Algorithms for Minimum-Cost Paths in Time-Dependent Networks with Waiting Policies

Brian C. Dean

M.I.T., Cambridge, MA 02139, USA,
bdean@theory.lcs.mit.edu

Abstract. We study the problem of computing minimum-cost paths through a time-varying network, in which the travel time and travel cost of each arc are known functions of one's departure time along the arc. For some problem instances, the ability to wait at nodes may allow for less costly paths through the network. When waiting is allowed, it is constrained by a (potentially time-varying) waiting policy that describes the length of time one may wait and the cost of waiting at every node. In discrete time, time-dependent shortest path problems with waiting constraints can be optimally solved by straightforward dynamic programming algorithms; however, for some waiting policies these algorithms can be computationally impractical. In this paper we survey several broad classes of waiting policies and show how techniques for speeding up dynamic programming can be effectively applied to obtain practical algorithms for these different problem variants.

Keywords: shortest paths, time-dependent networks, time-varying networks, waiting.

Introduction

Time-dependent shortest path (TDSP) problems often arise in vehicular transportation applications, where arc travel times and costs vary with time in a predictable fashion and at a fast enough rate that conditions may change significantly during transit through the network. Take $G = (N, A)$ to be a directed graph with $n = |N|$ nodes and $m = |A| = \Omega(n)$ arcs. Along with each arc is specified an arrival time function $a_{ij}(t)$ and a travel cost function $c_{ij}(t)$, which give respectively the arrival time at node j and the cost of traveling on the arc, if one departs from node i at time t. The all-to-one minimum-cost TDSP problem involves computing a minimum-cost path from every node and every point in time to a specified destination node $d \in N$, with no restriction on the arrival time at d. Similarly, for the one-to-all minimum-cost TDSP problem one must compute minimum-cost paths from a single source node and departure time to every other node, where again there is no restriction on the arrival time at these nodes. In this paper we focus exclusively on the all-to-one problem, although all of our results are straightforward to adapt to the one-to-all problem without loss of running time.

TDSP problems were first introduced in a discrete-time framework in 1966 by Cooke and Halsey [6] and have been subsequently studied in both discrete-time [3, 4, 5, 21] and continuous-time settings [13, 19, 20]. At the present time one finds almost exclusive adoption of discrete-time solutions in practice due to their relative simplicity and more predictable performance guarantees. This paper focuses entirely on the discrete-time domain, involving time values $t \in [T] \equiv \{0, 1, 2, \ldots, T-1\}$, where T denotes the length of time until the "planning horizon" for the problem instance under consideration. We can either represent the functions a_{ij} and c_{ij} for a particular arc (i, j) as length-T vectors or assume an "oracle" model in which these functions are analytically specified in some more concise form (e.g. piecewise linear) that can be queried in constant time. In discrete time the goal of the all-to-one problem is to compute for each $(i, t) \in N \times [T]$ the cost $P_i(t)$ of the optimal path to the destination departing from node i at time t. For now we assume for simplicity that travel at any point in time $t \geq T$ beyond the planning horizon is forbidden. This restriction can be relaxed, if we assume that network characteristics remain static for $t \geq T$, although this has a slight impact on the running times of our algorithms — we discuss this issue further in Section 4.

It is a well-known result that waiting at nodes is never beneficial if one wishes to compute minimum-time paths and all a_{ij}'s are non-decreasing (known as the FIFO condition). On the other hand, if we either want minimum-cost paths or if the FIFO condition fails to hold, then waiting at nodes may allow for better paths to the destination. The simplest waiting models in the literature assume that waiting at a given node is either forbidden or allowed for an unrestricted amount of time. However, more generally one may specify for each node $i \in N$ a function $C_i(t, \tau)$, which gives the cost of waiting for τ units of time after arriving at node i at time t, departing at time $t+\tau$. Waiting can be forbidden at certain times and for certain durations by appropriate use of infinite waiting costs. Unfortunately, by allowing such a general waiting cost function as input no TDSP algorithm can hope to achieve a better worst-case running time than $O(nT^2)$ due to the need to potentially examine all of its input. As a result, we wish to consider more succinct waiting policies such as the following:

