Discrete Dynamic Shortest Path Problems in Transportation Applications
Complexity and Algorithms with Optimal Run Time

ISMAIL CHABINI

A solution is provided for what appears to be a 30-year-old problem dealing with the discovery of the most efficient algorithms possible to compute all-to-one shortest paths in discrete dynamic networks. This problem lies at the heart of efficient solution approaches to dynamic network models that arise in dynamic transportation systems, such as intelligent transportation systems (ITS) applications. The all-to-one dynamic shortest paths problem and the one-to-all fastest paths problems are studied. Early results are revisited and new properties are established. The complexity of these problems is established, and solution algorithms optimal for run time are developed. A new and simple solution algorithm is proposed for all-to-one, all departure time intervals, shortest paths problems. It is proved, theoretically, that the new solution algorithm has an optimal run time complexity that equals the complexity of the problem. Computer implementations and experimental evaluations of various solution algorithms support the theoretical findings and demonstrate the efficiency of the proposed solution algorithm. The findings should be of major benefit to research and development activities in the field of dynamic management, in particular real-time management, and to control of large-scale ITSs.

The shortest paths problem in networks has been the subject of extensive research for many years (*1*). The analysis of transportation networks is one of the many application areas in which the computation of shortest paths is one of the most fundamental problems. The majority of published research on shortest paths algorithms, however, dealt with static networks that have fixed topology and fixed link costs.

Interest in the concept of dynamic management of transportation systems has been increasing. New advances have brought renewed interest in the study of shortest paths problems with a new twist: Link costs generally depend on the entry time of a link. This results in a new family of shortest paths problems known as dynamic, or time-dependent, shortest paths problems.

Chabini (*2*) distinguishes various types of dynamic shortest path problems depending on the following: (*a*) fastest versus minimum cost (or shortest) path problems; (*b*) discrete versus continuous representation of time; (*c*) first-in-first-out (FIFO) networks versus non-FIFO networks, in which one can depart later at the beginning of one or more arcs and arrive earlier at their end; (*d*) waiting is allowed versus waiting is not allowed at nodes; (*e*) types of shortest path questions asked: one-to-all for a given departure time or all departure times, and all-to-one for all departure times; and (*f*) integer versus real valued link travel times and link travel costs.

In fastest path problems, the cost of a link is the travel time of that link. In minimum cost paths problems, link costs can be of general form. Although the fastest paths problem is a particular case of the minimum cost paths problem, the distinction between the two is particularly important for the design of efficient solution algorithms. Time-dependent marginal travel times encountered in system optimum dynamic traffic assignment models are an example of general form costs.

Depending on how time is treated, dynamic shortest paths problems can be subdivided into two types: discrete and continuous. In discrete dynamic networks, time is modeled as a set of integers. Orda and Rom (*3*) establish various properties for continuous dynamic networks where time is treated as real numbers. The results for the one-to-all fastest paths problem studied here are valid for both discrete and continuous representations of time.

It is important to note that a discrete dynamic network can be viewed alternatively as a static network obtained by using a time-space expansion representation. The size of the equivalent representation depends on the waiting policy at nodes. As may be expected, it is not the best approach to explicitly use the time-space expansion representation to compute dynamic shortest paths. The time-space expansion network has, however, some particular properties that can be exploited in the design of efficient shortest paths algorithms. The challenge is to discover particular properties of these networks and appropriately exploit them in designing better algorithms.

Compared with that for static shortest paths problems, the literature on dynamic versions is very limited. The main known result is the condition under which static shortest path algorithms can be used, at no extra cost, to solve the one-to-all fastest paths problem in dynamic networks: Arc travel times must satisfy the FIFO condition. Hidden behind this result are various limitations:

1. This result applies to the fastest paths problem only and not to the minimum cost paths problem.
2. This result is limited to forward-search labeling algorithms only. Transportation applications need the computation of fastest paths from all nodes to a set of destinations. Hence, static methods typically would first compute fastest paths from all nodes to all nodes and then extract needed solutions. Because the number of destinations in a transportation network usually is a small fraction of total number of nodes, a static approach may not lead to the best solution algorithm possible.
3. Transportation applications do not necessarily satisfy the FIFO condition.
4. A static algorithm computes shortest paths for one departure time interval only, but shortest paths for all possible departure time intervals are generally sought.

