MATHEMATICS OF OPERATIONS RESEARCH
Vol. 12, No. 3, August 1987
Printed in U.S.A.

THE COMPLEXITY OF MARKOV DECISION PROCESSES*[†]

CHRISTOS H. PAPADIMITRIOU[‡] AND JOHN N. TSITSIKLIS[§]

We investigate the complexity of the classical problem of optimal policy computation in Markov decision processes. All three variants of the problem (finite horizon, infinite horizon discounted, and infinite horizon average cost) were known to be solvable in polynomial time by dynamic programming (finite horizon problems), linear programming, or successive approximation techniques (infinite horizon). We show that they are complete for *P*, and therefore most likely cannot be solved by highly parallel algorithms. We also show that, in contrast, the deterministic cases of all three problems can be solved very fast in parallel. The version with partially observed states is shown to be PSPACE-complete, and thus even less likely to be solved in polynomial time than the NP-complete problems; in fact, we show that, most likely, it is not possible to have an efficient on-line implementation (involving polynomial time on-line computations and memory) of an optimal policy, even if an arbitrary amount of precomputation is allowed. Finally, the variant of the problem in which there are no observations is shown to be NP-complete.

1. Introduction. In the past, Complexity Theory has been applied with considerable success to separate optimization problems with respect to their computational difficulty [GJ, PS]. Such results are valuable in that they succeed in formalizing the intuitive feeling by researchers in the area about the impossibility of certain approaches in solving some hard problems, and in some cases give a clear separation between the easy and the hard versions (special cases or generalizations) of important problems. In most of this work, NP-completeness is the main notion of complexity used. The important distinction here is between problems that are in P, that is, can be solved in polynomial time (linear programming, max flow, minimum spanning tree) and those that are NP-complete, and thus presumably cannot be so solved (examples are integer programming and the traveling salesman problem).

Recent research in parallel algorithms [Co] has succeeded in pointing out some important differences between problems that are in P, that is, can be solved in polynomial time. Some of these problems can be solved by algorithms that use cooperating processors so that the time requirements are reduced tremendously (such problems include, from the optimization point of view, minimum spanning tree problem, the shortest path problem, and others), whereas certain other problems (first of all linear programming, but also some special cases of it, such as maximum flow) do not appear to be susceptible to such massive parallelization. NC is the class of problems that have algorithms using a polynomial number of processors, and time delay which is polynomial *in the logarithm* of the input size. The question then arises, is

*Received June 10, 1985; revised February 21, 1986.
AMS 1980 subject classification. Primary: 90C7. Secondary: 68C25.
IAOR 1973 subject classification. Main: Programming: Markov Decision.
OR/MS Index 1978 subject classification. Primary: 117 Dynamic programming/Markov. Secondary: 568 Probability/Markov processes.
Key words. Markov decision processes, dynamic programming, computational complexity, parallel computation.
[†] This research was supported by an IBM faculty development award, and ONR contract N00014-85-C-0731.

[‡] Stanford University.
[§] Massachusetts Institute of Technology.

0364-765X/87/1202/$01.25

NC = P? In other words, is it true that all problems that can be solved satisfactorily by sequential computers can also take the maximum possible advantage of parallel computers? Most researchers believe that the answer is negative.

The notions of reductions and completeness offer some important evidence here: Linear programming, maximum flow, and some other problems have been shown to be P-*complete* [DLS, GSS]. This means, intuitively, that such problems are as hard as any problem in P; thus, they can be massively parallelized only if NC = P, that is, if *all* problems in P can, and thus there are no inherently sequential problems. Since this event is considered unlikely, P-completeness is taken as evidence that a problem is not in NC, and thus cannot be satisfactorily parallelized.

The design of optimal strategies for Markov decision processes (say, for the discounted infinite horizon case) has been a very central and well-solved optimization problem. There are basically two approaches to its solution. First, the problem can be expressed as a linear program and thus solved (either by generic techniques, or specialized "policy improvement" routines [Ho]); this technique appears to be unpromising from the parallel algorithm point of view, since it involves linear programming, a problem that seems inherently sequential. Secondly, there are iterative techniques that are known to converge fast on the optimal policy; however, such techniques are sequential also by nature. (Let us note in passing that the problem of deriving a "clean" polynomial time algorithm for this problem, without using general linear programming or approximate techniques, is an important open question that has not been emphasized in the literature as much as it deserves.) An interesting question arises: Is the Markov decision problem in NC, or is it complete for P, and thus, in some sense, the use of linear programming for its solution is inherent?

