JOURNAL OF COMPUTER AND SYSTEM SCIENCES 31, 288–301 (1985)

Games against Nature

CHRISTOS H. PAPADIMITRIOU

Stanford University, Stanford, California and
National Technical University of Athens, Athens, Greece

Received April 26, 1984; revised April 29, 1985

We present a new characterization of PSPACE in terms of problems in a classical area in optimization, decision-making under uncertainty. These problems are modeled by certain games played against a disinterested opponent who makes moves at random. We show several natural problems of this sort to be PSPACE-complete. © 1985 Academic Press, Inc.

1. INTRODUCTION

In computer science, important concepts usually come with a plethora of alternative characterizations. Take the class PSPACE, for example. It can be defined as the class of problems solvable in polynomially bounded space by multitape Turing machines or by a similar model (deterministic or non-deterministic). It can also be defined equivalently as the class of problems reducible to some polynomial-depth combinatorial two-person game [15, 14]. Alternation [2] is an interesting variant of the latter point of view. A more recent, also "problem-oriented" characterization of PSPACE is the one in terms of problems involving *periodic* objects [11].

In this paper we propose an alternative characterization of PSPACE based on some of the most classical and well-studied problems in optimization: *decision-making under uncertainty*. Problems in this class are usually characterized by a discrete-time random process, the parameters of which can be influenced by dynamic decisions. Decisions are based on the current state, and the next state is a random variable with distribution that depends on the current decision. The goal is to minimize the expectation of some cost functional of the history of states and decisions. (See the next section for a formal definition of such problems.) There is a vast literature on the numerous variants of this problem (see e.g., the books [5, 6, 10, 1]). Typically, such a problem is solved by dynamic programming, with time and space costs that are usually exponential functions of the size of the input. In a handful of now classical cases, more clever specialized techniques have yielded polynomial-time algorithms [10, 5]. Linear programming is sometimes employed.

We can formulate a problem of decision-making under uncertainty as a new sort of game, in which one opponent is "disinterested" and plays at random, while the other tries to pick a strategy that maximizes the probability of winning—a "game against Nature"! Let SAPTIME (for stochastic-alternating polynomial time) denote

0022-0000/85 $3.00

the complexity class defined by such games, where the number of moves per play is bounded by a polynomial in the size of the input (description of the starting position). An equivalent formulation of this class is polynomial time bounded Turing machines, alternating between an existential (non-deterministic) mode, and a *stochastic* mode; the machine accepts if more than half the leaves of the computation tree selected by the decisions at the existential nodes are accepting.

A basic result is that SAPTIME = PSPACE. This is not hard to prove. In one direction we can use a space-efficient technique for formula evaluation, and in the other we use a construction similar to the one employed to show that $NP \subset PP$ [9]. The fact that all problems of decision-making under uncertainty can be solved in polynomial space had apparantly not been observed before. Furthermore, this result immediately suggests a new problem complete for PSPACE: The satisfiability problem with quantifiers alternating between an existential and a *stochastic* one.

What is more exciting, we show that there are some natural problems of decision-making under uncertainty that are complete for PSPACE. These problems are slight twists of classical tractable problems, and some of them had been well-known for their apparent intractability. We present four representative PSPACE-complete problems of this sort: (1) a variant of the scheduling problem under exponentially disturbed processing times (in fact, this was the motivating example for this work); (2) a variant of the Markov chain decision problem in which costs (or transition probabilities, or decision options) are based on the state history; (3) a dynamic graph reliability problem; and (4) the linear optimal control problem with noise and non-convex objective function. Problem (1) with tree-ordered tasks and two processors can be solved in polynomial time [3]; problem (2) without memory can be solved by polynomial-time dynamic programming [10]; the static version of problem (3) is $\#P$-complete [16]; finally, we can obtain the optimum solution to the problem (4) with convex costs *in closed form* [1].

In the next section we introduce a formalism for problems of decision-making under uncertainty, and present the four basic problems examined here. In Section 3 we introduce SAPTIME, and show that it is yet another disguise of PSPACE. Finally, in Section 4 we show that the four problems are PSPACE-complete.