Traffic management centers in intelligent transportation systems (ITS) must operate in real time. The time taken to collect data, process them, and broadcast the resulting information may consti-

Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Room 1-138, 77 Massachusetts Avenue, Cambridge, MA 02139-4307.

tute a possible information bottleneck. To avoid this bottleneck, the ITS models and algorithms must run much faster than real time. Furthermore, transportation applications usually involve very-large-size networks. Typical real-life networks involve thousands of links and nodes and hundreds of time intervals representing the time dimension. For instance, a network model of the city of Boston contains 25,000 links and 7,000 nodes. The time dimension depends on the length of analysis period and the discretization interval. If 15-sec time discretization is adopted, 480 time intervals are required to model a 2-hour morning peak period.

Given these key issues in ITS and the importance of shortest paths in ITS models and algorithms, one then must find the most efficient methods to solve shortest path problems in dynamic networks. Although other researchers have made valuable contributions in addressing this problem, a question remained unanswered for the last 3 decades: What is the exact complexity of these problems and could one discover algorithms that have the best possible run time? The main objective of this study was to answer this question.

Three types of shortest path problems in discrete dynamic networks are addressed in this study: the one-to-all fastest paths problem departing origin node at a given time interval, the all-to-one fastest paths problem for all departure time intervals, and the all-to-one minimum cost paths problem for all departure time intervals. The latter two problems are of particular interest in the context of ITS applications. Note that most of the published papers on these topics focus mainly on problems of the first type. The objective in restudying the one-to-all version of the problem essentially was to revisit and extend early results. These are established by using alternative arguments which are shorter, simpler, and more insightful. For instance, new results on the properties of the problem are established. These are exploited in designing new and efficient solution algorithms. A new all-to-one discrete dynamic shortest paths algorithm that is demonstrated to be the most efficient solution algorithm possible is proposed. Numerical tests on large networks support the analytical findings. It is the first time that a solution algorithm with an optimal computation time complexity is proposed for what appears to be a 30-year-old problem first addressed by Cooke and Halsey (*4*).

DEFINITIONS AND NOTATION

Let $G = (N, A, D, C)$ be a directed network where $N = \{1, \ldots, n\}$ is the set of nodes and $A = \{(i,j) \in N x N\}$ is the set of arcs (or links). The number of links (arcs) is denoted by *m*. We denote by $D = \{d_{ij}(t) | (i,j) \in A\}$ the set of time-dependent link travel times and by $C = \{c_{ij}(t) | (i,j) \in A\}$ the set of time-dependent link travel costs. Functions $d_{ij}(t)$ have integer-valued domain and positive integer-valued range. A function $d_{ij}(t)$ is then a discrete and time-dependent function that, it is assumed, takes a static value after a finite number of time intervals M. $S = \{0, \ldots, M - 1\}$ is the set of departure time intervals for which link travel times are time-dependent. Functions $c_{ij}(t)$ have real-valued range and integer-valued domain. $c_{ij}(t)$ is static when the departure time is greater than or equal to $M - 1$. We assume that there is no negative cycle after departure time interval $M - 1$. $B(i)$ denotes the set of nodes having an outgoing arc to node i and $A(i)$ denotes the set of nodes having an ingoing arc from node i. $G = (N, A, D, C)$ is called a discrete dynamic network.

Arc travel times may possess some properties useful in studying and developing algorithms for dynamic networks. A well-celebrated property is the FIFO condition (*5*), which may be defined in various mathematical forms. For instance, the FIFO condition is valid if and only if the following system of inequalities holds:

$$\forall(i, j, t),\ t + d_{ij}(t) \leq (t+1) + d_{ij}(t+1)$$

When the FIFO condition holds, we say that the dynamic network is FIFO. The FIFO condition also is known as the nonovertaking condition in traffic theory. The above equivalent mathematical condition simply says that link exit time functions are nondecreasing. Intuitively, this holds if no overtaking takes place.

When the FIFO condition is not satisfied, sometimes it may be preferable to wait a certain amount of time at the start node of a link before embarking on that link. Such waiting may or may not be allowable depending on the application at hand. Therefore, two policies of waiting at nodes are considered: waiting is allowed at nodes and waiting is not allowed at nodes. The next section demonstrates that in studying fastest paths problems, the former waiting policy is a particular case of the latter one.