In this paper we show that computing the optimal policy of a Markov decision process is complete for P, and therefore most probably is inherently sequential. This is true also for the average cost case, as well as the finite horizon (nonstationary) case. However, we also show that the *deterministic* special cases of all three problems (that is, in which all possible stochastic matrices have zero-one entries) are in NC.

Next, we address the question of the complexity of the *partially observed* generalization of this problem; this is an important problem arising in many applications in control theory and operations research [Be]. We show that the partially observed version of the finite horizon problem is PSPACE-complete. We do not consider infinite horizon partially observed problems because these are not combinatorial problems and do not seem to be exactly solvable by finite algorithms. Notice that only recently has there been work relating combinatorial optimization problems [Or, Pa1] to this notion of complexity, which is stronger than NP-competeness (see §2 for definitions). In fact, we show that, most likely, it is not possible to have an efficient on-line implementation (involving polynomial time on-line computations and memory) of an optimal policy, even if an arbitrary amount of precomputation is allowed. Finally, we show that the same problem with no observations is NP-complete.

In the next section we introduce the necessary concepts from Markov Decision Processes, and Complexity Theory. In §3 we present our results concerning the possibility of designing massively parallel algorithms for these problems. Finally, in §4 we prove our results on partially observable Markov processes.

2. Definitions.

Complexity. For a general introduction to Computational Complexity see the bibliography in [Pa2]. Our model of (sequential) computation can be any one of the Turing machine, random access machine, etc. [AHU, LP]; all these models of computation are known to be equivalent within a polynomial. We denote by P the class of all problems that can be solved in polynomial time (in such a model of computation). As usual, by "problem" we mean an infinite set of instances, each of which is a "yes-no" question. Optimization problems can be transformed into such problems by asking, not for the optimal feasible solution, but simply whether the optimal cost is below a given bound.

NP is the class of problems that can be solved *nondeterministically* in polynomial time; a nondeterministic machine essentially has the power of branching out and following both computation paths; this can be repeated in each of the paths, and so on, creating a potentially exponential set of computations. (Equivalently, NP is the class of "yes-no" problems that have the "succinct certificate property", see [PS].)

Besides time complexity, we shall concern ourselves with the space complexity of algorithms. The space consumed by a computation is the number of memory cells necessary for the computation; notice that this does not count the space necessary for the input, so the space requirement of an algorithm may be less than linear (for example, logarithmic). An important class is PSPACE, the class of all problems that can be solved in polynomial space. If a computation can be carried out in time (or nondeterministic time) $T(n)$, then it requires only $T(n)$ space; it follows that P and NP are subsets of PSPACE. If the space requirements are $T(n)$, then the time requirement cannot exceed $c^{T(n)}$, for some constant c.

We say that a problem A is *reducible* to another B, if for any instance x of A, we can construct *using space* $\log(|x|)$ an instance $f(x)$ of B such that the answer for x, considered as an instance of A, is the same as the answer of $f(x)$, considered as an instance of B (here x is a string encoding an instance of problem A, and $|x|$ is its length). Notice that we insist that the function f be computed in logarithmic space; this is stricter than the polynomial time usually required. In practice this is not a major restriction as most known polynomial reductions can in fact be accomplished in logarithmic space. Also, it can be shown that log-space reduction is transitive. Let A be a problem, and $\mathscr{C}$ be a class of problems (such as P, NP, PSPACE). We say that A is complete for $\mathscr{C}$, or *$\mathscr{C}$-complete* if (a) A is a member of $\mathscr{C}$, and (b) for every member B of $\mathscr{C}$, B is reducible to A.

Although there are many NP-complete problems in the literature, the notion of PSPACE-completeness is somewhat less well-known. A fundamental PSPACE-complete problem is that of *quantified satisfiability* (*QSAT*) [SM]: We are given a quantified Boolean formula $\exists x_1 \forall x_2 \exists x_3 \ldots \forall x_n F(x_1, \ldots, x_n)$, where F is an ordinary Boolean formula in conjunctive normal form with three literals per clause. We are to determine whether this formula is true, that is, whether there exists a truth value for x_1 such that, for all truth values of x_2, etc. for all truth values of x_n, F comes out true.