2. The Problems

We shall present a formalism for problems of decision-making under uncertainty with a polynomially bounded number of sequential decisions. An *instance* of such a problem Π is a string over some alphabet. Given an instance x, there are $p(|x|)$ sets of states $S_0, S_1, \ldots, S_{p(|x|)}$ (where S_j contains the set of possible states after the jth decision). Each state in S_j is a string of length $p(|x|)$, where p is a fixed polynomial (for simplicity we use the same polynomial p throughout; naturally, if different polynomials are involved in our problem, we can always take the largest). S_0 contains just *the initial state*, and $S_{p(|x|)}$ the *final states*. Each state that is not final has a constant number of *decisions* $\{1, 2, \ldots, d\}$ associated with it, where d is a constant

depending on Π. If decision i is made at state $s \in S_j$, this results in a probabilistic distribution over S_{j+1}, say $D_s^i = \{(s_1^i, p_1^i), \ldots, (s_m^i, p_m^i)\}$, with all but $p(|x|)$ of the probabilities equal to zero. The s_k^i's are states in S_{j+1}, and the probabilities p_k^i are $p(|x|)$-bit real numbers between 0 and 1. A final state has an integer *cost* associated with it. The costs are numbers smaller than $2^{p(|x|)}$.

There are two fixed polynomial-time algorithms A and B associated with Π. Given an instance x, a state, a decision number i, and an integer k, A returns the pair (s_k^i, p_k^i), consisting of the kth next state with nonzero probability, under the given decision, together with the probability of transition. Algorithm B, given x and a state, decides whether the state is a final, and computes its cost. (In the examples that we shall be discussing, each decision and transition incurs some additive cost; however, notice that the costs incurred by decisions can be absorbed into states, until the final one is reached). Notice that, according to this definition, the structure of the problem is such that the total length of any chain of possible next states starting from an initial state is bounded by a polynomial in the size of the instance. Our goal is to devise a *strategy* for making decisions at the states so as to minimize the expected cost. As usual, in proving completeness we shall consider the decision problem of telling whether a strategy exists that achieves an expected cost below a given limit.

We introduce below several interesting examples of such problems. The verification that each of these (except the fourth) meets the requirements of our general definition is left to the reader.

1. *Stochastic Scheduling*

Let $\tau = \{T_1, \ldots, T_n\}$ be a finite set of tasks with *precedence constraints* [17] among them. The precedence constraints form a rooted in-tree (directed acyclic graph with outdegrees at most one) $P = (\tau, E)$ with the tasks as vertices. Intuitively, a precedence constraint (T_i, T_j) has the effect that task T_j cannot start executing until T_i has finished its execution. Let t_i, $i = 1, \ldots, n$ be *execution times* for the tasks. Then a *preemptive schedule of* τ *on* m *processors* in an association to each task T_i of a set of the form $\{(i_1, [a_1, b_1]), \ldots, (i_k, [a_k, b_k])\}$, where (1) the i_j's are indices of processors ($1 \leqslant i_j \leqslant m$) and the $[a_j, b_j]$'s are intervals of positive reals; (2) for any two pairs in the set assigned to a task, the intervals are disjoint; (3) for any two pairs in any set either the processors are distinct or the intervals are disjoint; (4) the lengths of the intervals in the set for T_i add to t_i; and (5) if (T_i, T_j) is an edge of the tree of precedence constraints, all b's in the set for T_i are no larger than any a in the set for T_j. The *makespan* of the schedule is the largest of the b_j's used.

We shall consider a dynamic, stochastic version of this problem, in which the execution times are *random variables with identical exponential distributions.* That is, the distribution of the execution time of each task is e^{-t}, independent of other execution times. We wish to minimize the expected completion time. We are interested in constructing a preemptive schedule of the tasks *on-line.* This is done as follows: We start by choosing a subset L of up to m leaves of the precedence tree, and assigning them to processors. When the first task ends (after an amount of time

that is a random variable exponentially distributed with mean $1/|L|$), we delete it from the tree, and assign processors to tasks that are leaves in the new tree, and so on, until there are no tasks left. Recall that the exponential distribution has the *memoryless property*, that is, the distribution of the remaining execution time is the same as the initial distribution. Thus, it is not a loss of generality to assume that scheduling decisions are made only at the times that tasks are completed.

This problem, which we call *Poisson-tree scheduling*, is a typical problem of decision-making under uncertainty. At each decision point, the state is a tree which we wish to schedule in minimum expected time. A decision entails choosing a set L of leaves of the tree of the remaining tasks, and starting them on the processors. We would like to minimize the sum of the expected lengths $1/|L|$ over all stages. The random element is the unpredictable choice by "nature" of the task among the running ones to finish first. The next states are the trees by deleting each of the leaves deleted; all next states are equiprobable.