Shortest paths problems also depend on the type of questions they answer. Two questions are of concern here: What are the shortest paths from one origin to all destinations departing at instant 0? and What are the shortest paths from all nodes to one destination node for all departure times? The second question is perhaps the most relevant one in the context of dynamic management of transportation systems. Most published work has dealt with the first question, with the exception of the work by Cooke and Halsey (*4*) and Ziliaskopoulos and Mahmassani (*6*).

Although algorithms that answer the first question can be used to answer the second question as well, such approaches would not be efficient for transportation networks because only a reduced subset of nodes are actually destination nodes. For instance, a network model for a city like Boston contains about 7,000 nodes, only 10 percent of which are destination nodes.

FORMULATIONS AND ALGORITHMS FOR ONE-TO-ALL FASTEST PATHS PROBLEM

Waiting at Nodes Is Not Allowed

This is the most studied variation of the problem (*4,5,7*). The most celebrated result for this variation of the problem is the following: When the FIFO condition is valid, Dijkstra's (*8*) algorithm can be generalized to solve the time-dependent fastest paths problem with the same time complexity as the static shortest paths problem. Dreyfus (*7*) was the first to mention this generalization. Later, Kaufman and Smith (*5*) formally proved that this generalization is valid only if the FIFO condition is satisfied. Ahn and Shin (*9*) provided an even earlier proof.

A proof that is more insightful and simpler follows. The proof is based on a different formulation of the problem. Let f_j denote the minimum travel time from origin node o to node j, leaving the origin node at time interval 0. The key idea is to consider, when writing optimality conditions for node j, only those paths that visit previous node i at a time greater than or equal to f_i. Minimum travel times are then defined by the following functional form:

$$f_j = \begin{cases} \min_{i \in B(j)} \min_{t \geq f_i}\left(t + d_{ij}(t)\right) & ; j \neq o \\ 0 & ; j = o \end{cases}$$

Proposition 1: If the FIFO condition is satisfied, the above formulation of the fastest paths problem is equivalent to the following equations:

$$f_j = \begin{cases} \min_{i \in B(j)} \left(f_i + d_{ij}(f_i) \right); j \neq 0 \\ 0 \quad ; j = 0 \end{cases}$$

Proof: The equivalence holds because $\min_{t \geq f_i} (t + d_{ij}(t)) = f_i + d_{ij}(f_i)$ if the FIFO condition holds.

The formulation shown in Proposition 1 provides a strong basis for the development of efficient algorithms. For instance, it shows that some static shortest paths algorithms can be extended, at no extra time, to solve the fastest paths problem if the FIFO condition holds. This result is summarized in the following proposition:

Proposition 2: If the FIFO condition is satisfied, the solution of a fastest paths problem in such dynamic networks is equivalent to an associated static shortest paths problem.

The generalization of Dijkstra's algorithm first proposed, as a heuristic, by Dreyfus (*7*), and later proved by Kaufman and Smith (*5*) and by Ahn and Shin (*9*), can be viewed as a particular case of Proposition 2. Note that these generalizations are subject to restrictions: Only a static forward labeling process would be permitted, and it is supposed that running time would depend on the number of nodes and links only (static shortest path algorithms may depend on link travel costs as well). These restrictions were never made explicit in the literature.

Proposition 3: If the FIFO condition is satisfied, any forward label setting algorithm based on functional equations of Proposition 1 solves the dynamic fastest paths problem and the static shortest paths problem in the same time complexity.

When the FIFO condition is not satisfied, no algorithm that does not explicitly use the time-space expansion representation has yet been published. On the basis of this formulation, such type of solution algorithms were designed.

Waiting at Nodes Is Allowed

The case of when waiting is allowed at all nodes was analyzed (results can be generalized if waiting is allowed at a subset of nodes only). The main result of this was that this variant of the problem is a special case of the no-waiting-is-allowed policy variant of the problem studied above.

Denote by $w_i(t)$ the maximum waiting time allowed at node *i* and at departure time interval *t.* First note that when waiting is allowed at node *i,* the minimum travel time possible on arc (i,j) is given by the function $D_{ij}(t) = \min_{w_i(t)+t \geq s \geq t}(s - t + d_{ij}(s))$. It is easy to prove that this function verifies the FIFO condition if unlimited waiting is allowed.