- **Duration-dependent waiting costs**. The waiting cost function of node i is duration-dependent if $C_i(t, \tau) = DWC_i(\tau)$, where $DWC_i(\tau)$ is called the duration waiting cost function of node i. Lower and upper bounds on the amount of time one may wait at i may be imposed by setting $DWC_i(\tau)$ to infinity for appropriate values of the waiting duration τ.

- **Location-dependent waiting costs**. Let us assign a unit waiting cost $C_i(t, 1)$ to every $(i, t) \in N \times [T]$. We say waiting costs are location-dependent if they only depend on the "locations", in time and space, at which waiting occurs; that is,
$$C_i(t, \tau) = \sum_{u=t}^{u=t+\tau-1} C_i(u, 1).$$
One may also characterize location-dependent costs as being "memoryless", because in this case the cost of waiting at every (node, time) location is independent of the amount of waiting that has previously taken place in the network. Infinite unit waiting costs can be used to prohibit waiting at certain nodes during certain time intervals. We will find it convenient to define location-dependent waiting costs in terms of a cumulative cost of waiting function $CCW_i(t) = \sum_{u=0}^{t-1} C_i(u, 1)$, so that $C_i(t, \tau) = CCW_i(t+\tau) - CCW_i(t)$.

- **Mixed waiting costs**. Waiting cost in this case is the sum of a duration-dependent component plus a location-dependent component: $C_i(t, \tau) = DWC_i(\tau) + CCW_i(t+\tau) - CCW_i(t)$.

The contribution of this paper is a set of more efficient algorithms for the minimum-cost TDSP problem with mixed waiting costs, where the duration-dependent component is either convex, concave, or a piecewise combination of convex and concave functions (we define a convex function as having a non-negative second derivative). This class is sufficiently broad that it should encompass many waiting policies found in practice. As noted above, we can restrict both the time intervals during which waiting may occur as well as the allowable durations of waiting. Parking a vehicle in a private lot or garage typically involves a concave cost function over the duration of waiting. Convex functions may be used in locations where only brief waiting is desired, since the marginal cost of waiting becomes increasingly larger over time. Our results are based on techniques for speeding up dynamic programming that were initially developed for a host of applications, most notably sequence alignment in computational biology.

1 Previous Work and Summary of Results

If waiting costs are location-dependent, simple dynamic programming algorithms [3, 4, 21] can be used to solve the all-to-one TDSP problem in $O(mT)$ time, based on the following recursive formulation:

$$P_i(t) = \begin{cases} +\infty, & \text{if } t \geq T, \\ 0, & \text{if } t < T \text{ and } i = d, \\ \min(C_i(t,1) + P_i(t+1), \min\{c_{ij}(t) + P_j(a_{ij}(t)) : (i,j) \in A\}), & \text{if } t < T \text{ and } i \neq d. \end{cases} \quad (1)$$

We arrive at a nice interpretation for dynamic programming algorithms based on (1) by noting that these equations give precisely Bellman's optimality conditions for an equivalent static shortest path problem in a time-expanded network (Fig. 1a). Since the pioneering work of Ford and Fulkerson [9], many dynamic network optimization problems have been shown to be equivalent to static problems appropriately cast within a time-expanded network, and the TDSP problem is no exception: here, we wish to compute the shortest path from every time-expanded node $(i, t) \in N \times [T]$ to the set of destination nodes $d \times [T]$. Assuming that $a_{ij}(t) > t$ for all $(i, j) \in A$, the time-expanded network is acyclic. It is well-known that the single-source