As mentioned above, PSPACE contains NP. In fact, NP can be thought of as the class of problems reducible to the special case of QSAT, called SAT, in which all quantifiers are existential (that is, there is no alternation of quantifiers). In fact, there is a number of intermediate classes[1] based on less restricted alternation; this is called the *polynomial hierarchy* [SM]. The next level above NP in this hierarchy is the class Σ_2^p, which can be thought of as all problems that reduce to the problem of telling whether a formula with one string of existential quantifiers, followed by a string of universal quantifiers, is true. An example of such a formula is $\exists x_1 \exists x_2 \ldots \exists x_n \forall y_1 \forall y_2 \ldots \forall y_m F(x_1, \ldots, y_m)$.

In the interest of formalizing parallel computation, we can define a parallel random access machine (PRAM), that is, a set of RAMs operating in unison, and communicat-

[1] Of course, it should be borne in mind that, for all we know, PSPACE, NP, even P might all coincide, in which case so would these "intermediate" classes.

ing through a sequence of common registers, some of which initially hold the input [Co]. The measures of complexity here are the delay until the final answer is produced, and the number of processors that have to be involved. A problem is considered to be solved in a satisfactory way by such a supercomputer if the delay is polynomial in the logarithm of the length of the input, and the number of processors involved are polynomial in the length of the input. The class of problems solvable under these terms is called NC. Obviously (by multiplying delay times number of processors) NC is a subset of P. The great enigma for parallel computation, analogous to P = ?NP for sequential computation, is whether NC = P. That is, while P = NP asks whether there are (reasonable) problems that are inherently exponential, NC = P asks whether there are problems that are inherently sequential. The bets are that such problems do exist, but there is no proof. Now, if inherently sequential problems exist (equivalently, if NC $\neq$ P) then the problems that are complete for P will certainly be among them. Thus, P-completeness plays vis-a-vis the NC = P problem the same role enjoyed by NP-completeness in relation to the P = NP problem.

One basic P-complete problem is the *circuit value problem* (CVP). A *circuit* is a finite sequence of triples $C = ((a_i, b_i, c_i),\ i = 1, \ldots, k)$. For each $i \leqslant k$, a_i is one of the "operations" **false**, **true**, **and**, and **or**, and b_i, c_i are nonnegative integers smaller than i. If a_i is either **false** or **true** then the triple is called an *input*, and $b_i = c_i = 0$. If a_i is either **and** or **or** then the triple is called a gate and $b_i, c_i > 0$. The *value* of a triple is defined recursively as follows: First, the value of an input (**true**; 0, 0) is **true**, and that of (**false**, 0, 0) is **false**. The value of a gate (a_i, b_i, c_i) is the Boolean operation denoted by a_i applied to the values of the b_ith and c_ith triples. The value of the circuit C is the value of the last gate. Finally, the CVP is the following problem: Given a circuit C, is its value **true**?

Markov Decision Processes. In a Markov decision process we are given a finite set S of states, of which one, s_0 is the initial state. At each time $t = 0, 1, \ldots$ the current state is $s_t \in S$. For each state $s \in S$ we have a finite set D_s of decisions. By making decision $i \in D_{s_t}$ at time t, we incur a cost $c(s, i, t)$, and the next state s' has probability distribution given by $p(s, s', i, t)$. We say the process is *stationary* if c and p are independent of t, in which case we write p as $p(s, s', i)$. A *policy* δ is a mapping assigning to each $t \in \{0, 1, 2, \ldots\}$ and state $s \in S$ a decision $\delta(s, t)$. A policy is *stationary* if $\delta(s, t)$ is independent of t. We shall consider three variants of the problem. First, in the *finite horizon* problem, we are given a (nonstationary) Markov decision process and an integer T. We wish to find a policy that minimizes the expectation of the cost $\Sigma_{t=0}^{T} c(s_t, \delta(s_t, t), t)$. The other variants that we deal with have *infinite horizon*, and thus we shall only consider their stationary special cases (otherwise we need the description of an infinite object, namely the parameters c and p for all times). In the *discounted* (infinite horizon) problem we are given a Markov decision process and a number $\beta \in (0, 1)$, and we wish to minimize the expectation of the discounted cost $\Sigma_{t=0}^{\infty} c(s_t, \delta(s_t, t))\beta^t$. Finally, in the average cost (infinite horizon) problem we are given a Markov decision process, and we wish to minimize the expectation of the limit $\lim_{T \to \infty}(\Sigma_{i=0}^{T} c(s_t, \delta(s_t, t))/T)$. It is well known that in the two latter cases (those for which the optimal policy is a potentially infinite object), there exist stationary optimal policies, and therefore optimal policies admit a finite description. For all three cases of the problem, there are well-known polynomial-time algorithms.

