DYNAMIC SHORTEST PATHS IN ACYCLIC NETWORKS WITH MARKOVIAN ARC COSTS

HARILAOS N. PSARAFTIS

National Technical University of Athens, Athens, Greece

JOHN N. TSITSIKLIS

Massachusetts Institute of Technology, Cambridge, Massachusetts

(Received August 1990; revisions received March, October 1991; accepted December 1991)

We examine shortest path problems in acyclic networks in which arc costs are known functions of certain *environment variables* at network nodes. Each of these variables evolves according to an independent Markov process. The vehicle can wait at a node (at a cost) in anticipation of more favorable arc costs. We first develop two recursive procedures for the individual arc case, one based on successive approximations, and the other on policy iteration. We also solve the same problem via parametric linear programming. We show that the optimal policy essentially classifies the state of the environment variable at a node into two categories: *green* states for which the optimal action is to immediately traverse the arc, and *red* states for which the optimal action is to wait. We then extend these concepts for the entire network by developing a dynamic programming procedure that solves the corresponding problem. The complexity of this method is shown to be $O(n^2K + nK^3)$, where n is the number of network nodes and K is the number of Markov states at each node. We present examples and discuss possible research extensions.

In this paper, we examine shortest path problems in a stochastic and dynamic setting. We assume a directed acyclic graph $G = (N, A)$. A vehicle plans to traverse G, starting from a specified node 1, and ending at another specified node n. Assume also that arc-traversal costs on G are stochastic and dynamic, in the following sense. The cost of traversing each arc (i, j) of G is a known function $f_{ij}(e^i)$ of the state e^i of a certain *environment variable* at node i at the time the vehicle departs from node i on its way to node j. Environment variables are mutually independent, and each is governed by a finite-state Markov process of a known transition probability matrix. State transitions occur in discrete time. The actual state e^i of the environment variable at node i is revealed to the vehicle only when it is at node i (and if it chooses to visit that node). Once at node i, the vehicle either may immediately depart from i toward some other node j and incur the cost $f_{ij}(e^i)$ associated with the prevailing state e^i at departure time, or, it may choose to wait, in anticipation of a "more favorable" environment state at i. Waiting at a node may last as long as desired, but it will cost the vehicle an amount of C per state transition. Based on the above, what is the policy that minimizes the expected total cost of a traversal from node 1 to node n?

Some further clarifications about this problem are in order:

1. We assume that the number of possible states of the environment variable at each node is K (the assumption that K is the same for all nodes is not needed in what follows, and is only made for notational convenience). We denote these states as $e_1^i, e_2^i, \ldots, e_K^i$. We also define p_{mk}^i as the probability of a direct transition of e^i from state e_m^i to e_k^i. All transition matrices $[p_{mk}^i]$ are assumed to be known.

2. All Markov processes are assumed to have the ergodic property, that is, any state can be reached from any other state in a finite number of transitions. We define as Π_m^i the steady-state probability that the environment at node i is in state e_m^i (this probability can be calculated from matrix $[p_{mk}^i]$). The Markov processes at different nodes are assumed to be statistically independent and in steady state.

Subject classifications Networks/graphs, distance algorithms: Markovian cost structure. Networks/graphs, stochastic: dynamic shortest paths. Transportation, models, network: dynamic and stochastic problems.

Area of review DISTRIBUTION, TRANSPORTATION AND LOGISTICS (SPECIAL ISSUE ON STOCHASTIC AND DYNAMIC MODELS IN TRANSPORTATION).

0030-364X/93/4101-0091 $01.25

© 1993 Operations Research Society of America

Copyright © 2001 All Rights Reserved

3. We assume that C is positive to avoid the "singular" solution of indefinitely waiting, which would be optimal if $C = 0$ and the arc costs were nonnegative (as with K, the assumption that C is the same for all nodes is again nonbinding and was made only for notational convenience).

4. The assumption of an acyclic network means that there is a numbering of the nodes such that if there exists an arc from node i to node j, then $i < j$. A discussion of what might happen if this assumption is relaxed is presented in Section 3. It is useful to define for each node i as $J(i)$ the set of nodes j (with $j > i$) that are endpoints of arcs emanating from node i.

Perhaps the earliest work in which the problem of shortest paths in random graphs was treated can be attributed to Frank (1969), where, among other things, a procedure for obtaining the probability distribution of the cost of the shortest path was outlined. Since that time, however, the related literature has been scant, and, to our knowledge, there has been little or nothing on the specific problem class considered in this paper (dynamic shortest path policies in the presence of Markovian arc costs). Nevertheless, we mention that in the general context of random arc costs several researchers have considered the problem of finding the path with the maximum probability of being the shortest, or the minimum variance path (Andreatta, Ricaldone and Romeo 1985, Frieze and Grimmet 1985). In addition, Jaillet (1989) considered shortest path problems in the presence of node failures, that is, problems in which the nodes of the network may or may not be operable. The problem examined by Hall (1986), on the fastest path through a network with random, time-dependent travel times also belongs to the same general category of problems. With the exception of the work of Hall, who examines time-adaptive decision rules, all other versions reported above are essentially *static* because they call for the a priori selection of a *fixed* path optimizing an appropriate objective, and not for a policy specifying what should be done for each particular real-time scenario. A priori optimization is a general scheme that has also been applied in other stochastic routing problems, most notably for the Probabilistic Traveling Salesman Problem and other related problems (Jaillet 1985, Bertsimas 1988). All these routing problems are essentially static. A general classification of dynamic routing problems has been examined by Psaraftis (1988). Dynamic routing policies in a stochastic setting have been analyzed by Bertsimas and van Ryzin (1991) for the Dynamic Traveling Repairman Problem (a variation of the dynamic TSP). The focus of that paper was on the queueing aspects of the problem, and the stochasticity of the problem was the random appearance of customers in the Euclidean plane (as opposed to random arc costs).