This problem can be formulated as a dynamic programming recurrence. The intuition is that, once we have computed the optimal strategy for all subtrees of a given tree, we can compute the optimal decision for this tree by picking the set of leaves which has the optimum expected behavior. The precise recurrence for the optimum expected scheduling time $c(T)$ of a tree T is:

$$c(T) = \min_{L \text{ set of } \leqslant m \text{ leaves of } T} \frac{1}{|L|}\left(1 + \sum_{l \in L} c(T-l)\right).$$

Here the minimum is taken among all possible sets L of leaves of T, $|L| \leqslant m$. It is easy to verify that, in general, the algorithm implicit in this recurrence requires exponential time *and* space.

An interesting and difficult result [3] is that, for $m = 2$, a polynomial algorithm exists for reaching the optimal decision—namely, the intuitive "highest level first" strategy, which was known to be optimal for the deterministic case for *any* number of processors [17]. Unfortunately, the same strategy fails to work in the case of three or more processors, and no other algorithm for that problem is known.

An immediate reaction of a complexity-theorist to such a situation would be to try to prove that the three-processor problem is NP-complete. There is a catch, though: It is not clear that this problem is in *NP*—not even PSPACE! What is the complexity of *Poisson-tree scheduling* with three processors? It follows from the results of the next section that this problem is in PSPACE; furthermore, in Section 4 we show that a more complicated variant is PSPACE-complete.

2. *Markov Decision Process*

Suppose that we have a finite Markov process described by the stochastic matrix P. The rows and columns of the matrix are indexed by the states of the process, and the entry p_{ij} is the probability that the next state is j, given that the present state is i. In fact, let us assume for simplicity that the graph with the states as nodes and an

arc from i to j whenever $p_{ij} \neq 0$ is *layered*, that is, the states are partitioned into indexed stages, and the process proceeds from one stage to the next (in *optimization*, such processes model *finite horizon* problems). Suppose now that at each state s we can choose from a set D_s of *decisions*. Each decision $d \in D_s$ carries a cost $c(d)$, and influences the immediate transition probabilities. That is, once d is chosen, the transition matrix becomes $P + R_d$, where R_d is a matrix which is everywhere 0 except for the sth row; in that row it contains reals summing up to 0, and yielding nonnegative entries when added to P. The next state will be j with probability equal to the sjth entry of $P + R_d$. What is the policy which leads us from a source state to a sink state with the minimum expected cost?

This is a classic among problems that can be solved by dynamic programming in polynomial time [10]. A more general question, however, remains: What if decisions can influence future transition probabilities? Or, if the costs are affected by the *history* of states? These open questions were mentioned, for example, in [1]. To pick a concrete problem, just relax the requirement in the above problem that R_d be nonzero only at the sth row. We call the resulting problem *Dynamic Markov Process*. Dynamic programming now requires exponential time and space, and *Dynamic Markov Process* is not easily seen to belong in any complexity class below that. We shall show that it is complete for PSPACE.

3. *Dynamic Graph Reliability*

Valiant has studied the problem of computing the reliability of a graph [16]. We are given a directed, acyclic graph G with a source s and a sink t, and with a probability of failure $p(e)$ defined for each edge e. What is the probability that there is a path from s to t consisting exclusively of edges that have not failed? It is known that this problem is complete for $\#P$ [16].

We ask a harder question, called *Dynamic Graph Reliability*. Suppose that we wish to traverse the graph, from s to t, while it is falling apart. At any moment, the probability that edge e will fail before our next move is $p(e, v)$, where v is the node which we are currently visiting. We are seeking the strategy that maximizes the probability of successful traversal.

4. *Optimal Control*

A discrete-time linear system obeys the equation

$$x_{t+1} = Ax_t + Bu_t + w_t,$$

where $x_t \in \mathfrak{R}^n$ is the state vector at time t and x_0 is given, $u_t \in U \subset \mathfrak{R}^m$ is the control vector (our decision variable), and w_t is a random variable of given distribution—the "weather." U is the space of acceptable controls, typically a hyperrectangle. The problem is to find the optimal strategy $u_1, u_2, \ldots, u_{T-1}$ which minimizes the functional $(x_T - \hat{x}_T)' Q(x_T - \hat{x}_T) + \sum_{t=0}^{T-1} (u_t - \hat{u}_t)' R(u_t - \hat{u}_t))$, where Q and R are square matrices, and $\hat{x}_T$ and $\hat{u}_t$ are the desired final state and the sequence of desired controls.