3. P-completeness and parallel algorithms.

We show the following result:

THEOREM 1. *The Markov Decision Process problem is P-complete in all three cases (finite horizon, discounted, and average cost).*

PROOF. We shall reduce the CVP to the finite horizon version first. Given a circuit $C = ((a_i, b_i, c_i), i = 1, \ldots, k)$, we construct a stationary Markov process $M = (S, c, p)$ as follows: S has one state i for each triple (a_i, b_i, c_i) of C, plus an extra state q. If (a_i, b_i, c_i) is an input, then the corresponding state i has a single decision 0, with $p(i, q, 0) = 1$, and cost $c(i, 0) = 1$ if $a_i =$ **false**, and 0 otherwise. All other costs in this process are 0. From q we only have one decision, which has $p(q, q, 0) = 1$. If a_i is an **or** gate, there are two decisions 0, 1 from state i, with zero cost, and $p(i, b_i, 0) = 1$, $p(i, c_i, 1) = 1$. That is, we decide whether the next state is going to be b_i or c_i. If a_i is an **and** gate, then there is one choice 0 in D_i, with $p(i, b_i, 0) = p(i, c_i, 0) = 1/2$, that is, the next state can be either b_i or c_i. The initial state is k, the last gate, and the time horizon also equals k, the size of the circuit.

We claim that the optimum expected cost is 0 or less iff the value of C was true. In proof, suppose that the expectation is indeed zero (it cannot be less). Then it follows that there are decisions so that the states with positive costs are impossible to reach. However, these states correspond to the **false** inputs, and thus these decisions are choices of a **true** gate among b_i, c_i for each **or** gate i of the circuit so that its overall value is **true**. Conversely, if the value is **true**, there must be a way to choose an input gate for each **or** gate so that the **false** inputs are not reachable, or, equivalently, the states with positive costs are not possible.

Essentially the same construction works for the discounted case; for the average cost problem, a modification is necessary: We must first make sure that all paths from the inputs to the last gate of C have the same number of gates on them, and then have transitions from the states corresponding to the inputs not to a state q, but back to the initial state. ■

We note that the above proof shows that even the stationary special case of the finite horizon problem is P-hard. It is not known whether the stationary finite-horizon problem is in P, because of the following difficulty: We could be given a stationary process with n states, and a horizon T up to 2^n, and the input would still be of size $O(n)$. Still, the dynamic programming algorithm for this problem would take time proportional to nt, and thus exponential in the input. (In the nonstationary case, the input must specify c and p for each $i \leqslant T$, and so the input size is at least T.)

Deterministic problems. Consider now the special case of these problems, in which the only allowed values for p are 0 and 1; we call this the *deterministic case*. We shall show below that the deterministic cases of the finite horizon (stationary and nonstationary), discounted, and average cost problem are in NC. Our approach is to look at these problems as variants of the graph-theoretic shortest-path problem [La]. This is done as follows: Given a deterministic Markov decision process, we construct a directed graph $G = (V, A)$, where the nodes are the states, and the arcs emanating from a node are the decisions available at this state. The node pointed to by an arc is the state to which the decision is certain to lead (recall that there are only 0 or 1 probabilities). There is a weight $c(u, v)$ on each arc (u, v), equal to the cost of the corresponding decision. We denote by u_0 the node corresponding to the initial state s_0. The particular variants of the problem are then equivalent to certain variants of the shortest path problem.

The nonstationary finite horizon problem. The parallel algorithms that we describe in this section employ a technique used in the past to yield fast parallel (or space efficient) algorithms known as "path doubling" [Sa, SV]. The idea is, once we have computed all optimal policies between any two states, where each policy starts at time t_1 and ends at time t_2, and similarly between t_2 and t_3, to compute in one step all optimal policies between t_1 and t_3. We can think of this as multiplying two $|V| \times |V|$ matrices $A(t_1, t_2)$

and $A(t_2, t_3)$, where the (u, v)th entry of $A(t_1, t_2)$ is the cost of the optimal policy from state u to v between times t_1 and t_2. This "matrix multiplication" is the variant in which multiplication of reals is replaced by addition, and addition of reals by the operation of taking the minimum. Using n^3 processors, we can "multiply" $n \times n$ matrices in $\log n$ parallel steps, by independently computing each of the n^2 inner products using n processors in $\log n$ steps. This latter can be done by computing the n terms of the inner product independently, and combining them in $\log n$ stages.