Although the purpose of this paper is to focus only on the abstract problem defined earlier, it is interesting to note that some motivation about the problem may come from real-world contexts that generally involve the motion of a vehicle across a stochastic terrain. One real-world problem that belongs to this family concerns the routing of a ship across the ocean under uncertain and dynamically changing weather conditions. Here, graph nodes approximately represent geographical regions across which the ship will transit. The environment variable at each region is a vector of meteorological variables that describes the state of the weather in that region. Weather conditions change dynamically and stochastically in time. The cost of moving from a certain region to an adjacent region mainly consists of the cost of fuel consumed in doing so, and this is a function of the prevailing weather conditions. If this cost is high enough (due to adverse weather, a storm, etc.), it may make sense for the ship to wait (at a cost) in anticipation of more favorable weather later on.

Of course, some caveats are in order: The real-world ship weather routing problem is much more complex than the abstract one defined earlier, and may be based on different assumptions. For instance, advance information about weather conditions along the projected ship path may be available. This means that the vessel will have some information about the state of the environment in other regions of the network, and not only for the region it currently visits. Also, in general there will be a cross-coupling of environmental conditions not only in the time dimension, but also in the space dimension, and this may violate the assumption of independence among the Markov processes at different nodes. Actually, weather may not evolve according to a Markov process. Finally, the vessel may choose the speed at which it will sail (and, hence, control fuel consumption costs), and it may simply slow down instead of stopping completely if the weather is bad (see Chen 1978, and Perakis and Papadakis 1988 for a treatment of this problem).

With these important caveats in mind, we now describe how the rest of this paper is organized: Section 1 focuses on an individual arc of the network, develops policies for traversing that arc, and investigates the properties of such policies. Section 2 examines the problem on the entire network, and extends the previous procedures for the problem. Finally, Section 3

Copyright © 2001 All Rights Reserved

discusses the results of this work and examines possible extensions.

1. INDIVIDUAL ARC POLICY

Before we proceed with the problem, it is of interest to isolate an individual arc (i, j) of the network, and attempt to solve the same problem just on that arc. The solution of this special case will be an important building block for the treatment of the general case in Section 2.

Imagine the vehicle at the starting node i of the arc, and contemplate to traverse the arc to the ending node j. For this individual arc alone, the question is: What should be the vehicle's best action as a function of the state of the environment that happens to prevail at node i? Clearly, for some environment states (to be determined) the vehicle should immediately depart from i to j (we will call such states *green*—see Figure 1), while for the remaining states it should wait (we will call such states *red*).

To simplify notation, we temporarily drop indices i and j, and assume that the state of the environment variable ranges from e_1 to e_K, with the corresponding arc cost ranging from f_1 to f_K. The simplified notation for the transition probabilities is p_{mk}.

Defining X_m as the minimum achievable total expected cost of going across the arc given that the state of the environment is e_m, it is clear that there are two possible choices for the vehicle: either depart immediately, and incur a cost of f_m, or wait, and incur an immediate waiting cost of C, and a residual expected cost of $\sum_{k=1}^{K} p_{mk}X_k$. The latter is the expected sum, over each possible "next" environment state e_k, of the minimal expected cost associated with the vehicle facing a variable of state e_k after the transition, and assuming an optimal policy thereafter.

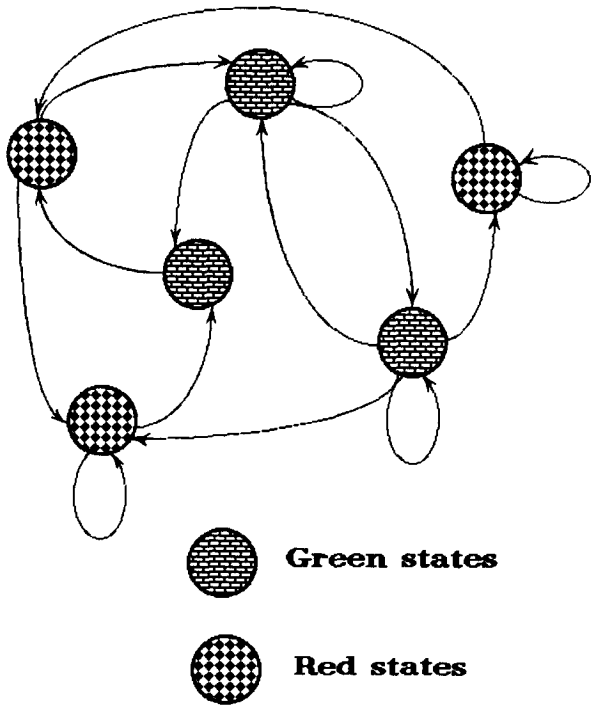

Figure 1. Green and red environment states.

Thus, it is clear that the cost-to-go variables X_m satisfy the system of nonlinear equations:

$$X_m = \min\left(f_m, C + \sum_{k=1}^{K} p_{mk}X_k\right), \quad m = 1, \ldots, K. \qquad (1)$$

The variables X_m are the unique solution of the system (1), as can be seen by applying the results of Bertsekas and Tsitsiklis (1988). The only assumption needed in order to apply these results is that any policy that waits forever has infinite cost. Our problem has this property because $C > 0$.

The set of recursive relationships implied by (1) cannot be solved in closed form in the general case. Given that the duration of the decision process implied by (1) is unbounded, the underlying problem belongs to the general class of infinite-horizon, total cost, stochastic dynamic programming problems, and, as such, can be solved by the spectrum of techniques available for this class (see Bertsekas 1987, pp. 188–205). Of those, we focus on the methods of successive approximations (**SA**), of policy iteration (**PI**), and of parametric linear programming (**PLP**). We also discuss solving the problem (whenever possible) in *closed form*.