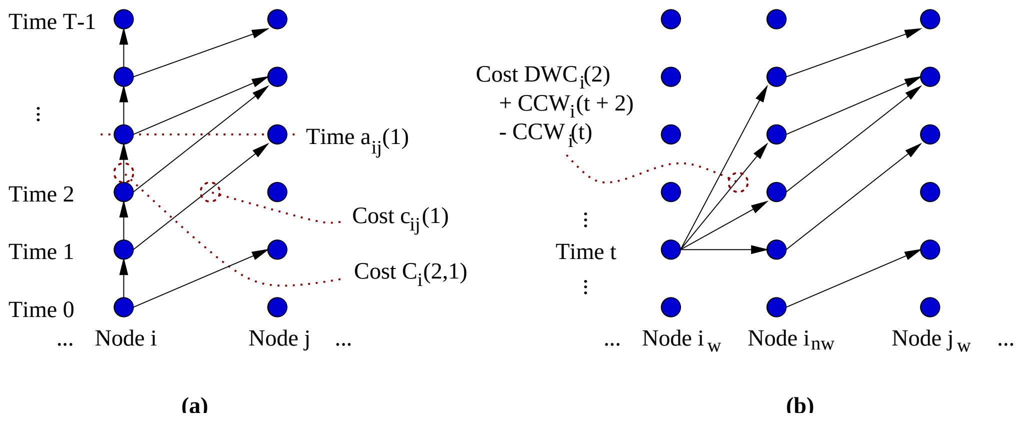

Figure 1: Graphical depiction of time-expanded networks. In (a), waiting is allowed for unlimited duration with location-dependent costs at node i, and is forbidden at node j (the "waiting arcs" from (j, t) to $(j, t + 1)$ have infinite cost and are not drawn). In (b), waiting is allowed at i with mixed waiting costs.

shortest path problem can be solved in acyclic networks in linear time, in this case $O(mT)$ time. Acyclic shortest path algorithms work via dynamic programming by processing nodes in a topological ordering, which in this case corresponds to enumerating the nodes of the time-expanded network in successive time levels in reverse chronological order. In the input model where the a_{ij} and c_{ij} functions are represented by length-T vectors, the $O(mT)$ running time is worst-case optimal and linear in the input size.

A slightly more complicated cost policy involves the use of location-dependent costs where waiting of only bounded duration is allowed. The bounded duration constraint can be specified by a duration-dependent cost function that jumps to infinity at some point. For zero-cost bounded-duration waiting, Cai, Cloks, and Wong [3] provide an algorithm with running time of $O(nT \log T + mT)$. Chabini and Dean [5] later improved the running time to $O(mT)$ while also allowing for location-dependent waiting costs.

For the more general mixed waiting cost policy, a slightly more complicated dynamic programming formulation yields an $O(nT^2 + mT)$ algorithm. Let us define $P_i^w(t)$ to be the optimal path cost from (i, t) to the destination where waiting is allowed prior to departure, and let $P_i^{nw}(t)$ be the optimal path cost where no waiting is allowed. Then we have

$$P_i^{nw}(t) = \begin{cases} +\infty, & \text{if } t \geq T, \\ 0, & \text{if } t < T \text{ and } i = d, \\ \min\{c_{ij}(t) + P_j^w(a_{ij}(t)) : (i,j) \in A\}, & \text{if } t < T \text{ and } i \neq d, \end{cases} \quad (2)$$

$$P_i^w(t) = \min_{\tau \geq 0}\{DWC_i(\tau) + CCW_i(t+\tau) + P_i^{nw}(t+\tau)\} - CCW_i(t). \quad (3)$$

