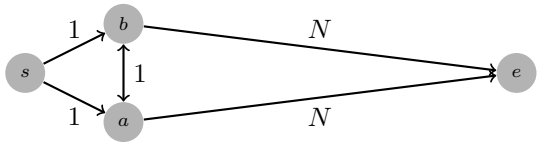

Figure 5: A simple graph with constant edge delays (shown) for which the algorithm of [12] runs for $\Theta(N)$ time, even though n, m, and $|A[s,i]|$ for all nodes i are all $O(1)$.

5.3 $(1+\epsilon)$-Approximation of Travel Time

Although the worst-case complexity of $A[s,d]$ is $n^{\Theta(\log n)}$, in practice one may be satisfied with a lower-complexity approximation of the arrival time function. Let $D(t) = D[s,d](t) = A[s,d](t) - t$ be the *delay function* for the (s,d) node pair. A function $D'(t)$ is a $(1+\epsilon)$-approximation of $D(t)$ if $|D(t) - D'(t)| \le \epsilon \cdot D(t)$.

One direct approach to finding a $(1+\epsilon)$-approximation for $D(t)$ is to build on the idea of Section 5.1, mapping the problem to an (α,β)-slope graph, with an appropriate choice of *slope buckets*. However, this does not seem to work—we end up with an exponential term, either in the number of buckets or the relative arrival time error.

A simple but effective way to approximate $D(t)$ is to consider its functional form. Between any two consecutive primitive breakpoints, the function is concave. Since the number of primitive breakpoints is only $O(K)$, we can reduce the problem of approximating $D(t)$ to $O(K)$ approximation problems for concave chains. While we do not have access to the concave chains explicitly, we can *query* them efficiently: in particular, given a departure time t, we can use Dijkstra's algorithm to find its image in $A[s,d]$, and similarly, given any arrival time t at d, we can run a *backward* Dijkstra to find the corresponding departure time.

The approximate representation

The first step of the approximation is to break $A[s,d]$ (and hence $D[s,d]$) at all primitive images. Instead of trying to identify exactly those primitive breakpoints with images in $A[s,d]$, we simply break $A[s,d]$ at the images of all primitive breakpoints, even those that do not propagate into $A[s,d]$. For each edge $e = (u,v)$ with a breakpoint at time b in u's frame of reference, we project b back to the temporal frame of reference of the source s. To do this we run Dijkstra's algorithm on the graph G^r obtained by reversing all edges of G and inverting the arrival time function for each edge.[2] By running Dijkstra on the reversed graph, we find the time b' in s's frame of reference at which one has to depart in order to hit the breakpoint b on edge e. This is the primitive image in $A[s,d]$, if any, that is the image of b. The union of all these breakpoint images, denoted PI, is a superset of the primitive images of $A[s,d]$; between any two consecutive times in PI, $A[s,d]$ is concave.

For each pair of consecutive breakpoints $b, b' \in PI$, we use (forward) Dijkstra to find the corresponding values of $A[s,d]$ (and hence the values $D(b)$ and $D(b')$). The function $D(t)$ is concave between b and b'; that is, the point $(t_2, D(t_2))$ lies on or above the line segment from $(t_1, D(t_1))$ to $(t_3, D(t_3))$. The minimum value of $D(t)$ in $[b,b']$ is $D_{\min} \equiv \min(D(b), D(b'))$, and the maximum value $D_{\max}$ is at most $2D(\bar{b})$, where $\bar{b} = (b+b')/2$.

A conceptually simple approximation for $D(t)$ slices the function horizontally: find the intersections of $D(t)$ with the lines $y = (1+\epsilon)^k D_{\min}$, for each $k \ge 0$ such that $(1+\epsilon)^k D_{\min} \le D_{\max}$. Let the resulting sample points be $(t_i, D(t_i))$. Because $D(t)$ is concave in $[b,b']$, each slice generates at most two sample points, so the total number of samples is at most $2\log(D_{\max}/D_{\min})/\log(1+\epsilon) \approx \frac{2}{\epsilon}\log(D_{\max}/D_{\min})$. Linearly interpolating between

[2] Discontinuities in $A[e]$ map to horizontal edges in $A[e^r]$ and vice versa. For example, if an edge e has a discontinuous function $A[e]$, i.e., $A[e](\tau^-) = x$ and $A[e](\tau^+) = y$ with $x < y$, then in $A[e^r]$ each time in the range $[x,y]$ maps to τ: when traveling from s, one must arrive at edge e at or before $t = \tau$ in order to reach the other end of e at or before any time in the range $[x,y]$.

consecutive sample points $(t_i, D(t_i))$ gives an approximation that is never greater than $D(t)$; the lower envelope of the supporting lines at all the sample points is a concave curve that is never less than $D(t)$. The worst-case error of either approximation is at most ϵ times the actual value of $D(t)$, giving a $(1+\epsilon)$-approximation, as desired.