1.1. Successive Approximations (SA)

The **SA** algorithm consists of the following steps:

STEP 1. (**Initialize**)
For all $m = 1, \ldots, K$, set $X_m := f_m$, and
ARC_COLOR(e_m) := green.

STEP 2. (**Update**)
For all $m = 1, \ldots, K$, reset

$$X_m := \min\left(f_m, C + \sum_{k=1}^{K} p_{mk}X_k\right).$$

If ARC_COLOR(e_m) = green, and if $X_m < f_m$, reset ARC_COLOR(e_m) := red.

STEP 3. (**Iterate**)
If convergence has been achieved, go to Step 4. Else go to Step 2.

Copyright © 2001 All Rights Reserved

STEP 4. **(Optimal policy)**

- If the environment variable is in state e_m and ARC_COLOR(e_m) = green, immediately traverse the arc.
- If the environment variable is in state e_m and ARC_COLOR(e_m) = red, wait for a state transition, and apply the policy to the next state.
- In all cases, the optimal expected cost given state e_m is X_m.

We mention this method because it is often used in practice for the solution of problems that belong to this general class (infinite-horizon, stochastic dynamic programming), and because for the particular problem the method has some interesting properties. These can be summarized as follows (proofs are straightforward and hence are omitted):

1. For all $m = 1, \ldots, K$, the sequence of X_m's in the iterations of **SA** is bounded and nonincreasing.
2. If in **SA** ARC_COLOR(e_m) is reset to red for a certain state e_m, that color need not be reconsidered throughout the rest of the procedure.
3. At each iteration of **SA**, there is *at least one* green state.
4. States e_m for which $C < f_m - \sum_{k=1}^{K} p_{mk} f_k$ are always red.
5. *All* states are green if $C \geq f_m - \sum_{k=1}^{K} p_{mk} f_k$ for all $m = 1, \ldots, K$. In this case, $X_m = f_m$ for all m.

A disadvantage of **SA** is that there is no guarantee for the number of iterations needed until an optimal policy is generated. The example shown in Figure 2 illustrates this point. There are three possible states, 1, 2, and 3, with the corresponding direct travel times of 1, 5, and 10 shown inside the squares in the figure. The transition probabilities are also shown.

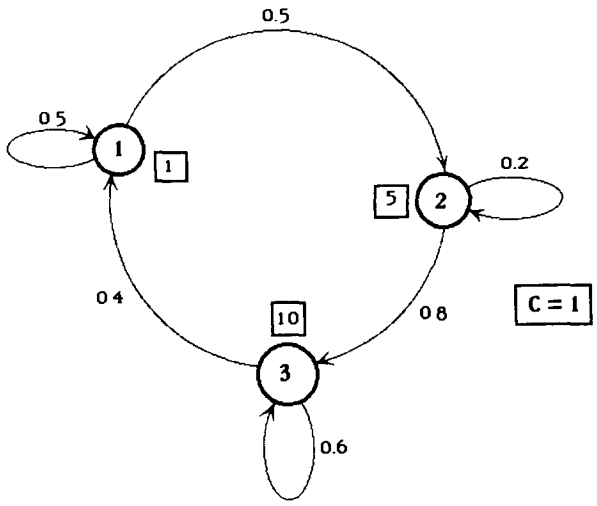

Figure 2. Example for the comparison of procedures **SA** and **PI**.

In this example, the first eleven iterations of **SA** produce the results given in Table I. If these iterations are pursued further, X_2 and X_3 will ultimately converge to 4.75 and 3.5, respectively, with all other variables already having achieved convergence. However, convergence of these two variables will be extremely slow, and the ultimate number of iterations will be unbounded if the "tolerance factor" is arbitrarily small. Notice also that the second state converges to its ultimate red color only at the ninth iteration of the algorithm, even though between iterations 2 and 8 array ARC_COLOR gives the (false) appearance of having "stabilized" to a policy of (G, G, R)! Iteration 9 shows that this is *not* the ultimate optimal policy, which is (G, R, R).

The conclusion from all this is that **SA** may achieve convergence in both arrays X and ARC_COLOR very

Table 1
Iterations of **SA**

	Array X $m =$			Array ARC_COLOR $m =$		
Iteration	1	2	3	1	2	3
1	1	5	10	G	G	G
2	1	5	7.4	G	G	R
3	1	5	5.84	G	G	R
4	1	5	4.904	G	G	R
5	1	5	4.342	G	G	R
6	1	5	4.005	G	G	R
7	1	5	3.803	G	G	R
8	1	5	3.682	G	G	R
9	1	4.946	3.609	G	R	R
10	1	4.876	3.566	G	R	R
11	1	4.828	3.539	G	R	R

Copyright © 2001 All Rights Reserved

slowly, and that even if or when convergence in policy (array ARC_COLOR) has been achieved, convergence of array X may still need a significant number of additional iterations.

The convergence shortcoming is alleviated in the policy iteration procedure **PI**, which can be shown to terminate in a fairly small number of iterations. **PI** works as follows.

1.2. Policy Iteration (PI)

The **PI** algorithm is:

STEP 1. **(Initialize)**
For all $m = 1, \ldots, K$, set ARC_COLOR(e_m) := green.

STEP 2. **(Evaluate policy)**
For all $m = 1, \ldots, K$, calculate array X by solving the system of linear equations:

$$X_m := f_m \quad \text{if ARC\_COLOR}(e_m) = \text{green, and}$$

$$X_m = C + \sum_{k=1}^{K} p_{mk} X_k \quad \text{if ARC\_COLOR}(e_m) = \text{red.}$$

STEP 3. **(Improve policy)**
For all $m = 1, \ldots, K$, calculate

$$T_m(X) := \min\left(f_m, C + \sum_{k=1}^{K} p_{mk} X_k\right).$$

If $T_m(X) < f_m$, set ARC_COLOR(e_m) := red; else set ARC_COLOR(e_m) := green.

