DISCRETE DYNAMIC SHORTEST PATH PROBLEMS IN TRANSPORTATION APPLICATIONS: COMPLEXITY AND ALGORITHMS WITH OPTIMAL RUN TIME

(Paper Acccepted for Publication in Transportation Research Records, 1997)

Ismail Chabini

Massachusetts Institute of Technology
Department of Civil and Environmental Engineering
77 Massachusetts Avenue, Room 1-138
Cambridge, MA 02139-4307, U.S.A.
Email: chabini@mit.edu
Tel.: (617) 253-0464
Fax: (617) 258-8073

Abstract: This paper solves what appears to be a 30 years old problem dealing with the discovery of 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. While the main objective of this paper is the study of the all-to-one dynamic shortest paths problem, one-to-all fastest paths problems are studied as well. Early results are revisited and new properties are established. We establish the exact complexity of these problems and develop optimal, in the run time sense, solution algorithms. A new and simple solution algorithm is proposed for all-to-one all departure time intervals shortest path 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. We expect our findings to be of major benefit to research and development activities in the field of dynamic, in particular real-time, management and control of large-scale Intelligent Transportation Systems (ITS).

Keywords: Intelligent Transportation Systems Applications, Algorithms, Networks, Computing, Computational Methods, Operations Research, Optimization, Dynamic Networks.

1. INTRODUCTION

The shortest paths problem in networks has been the subject of extensive research for many years resulting in the publication of a large number of scientific papers (Deo and Pang, 1984). The analysis of transportation networks is one of the many application areas where the computation of shortest paths is one of the most fundamental problems.The vast majority of published research on shortest paths algorithms dealt, however, with static networks that have fixed topology and fixed link costs.

In recent years, there has been an increasing interest in the concept of dynamic management of transportation systems. These new advances have brought renewed interest in the study of shortest paths problems with, however, 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 (1997) distinguishes various types of dynamic shortest path problems depending on the following: (1) Fastest vs. minimum cost (or shortest) path problems. (2) Discrete vs. continuous representation of time. (3) FIFO networks vs. non-FIFO networks, where one can depart later at the beginning of one or more arcs and arrive earlier at their end. (4) waiting is allowed vs. waiting is not allowed at nodes. (5) 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. (6) Integer vs. 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 path problems, link costs can be of general form. While 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 further subdivided into two types: discrete and continuous. In discrete dynamic networks, time is modelled as a set of integers. In continuous dynamic networks (Orda and Rom, 1991), time is treated as real numbers.

It is important to mention that a discrete dynamic network can alternatively be viewed 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 one may expect, 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.This paper provides answers to such questions.

Compared to static shortest paths problems, the literature on dynamic versions, surprisingly, 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 First-In-First-Out (FIFO) condition (a mathematical definition is given in the next section). 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 labelling algorithms only. Transportation applications need the computation of fastest paths from all nodes to a set of destinations. Hence, static methods would typically first compute fastest paths from all nodes to all nodes and then extract needed solutions. Since the number of destinations in a transportation network is usually 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, while shortest paths for all possible departure time intervals are generally sought.

Traffic Management Centers in Intelligent Transportation Systems, must operate in real-time. The time taken to collect data, process it and broadcast the resulting information may constitute a possible information bottleneck. To avoid this bottleneck, the ITS models and algorithms must run in a time 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 25000 links and 7000 nodes. The time dimension depends on the length of analysis period and the discretization interval. If 15 second time discretization is adopted, 480 times intervals are required to model a 2 hour morning peak period.

Given the above key issues in ITS and the importance of shortest paths in ITS models and algorithms, one then need to find the most efficient methods to solve shortest path problems in dynamic networks. While other researchers have made valuable contributions in addressing this problem, the following question remained answered for the last three 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 paper is to answer this question.

In this paper, we study three types of shortest path problems in discrete dynamic networks: 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 so far published papers on these topics have focused mainly on problems of first type. Our objective in re-studying the one-to-all version of the problem is essentially to revisit and extend early results. These are established by using alternative arguments which we believe 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. We propose a new all-to-one discrete dynamic shortest paths algorithm that is demonstrated to be the most efficient solution algorithm possible. Numerical tests on large networks support our 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 in the paper by Cooke and Halsey (1966).

This paper is organized as follows. In next section, we introduce some notation and give some definitions. In section 3, we study the one-to-all fastest paths problem under different waiting scenarios at nodes.This study is based on an appropriate formulation of the problem. In section 4, we revisit a well known formulation of the all-to-one fastest paths problem and show that it has some interesting properties. These are then exploited to design a Decreased-Order of Time (DOT) solution algorithm. The results of section 4 are extended to the minimum cost paths problem in section 5. We present a theoretical evaluation of algorithm DOT and prove its optimality in the sense of computational complexity. Section 6 presents results of an experimental evaluation of computer implementations of algorithm DOT and of other algorithms based on other dynamic adaptations of static labelling algorithms.