Because waiting is allowed at nodes, minimum travel times are now given by the following functional form:

$$f_j = \begin{cases} \min_{i \in B(j)} \min_{t \geq f_i} \left(t + D_{ij}(t) \right); j \neq 0 \\ 0 \quad ; j = 0 \end{cases}$$

Proposition 4: The waiting-is-allowed variant is a particular case of the waiting-is-not-allowed variant studied above. Moreover, if unlimited waiting is allowed at all nodes, results of propositions 1, 2, and 3 hold without the FIFO requirement on link travel times.

Proof: The above functional form shows that when waiting is allowed at all nodes, this variant of the fastest paths problem is equivalent to a fastest paths problem without waiting at nodes and with link delay functions $D_{ij}(t)$ [instead of $d_{ij}(t)$]. Because $D_{ij}(t)$ verifies the FIFO condition if unlimited waiting is allowed, results of propositions 1, 2, and 3 hold.

The fastest paths problem with no waiting is allowed at nodes and FIFO condition not satisfied is the most difficult variant of the fastest paths problem.

ALL-TO-ONE FOR ALL DEPARTURE TIMES FASTEST PATHS PROBLEM

As shown above, the all-to-one for all starting times fastest paths problem is the most relevant variant of fastest paths problems in the context of dynamic management of transportation systems. A backward star formulation of this problem, when waiting is not allowed, is presented here, and a property that is then exploited in developing a new solution algorithm is exploited.

Formulation

Denote by $\pi_i(t)$ the fastest travel time to destination *q* departing node *i* at time *t.* The minimum travel times are then defined by the following functional form:

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

This is a well-known optimality condition that was used by Cooke and Halsey (*4*) and Ziliaskopoulos and Mahmassani (*6*) to design solution algorithms that have respectively $O(n^3M^2)$ and $O(nmM^2)$ as worst-case running time complexities. An optimal algorithm for this problem is presented below, based on the following proposition:

Proposition 5: Labels $\pi_i(t)$ can be set in a decreasing order of departure time intervals.

Proof: Because all arc travel times are positive integers, labels corresponding to time step *t* never update labels corresponding to time steps greater than *t.*

The above result implicitly reflects the acyclic property, along the time dimension, of the time-space expansion of a discrete dynamic network.

Decreasing Order of Time Algorithm

For departure time intervals greater than or equal to $M - 1$, the computation of fastest paths is equivalent to a static shortest paths problem. A decreasing order of time (DOT) algorithm is proposed here that is based on the result of Proposition 5. The main loop of the algorithm is carried out in decreasing order of time. The algorithm assumes that a static shortest paths procedure, with an optimal running time SSP, is given:

```
Step 0 (Initialization):

$\pi_i(t) = \infty, \forall(i \neq q), \pi_q(t) = 0, \forall(t < M - 1)$

        $\pi_i(M-1) = \text{Static Shortest Paths}\left(d_{ij}(M-1), q\right)\forall(i)$

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

Step 2 (Main Loop):

For $t = M - 2$ down to 0 do:
For $(i, j) \in A$ do:    $\pi_i(t) = \min\left(\pi_i(t), d_{ij}(t) + \pi_j\left(t + d_{ij}(t)\right)\right)$
```

Proposition 6: Algorithm DOT solves for the all-to-one fastest paths problem, with a serial computation time in $\theta(\text{SSP} + nM + mM)$.

Proof: The optimality of the algorithm follows from Proposition 5. The order analysis of running time follows in a straightforward manner by counting the number of operations appearing in the statements of algorithm DOT.

Proposition 7: The complexity of the all-to-one fastest paths problem, for all starting times, is in $\Omega(\text{SSP} + nM + mM)$. Hence, algorithm DOT has an optimal running time complexity (no algorithm with a better running time complexity can be found).

Proof: The problem has the above complexity because every solution algorithm has to access all arc data (mM), initialize network node labels because fastest paths for all departure times are sought (nM), and compute an all-to-one fastest paths tree for departure time intervals beyond departure time $M - 1$ (SSP). Using results of Proposition 6, algorithm DOT is then optimal.

ALL-TO-ONE FOR ALL DEPARTURE TIMES MINIMUM COST PATHS PROBLEM

In this section, the results established in previous sections are expanded to the minimum cost paths problem.