STEP 4. **(Iterate)**
If for all $m = 1, \ldots, K$, it is $T_m(X) = X_m$, then STOP, arrays X and ARC_COLOR are optimal, and the optimal policy is as per Step 5 of **SA**. Otherwise, go to Step 2.

In contrast to **SA**, here array X is explicitly computed from the current policy (array ARC_COLOR) in Step 2 by solving a system of linear equations. Once this is done, array ARC_COLOR is updated in the policy improvement step (Step 3). In case no improvement is registered, which happens if $T_m(X) = X_m$ for all $m = 1, \ldots, K$, the algorithm terminates. Otherwise, the algorithm loops to Step 2.

Finite termination of the **PI** algorithm follows from standard results in dynamic programming. However, due to the special structure of this problem, much more can be said. In particular, once a state becomes red, it always stays red, as shown in Theorem 1; this readily leads to a complexity estimate.

Theorem 1

a. *If at some iteration of the policy iteration algorithm we have ARC_COLOR(e_m) = red, this property remains true for all subsequent iterations.*
b. *The number of iterations is at most K.*

Proof. Let X_k^* and X_k^{**} be the values of X_k at two successive iterations. It is a general property of the policy iteration algorithm that $X_k^{**} \leq X_k^*$ for all k. If ARC_COLOR(e_m) = red, then $f_m > C + \sum_{k=1}^{K} p_{mk} X_k^*$, which implies that $f_m > C + \sum_{k=1}^{K} p_{mk} X_k^{**}$ at the next iteration as well, and ARC_COLOR(e_m) is again red.

Part b follows immediately from part a. Since a red state cannot switch back to green, and since there can be at most $K - 1$ red states, there can be at most $K - 1$ policy improvements (changes in array ARC_COLOR). Given that if there is no policy improvement the algorithm terminates, the maximum number of iterations is K.

Theorem 1 leads to a complexity estimate for policy iteration. There are $O(K)$ iterations, and each consists mainly of the solution of a linear system of K equations in K unknowns, which can be accomplished with $O(K^3)$ arithmetic operations. Thus, we have a strongly polynomial time algorithm, with complexity $O(K^4)$. In contrast, no strongly polynomial algorithm is known for the general Markov decision problem.

By being a little more clever, the complexity of the algorithm can be improved further. Let n_i be the number of states that changes color at iteration i. Using part a of Theorem 1, we have $\sum_i n_i \leq K$. The system of linear equations solved in Step 2 has the general structure $X = A_i X + b_i$, where A_i is a $K \times K$ matrix and b_i is a K-dimensional vector. Changing the color of n_i states amounts to changing n_i rows of matrix A_i. Thus, $A_{i+1} - A_i$ has rank n_i. It follows that A_{i+1}^{-1} can be computed from A_i^{-1} with only $O(n_i K^2)$ arithmetic operations. For example, we can use the Sherman-Morrison-Woodbury formula (see p. 3 of Golub and van Loan 1983). Thus, the total complexity of the algorithm is reduced to $O(K^3)$, provided that Step 2 is carried out by explicitly computing the inverse of A_i.

An alternative view is provided by the well known relation between policy iteration and simplex-like methods for solving a related linear programming problem (this is formulated in the next subsection; it has K variables and $O(K)$ constraints). Under this relation, changing the policy at a single state is equivalent to a simplex iteration (pivot). Theorem 1

Copyright © 2001 All Rights Reserved

guarantees that only $O(K)$ iterations are needed, and the $O(K^3)$ complexity estimate follows.

The above discussion establishes a theoretical superiority of **PI** over **SA**, at least with respect to a bound in the number of iterations. However, an important caveat is in order. Each iteration of **SA** requires only $O(K^2)$ computations, and if an optimal policy can be produced in much less than K iterations, then **SA** can be faster. **SA** may have a further advantage if the matrix of transition probabilities is sparse, in which case the complexity of each iteration can be as low as $O(K)$.

Coming back to the previous example, **PI** can be solved in only three iterations. These are shown in Table II. Since $T(X) = X$ at iteration 3, convergence is achieved, and the algorithm terminates.

1.3. Parametric Linear Programming (PLP)

As mentioned earlier, linear programming can be used to solve infinite-horizon dynamic programming problems (Bertsekas 1987, p. 206). For our specific problem, an LP formulation would be as follows.

Problem P

$$\text{Maximize} \sum_{k=1}^{K} X_k$$

subject to

$$X_m - \sum_{k=1}^{K} p_{mk} X_k \leq C \quad \text{for } m = 1, \ldots, K,$$

$$0 \leq X_m \leq f_m \quad \text{for } m = 1, \ldots, K.$$

We then have ARC_COLOR(e_m) = green if $X_m = f_m$, otherwise ARC_COLOR(e_m) = red.

We observe that the *parametric dual simplex* method can be used to solve this linear program and to obtain an optimal policy for all positive values of C. Due to the special structure of the problem, this method admits a complexity estimate comparable to the one derived for policy iteration, as we will show next.

Let us use the notation $X_m(C)$ to denote the value of the optimal cost-to-go (the solution of **P**) as a function of C. The following result summarizes some of the properties of $X_m(C)$.

Theorem 2. *For every m, $X_m(C)$ is piecewise linear, concave, and nondecreasing.*

Proof. For every stationary policy π, let $X_m(\pi; C)$ be the expected cost-to-go starting from state e_m, as a function of C. Also, $X_m(\pi; C)$ is affine and nondecreasing in C. Note that $X_m(C) = \min_\pi X_m(\pi; C)$. It follows that $X_m(C)$ is concave and nondecreasing. Furthermore, since the set of all stationary policies is finite, we see that $X_m(C)$ is piecewise linear.

The key to the efficiency of the parametric simplex method is provided by the following theorem.