2. DEFINITIONS AND NOTATION

Let $\mathbf{G} = (\mathbf{N}, \mathbf{A}, \mathbf{D}, \mathbf{C})$ be a directed network where $\mathbf{N} = \{\mathbf{1}, \ldots, \mathbf{n}\}$ is the set of nodes and $\mathbf{A} = \{(\mathbf{i}, \mathbf{j}) \in \mathbf{NxN}\}$ is the set of arcs (or links). The number of links (arcs) is denoted by $\mathbf{m}$. We denote by $\mathbf{D} = \{\mathbf{d_{ij}}(\mathbf{t}) \mid (\mathbf{i}, \mathbf{j}) \in \mathbf{A}\}$ the set of time-dependent link travel times and by $\mathbf{C} = \{\mathbf{c_{ij}}(\mathbf{t}) \mid (\mathbf{i}, \mathbf{j}) \in \mathbf{A}\}$ the set of time-dependent link travel costs. Functions $\mathbf{d_{ij}}(\mathbf{t})$ have integer-valued domain and positive integer-valued range. A function $\mathbf{d_{ij}}(\mathbf{t})$ is then a discrete and time-dependent function which, we assume, takes a static value after a finite number of time intervals $\mathbf{M}$. $\mathbf{S} = \{\mathbf{0}, \ldots, \mathbf{M-1}\}$ is the set of departure time intervals for which link travel times are time-dependent. Functions $\mathbf{c_{ij}}(\mathbf{t})$ have real-valued range and integer-valued domain. $\mathbf{c_{ij}}(\mathbf{t})$ is static when the departure time is greater or equal to $\mathbf{M-1}$. We assume that there is no negative cycle after departure time interval $\mathbf{M-1}$. $\mathbf{B}(\mathbf{i})$ denotes the set of nodes having an outgoing arc to node $\mathbf{i}$ and $\mathbf{A}(\mathbf{i})$ denotes the set of nodes having an ingoing arc from node $\mathbf{i}$. $\mathbf{G} = (\mathbf{N}, \mathbf{A}, \mathbf{D}, \mathbf{C})$ 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 First-In-First-Out (FIFO) condition (Kaufman and Smith, 1993) 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:

$$\forall(\mathbf{i}, \mathbf{j}, \mathbf{t}), \quad \mathbf{t} + \mathbf{d_{ij}}(\mathbf{t}) \le (\mathbf{t}+\mathbf{1}) + \mathbf{d_{ij}}(\mathbf{t}+\mathbf{1}) \quad .$$

When the FIFO condition holds, we say that the dynamic network is FIFO. The FIFO condition is also known as the non-overtaking 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, it may sometimes 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 (WA)* and *waiting is not allowed at nodes (WN).* In the next section, we will show that in studying fastest path 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. In this paper we concentrate on two questions: *question (1)* what are the shortest paths from one origin to all destinations departing at instant 0? and *question (2)* 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. To the best of our knowledge, most of published work dealt, however, with the first question with the exception of the work by Cooke and Halsey (1966) and Ziliaskopoulos and Mahmassani (1993).

Although algorithms that answer question (1) can be used to answer question (2) as well, such approaches would not be efficient for transportation networks since 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% of which are destination nodes. Hence, even if $\mathbf{M}$ answers to question (1) would take as much time as one answer to question (2) (we will see that this is a very optimistic hypothesis), a procedure which is based on answers to question (1) would be 10 times slower that one which is based on answers to question (2).

3. FORMULATIONS AND ALGORITHMS FOR THE ONE-TO-ALL FASTEST PATHS PROBLEM

In this section we consider the computation of fastest paths from one origin node, departing at time interval 0, to all other nodes. We analyse the problem for the two types of waiting-at-node policies defined above.

3.1 Waiting at nodes is not allowed

This is the most studied variation of the problem (Cook and Halsey, 1966; Dreyfus, 1969; Kaufman and Smith, 1993). The most celebrated result for this variation of the problem is the following: when the FIFO condition is valid, Dijkstra's algorithm can be generalized to solve the time-dependent fastest paths problem with the same time complexity as the static shortest paths problem. Dreyfus (1969) is the first to mention this generalization. Later Kaufman and Smith (1993) formally proved that this generalization is valid only if the FIFO condition is satisfied. During our literature review, we found an even earlier proof due to Ahn and Shin (1991).