This approach immediately suggests an NC parallel algorithm for the finite horizon nonstationary problem: We start from the matrices $A(t, t+1)$, $t = 0, \ldots T-1$; the (u, v)th entry of $A(t, t+1)$ equals the cost of the decision leading at time t from state u to state v, if such a decision exists at time t, and is equal to ∞ otherwise. Then to compute the required $A(0, T)$ we multiply in $\log T$ stages these matrices. The total number of processors is Tn^3, and the total parallel time $\log T \log n$; since the size of the input for the nonstationary problem is $n^2 T$, this is an NC algorithm.

THEOREM 2. *The finite-horizon, nonstationary deterministic problem is in NC.* ■

Notice that this technique does not solve the stationary problem, whose input is of length $n^2 + \log T$, and which will have to be attacked by a more sophisticated technique, explained later.

The infinite horizon undiscounted case. It is easy to see that the infinite horizon average cost problem is equivalent to finding the cycle in the graph corresponding to the process that is reachable from u_0, and has the smallest average length of arcs. In proof of the equivalence, the limit of the average cost of an infinite path which starts at u_0, reaches this cycle, and then follows this cycle for ever, equals the optimum average cost, and this cannot be improved.

To make sure that we do not consider solutions that are not reachable from u_0, we first compute in $\log n$ parallel time [SV] the nodes reachable from u_0, and in the sequel consider only these. The cycle with the shortest average cost can be found by computing, in parallel, for each $k = 1, \ldots, n$ the shortest cycle of length k, and companying the results, each divided by k. To compute the shortest cycle of length k, we essentially have to compute the kth power of matrix A, whose (u, v)th entry is equal to the cost of the decision leading from state u to state v, if such a decision exists, and ∞ otherwise. This is done with $n^2 k$ processors in $\log k \log n$ parallel steps. The total time is $\log^2 n + 2 \log n$, using n^4 processors.

THEOREM 3. *The infinite-horizon, average cost deterministic problem is in NC.* ■

The infinite horizon, discounted case. Define a *sigma* in a directed graph to be a path of the form $(u_0, \ldots, u_k, v_1, \ldots, v_l, v_1)$, where all nodes indicated are distinct. In other words, a sigma is a path from u_0 until the first repetition of a node. The *discounted cost* of a sigma $P = (u_0, \ldots, u_k, v_1, \ldots, v_l, v_1)$ is defined to be

$$c(P) = \sum_{i=0}^{k} c(u_j, u_{j+1})\beta^i + \frac{\beta^{k+1}}{(1-\beta^l)} \sum_{j=1}^{l} c(v_i, v_{i+1 \bmod l})\beta^j.$$

That is, the discounted cost of a sigma coincides with the discounted cost of an infinite path that follows the sigma and repeats the cycle forever. It follows from the fact that the (general) discounted problem has a stationary optimal policy, that the cost of this optimal policy is the optimum discounted cost of a sigma in the corresponding directed graph.

We can compute the optimum sigma as follows: First, we compute the shortest discounted path of length $j = 0, 1 \ldots, n$ among any pair of nodes by "multiplying" the matrices $A, \beta A, \ldots, \beta^{j-1}A$, where A is the matrix defined in the paragraph before Theorem 3. This can be done in $\log^2 n$ steps by using n^4 processors. Let $B_1, \ldots, B_n$ be the resulting products; the (u, v)th entry of B_j is the length of the shortest path with j arcs from u to v. Once this is done, for each node u and for each $k, l = 0, 1, \ldots, n$ we compute $\lambda + (\beta^{k+1}\mu/(1 + \beta^l))$, where λ is the (u_0, u)th entry of B_k, and μ is the u, u entry of B_l. We pick the best result. This requires a total of n^4 processors, and $\log^2 n + 3 \log n$ parallel steps.

Theorem 4. *The infinite-horizon, discounted deterministic problem is in NC.* ■

The finite horizon, stationary case. Given a stationary deterministic process, the stationary finite horizon problem with horizon T is equivalent to finding the shortest path with T arcs in the corresponding graph, starting from the node u_0. The algorithm should run in a number of parallel steps that is polynomial to $\log \log T$; therefore the "path-doubling" technique would not give the desired result. In the sequel we assume that $T > n^2$, otherwise the previous technique applies.