When Q and R are positive-definite matrices (and therefore the cost funtional convex), and $U = \mathfrak{R}^m$, there is a well-known method, called the *matrix Ricatti equation* for solving this problem [1]. In fact, this method obtains the optimum sequence of controls in *closed form*. A polynomial-time algorithm exists for the case of U being a hyper-rectangle $U = \{u \in \mathfrak{R}^m: a \leqslant u \leqslant b\}$ for some m-vectors a and b; however, Khacian's "ellipsoid" algorithm for linear programming must be employed. In contrast, we show that when R is not positive-definite, the problem is PSPACE-hard. This resembles the situation with *quadratic programming*, the problem of minimizing a quadratic form over a convex polytope. Quadratic programming is solvable by Khacian's algorithm if the quadratic form involved is positive-definite, but is *NP*-hard in general [12].

It is not clear that *Stochastic Control* is in PSPACE, the reason being that we cannot show any useful bounds on the precision required in order to compute the components of the state space and the controls. For the same reason, it cannot be formulated in the precise formalism of problems of decision-making under uncertainty introduced earlier in this section, since the state space in not a priori finite.

3. SAPTIME

Without loss of generality, we assume that our nondeterministic Turing machines have two choices at each step, and that, for each input, all computation paths have length equal to a fixed polynomial (depending on the machine) in the size of the input. A *stochastic* Turing machine is such that a nondeterministic Turing machine, with the following unusual acceptance convention: Odd-numbered steps are considered *stochastic* steps, while even-numbered steps are *existential* ones. Given the computation tree of such a machine on some input, we define an *admissible subtree* to be any subtree that results if we delete from each existential step one choice (and the subtree hanging from it). We say that the machine *accepts* the input if there is an admissible tree with more accepting leaves than rejecting ones. Let SAPTIME denote the class of all languages accepted by such machines.

PROPOSITION 1. *Any problem of decision-making under uncertainty is in SAPTIME. Conversely, any problem in SAPTIME can be formulated as a problem of decision-making under uncertainty.*

Sketch. Let Π be a problem of decision-making under uncertainty (recall the definition of our previous section); we assume that we have turned it into a "yes–no" problem in the usual manner. That is, in Π we are given an instance x and an integer bound B for the expected cost, and we are asked whether a policy exists that yield expectated cost B or less. We shall design a stochastic Turing machine M that accepts x iff the answer is "yes."

Our description of M will not strictly alternate between stochastic and nondeterministic nodes, but this can be rectified by inserting "dummy" nodes of either kind.

For each step of the decision problem, M starts with a configuration that encodes the current state in the tape. M then "guesses" the next decision one bit at a time. For each decision, we have several possible outcomes, with probabilities that have $2^{-p(|x|)}$ as a divisor. Then M creates a tree of stochastic steps with depth $p(|x|)$, and assigns the appropriate number of leaves of this tree to each state that is a possible outcome of the present one (the algorithm A in our definition is used here). For the final states, a tree of nondeterminstic decisions with a carefully calculated number of accepting leaves is created. If the cost of the final state is c, then we create at this particular final state a tree of depth $p(|x|)$ with $2^{p(|x|)} - c$ accepting leaves (recall that $c \leqslant 2^{p(|x|)}$; here we used algorithm B). Finally, at the initial configuration of the machine we create a stochastic step that leads to another computation,with the same number s of stochastic steps in each path as in any path of the main computation tree of the machine, in which all strategies yield exactly $2^s - 2^{p(|x|)}B + 1$ accepting leaves. It follows that there is a strategy with more than half the leaves accepting iff there is a strategy in the original game with cost B or less

For the converse, it is immediate that the configurations of a stochastic machine on input x are the states of a problem of decision-making under uncertainty, in which the stages are the time units, we have binary decisions corresponding to the nondeterministic steps, and, for each decision, two equiprobable outcomes; the cost of a final state is either zero or one. ▌

This proposition justifies our claim that SAPTIME is the right complexity class for problems of decision-making under uncertainty. The next result classifies these problems in terms of more conventianal concepts in complexity:

THEOREM 1. SAPTIME = PSPACE.