Below we present yet another proof which we feel is more insightful and simpler. Our proof is based on a different formulation of the problem. Let $\mathbf{f_j}$ denote the minimum travel time from origin node $\mathbf{o}$ to node $\mathbf{j}$ leaving the origin node at time interval 0. The key idea is to consider, when writing optimality conditions for node $\mathbf{j}$, only those paths that visit previous node $\mathbf{i}$ at a time greater or equal to $\mathbf{f_i}$. Minimum travel times are then defined by the following functional form:

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

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

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

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 algorithm can be extended, at no extra time, to solve the fastest paths problem if the FIFO condition holds. We summarize this result 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 (1969), and later proved by Kaufman and Smith (1993) and by Ahn and Shin (1991) can be viewed as a particular case of proposition 2. Note that these generalizations are subject to the following restrictions: only a static forward labelling 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, is published yet. Based on our formulation, we have designed such type of solution algorithms. We will report on them in a forthcoming paper.

3.2 Waiting at nodes is allowed

We now turn our attention to the case when waiting is allowed at nodes. For editorial reasons, we analyse the case when waiting is allowed at all nodes (results can be generalized if waiting is allowed at a subset of nodes only). The main result of this subsection is 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 $\mathbf{w_i(t)}$ the maximum waiting time allowed at node $\mathbf{i}$ and at departure time interval $\mathbf{t}$. First note that when waiting is allowed at node $\mathbf{i}$, the minimum travel time possible on arc $(\mathbf{i}, \mathbf{j})$ is given by function $\mathbf{D_{ij}(t)} = \min_{w_i(t) + t \geq s \geq t} (s - t + d_{ij}(s))$. It is easy to prove that this function verifies the FIFO condition if unlimited waiting is allowed.

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

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

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 $d_{ij}(t)$). Since $D_{ij}(t)$ verifies the FIFO condition if unlimited waiting is allowed, results of propositions 1, 2 and 3 hold.

3.3 Conclusion

A main conclusion of this section is that 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.

4. THE ALL-TO-ONE FOR ALL DEPARTURE TIMES FASTEST PATHS PROBLEM

Since we demonstrated that the waiting-is-allowed policy is conceptually a particular case of the waiting-is-not-allowed policy, the remainder of this paper is dedicated to studying the latter only. As showed 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. Below, we present a backward star formulation of this problem, when waiting is not allowed, and establish a property that is then exploited in developing a new solution algorithm.

4.1 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:

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

This is a well known optimality condition that was used by Cooke and Halsey (1966), and by Ziliaskopoulos and Mahmassani (1993) to design solution algorithms that have respectively $O(n^3M^2)$ and $O(nmM^2)$ as worst case running time complexities. Below we present an optimal algorithm for this problem, based on the following proposition.

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

Proof: Since all arc travel times are positive integers, labels corresponding to time step t never update labels corresponding to time steps greater than t (see equations above).

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

4.2 A Decreasing Order of Time (DOT) Algorithm

Note that for departure time intervals greater than or equal to $M - 1$, the computation of fastest paths is equivalent to a static shortest paths problem. Below we propose an algorithm which 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.

Algorithm DOT

Step 0 (Initialization):

```
$\pi_i(t) = \infty, \forall (i \neq q), \pi_q(t) = 0, \forall (t < M-1)$
$\pi_i(M-1) = \mathbf{StaticShortestPaths}(d_{ij}(M-1), q) \forall (i)$
($\mathbf{Note:}$ $\pi_i(t) = \pi_i(M-1), \forall (t \geq M-1, i)$)
```

Step 2 (Main Loop):

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

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

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*

$\Omega(\mathbf{SSP} + \mathbf{nM} + \mathbf{mM})$. *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 since every solution algorithm has to: access all arc data ($\mathbf{mM}$), initialize network node labels because fastest paths for all departure times are sought ($\mathbf{nM}$), and compute an all-to-one fastest paths tree for departure time intervals beyond departure time $\mathbf{M-1}$ ($\mathbf{SSP}$). Using results of proposition 6, algorithm DOT is then optimal.

5. THE ALL-TO-ONE FOR ALL DEPARTURE TIMES MINIMUM COST PATHS PROBLEM

The objective of this section is to extend results established in previous section, to the minimum cost paths problem. We first show a formulation of the problem. Then we use it to determine the exact complexity of the problem and to design an efficient solution algorithm that generalizes algorithm DOT.