LEMMA 5.1. *If the edge arrival time functions have a total of K linear pieces and the maximum value of $D_{\max}/D_{\min}$ over all the concave intervals of $D[s,d]$ is at most R, then there exists a data structure of size $O(K\frac{1}{\epsilon}\log R)$ that represents $D[s,d]$ with relative error ϵ.*

Computing the representation

Finally, we discuss the time complexity of computing the $D(t)$ samples needed by our procedure. It is not possible to perform the horizontal slicing of $D(t)$ directly, because we have access to $A[s,d](t)$, not to $D(t)$. However, we observe that we can obtain an adequate approximation without precisely locating the t_i such that $D(t_i) = (1+\epsilon)^i D_{\min}$ for every $i \le 2\log(D_{\max}/D_{\min})/\log(1+\epsilon)$. It suffices to find approximate sample points within a finer partition of the range. That is, we imagine slicing $D(t)$ into fragments with the horizontal lines $y = (1+\epsilon)^{k/2} D_{\min}$ (using twice as many values of k), and then find a sample point on each fragment. The values of $D(t)$ at consecutive sample points differ by at most a factor of $1+\epsilon$, so this sampling satisfies the criteria needed to prove Lemma 5.1.

We consider the problem of finding approximate samples for a single concave chain delimited by two consecutive primitive breakpoint images b and b'. The horizontal span of the chain is $L = b' - b$, and the vertical range lies between $D_{\min}$ and $D_{\max}$. Note that the slope of $D(t)$ is monotone decreasing inside the interval (b, b'), and by the FIFO assumption, the slope of $D(t)$ at $t = b'$ is at least -1.

We consider how to locate a target value by using two kinds of probes: *forward probes* run Dijkstra for a given t to determine $D(t)$; *reverse probes* run Dijkstra in reverse, starting from a value A and determining the time t such that $A[s,d](t) = A$. Note that reverse probes have a rather loose connection to $D(t)$, since the value of t is not known *a priori*.

We search for sample points on $D(t)$ using reverse probes on a prefix of $[b, b']$ and use forward probes on the remainder. Starting from $t = b$, so long as the slope of $D(t)$ is at least 1, we perform a reverse probe for $A = t + D(t)\cdot\sqrt{1+\epsilon}$, then set t to the resulting time value t'. Because $t' \ge t$, we have $D(t') = A(t') - t' \le A(t') - t = D(t)\cdot\sqrt{1+\epsilon}$; that is, $(t', D(t'))$ is in the current fragment or the next one. But if the slope of $D(t')$ is at least 1, then $A[s,d]$ has slope at least 2 between t and t', implying that $A = t + D(t)\sqrt{1+\epsilon} \ge t + D(t) + 2(t'-t)$. Thus $t' \le \frac{1}{2}D(t)(\sqrt{1+\epsilon}-1) + t$, which implies that $D(t') = t + D(t)\sqrt{1+\epsilon} - t' \ge \frac{1}{2}D(t)(\sqrt{1+\epsilon}+1)$. But $(\sqrt{1+\epsilon}+1)/2$ is a constant fraction of $\sqrt{1+\epsilon}$, so this procedure performs a constant number of reverse probes in each fragment.

Once the reverse probes produce an interval $[t, b']$ with initial slope at most 1, we use bisection with forward probes to find the rest of the sample points. Observe that if the slopes of $D(t)$ at the ends of an interval of length ℓ differ by Δ, then the maximum possible vertical error in the interval is bounded by $\ell\Delta$. Suppose that the current interval has length ℓ, slope difference Δ, and minimum $D(t)$ value $\overline{D}$ (obtained at an interval endpoint). If $\ell\Delta > \overline{D}(1+\epsilon)$, then we bisect the interval at its middle t value with a forward probe and recursively partition the two subintervals. This gives a worst-case approximation error of at most $1+\epsilon$ between consecutive samples.

We can bound the number of bisection probes by considering the binary recursion tree generated by the bisection process described. The leaves of the tree correspond to the final intervals; the internal nodes correspond to the bisections. Each bisection halves the current interval. After k bisections, the current subinterval has length $L/2^k$, and therefore since $\Delta \le 2$ the maximum error in this interval is at most $2L/2^k$. This implies that the height of the recursion tree is at most $\log(2L/(\epsilon D_{\min}))$. The internal nodes of maximal depth correspond to intervals with $\ell\Delta > \overline{D}(1+\epsilon)$, therefore in *any* $(1+\epsilon)$-approximation of the initial interval, at least one point must come from each such interval. The overhead cost of finding each approximation point using the bisection algorithm (Lemma 5.1 states that there are $O(\frac{1}{\epsilon}\log(D_{\max}/D_{\min}))$ points in the worst case) is then the number of bisections on a root-to-leaf path of the bisection tree. This is bounded by the height of the recursion tree. Overall, the total number of Dijkstra probes needed to compute the approximation of Lemma 5.1 is at most

$$O\left(\frac{1}{\epsilon}\log\left(\frac{D_{\max}}{D_{\min}}\right)\log\left(\frac{L}{\epsilon D_{\min}}\right)\right).$$

5.4 Minimum Delay Queries

One important application of time-dependent shortest paths is finding the minimum delay for a departure time in a given query window. Because the delay function is concave between primitive breakpoints,

the minimum occurs at those breakpoints or at the endpoints of the query interval. We can find the answer to minimum delay queries in polynomial time. If the query interval is $[t_1, t_2]$, we perform forward Dijkstra probes at t_1, t_2, and all primitive breakpoint images inside $[t_1, t_2]$. Those values can be precomputed, if desired, or determined on the fly: after the two Dijkstra probes at the interval endpoints, we consider the primitive breakpoints on each edge (u, v). Any primitive breakpoint b on (u, v) such that $A[s, u](t_1) < b < A[s, u](t_2)$ might have a primitive image in $A[s, d]$ during $[t_1, t_2]$. We determine those primitive images by reverse Dijkstra, then find their delay values by forward Dijkstra.