Proof. To show that SAPTIME $\subset$ PSPACE, we have to program a nondeterministic polynomial-space bounded machine to guess a strategy and count the total number of accepting leaves in a computation of a stochastic machine. This is done as follows: Our nondeterministic machine traverses the computation tree of the stochastic one. Going down from an existential node, it guesses "left" or "right"; going down from a stochastic node, it takes the left branch. Once it reaches a leaf, it adds one to its counter iff the leaf is accepting, and starts upwards. Going up to an existential node, it continues upwards; going up to a stochastic node, if it came from the left it next goes down to the right, otherwise it continues upwards. If the total count has more than $p(|x|)/2 - 1$ bits (the number of stochastic nodes minus one), then the machine accepts, otherwise it rejects.

For the other direction, suppose that we have an alternating Turing machine A. We design a stochastic one, S, which behaves as follows: S starts by a stochastic step, which creates two equiprobable possibilities: In the first, S simulates A faithfully. In the second possibility, S has computations of the same length as A, all of which *except for one* are rejecting. It turns out that the only way that an admissible subtree with more accepting than rejecting leaves can be found in the

computation tree of S is by having an accepting alternating computation in the first possibility (i.e., an admissible subtree with only accepting leaves), reinforced by the single accepting leaf of the second possibility. (Notice that this is essentially the argument used by Gill to show that $NP \subset PP$ [9]. ▮

4. Complete Problems

We first introduce our "generic" problem for SAPTIME. (A related problem had been introduced (and shown PSPACE-complete) by [13].)

Stochastic Satisfiability (SSAT)

We are given a Boolean formula F in conjunctive normal form and with three literals per clause, involving variables $x_1,..., x_n$ (where n is even). We are asked whether there is a choice of Boolean value for x_1 such that, for a random choice (with probability of true equal to $\frac{1}{2}$) of truth value for x_2 there is a choice for x_3, etc., so that the probability that F comes out true under these choices is more than half. We denote this as follows (read $\mathbf{R}x$ as "for random"):

$$F = \exists x_1,\ \mathbf{R}x_2 \exists x_3 ... \mathbf{R}x_n\ [\Pr(F(x_1,..., x_n) = \textbf{true}) > \tfrac{1}{2}].$$

Theorem 2. *SSAT is* PSPACE-*complete.*

Proof. It is clearly in PSPACE, by Proposition 1. To show completeness, we use a variant of the proof of Cook's theorem [4]. We have as usual a Boolean variable (existentially quantified) for each combination of time unit, tape cell, state, and head position; also, we have an existential variable for each time unit in which the machine makes a decision. Also, we have a stochastic Boolean variable for each time unit in which the machine makes a stochastic move. By inserting "dummy" Boolean variables, we can have a strictly alternating sequence of variables. Finally, the formula says that the machine accepts. ▮

From this, we can show the problems in Section 2 to be PSPACE-hard. We start with *Dynamic Markov Process.*

Theorem 3. *Dynamic Markov Process is* PSPACE-*complete.*

Proof. In order to reduce the SSAT problem to *Dynamic Markov Process*, we employ a new variant of the standard methodology for showing *NP*-completeness of problems related to the traveling salesman problem. (see, e.g., [8, 12]). We are given a quantified formula with variables $x_1,..., x_n$, and clauses $C_1,..., C_m$. We shall create an example of *Dynamic Markov Process*, that is, a set of states, decisions for each state, transition probabilities, costs for each decision, modifiers for the transition matrix, and a bound B, such that there is a strategy achieving an expected cost of B or less iff there is a sequence of choices for the existential variables, each

depending on previous existential choices and the previous stochastic values, such that more than half of the assignments come out satisfying the formula.

For each clause C_i we have a separate stage in our process, with a single state s_i. There are four possible next states from this stage. The first three l_{i1}, l_{i2}, l_{i3}, correspond to each of the three literals of the clause being **true**; the last, s_i' to the clause being **false**. The latter state has a single decision, with a prohibitive cost of one. This decision leads immediately to the final stage (a simple alternative construction retains the layered structure). Initially, the transition probabilities from s_i to each of these four states is 0.25. There are cost-free decisions at s_i modifying these transitions probabilities by shifting a total of three multiples of 0.25 to a particular one among the three first states, a total of twelve decisions (one for each literal, and each subset of the remaining two). Thus, we can always make the transition to s_i' impossible, unless it has already accumulated all probability (see the next paragraph for the ways that this can happen).