Theorem 3. *Fix a value of C and suppose that state e_m is red under an optimal policy. Then e_m is red under an optimal policy for any positive $C^* < C$.*

Proof. Since e_m is red,

$$X_m(C) = C + \sum_{k=1}^{K} p_{mk} X_k(C) < f_m.$$

Let $0 < C^* < C$. Since $X_m(C)$ is nondecreasing (Theorem 2), we have

$$C^* + \sum_{k=1}^{K} p_{mk} X_k(C) < f_m,$$

which means that state e_m is red under an optimal policy for C^*.

Theorem 3 implies that as we change C, the optimal policy changes at most $K - 1$ times. These changes occur at the values C_m where a state e_m changes color. Let us assume for simplicity that distinct states change color at different values of C. Thus, there are exactly $K - 1$ values of C where the optimal policy changes. The parametric simplex method only performs pivots at these special values of C. Note that at $C = C_m$ constraint $X_m = f_m$ exits the dual basis and constraint $X_m = C + \sum_{k=1}^{K} p_{mk} X_k$ enters the dual basis (this corresponds to state e_m changing from green to red).

Table 2
Iterations of **PI**

	Array ARC_COLOR $m =$			Array X $m =$			Array T(X) $m =$		
Iteration	1	2	3	1	2	3	1	2	3
1	G	G	G	1	5	10	1	5	7.4
2	G	G	R	1	5	3.5	1	4.8	3.5
3	G	R	R	1	4.75	3.5	1	4.75	3.5

Copyright © 2004 All Rights Reserved

Thus, the total number of pivots in the parametric simplex method is $K - 1$ and the total computational complexity $O(K^3)$ (this argument can be generalized to the case where several states change color at the same value of C). The complexity of the parametric simplex method is the same as the complexity of policy iteration, but it has the added advantage that an optimal policy can be obtained for all values of C.

1.4. Closed-Form Solution

Finally, there are often cases of lower dimension for which an optimal policy can be determined *in closed form*. Consider, for instance, the case $K = 2$ with $p_{11} = a$, $p_{12} = 1 - a$, $p_{21} = 1 - b$, and $p_{22} = b$. Without loss of generality we assume that $f_1 \leq f_2$.

For this case, it is straightforward to show that if $C \geq (f_2 - f_1)(1 - b)$, both states are green, whereas if $C < (f_2 - f_1)(1 - b)$, only e_1 is green and e_2 is red. In both cases, since e_1 is green, $X_1 = f_1$. In the former case, $X_2 = f_2$. In the latter case, $X_2 = f_1 + C/(1 - b)$.

We close our investigation of the individual arc policy by reintroducing indices i and j for the arc in question, and by defining as $D_{ij}(e_m^i)$ what we have so far defined as X_m, that is, the minimum expected cost of traversing arc (i, j), *given* that the current state of the environment variable at node i is e_m^i. The full notation for the color array is $\text{ARC\_COLOR}_{ij}(e_m^i)$ (notice the color of a state is specific to the arc (i, j), and two arcs out of the same node will in general have different arrays ARC_COLOR). For each arc (i, j), arrays D_{ij} and ARC_COLOR_{ij} can be obtained by applying either the successive approximation algorithm or the policy iteration procedures described above.

2. POLICY FOR THE ENTIRE NETWORK

It is not immediately clear how (or whether) the above individual arc considerations are relevant to the original problem. For one thing, in addition to the "go/no-go" decision at each node, in the original problem one also has to choose which arc to traverse (direction decision). Moreover, that the costs of all arcs emanating from a given node are correlated (being functions of the same variable) introduces a cross-coupling among arcs. Even the notion of green versus red states becomes ambiguous in this case because environment states which are of a certain color for a certain arc (i, j) emanating from node i may or may not be of the same color for another arc (i, j') emanating from the same node, unless some "regularity" assumptions are made. But even if this ambiguity does not exist, it is not clear what the optimal traversal policy should be.

In addressing the general problem we first try a naive approach and show that it can be far from optimal. Then we develop an algorithm (**OPT**) based on dynamic programming that yields an optimal policy, and we estimate its computational complexity.

2.1. A Naive Approach

In a naive approach to the problem, we break the decision into two parts: We first somehow decide which of the arcs out of node i is preferable (in general, this will be a function of the prevailing state of the environment variable at that node), and then, once we are committed to that arc, make the go/no-go decision based on the policy suggested by array ARC_COLOR *for that arc*. The latter policy is assumed to have been computed separately for all arcs of the network, as described in the previous section. Such an approach is suboptimal precisely because we structure the decision process so that we are committed to a direction decision *before* the go/no-go decision is made (this point will be further discussed later on).

This point is illustrated by the simple example shown in Figure 3. Figure 3a shows the 4-node network under consideration, with only arcs (1, 2) and (1, 3) having Markovian costs; the other two arcs (2, 4) and (3, 4) have zero costs. Assume a waiting cost C. In this case, the only relevant decision is that at node 1, and the only relevant environment variable is at that node. Figure 3b shows that this variable may have three possible states, with Markov transition probabilities as shown in the figure. The direct traversal costs of arcs (1, 2) and (1, 3) corresponding to these three states are shown in the figure inside squares. Let us assume that M is very large.

One can immediately notice that an optimal policy at node 1 is the following:

If $m = 1$ (state is e_1^1), immediately go to node 2.
If $m = 2$ (state is e_2^1), wait.
If $m = 3$ (state is e_3^1), immediately go to node 3.

The expected cost of this policy is 0 if $m = 1$ or 3 (state is e_1^1 or e_3^1), and C if $m = 2$ (state is e_2^1).