Again, we can interpret this dynamic program as the computation of shortest paths through an acyclic time-expanded network (Fig. 1b), where each time-expanded node (i, t) is split into a "waiting node" (i_w, t) and a "non-waiting" node (i_{nw}, t). In contrast to the previous $O(mT)$ algorithm, however, the $O(nT^2 + mT)$ running time of this algorithm is not linear in the input size, since a mixed waiting policy is specified by at most $O(nT)$ elements of data. Therefore, in this case we have some hope of achieving more efficient running times. In practice we can typically realize substantial savings by only applying (2) and (3) for nodes with duration-dependent or mixed waiting costs — for example if waiting happens to be prohibited at most nodes in our network. This simplification reduces the running time to $O(n'T^2 + mT)$, where n' is the number of nodes with "complicated" waiting policies; however, the troublesome T^2 term still persists. In Section 3, we describe how to speed up the dynamic programming computation so as to reduce the running time from $O(nT^2 + mT)$ down to $O(nT\alpha(T) + mT)$ where each duration-dependent cost component has a constant number of convex and/or concave pieces and $\alpha(\cdot)$ denotes a version of the inverse Ackermann function to be defined later. This reduction in running time has significant implications for the computational feasibility of solving large problem instances.

2 Bounded Waiting with Location-Dependent Waiting Costs

For completeness, we briefly summarize the result of Chabini and Dean [5] that yields an $O(mT)$ running time when waiting costs are location-dependent and waiting durations are bounded. Their basic dynamic programming algorithm repeatedly applies (2) and (3) in reverse chronological order to determine the optimal labels for nodes in the time-expanded network. In total (2) requires $O(mT)$ time and (3) requires $O(nT^2)$ time. We will show how to evaluate each of the nT occurrences of (3) in constant amortized time, reducing the total back down to $O(mT)$.

We define the bounded waiting constraint by means of a function $M_i(t)$ that specifies the latest time until which one may wait after arriving at node i at time t. If bounded waiting is specified by a duration-dependent waiting cost function that jumps to infinity after some duration D, we get the constant function $M_i(t) = D$. However, more generally we can consider $M_i(t)$ to be any non-decreasing function. In other words, we can think of there being a window of feasible waiting durations $[t, M_i(t)]$, and as t monotonically decreases over the course of our dynamic programming algorithm, the upper endpoint of the window $M_i(t)$ will also be monotonically decreasing. By setting $Q_i(t) = P_i^{nw}(t) + CCW_i(t)$, we can simplify (3) to

$$P_i^w(t) = \min\{Q_i(t+\tau) : t \le t+\tau \le M_i(t)\} - CCW_i(t). \tag{4}$$

As we scan backward through time, we can think of the evaluation of (4) as the selection of the minimum element from a FIFO queue whose elements represent the cost of waiting plus departure at each feasible waiting duration. When time is reduced from t to $t-1$, a new element $Q_i(t-1)$ is added to the back of the queue and a set of $M_i(t)-M_i(t-1)$ elements is removed from the front. Therefore, we can reduce our problem to the following data structure problem: design a queue-like data structure that supports *insert*, *remove*, and *find-min* operations each in $O(1)$ amortized time. It is not difficult to build such a data structure: it is a standard FIFO queue in which elements will have increasing values as we move from the front of the queue to the back. Whenever an element is inserted in the back of the queue, we delete all elements in front of it having larger value (a contiguous block of elements), and record the size of this deleted block alongside the new element (to accommodate subsequent calls to remove, in the case that we ask to remove an already-deleted element).

2.1 Piecewise Linear Duration-Dependent Waiting Costs

If we have mixed waiting costs involving duration-dependent components that are piecewise linear with at most P total pieces across all arcs, the result above can be used to obtain an $O(PT + mT)$ running time via a node splitting transformation. Suppose the duration-dependent component $DWC_i(\tau)$ of the waiting cost at node i is piecewise linear with p pieces, so for $k = 1 \ldots p$,

$$DWC_i(\tau) = \alpha_k(\tau - \tau_{k-1}) + \beta_k \quad \text{for } \tau_{k-1} \le \tau < \tau_k,$$