6 Closing Remarks

In this paper we resolved a conjecture regarding time-dependent shortest paths, and showed that even with linear cost functions and FIFO edges, the arrival time function at a node can have complexity $n^{\Omega(\log n)}$. We also presented an upper bound of $Kn^{O(\log n)}$ if the total number of linear pieces over all the piecewise linear edge cost functions is at most K. If K is polynomial in n, the arrival time function's complexity is $n^{\Theta(\log n)}$. Nevertheless, and somewhat surprisingly, we presented a polynomial-time algorithm to find the departure time in a query interval that minimizes delay.

The arrival time functions for time-dependent and parametric shortest paths have equivalent complexity if the edge cost functions are (piecewise) linear, but their behavior diverges for higher-order edge cost functions. In particular, if the edge costs are represented by polynomials of degree d, then the arrival time function for the parametric shortest path remains a polynomial of degree d, while the time-dependent shortest path arrival function may have degree d^n.

Several open problems naturally arise from our work: Are there natural but rich classes of graphs for which the arrival time complexity remains polynomial? For instance, what is the complexity for planar graphs? Our lower bound proof easily extends to *constant degree* graphs, so sparsity alone cannot lead to a better bound. We showed that if the slopes of the cost functions belong to a restricted class (powers of α), then the complexity is polynomial. What other natural classes of linear functions lead to similarly reduced complexity? Another natural direction is to investigate the smoothed complexity: does a small random perturbation of the edge arrival functions lead to small expected complexity of the arrival time function?

References

[1] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *J. Alg.*, 31(1):1–28, 1999.

[2] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[3] X. Cai, T. Kloks, and C. K. Wong. Time-varying shortest path problems with constraints. *Networks*, 29(3):141–150, 1997.

[4] P. J. Carstensen. Parametric cost shortest path problems. Unpublished Bellcore memo, 1984.

[5] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record: Journal of the Transportation Research Board*, 1645:170–175, 1998.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[7] B. C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[8] B. C. Dean. Shortest paths in FIFO time-dependent networks: Theory and algorithms. Technical report, 2004.

[9] F. Dehne, M. T. Omran, and J.-R. Sack. Shortest paths in time-dependent FIFO networks using edge load forecasts. In *IWCTS '09: Proceedings 2nd International Workshop on Computational Transportation Science*, pages 1–6. ACM, 2009.

[10] D. Delling and D. Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

[11] C. Demetrescu and G. F. Italiano. Dynamic shortest paths and transitive closure: An annotated bibliography (draft). `www.diku.dk/PATH05/biblio-dynpaths.pdf`, 2005.

[12] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, pages 205–216. ACM, 2008.

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

[14] J. Erickson. Maximum flows and parametric shortest paths in planar graphs, 2010. Submitted to the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (2010).

[15] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[16] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

[17] D. M. Gusfield. *Sensitivity analysis for combinatorial optimization*. PhD thesis, University of California, Berkeley, 1980.

[18] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *Proceedings of the 22st International Conference on Data Engineering (ICDE)*, 2006.

[19] R. M. Karp and J. B. Orlin. Parameter shortest path algorithms with an application to cyclic staffing. Technical report, Massachusetts Institute of Technology, Operations Research Center, 1980.

[20] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 81–90, 1999.

[21] C. S. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *Symposium on Computational Geometry*, pages 264–273, 1997.

[22] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

[23] K. Mulmuley and P. Shah. A lower bound for the shortest path problem. In *Proceedings 15th Annual IEEE Conference on Computational Complexity*, pages 14–21, 2000.

[24] K. Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166, 1995.

[25] G. Nannicini, D. Delling, L. Liberti, and D. Schultes. Bidirectional A^* search for time-dependent fast paths. In *Workshop on Experimental Algorithms*, volume 5038 of *Lecture Notes in Computer Science*, pages 334–346. Springer, 2008.

[26] E. Nikolova, M. Brand, and D. R. Karger. Optimal route planning under uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling*, 2006.

[27] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *14th European Symposium on Algorithms*, volume 4168 of *Lecture Notes in Computer Science*, pages 552–563. Springer, 2006.

[28] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

[29] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *12th Annual European Symposium on Algorithms*, volume 3221 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2004.

[30] H. D. Sherali, K. Ozbay, and S. Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259–272, 1998.

[31] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 112–119. ACM, 2005.

[32] N. Young, R. Tarjan, and J. Orlin. Faster parametric shortest path and minimum balance algorithms. *Networks*, 21(2):205–221, 2006.