Formulation

Let $C_i(t)$ denote the minimum travel cost from node i, departing at time t, to destination q. Minimum travel costs are then defined by the following functional form:

$$C_i(t) = \begin{cases} \min_{j \in A(i)} c_{ij}(t) + C_j\left(t + d_{ij}(t)\right) & ; i \neq q \\ 0 & ; i = q \end{cases}$$

These equations extend the optimality conditions described in previous sections for the fastest paths problem. Here they are used to extend results established for the fastest paths problem. In particular, the exact complexity of the minimum cost paths problem is determined and an extension of algorithm DOT that runs in an optimal time is shown.

Proposition 8: Labels $C_i(t)$ can be set in a decreasing order of departure time intervals.

Proof: Because all arc travel times are positive integers, labels corresponding to time steps t do not depend on labels corresponding to time intervals smaller than t. This result implicitly reflects the acyclic property, along the time dimension, of the time-space expansion of a discrete dynamic network. Note that link travel costs can take any real number value, including negative numbers.

Extension of Algorithm DOT To Compute Minimum Cost Paths

Note that for departure time intervals greater than or equal to $M - 1$, link travel times and costs become static. Labels $C_i(M-1)$ are then the minimum travel costs for all departures taking place at a time greater or equal to $M - 1$. Moreover, solving for these labels is equivalent to solving a static shortest paths problem using $c_{ij}(M-1)$ as link distances.

Static shortest paths algorithms to be used to solve for $C_i(M-1)$ depend on the assumptions on $c_{ij}(M-1)$. An assumption required by all static shortest paths algorithms is that there is no negative cycle in the network; otherwise, one can circulate an infinite number of times leading to an infinite decrease in label values. Label setting algorithms require that $c_{ij}(M-1)$ are nonnegative. Label correcting algorithms work even for nonnegative link distances. The algorithm proposed below assumes the availability of an appropriate static shortest paths procedure. If many alternatives are available, one should use a procedure with the fastest run time if the objective is to obtain algorithms with optimal run time.

An extension of algorithm DOT is proposed for minimum cost paths and is based on the result of Proposition 8. The main loop of the algorithm is carried out in decreasing order of time. The algorithm assumes that a valid static shortest paths procedure, with an optimal running time SSP, is available. This is not a restriction because very efficient static shortest path solvers are widely available and are based on a vast body of research results published during the last 4 decades. The algorithm DOT for minimum cost paths is as follows:

```
Step 0 (Initialization):

$C_i(t) = \infty, \forall(i \neq q), C_q(t) = 0, \forall(t < M-1)$

$C_i(M-1) = \text{Static Shortest Paths}\left(c_{ij}(M-1), q\right)\forall(i)$

where $C_i(t) = C_i(M-1), \forall(t \geq M-1, i)$

Step 2 (Main Loop):

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

 For $(i, j) \in A$ do:

 $C_i(t) = \min\left(C_i(t), c_{ij}(t) + C_j\left(t + d_{ij}(t)\right)\right)$
```

Proposition 9 and Proposition 10 respectively generalize Proposition 6 and Proposition 7. The proofs are not provided because these are straightforward generalizations of respective proofs given in Section 4.

Proposition 9: Algorithm DOT solves for the all-to-one minimum cost paths problem, with a serial computation time in $\theta(\text{SSP} + nM + mM)$.

Proposition 10: The complexity of the all-to-one shortest paths problem, for all starting times, is in $\Omega(\text{SSP} + nM + mM)$. Hence, the above extension of algorithm DOT has an optimal running time complexity (no algorithm with a better running time complexity can be found).

COMPUTER IMPLEMENTATIONS AND EXPERIMENTAL EVALUATION

The four implemented solution algorithms for the all-to-one fastest paths problem are algorithm DOT and three dynamic adaptations of label-correcting algorithms using three types of data structures for node candidates list: the Deque data structure (*10*) as described in Ziliaskopoulos and Mahmassani (*6*), the two-queue data structure (11), and a three-queue data structure that extends the notion of the two-queue data structure (*12*). All algorithms were coded in C++.

Algorithm DOT has very attractive properties. First, its computer coding is a very easy task. Second, algorithm DOT does not need complicated data structures; a basic two-dimensional array suffices.