In the naive approach, we have to commit ourselves to a choice of the next node (2 or 3) ahead of time. Suppose that the initial environment state at node 1 is e_2^1. Then, by symmetry, it does not make any difference which arc we commit ourselves to traverse. Suppose we choose arc (1, 2). As long as M is very large (in fact, $M \gg 1/\epsilon$), the optimal policy will be to wait until the first time that a state of e_1^1 occurs. It is straightforward to check that the expected cost of the

Copyright © 2001 All Rights Reserved

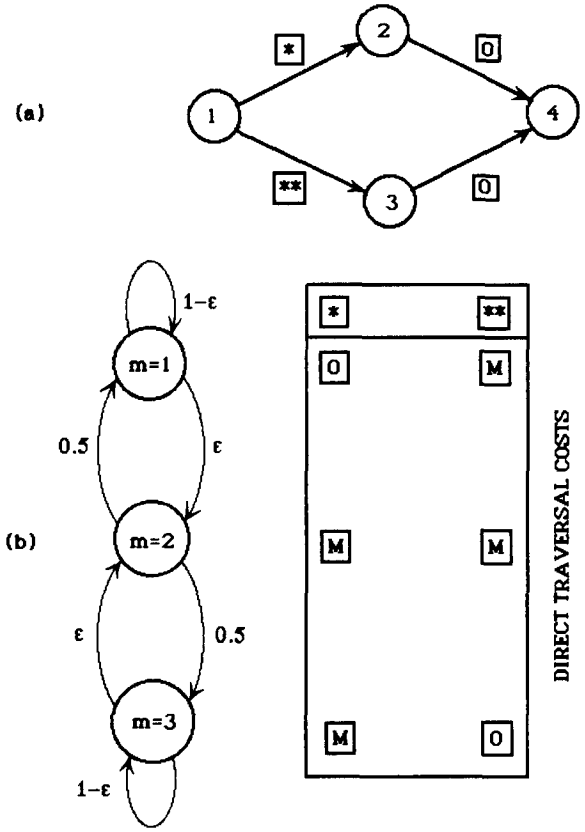

Figure 3. An example in which the naive procedure performs arbitrarily poorly.

naive policy is $C(2 + 1/\epsilon)$. We conclude that the naive approach can produce policies whose performance differs from the optimum by an arbitrarily large multiplicative factor.

2.2. Optimal Approach

From the above considerations it is clear that the way to structure the decision process is to *reverse* the order of decisions from that of the naive approach, as follows: First we decide whether or not we should wait at a node (this will be a function of the prevailing state of the environment variable at that node), and if the decision is go, we decide what the next node should be.

In procedure **OPT**, the decision to be made when the vehicle is at node i is again based on the prevailing state e_m^i of the environment variable at that node. Let us define the optimal value function $V_i(e_m^i)$ as the minimum achievable total expected cost of traversing the network from node i to the terminal node n, given that while at i the prevailing environment variable is in state e_m^i.

The options for the vehicle are either to immediately traverse arc (i, j), among those that are emanate from node i (in which case the vehicle must also choose node j, the next node to visit), or wait, but with no commitment as to where it should go next. The immediate cost incurred is $f_{ij}(e_m^i)$ in the former case, and C in the latter case.

As before, if arc (i, j) is immediately traversed, the environment variable at node j will be in state e_k^j with probability Π_k^j when the vehicle arrives there. The cost-to-go from then on is given by $V_j(e_k^j)$, assuming an optimal policy is followed. If, on the other hand, the vehicle waits at node i, the environment variable will be in state e_k^i with probability p_{mk}^i at the next stage, and the cost-to-go from then on will be $V_i(e_k^i)$, assuming again an optimal policy thereafter.

Based on all this, it is clear that $V_i(e_m^i)$ obeys the following system of equations for all e_m^i. For all $i \neq n$:

$$V_i(e_m^i) = \min\left\{\min_{j\in J(i)}\left\{f_{ij}(e_m^i) + \sum_{k=1}^{K} \Pi_k^j V_j(e_k^j)\right\},\right.$$

$$\left. C + \sum_{k=1}^{K} p_{mk}^i V_i(e_k^i)\right\} \tag{4}$$

with boundary condition $V_i(e_m^i) = 0$ for $i = n$.

A close look at (4) reveals that it is very similar in structure to (1) that was developed for the individual arc case. Other than the difference in notation, the second term in the outer braces is identical to the corresponding second term $C + \sum_{k=1}^{K} p_{mk}X_k$ in the braces of (1). The only other difference between (1) and (4) is that the first term in the outer braces, namely

$$\min_{j\in J(i)}\left\{f_{ij}(e_m^i) + \sum_{k=1}^{K} \Pi_k^j V_j(e_k^j)\right\},$$

replaces variable f_m in (1).

Since the network is acyclic, recursion (4) can be solved by backward dynamic programming. However, since $V_i(e_m^i)$ appears in both the left- and right-hand sides, any algorithm of Section 1 can be used to obtain the value of $V_i(e_m^i)$. Thus, we have the following algorithm.

Optimal Procedure (OPT)

STEP 1. (**Set boundary condition**)
Set $i := n$, and $V_i(e_m^i) := 0$ for all $m = 1, \ldots, K$.

STEP 2. (**Backtrack**)
Set $i := i - 1$.

Copyright © 2001 All Rights Reserved

STEP 3. **(Compute cost of direct traversal)**
For all $m = 1, \ldots, K$ define

$$F_m := \min_{j \in J(i)} \left\{ f_{ij}(e_m^i) + \sum_{k=1}^{K} \Pi_k^i V_j(e_k^j) \right\},$$

and DIRECTION$_i(e_m^i)$:= a node j that minimizes the right-hand side of the above expression.

STEP 4. Use any one of the algorithms of Section 1 to solve the individual arc problem, with f_m replaced by F_m, X_m replaced by $V_i(e_m^i)$, and ARC_COLOR(e_m) replaced by a new array, named NODE_COLOR$_i(e_m^i)$.

STEP 5. **(Check if done)**
If $i > 1$ go to Step 2. Else go to Step 6.