5.1 Formulation

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

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

These equations extend the optimality conditions described in previous section for the fastest paths problem. Below we use them to extend results established for the fastest paths problem. In particular, we will determine the exact complexity of the minimum cost paths problem and show an extension of algorithm DOT that runs in an optimal time.

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

Proof: Since all arc travel times are positive integers, labels corresponding to time steps $\mathbf{t}$ do not depend on labels corresponding to time intervals smaller than $\mathbf{t}$ (see equations above).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.

5.2 Extension of Algorithm DOT to Compute Minimum Cost Paths

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

Static shortest paths algorithms to be used to solve for $\mathbf{C_i(M-1)}$ depend on the assumptions on $\mathbf{c_{ij}(M-1)}$. An assumption required by all static shortest path 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 $\mathbf{c_{ij}(M-1)}$ are non-negative. Label correcting algorithms work even for non-negative link distances. The algorithm that we propose below assume the availability of an appropriate static shortest paths procedure. If many alternatives are available, one then should use a procedure with the fastest run time if the objective is to obtain algorithms with optimal run time.

Below we propose an extension of Algorithm DOT, which 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 since very efficient static shortest path solvers are widely available and are based on a vast body of research results published during the last four decades.

Algorithm DOT for Minimum Cost Paths

```
Step 0 (Initialization):
$\mathbf{C_i(t) = \infty, \forall(i \neq q), C_q(t) = 0, \forall(t < M-1)}$
$\mathbf{C_i(M-1) = StaticShortestPaths(c_{ij}(M-1), q) \forall(i)}$
$\mathbf{(Note: C_i(t) = C_i(M-1), \forall(t \geq M-1, i))}$

Step 2 (Main Loop):

For $\mathbf{t = M-2}$ down to 0 do:
For $\mathbf{(i, j) \in A}$ do:
$\mathbf{C_i(t) = min(C_i(t), c_{ij}(t) + C_j(t + d_{ij}(t)))}$
```

Proposition 9 and Proposition 10 below respectively generalize Proposition 6 ad Proposition 7. We do not provide the proofs since 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* $\theta(\mathbf{SSP} + \mathbf{nM} + \mathbf{mM})$.

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

6. COMPUTER IMPLEMENTATIONS AND EXPERIMENTAL EVALUATION

In this section we report on implementations of five solution algorithms: four algorithms that compute for the all-to-one fastest paths problem and the extension of algorithm DOT that solves for all-to-one minimum cost paths. The four implemented solution algorithms for the all-to-one fastest paths problem are: algorithm DOT introduced in section 4 and three dynamic adaptations of label correcting algorithms using three types of data structures for node candidates list: the Deque data structure (Papa, 1974) as described in Ziliaskopoulos and Mahmassani (1993), the 2-queue data structure (Gallo and Pallotino,1988) and a 3-queue data structure that extends the notion of the 2-queue data structure (Chabini, 1997b). All algorithms were coded in C++. Codes are available upon request.

Algorithm DOT has very attractive properties. First, its computer coding is a very easy task (if one has a static shortest paths code, it should take him/her less time to code algorithm DOT than to read this paper!). Second, algorithm DOT does not need complicated data structures: a basic two-dimensional array suffices. Finally algorithm DOT has an optimal running time and does not suffer from worst case behaviour that is typical to other solution algorithms such as label correcting algorithms.

An extensive evaluation of above algorithms was carried out. Computational results will be included in a longer version of this paper. We also implemented other known algorithms and evaluated them. Label correcting algorithms had generally the closest performances to algorithm DOT. We then only report on the comparison of algorithm DOT to label correcting methods.

Performance of label correcting algorithms depend on network topology and the dynamics of link travel times. These algorithms present best as well as worst case behaviours. Tables 1, 2 and 3 show results for instances of the problem where label correcting algorithms gave their best computation times. These instances correspond to very sparse networks that contain less cycles and are then less difficult to be solved by label correcting algorithms. These instances were generated by using a discrete dynamic network generator which 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 3000 nodes and 9000 arcs.

As one may expect from the above theoretical analysis, algorithm DOT has proven to be better than label correcting algorithms. Since numerical results presented here correspond to best cases of label correcting algorithms, algorithm DOT should perform always better than label correcting methods.

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 behaviour as demonstrated theoretically in Sections4 and 5. Label correcting methods may lead to a higher than linear running time increase as a function of the number of nodes and links. In this paper our objective is 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 then consistent with our choice to only report on problem instances where label correcting algorithms gave their best run times (The objective is not to compare algorithm DOT to worse case behaviours 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 a first glance, this may seem incorrect. This is explained as follows. Since the number of links is constant, as the number of nodes increases the network “tends toward a tree”. This lead to a reduction in the number of node revisits and hence lead to a reduction in running time. Algorithm DOT on the other hand 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 one may expect, 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.