TABLE 1 Computation Times for 3,000 Nodes, 9,000 Links, and Different Time Intervals for All-to-One Fastest Paths Problem

M	Deque	2-Q	3-Q	DOT
30	0.39	0.40	0.40	0.21
50	0.65	0.65	0.66	0.30
70	0.89	0.92	0.91	0.41
90	1.05	1.06	1.16	0.51
110	1.31	1.30	1.31	0.69
130	1.55	1.53	1.53	0.76
150	1.74	1.74	1.74	0.90

NOTE: Times are given in seconds.

Finally, algorithm DOT has an optimal running time and does not suffer from worst-case behavior that is typical to other solution algorithms such as label-correcting algorithms.

An extensive evaluation of the algorithms was carried out. Other known algorithms were implemented and evaluated. Label-correcting algorithms generally had the closest performances to algorithm DOT.

Performance of label-correcting algorithms depends on network topology and the dynamics of link travel times. These algorithms present best-case as well as worst-case behaviors. Tables 1, 2, and 3 show results for instances of the problem in which label-correcting algorithms gave their best computation times. These instances correspond to very sparse networks that contain fewer cycles and then are less difficult to solve by using label-correcting algorithms. These instances were generated by using a discrete dynamic network generator that was developed to test computer codes on networks having different topologies, number of cycles, densities, and link travel times. Running times are wall clock times obtained on an SGI Indy workstation.

To give an indication of how fast algorithm DOT is, Step 2 typically requires only three times of what is required as computing time to initialize node labels to infinity on a network composed of 3,000 nodes and 9,000 arcs.

As may be expected from the above theoretical analysis, algorithm DOT has proved to be better than label-correcting algorithms. Because numerical results presented here correspond to best cases of label-correcting algorithms, algorithm DOT always should perform better than label-correcting methods.

TABLE 2 Computation Times for 800 Nodes, 60 Times Steps, and Varying Number of Links for All-to-One Fastest Paths Problem

m	Deque	2-Q	3-Q	DOT
799	0.07	0.05	0.05	0.05
2799	0.26	0.25	0.26	0.10
4799	0.48	0.48	0.47	0.15
6799	0.66	0.66	0.67	0.21
9799	1.03	1.02	1.01	0.28
12799	1.27	1.28	1.26	0.35
14799	1.40	1.41	1.39	0.40

NOTE: Times are given in seconds.

TABLE 3 Computation Times for 10,000 Links, 60 Times Steps, and Varying Number of Nodes for All-to-One Fastest Paths Problem

n	Deque	2-Q	3-Q	DOT
2000	1.34	1.37	1.34	0.38
4000	1.07	1.11	1.09	0.50
6000	0.86	0.88	0.71	0.60
8000	0.76	0.78	0.77	0.76

NOTE: Times are given in seconds.

Table 1 and 2 show that algorithm run times are linear in the number of nodes and links. For algorithm DOT this is a general behavior as demonstrated theoretically above. Label-correcting methods may lead to a higher than linear running time increase as a function of the number of nodes and links. The objective here was not to analyze label-correcting algorithms but instead to evaluate the relative performance of algorithm DOT and label-correcting methods. The results in Table 1 and 2 are consistent with the objective to report only on problem instances in which label-correcting algorithms gave their best run times. (The objective was not to compare algorithm DOT to worse-case behaviors of label-correcting algorithms).

Results of Table 3 show that the run times of label-correcting algorithms decrease as a function of the number of nodes. At first glance, this may seem incorrect. However, because the number of links is constant, as the number of nodes increases the network tends toward a tree. This led to a reduction in the number of node revisits and hence led to a reduction in running time. Algorithm DOT, conversely, has an increasing run time as a function of the number of nodes, as shown theoretically.

Table 4 compares two versions of algorithm DOT: the fastest paths version and the minimum cost paths version. As may be expected, run times are almost similar. The latter version takes an extra small fraction of run time. This is attributable to the extra time required to manage the link travel costs array.