where $\tau_0 = 0$ and $\tau_p = T$. We split i into $p+2$ nodes i', i'', and $i_1 \ldots i_p$ (each of which corresponds to one of the pieces of DWC_i). There is a directed arc from i' to each of the nodes $i_1 \ldots i_p$ and a directed arc from each of $i_1 \ldots i_p$ to i'', and i'' will have the same set of outgoing arcs as the original node i. No waiting is allowed at i' and i'', and waiting at i_k is allowed for at most $\tau_k - \tau_{k-1} - 1$ units of time. While traveling through the time-expanded network, a commodity will arrive first at i', at which point it selects the linear piece k, $1 \le k \le p$, of DWC_i until which it will wait. It then moves to node i_k, waits for some amount of time, and finally moves to i'' from which it immediately departs to some other node. The time-dependent characteristics of i', i'', and $i_1 \ldots i_p$ are as follows:

$$\begin{aligned} c_{i'i_k}(t) &= CCW_i(\tau_{k-1}) - CCW_i(t), & a_{i'i_k}(t) &= t + \tau_{k-1}, \\ c_{i_k i''}(t) &= \beta_k, & a_{i_k i''}(t) &= t, \\ c_{i''j}(t) &= c_{ij}(t), & a_{i''j}(t) &= a_{ij}(t), \\ C_{i_k}(t,1) &= C_i(t,1) + \alpha_k. \end{aligned}$$

Since nodes $i_1 \ldots i_p$ each contribute 2 new edges, a total of $O(PT)$ extra time will be required by the algorithm beyond its original $O(mT)$ running time. Also note that we had previously assumed that $a_{ij}(t) > t$ for all edges (i,j) in order to assure acyclicity of the time-expanded network. The transformation above does create some edges for which $a_{ij}(t) = t$, but these new edges never create time-expanded cycles. Thus, our dynamic programming algorithm can still be successfully applied, but we must now however be careful to process nodes within each time-level in a proper topological ordering.

3 Duration-Dependent Waiting Costs

In this section we consider mixed waiting cost policies with duration-dependent components made up from convex and concave pieces. If all such pieces are convex and there are at most P pieces across all arcs in the network, we will achieve a running time of $O(PT + mT)$. If concave pieces exist, the running time increases very slightly to $O(PT\alpha(T) + mT)$. In both cases, however, there is a substantial improvement over the naïve algorithm that includes an $O(nT^2)$ term. The function $\alpha(T)$ is a version of the inverse Ackermann function defined as $\alpha(T) = \min\{k | f_k(T) \leq k\}$, where $f_{-1}(n) = n/2$ and $f_k(n) = \min\{i | f_{k-1}^{(i)} \leq 1\}$ ($f^{(i)}$ denotes the function f applied for i iterations). The function $\alpha(T)$ grows much slower than $\log T$ and can be informally thought of as a constant for all conceivable values of T encountered in practice; for example, $\alpha(2^{256}) \leq 3$.

We utilize techniques for speeding up certain classes of "one-dimensional" dynamic programming algorithms that were initially motivated by a host of problems from various fields, most notably by sequence alignment problems from computational biology. These problems are based on the following fundamental recurrence for computing $D(t)$ and $E(t)$ for $t \in [T]$:

$$E(t) = \min_{\tau \geq 0}\{D(t+\tau) + f(\tau)\}, \tag{5}$$

where $D(t)$ can be computed from $E(t)$ in constant time. In the context of sequence alignment between two strings A and B, $D(t)$ represents the optimal cost of aligning a given suffix of A starting exactly at position t in B, and $E(t)$ represents the optimal cost of alignment at position t in B where a "gap" of some length τ is allowed (so the suffix of A would actually be aligned starting at position $t + \tau$ in B). The function $f(\tau)$ specifies a gap penalty that depends on the gap length, which is often concave for sequence alignment in computational biology.