STEP 6. **(Optimal policy)**
For all $i = 1, \ldots, n$ and all $m = 1, \ldots, K$, the optimal policy is:

- If the environment variable at node i is in state e_m^i of NODE_COLOR$_i(e_m^i)$ = green, immediately depart to node DIRECTION$_i(e_m^i)$.
- Else (i.e., if NODE_COLOR$_i(e_m^i)$ = red), wait for a state transition.

The following points are now in order:

1. As in **SA** or **PI**, this algorithm classifies the state of the environment variable into two categories, this time by array NODE_COLOR. As before, this state can be either green, or red. Also, as before, if the environment variable is in the red state, the vehicle should wait until a green state is reached. As soon as the latter happens, the vehicle immediately departs to the node designated by the array DIRECTION. Note that DIRECTION is not fixed, but generally depends on the state of the environment variable at the node. It is the optimal next node, given the state of the environment variable, and given NODE_COLOR = green (there is no designation for the best next node if NODE_COLOR = red).

2. To estimate the complexity of **OPT**, we first note that evaluating variable F_m in Step 3 takes $O(nK)$ time. If the single arc problem is solved using either **PI** or **PLP**, the discussion of Section 2 implies that Step 4 takes $O(K^3)$ time. Given that **OPT** loops over every node of the network, the total complexity of **OPT** becomes $O(n^2K + nK^3)$.

3. If we wish to solve the problem on the entire network using parametric linear programming *for all values of C*, then expression F_m (Step 3 of **OPT**) becomes a piecewise linear function of C. In this respect, the parametric problem is more difficult than the equivalent problem in the individual arc case, where f_m is a constant and independent of C. In particular, the time complexity of performing Step 4 for all values of C using parametric LP will be proportional to the number of breakpoints of the function $F_m(C)$. In principle, the number of these breakpoints can increase exponentially with the size of the graph and we obtain an unfavorable, worst case computational complexity.

The added complexity of a parametric solution of the problem on the entire network is illustrated by the following example which shows that as C changes monotonically, a state color can change from green to red and then back to green (Theorem 3 showed that this cannot happen in the individual arc case).

Consider the problem illustrated in Figure 4. There are three nodes and we wish to go from node 1 to node 3. The figure shows the original network

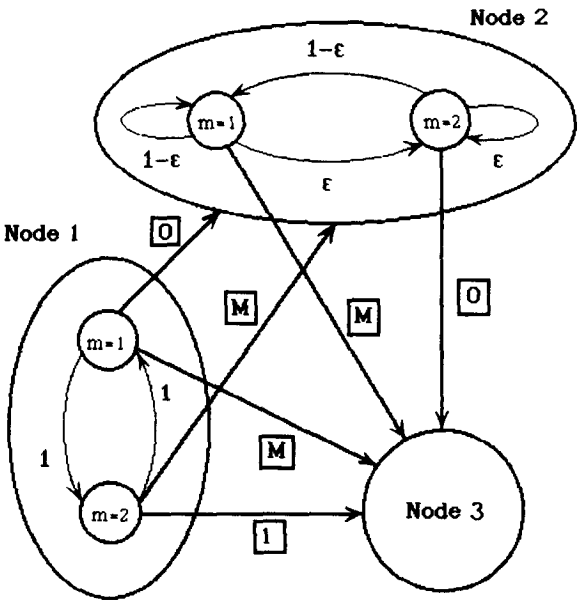

Figure 4. Example of a 3-node network in which a state color (here, that of the 1st state at node 1) can change from green to red and back to green. This is a combined graph in which states at nodes 1 and 2 are shown as small circles. Nodes 1 and 2 are shown as large ellipses. Arcs are shown in bold and arc costs are shown inside boxes. Also shown are the dynamics of the Markov processes at nodes 1 and 2 and the transition probabilities.

Copyright © 2001 All Rights Reserved

$G(N, A)$ for these three nodes and the Markov state graphs for nodes 1 and 2 in combination, the dynamics of the environment variables shown directly in the figure. Also shown are the dependencies of the arc-traversal costs on the states of the environment variables (e.g., if we are at node 1 with a state of e_1^1, and we choose to go to node 2, the arc-traversal cost would be equal to M). We assume that $\epsilon/(1-2\epsilon) < (1-\epsilon)M$.

The optimal cost-to-go at node 2 satisfies:

$V_2(e_2^2) = 0$, and

$V_2(e_1^2) = \min(M, C + (1-\epsilon)V_2(e_1^2))$, which yields

$V_2(e_1^2) = \min(M, C/\epsilon)$.

Let us now look at node 1. If state e_1^1 is green, the cost-to-go is equal to $\Pi_1^2\ V_2(e_1^2)$ (the probability of finding the environment variable at node 2 at state e_1^2 times $V_2(e_1^2)$), that is, equal to $(1-\epsilon)\min(M, C/\epsilon)$.

If, on the other hand, state e_1^1 is red, then state e_2^1 must be green, and the cost-to-go (starting from e_1^1) is $C+1$. It follows that e_1^1 is red if $C+1 \leq (1-\epsilon)\min(M, C/\epsilon)$, or, equivalently, if $\epsilon/(1-2\epsilon) \leq C \leq (1-\epsilon)M - 1$.

Thus, as C increases, e_1^1 changes color twice: at $C = \epsilon/(1-2\epsilon)$ (from green to red), and at $C = (1-\epsilon)M - 1$ (from red to green).