Without loss of generality, assume that the arc lengths are such that no ties in the lengths of paths are possible (this can be achieved by perturbing the lengths). Consider the shortest path out of u_0 with T arcs. Since $T > n^2$, there are many repetitions of nodes on this path. Consider the first such repetition, that is, the first time the path forms a sigma, and remove the cycle from the path. Then consider the first repetition in the resulting sequence. Continuing this way, we can decompose the path into a simple path (i.e., without repetitions of nodes) out of u_0, plus several simple cycles (notice that this decomposition may not be unique). We first need to show that we can assume that only one simple cycle is repeated more than n times, namely the one that has the shortest average length of its arcs. In proof, consider two cycles of length m_1 and m_2, repeated n_1 and n_2 times, respectively. If $n_1, n_2 \geqslant n$, then we can repeat the cycle with smaller average arc length, say the first, m_2 times less, and repeat the second m_1 times more, to obtain another path of smaller length. Thus, only one cycle is repeated more than n times. Furthermore, since we have no ties, only one cycle of each length is repeated. Therefore, the shortest path of length T has the following structure: It consists of a path with length $l < n^3$ (this is the simple path, plus the at most n repetitions of one cycle from each length), plus a simple cycle repeated many times to fill the required number of arcs. Therefore, for each value of $l < n^3$ and each node u and each possible number of arcs in the cycle k that divides $T - l$ we do, in parallel, the following: We compute the shortest path of length l from u_0 through u, the shortest cycle of length k through u, and the cost of adding $(T - l)/k$ copies of the cycle to the path. Of the resulting combinations, we pick the cheapest.

Theorem 5. *The finite-horizon, stationary deterministic problem is in NC.* ■

4. The partially observed problem.

We can define an important generalization of Markov decision processes if we assume that our policy must arrive at decisions with only partial information on the current state. In other words, we have a partition $\Pi = \{z_1, \ldots, z_p\}$ of S, where the z_i's are disjoint subsets of S that exhaust S, and at any time we only know the particular set in Π to which the current state belongs (naturally, we also remember the previous such sets).[2] Each set $z \in \Pi$ has a set D_z of decisions associated with it, and each decision $i \in D_z$ has a cost $c(z, i)$ and a probability distribution $p(s, s', i)$ of the next state s', given the current state $s \in z$.

[2] In the traditional formulation of the partially observed problem [Be], the observation at time t is a random variable, for which we know its conditional distribution given the current state and time. Here we essentially deal with the special case in which the observation is a deterministic function of the current state. Clearly, the general case cannot be any easier.

The initial state is known as before to be s_0. A policy is now a mapping from *sequences of observations* $z_1, \dots, z_t$ to decisions in D_{z_t}.

This problem is often treated by converting it to an equivalent perfectly observed problem by redefining the state to be the vector p whose ith component is the conditional probability that the state of the original problem equals i [Be]. In this formulation the state space is infinite. Nevertheless, it is known [SS] that the function assigning to each probability vector and time t the optimum future cost (cost-to-go) is of the form $J(p, t) = \min_{y \in K_t}[A_k + B_k p]$, where A_k is a scalar and B_k is a vector of suitable dimension and K_t is a finite index set. This allows the solution in finite time of the problem even though the state space is infinite. Nevertheless, the cardinality of the index set K_t may be an exponential function of the time horizon and, therefore, this procedure is of limited practical use. The following result shows that most likely this is a generic property of the problem we are dealing with rather than a defect of the particular reformulation of the problem. Notice also that in the infinite horizon limit the index set K_t will be infinite, in general, which makes fairly doubtful whether there exists a finite-algorithm for the infinite horizon problem. For this reason, we only consider the problem in which we are asked to minimize the expected undiscounted cost over a finite time horizon T.

THEOREM 6. *The partially observed problem is* PSPACE-*hard, even if the process is stationary and T is restricted to be smaller than $|S|$ (and it is in* PSPACE *in the latter case).*

PROOF. To show PSPACE-hardness, we shall reduce QSAT to this problem. Starting from any quantified formula $\exists x_1 \forall x_2 \dots \forall x_n F(x_1, \dots, x_n)$, with n variables (existential and universal) and m clauses $C_1, \dots, C_m$, we construct a partially observed stationary Markov process and a horizon $T < |S|$, such that its optimum policy has cost 0 or less if and only if the formula is true.