We also experimented with algorithm DOT on a real-life network emanating from the highway network of the city of Amsterdam. The objective here was to test the effects on run times of the length of discretization intervals. The travel times used are obtained from the output results of an analytical dynamic traffic assignment model developed by the author and his collaborators (Chabini and Yiyi (1997) and Yiyi (1997)). 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, equivalently, is inversely proportional to the length of a time interval for a given analysis-period length.

7. CONCLUSION

In this paper two types of shortest paths problems in discrete dynamic networks are studied: the one-to-all for one departure time interval and the all-to-one for all departure time intervals. Early results are revisited. New properties are established and exploited in designing solution algorithms. A new solution algorithm is proposed for all-to-one all departure time intervals, fastest and minimum cost path problems. It is 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 3000 nodes, 9000 arcs and 90 times intervals. The number of destination nodes for such transportation networks is typically in the order of 300. Hence, the determination of shortest paths would require 2.50 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. We will report on this in a forthcoming paper.

8. REFERENCES

Ahn, B.H. and J.Y. Shin (1991). Vehicle Routeing with Time Windows and Time-varying Congestion. *J. Opl Res. Soc.*, **42**, 393-400.

Chabini, I. (1997a). A New Short Paths Algorithm for Discrete Dynamic Networks. Appeared in the *in the proceeding of 8th IFAC Symposium on Transport Systems*, pp. 551-556.

Chabini, I. (1997b). Fastest Paths in Dynamic Networks. Submitted to *Transportation Science*.

Chabini, I. and M. Florian (1995). High Performance Computation of Temporal Shortest Routes for Intelligent Transportation Systems Applications. Proceedings of *The Second World Congress on Intelligent Transportation Systems*, pp. 1971-1976.

Chabini, I., M. Florian and E. Le Saux (1997). Parallel and Distributed Computation of Shortest Paths and Network Equilibrium Models" appeared *in the proceeding of 8th IFAC Symposium on Transport Systems*, pp. 1328-1333.

Chabini, I. and Y. He (1997). An Analytical Approach to Dynamic Traffic Assignment with Applications to Intelligent Transportation Systems. Paper presented at *Optimization Days*, Montreal, May 12-14.

Cooke, K.L. and E. Halsey (1966). The shortest Route Through a Network with Time-Dependent Internodal Transit Times. *J. Math. Anal. Appl.*, **14**, 493-498.

Deo, N. and C.Y. Pang (1984). Shortest Path Algorithms: Taxonomy and annotation. *Networks*, **14**, 275-323.

Dijkstra, E.W. (1959). A Note on Two Problems in Connection with Graphs. *Numer. Math.*,**1**, 269-271.

Dreyfus, S. E. (1969). An Appraisal of Some Shortest-Path Algorithms. *Operations Research*, **17**, 395-412.

Gallo, G. and S. Pallotino (1988). Shortest Paths Algorithms. *Annals of Oper. Res.*, **13**, 3-79.

He, Y. (1997). A Flow-Based Approach to Dynamic Traffic Assignment Problem: Formulations, Algroithms and Computer Implementations. Master of Science Thesis, MIT (Supervisor: Prof. Ismail Chabini).

Kaufman, D. E. and R.L. Smith (1993). Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application. *IVHS Journal*, **1**, 1-11.

Orda, A. and R. Rom (1991). Minimum Wait Paths in Time-Dependent Networks. *Networks*, **21**, 295-319.

Pape, U. (1974). Implementation and Efficiency of Moore-Algorithms for the Shortest Route

Problem. *Mathematical Programming*, **7**, 212-222.

Ziliaskopoulos, A. K. and H. S. Mahmassani (1993). A Time-Dependent Shortest Path Algorithm for Real-Time Intelligent Vehicle/ Highway System. *Transportation Research Record* **1408**, 94-104.

TABLE 1. Computation times in sec. on an SGI Indy workstation, for 3000 nodes, 9000 links and different time intervals, for the 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

TABLE 2. Computation times in sec. on an SGI Indy workstation, for 800 nodes, 60 times steps and varying number of links, for the 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

TABLE 3. Computation times in sec. on an SGI Indy workstation, for 10000 links, 60 time steps and varying number of nodes, for the 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

TABLE 4. Computation times in sec. on an SGI Indy workstation, for 10000 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

TABLE 5. Computation times in sec. on an SGI Indy workstation, for a real highway network of Amsterdam with 196 nodes, 308 links and an analysis period of 2.75 hours. We vary the discretization interval

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