Algorithm DOT also was experimented with on a real-life network emanating from the highway network of the city of Amsterdam. The objective was to test the effects on run times of the length of discretization intervals. The travel times used were obtained from the output results of an analytical dynamic traffic assignment model developed by Chabini and He (*13*) and He (*14*). Table 5 summarizes the results of this experience. Table 5 shows that run time of algorithm DOT is proportional to the number of time intervals or, equiv-

TABLE 4 Computation Times for 10,000 Links, 60 Times Steps, and Varying Number of Nodes of Algorithm DOT for Fastest and Minimum Cost Path Problems

n	DOT fastest	DOT minimum cost
2000	0.38	0.39
4000	0.50	0.54
6000	0.60	0.62
8000	0.76	0.78

NOTE: Times are given in seconds.

TABLE 5 Computation Times for Amsterdam Highway Network with 196 Nodes, 308 Links, and Analysis Period of 2.75 Hours

M	DOT	M	DOT
33	0.07	660	1.48
165	0.37	990	2.22
330	0.75	1980	4.40

NOTE: Times are given in seconds.

alently, is inversely proportional to the length of a time interval for a given analysis-period length.

CONCLUSIONS

Two types of shortest paths problems in discrete dynamic networks were studied: the one-to-all for one departure time interval and the all-to-one for all departure time intervals. Early results were revisited. New properties were established and exploited in designing solution algorithms. A new solution algorithm was proposed for all-to-one all departure time intervals, fastest and minimum cost path problems. It was proved, analytically, that the proposed algorithm is the most efficient solution method possible. Extensive experimental evaluation supports analytical results. It is expected that these findings will be of major benefit to research and development activities in the area of dynamic management of transportation systems. On an SGI Indy workstation, the new algorithm, algorithm DOT, computes shortest paths from all nodes to one destination node for all departure times within 0.5 sec for dynamic networks composed of 3,000 nodes, 9,000 arcs, and 90 times intervals. The number of destination nodes for such transportation networks typically is in the order of 300. Hence, the determination of shortest paths would require 2.5 min of computation time on an SGI Indy workstation. To achieve faster run time, high performance implementations of algorithm DOT are the subject of ongoing research.

REFERENCES

1. Deo, N., and C. Y. Pang. Shortest Path Algorithms: Taxonomy and Annotation. *Networks,* Vol. 14, 1984, pp. 275–323.
2. Chabini, I. A New Short Paths Algorithm for Discrete Dynamic Networks. *Proc., 8th IFAC Symposium on Transport Systems,* 1997, pp. 551–556.
3. Orda, A., and R. Rom. Minimum Wait Paths in Time-Dependent Networks. *Networks,* Vol. 21, 1991, pp. 295–319.
4. Cooke, K. L., and E. Halsey. The Shortest Route Through a Network with Time-Dependent Internodal Transit Times. *Journal of Mathematical Analysis Applications,* Vol. 14, 1966, pp. 493–498.
5. Kaufman, D. E., and R. L. Smith. Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. *IVHS Journal,* Vol. 1, 1993, pp. 1–11.
6. Ziliaskopoulos, A. K., and H. S. Mahmassani. Time-Dependent Shortest Path Algorithm for Real-Time Intelligent Vehicle/Highway System. In *Transportation Research Record 1408,* TRB, National Research Council, Washington, D.C., 1993, pp. 94–104.
7. Dreyfus, S. E., An Appraisal of Some Shortest-Path Algorithms. *Operations Research,* Vol. 17, 1969, pp. 395–412.
8. Dijkstra, E. W. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik,* Vol. 1, 1959, pp. 269–271.
9. Ahn, B. H., and J. Y. Shin. Vehicle Routing with Time Windows and Time-Varying Congestion. *Journal of the Operational Research Society,* Vol. 42, 1991, pp. 393–400.
10. Pape, U. Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem. *Mathematical Programming,* Vol. 7, 1974, pp. 212–222.
11. Gallo, G., and S. Pallotino. Shortest Paths Algorithms. *Annals of Operations Research,* Vol. 13, 1988, pp. 3–79.
12. Chabini, I. Fastest Paths in Dynamic Networks. *Transportation Science* (in press).
13. Chabini, I., and Y. He. An Analytical Approach to Dynamic Traffic Assignment with Applications to Intelligent Transportation Systems. Presented at Optimization Days, Montreal, May 1997.
14. He, Y. *A Flow-Based Approach to Dynamic Traffic Assignment Problem: Formulations, Algorithms and Computer Implementations.* M.S. thesis. Massachusetts Institute of Technology, Cambridge, 1997.

Publication of this paper sponsored by Committee on Transportation Network Modeling.