3. CONCLUSIONS AND POSSIBLE EXTENSIONS

This paper has examined shortest path problems in acyclic networks in which arc costs are known functions of certain environment variables at network nodes, and each of these variables evolves according to an independent Markov process. Several procedures were developed for determining which of the environment states at each node are *green* (the vehicle departs immediately) and which are *red* (the vehicle waits). For the individual arc case, the successive approximations (**SA**), policy iteration (**PI**), and parametric linear programming (**PLP**) methods were examined. We see that **PI** and **PLP** have some computational advantages over **SA**, the most important one being termination after a maximum of K iterations, and an overall complexity of $O(K^3)$. These concepts were extended for the entire network by developing a dynamic programming procedure that optimally solves the corresponding problem. The complexity of this method was shown to be $O(n^2K + nK^3)$. This low-order polynomial complexity is to be contrasted with that of the general Markov decision problem for which no strongly polynomial algorithm is known to exist.

We now further discuss the results of this work, with an emphasis on possible extensions.

1. An interesting extension of these procedures would be to networks other than acyclic (e.g., undirected or mixed). The resulting procedures will be similar to the Bellman-Ford algorithm for the general shortest path problem in that they will incorporate in the state space an additional index that measures progress along the path. However, in contrast to the previous case, here the vehicle may visit the same node more than once on its way to the terminal node, and in doing so would have some *prior knowledge* about the possible state of the environment at that node. Any optimal algorithm should explicitly incorporate such knowledge into the formulation. Alternatively, if that prior knowledge is deliberately *not* taken into account (say, by developing a memoryless procedure that does not keep track of such information), a straightforward extension of the Bellman-Ford algorithm could be used *as a heuristic* to obtain an upper bound on the optimal total expected cost. Of course, for reasonable instances it may be unlikely for the vehicle to visit the same node twice, however, this could happen in the general case.

2. Another interesting, but far from straightforward, extension is to assume that information on the states of environment variables at nodes other than the one currently visited by the vehicle is also available. Such information may be useful in that it may provide a "look-ahead" capability which may be exploited in order to achieve a lower cost policy. However, to carry out such an extension one would have to consider a number of nontrivial issues. First, the state space in the DP formulation may become enormous. If the vehicle knows the state of the environment at *all* nodes, the state space will be $O(K^n)$, clearly intractable even if K is very small, say, 3. A less pronounced, but nevertheless equally bothersome state-space explosion would occur even if only a limited amount of additional information is assumed to be known, for instance, the state of the environment variables at the current node and all its adjacent nodes. An additional issue is that one has to explicitly consider the *time* it takes to traverse an arc (at least in multiples of the time interval between successive Markov transitions), in order to predict the environment state at the next node at the time the vehicle arrives there. Such a consideration was not necessary in our version of the problem because of the lack of information on environment conditions

Copyright © 2001 All Rights Reserved

at other nodes. Moreover, one may also have to consider spacewise dependencies among environment variables in addition to their timewise dependency. All of the above certainly add to the difficulty of the problem.

3. With respect to the last problem, it might be of interest to investigate suboptimal policies in which the vehicle receives *partial information* on the environment states. Such partial information may be either an upper bound B on the number of nodes for which information is available (our original problem is the case $B = 1$), or a more aggregate representation of the state of the environment variables (e.g., the various states may be classified into good or bad and the vehicle may only know that information as opposed to the detailed information about the actual state), or finally, anything that would effectively limit the growth of the problem's state space.

4. Finally, another very interesting extension is if we add a time window $[r, d]$ at the terminal node n, where r is a specified earliest arrival time and d is a specified latest arrival time (or deadline). Penalties for early and/or late arrivals may be specified, and these penalties would be part of the overall expected cost to be minimized. As in the previous extension, this extension requires the consideration of arc-traversal times. It also necessitates the inclusion of the time dimension into the state space.

ACKNOWLEDGMENT

The work of both authors was supported in part by C. S. Draper Labs under contract DL-H-404164. The work of the second author also received matching support from NSF grant ECS-8552419. We thank the area editor and two anonymous referees for their comments.

REFERENCES

ANDREATTA, G., F. RICALDONE AND L. ROMEO. 1985. Exploring Stochastic Shortest Path Problems. In *ATTI Giornale di Lavoro*, Tecnoprint, Bologna, Italy.

BERTSEKAS, D. P. 1987. *Dynamic Programming Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, N. J.

BERTSEKAS, D. P., AND J. N. TSITSIKLIS. 1988. An Analysis of Stochastic Shortest Path Problems. *Math. Opns. Res.* (to appear).

BERTSIMAS, D. J. 1988. Probabilistic Combinatorial Optimization Problems. Ph.D. Thesis, Operations Research Center, MIT, Cambridge, Mass.

BERTSIMAS, D. J., AND G. VAN RYZIN. 1991. A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane. *Opns. Res.* **39,** 601–615.

CHEN, H. 1978. A Dynamic Program for Minimum Cost Ship Routing Under Uncertainty. Ph.D. Thesis, Department of Ocean Engineering, MIT, Cambridge, Mass.

FRANK, H. 1969. Shortest Paths in Probabilistic Graphs. *Opns. Res.* **17,** 583–599.

FRIEZE, A., AND G. GRIMMET. 1985. The Shortest Path Problem for Graphs With Random Arc Lengths. *Discrete Appl. Math.* **10,** 57–77.

GOLUB, G. H., AND C. F. VAN LOAN. 1983. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Md.

HALL, R. W. 1986. The Fastest Path Through a Network With Random Time-Dependent Travel Times. *Trans. Sci* **20,** 182–192.

JAILLET, P. 1985. The Probabilistic Traveling Salesman Problem. Technical Report 185, Operations Research Center, MIT, Cambridge, Mass.

JAILLET, P. 1989. Shortest Path Problems With Node Failures. *Networks* (submitted).

PERAKIS, A. N., AND N. PAPADAKIS. 1988. New Models for Minimal Time Ship Weather Routing. *SNAME Trans.* **96,** 247–269.

PSARAFTIS, H. N. 1988. Dynamic Vehicle Routing Problems. In *Vehicle Routing: Methods and Studies*, B. Golden and A. Assad (eds.). North-Holland, Amsterdam.

Copyright © 2001 All Rights Reserved