We first describe S. Besides s_0, the initial state, S contains six states A_{ij}, A'_{ij}, T_{ij}, T'_{ij}, F_{ij}, F'_{ij} for each clause C_i, and variable x_j. There are also $2m$ states $A_{i,n+1}$, $A'_{i,n+1}$. We next describe the partition Π. The initial state s_0 is in a set by itself. For each variable j, the states A_{ij}, $i = 1, \dots, m$ and the states A'_{ij}, $i = 1, \dots, m$ form a set called A_j, the states T_{ij}, $i = 1, \dots, m$ form a set called T_j, the states T'_{ij}, $i = 1, \dots, m$ form a set called T_j' and so on, up to F_j'. The states $A_{i,n+1}$ and $A'_{i,n+1}$ are each on a different set.

Next we shall describe the decisions, the probabilities, and the costs. All decisions except those out of the set A'_{n+1} have zero cost. At s_0 there is only one decision, leading to the states A'_{i1}, $i = 1, \dots, m$ with equal probability. If x_j is an existential variable, there are two decisions out of the set A_j, leading certainly from A_{ij} to T_{ij} and F_{ij}, respectively; similarly there are two decisions out of the set A'_j, leading with certainty from A'_{ij} to T'_{ij} and F'_{ij}, respectively. If x_j is a universal variable, there is one decision out of the set A_j, leading with equal probability from A_{ij} to T_{ij} and F_{ij}; similarly there is one decision out of the set A'_j, leading with equal probability from A'_{ij} to T'_{ij} and F'_{ij}. From the T_j, F_j, T_j' and F_j' sets there is only one decision, which leads with certainty from T_{ij}, F_{ij}, T'_{ij}, and F'_{ij} to (respectively) $A_{i,j+1}$, $A_{i,j+1}$, $A'_{i,j+1}$, and $A'_{i,j+1}$, with two exceptions: If x_j appears positively in C_i, the transition from T'_{ij} is to $A_{i,j+1}$ instead of $A'_{i,j+1}$; and if x_j appears negatively, the transition from $F'_{i,j}$ is to $A_{i,j+1}$.

Finally, out of $A_{i,n+1}$ and $A'_{i,n+1}$ there is one decision, leading to a new state with certainty. In the entire process, all decisions have cost zero, except for the decision out of $A'_{i,n+1}$, which incurs a cost of 1. This completes the construction of the process; the horizon T is defined to be $2m + 2$ (just enough time for the process to reach one of $A_{i,n+1}$ or $A'_{i,n+1}$).

We claim that there exists a policy with expected cost zero iff the formula is true. Suppose such a policy exists. Recall that the transition from the initial state can be to any state A'_{i1} of A'_1; we think that the process "chooses" a clause C_i. Once this has happened, the process remains at states subscripted by i forever, without ever observing the real value of i. It follows that the policy has zero expected cost for all such initial choices. We next claim that the policy must guarantee that the process ends up in $A_{i,n+1}$. If not, that is, if for some choices of decisions for the universal variables the process ends up in $A'_{i,n+1}$, then this contributes an expected cost of at least $2^{-n}/m$, which is absurd. It follows that the policy must, based on the previous observations on the transitions at the sets corresponding to previous universal variables, pick at the existential variables those decisions that correspond to a truth assignment which satisfies the clause. Since all clauses must be satisfied, the formula was true.

Conversely, if the formula were true, there is a policy for setting the existential variables, based on the values of previous universal ones, so that all clauses are satisfied. This, however, can be translated into a policy for choosing the corresponding decisions at the sets corresponding to existential variables so that, for all choices of clauses, the state $A_{i,n+1}$ is reached. A policy with zero cost is thus possible. The proof is complete.

Finally, we sketch a proof that the special case of the problem examined here is in PSPACE. The construction is a bit tedious, but quite reminiscent of polynomial-space algorithms for other similar problems (see for example [Pa1]). We can think of the possible outcomes of the process within the finite horizon T as a tree of depth T, which has internal nodes both for decisions and transitions. The leaves of this tree can be evaluated to determine the total contribution to the expected cost of this path. To determine whether the optimal policy has expected cost less than some number B, we need to traverse this tree, making nondeterministic decision nodes, iterating over all transitions *and taking care to make the same decision every time the history of observations is identical.* This latter can be achieved by organizing the search in such a way that nodes at the same level of the tree are divided into intervals from left to right, corresponding to distinct observation sequences. Decisions from nodes in the same interval must be the same. That the problem is in PSPACE follows now from the fact that PSPACE is robust under nondeterminism. ■