In the context of the TDSP problem, we see that (5) is essentially the same as the equation used to compute $P_i^w(t)$ in (3). If $DWC_i(\tau)$ is the duration-dependent component of $C_i(t,\tau)$, we can express (3) as

$$P_i^w(t) = \min_{\tau \geq 0}\{Q_i(t+\tau) + DWC_i(\tau)\} - CCW_i(t), \tag{6}$$

which is equivalent in structure to (5). In our case waiting plays the same role as a gap in sequence alignment, and the waiting cost function $DWC_i(\tau)$ corresponds directly to the gap penalty function $f(\tau)$.

The straightforward dynamic programming algorithm for solving (5) for $t \in [T]$ runs in $O(T^2)$ time, and for arbitrary gap penalty functions f it is currently not known if one can do better. For convex f, Hirschberg and Larmore [14] show how to speed up the dynamic programming computation to run in $O(T \log T)$ time, and when f is concave Miller and Myers [18] give an $O(T \log T)$ algorithm. Galil and Giancarlo [11] also provide $O(T \log T)$ algorithms for both the convex and concave cases. Following these results, a significant breakthrough was achieved by Wilber [22], whose devised an $O(T)$ algorithm for the convex case based on fast matrix searching techniques due to Aggarwal et al. [1]. However, Wilber's technique cannot be used for our TDSP problem since it does not run in linear time for many problems consisting of several interleaved computations based on (5). Wilber's algorithm occasionally makes "mistakes" that require it to recompute $D(t)$ from $E(t)$ multiple times for a given t. If $D(t)$ can be computed from $E(t)$ in constant time this does not adversely affect the running time; however, in our case D and E correspond to the P^{nw} and P^w labels and to compute D from E requires application of (2) which is not a constant-time operation. Subsequent to Wilber's result several authors proposed enhancements that allowed for interleaved computation in $O(T)$ time, including Klawe [15], Galil and Park [10], and Eppstein [7]. For the concave case Klawe and Kleitman [16] give an $O(T\alpha(T))$ algorithm also based on matrix searching, and for functions composed of a series of p convex and concave pieces, Eppstein [7] shows how to achieve a running time of $O(pT\alpha(T/p)) = O(pT\alpha(T))$. See also Galil and Park [12] for a good survey of these results.

By applying these techniques, we can achieve an $O(mT)$ running time for the TDSP problem if duration-dependent costs are convex, and we obtain a running time of $O(PT\alpha(T) + mT)$ by applying Eppstein's technique, where among all arcs we find a total of P convex and/or concave pieces. If all pieces are convex, we can apply the node splitting transformation from the previous section in conjunction with [15, 10, 7] to reduce the running time back down to $O(PT + mT)$.

4 Concluding Remarks

We have discussed several techniques for speeding up computation of minimum-cost paths in time-dependent networks with various waiting policies. Our discussion has been centered on the all-to-one TDSP problem; however, all techniques carry over to the one-to-all problem with the same running time if we adapt them to "forward" dynamic programming that operates in increasing, rather than decreasing chronological order of time. We have shown that in the presence of duration-dependent waiting costs, the structure of the dynamic programming formulation for the TDSP problem is equivalent to well-studied dynamic programming problems from other fields; hence, any subsequent improvements for those problems (e.g. removal of the inverse Ackermann function for the concave case) would carry over to the TDSP problem.