Theorem 6 says that, unless P = PSPACE (an event even less likely than P = NP), there is no polynomial-time algorithm for telling whether a cost can be achieved in a given partially observable process. However, there is a more practical problem, the complexity of which is not settled by this result. It would still be satisfactory if we could analyze such a process (perhaps by preprocessing it for an exponential time or worse), and, based on the analysis, design a practically realizable controller (polynomial-time algorithm) for driving the process on-line optimally. This algorithm would presumably consult come polynomially large data structure, constructed during the analysis phase. We point out below that even this is very unlikely:

COROLLARY 1. *Unless PSPACE* $= \Sigma_2^p$, *it is impossible to have an algorithm* $\mathscr{A}$ *and a mapping* μ (*of arbitrary high complexity*) *from a partially observed Markov process* M *to a string* $\mu(M)$ *of length polynomial in the description of the process, such that the algorithm, with the string and the observations as input, computes the optimum decision for the process in polynomial time.*

Sketch. If such an $\mathscr{A}$ and μ existed, then we could express the question of whether M has zero expected cost (and thus, by the previous proof, whether an arbitrary quantified Boolean formula is true) as follows: "There exists a string $\mu(M)$ such that, for all possible transitions, the decisions of algorithm $\mathscr{A}$ lead to a zero cost." This last sentence, however, can be rewritten as a Boolean formula with two alternations of quantifiers. ■

Finally, considering the special case of the partially observed problem, in which the partition $\Pi = \{S\}$ (i.e., the case of unobserved processes or, equivalently, an open-loop control) we note that a simplification of the proof of Theorem 6 establishes that this problem is NP-complete:

COROLLARY 2. *Deciding whether the optimal policy in an unobserved Markov decision process has expected cost* (*undiscounted, over a finite horizon*) *equal to zero is an NP-complete problem.* ■

Acknowledgement. We wish to thank Professor M. Katehakis for a number of helpful discussions and insightful comments, and an anonymous referee for careful reading of the manuscript and a number of helpful suggestions. This research was supported by an IBM faculty development award, and ONR contract N00014-85-C-0731.

References

[AHU] Aho, A. V., Hopcrost, J. E. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

[Be] Bertsekas, D. P. (1976). *Dynamic Programming and Stochastic Control*. Academic Press, New York.

[Co] Cook, S. A. (1981). Towards a Complexity Theory of Synchronous Parallel Computation. *Enseign. Math*. **2, 27** 99–124.

[DLR] Dobkin, D., Lipton, R. J. and Reiss, S., (1979). Linear Programming is Log-Space Hard for P. *Informat. Process Lett.* **8** 96–97.

[GSS] Goldschlager, L. M., Shaw, R. A. and Staples, J. (1982). The Maximum Flow Problem is Log Space Complete for P. *Theoret. Comput. Sci.* **21** 105–111.

[GJ] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

[Ho] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge.

[La] Lawler, E. L. (1977). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

[LP] Lewis, H. R. and Papadimitriou, C. H. (1981). *Elements of the Theory of Computation*. Prentice-Hall, Englewood Cliffs, NJ.

[Or] Orlin, J. (1981). The Complexity of Dynamic Languages and Dynamic Optimization Problems. *Proc. 13th STOC*, 218–227.

[Pa2] Papadimitriou, C. H. (1985). Computational Complexity. In Combinatorial Optimization: Annotated Bibliographies. M. O'hEigeartaigh, J. K. Lenstra, A. H. G. Rinooy Kan (eds.), 39–51.

[Pa1] ______. (1985). Games Against Nature. *J. Comput. Systems Sci.* **31, 2** 288–301.

[PS] ______ and Steiglitz, K. (1982) *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

[Sa] Savitch, W. J. (1970). Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Systems Sci.* **4** 177–192.

[SS] Smallwood, R. D. and Sondik, E. J. (1973). The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Oper. Res.* **11** 1971–1088.

[SM] Stockmeyer, L. J. and Meyer, A. R. (1973). Word Problems Requiring Exponential Space. *Proc. 5th STOC*.

[SV] Shiloach, Y. and Vishkin, U. (1982). An $O(\log n)$ Parallel Connectivity Algorithm. *J. Algorithms* **3** 57–67.

PAPADIMITRIOU: DEPARTMENTS OF COMPUTER SCIENCE AND OPERATIONS RESEARCH, STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94305

TSITSIKLIS: DEPARTMENT OF ELECTRICAL ENGINEERING, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139