We have assumed throughout this paper that arc travel times are strictly positive, in order to ensure an acyclic time-expanded network. In the event that this assumption fails and we have "zero-delay" arcs, the same efficient approaches can still be used to handle waiting constraints and costs. However, we incur a slight increase in running time since the computation of (2) within each "time-level" of the time-expanded network will now require a shortest path computation. If $SSP(m)$ denotes the running time for solving a static shortest path problem with m arcs, the presence of zero-delay arcs raises the $O(mT)$ term in our running time expressions up to $O(SSP(m) \times T)$. A common special case of the zero-delay scenario arises when we wish to model the fact that travel beyond time $T - 1$ is permitted, given that arc travel times and costs remains static for $t \geq T - 1$. This is accomplished through the use of zero-delay arcs on the final time level, by setting $a_{ij}(T - 1) = T - 1$ for all $(i, j) \in A$, and it results in an additive penalty of $O(SSP(m))$ for all of our running times. In particular, this extension allows us to solve the all-to-one problem for all nodes and all values $t \geq 0$ in time, since the minimum-cost path departing node i at time $t \geq T$ is now the same as the minimum-cost path departing from i at time $t = T - 1$. Related to this special case is the case of networks with temporally-repeated time-dependent characteristics (see, for example [2]). It is not clear whether or not any of the techniques from this paper might carry over to the temporally-repeated case, and this might be an interesting direction for future research. One may also wish to investigate how well these results carry over to problems formulated in continuous-time settings, for example in networks with piecewise constant or piecewise linear time-dependent characteristics.

References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. Algorithmica 2 (1987), 195-208.

[2] R. Ahuja, J. Orlin, S. Pallottino, and M.G. Scutella. Minimum time and minimum cost path problems in street networks with periodic traffic lights. Transportation Science 36 (2002), 326-336.

[3] X. Cai, T. Kloks, and C.K. Wong, Time-varying shortest path problems with constraints. Networks 29 (1997), 141-149.

[4] I. Chabini. Discrete dynamic shortest path problems in transportation applications: complexity and algorithms with optimal run time. Transportation Research Record 1645 (1998).

[5] I. Chabini and B. Dean. Shortest path problems in discrete-time dynamic networks: complexity, algorithms, and implementations. Unpublished manuscript (1999).

[6] L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. Journal of Mathematical Analysis and Applications 14 (1966), 492-498.

[7] D. Eppstein. Sequence comparison with mixed convex and concave costs. Journal of Algorithms 11 (1990), 85-101.

[8] D. Eppstein, Z. Galil, and R. Giancarlo. Speeding up dynamic programming. Proceedings of the 29th IEEE Symposium on the Foundations of Computer Science (1988), 488-496.

[9] L.R. Ford and D.R. Fulkerson. Flows in networks. Princeton University Press, Princeton, N.J., 1962.

[10] Z. Galil and K. Park. A linear-time algorithm for concave one-dimensional dynamic programming. Information Processing Letters 33 (1990), 309-311.

[11] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. Theoretical Computer Science 64 (1989) 107-118.

[12] Z. Galil and K. Park. Dynamic programming with convexity, concavity and sparsity. Theoretical Computer Science 92 (1992), 49-76.

[13] J. Halpern. Shortest route with time-dependent length of edges and limited delay possibilities in nodes. Zeitschrift fur Operations Research 21 (1977), 117-124.

[14] D.S. Hirschberg and L.L. Larmore. The least weight subsequence problem. SIAM Journal on Computing 16(4) (1987), 628-638.

[15] M. Klawe. A simple linear-time algorithm for concave one-dimensional dynamic programming. Technical Report 89-16, Department of Computer Science, University of British Columbia (1990).

[16] M.M. Klawe and D.J. Kleitman. An almost linear time algorithm for generalized matrix searching. SIAM Journal on Discrete Mathematics 3(1) (1990), 81-97.

[17] L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. Journal of Algorithms 12 (1991), 490-515.

[18] W. Miller and E.W. Myers. Sequence comparison with concave weighting functions. Bulletin of Mathematical Biology 50(2) (1988), 97-120.

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

[20] A. Orda and R. Rom. Minimum weight paths in time-dependent networks. Networks 21(3) (1991), 295-320.

[21] S. Pallottino and M. Skutella. Shortest path algorithms in transportation models: classical and innovative aspects". Technical Report TR-97-06, Univerità di Pisa (1997).

[22] R. Wilber. The concave least-weight subsequence problem. Journal of Algorithms 9 (1988), 418-425.