For each variable our process has a stage, and in fact the stages of the variables are arranged in the order of quantification, and before the clause stages. If this variable is an existential variable, we have two decisions corresponding to **true** and **false**. If it is a stochastic one, there is no decision involved, but there are two equiprobable next states, corresponding to **true** and **false**, with a single decision in each. The modifiers of all these decisions corresponding to literals, when added to the transition matrix, make impossible the transition from any clause to the corresponding contradicting literal, by adding its 0.25 probability to the transition to s_i'. The costs of all decisions, except those from the s_i' states, are zero.

It is rather straightforward to argue that a path through this process has expected cost zero after the variable part is traversed, if and only if the implied truth assignment satisfies the formula; otherwise the cost is 0.25 (the 1 incurred at the first **false** clause, times the probability 0.25 of this transition happening). Therefore, a strategy that keeps the expected cost below 0.125 is one that satisfies the formula with probability more than half. ▮

THEOREM 4. *Dynamic Graph Reliability is* PSPACE*-complete.*

Proof. In this proof we shall use a different version of SSAT, called SSAT′. The intuitive reason why the original SSAT is not appropriate for this problem is the following: In SATT the stochastic steps are choices between two possibilities, whereas in *Dynamic Graph Reliability* the stochastic steps determine whether certain possibilities (i.e., edges) exist.

In SATT′ we are given again a Boolean formula with an alternating prefix; we are also given a rational number $b \in [0, 1]$. In this version, however, the stochastically quantified variables have a probability of 0.5 for each of their two values to become *unavailable*. A strategy chooses a truth value for x_1, and then it is determined which (if any) of the values true and false for x_2 are available (with probability 0.25 both are available, with probability 0.25 only **true** is available, etc.). Next, the strategy deterministically chooses one of the available values for x_2, and a

value for x_3 (based on the previous choices), then the availability of values for x_4 is determined, and so on. If at some point there is no value of the variable available, then the formula was not satisfied. We accept iff there is a strategy that satisfies the formula with probability at least b, where the probability space is now the set of available values for the stochastic variables.

LEMMA. *SSAT′ is* PSPACE-*complete.*

Proof. SSAT′ is a problem of decision-making under uncertainty, as defined in Section 2, and thus, by Proposition 1, it is in PSPACE. To prove completeness, take any instance of quantified satisfiability (QSAT), the traditional PSPACE-complete problem [15]. We obtain an equivalent instance of SSAT′ by simply interpreting the universal quantifiers (say, n of them) as stochastic ones, and having $b = (\frac{3}{4})^n$. It is not hard to show that, for any strategy of choices for the existential variables, the probability of satisfaction is at most $(\frac{3}{4})^n$; this is achieved if all leaves in the subtree corresponding to the existential choices are accepting. We conclude that there is a strategy that achieves probability b of satisfaction iff the instance of QSAT is satisfiable. ▮

Now, to reduce SSAT′ to *Dynamic Graph Reliability*, suppose we have an instance of SSAT′ with variables $x_1,..., x_n$ and clauses $C_1,..., C_m$, and with bound b. We construct a graph G as follows: For each variable x, G has three nodes x, x^T, x^F; they are connected by the arcs (x, x^T) and (x, x^F). There are arcs leading from the x^T, x^T nodes to the next variable. Also, for each clause C_j we have a node C_j, with three parallel arcs (corresponding to the three literals of the clause) going from C_j to C_{j+1}. There are also arcs from x_n^T, x_n^F to C_1, and an arc all the way from x_1 to C_{m+1}.

We shall now define the probabilities of failure $p(e, v)$. For each stochastic variable x, $p((x, x^T), x) = p((x, x^F), x) = 0.5$. For all variables x, if e is an arc in the clause part of the graph corresponding to the literal x, then we have $p(e, x^F) = 1$; if e corresponds to the literal **not** x, then $p(e, x^T) = 1$. Finally, the probability $p((x_1, C_{m+1}), x_1)$ is taken to be $\frac{1}{2} + b + \varepsilon$, where ε is a number below the precision required for the joint probabilities; say, $\varepsilon = 2^{-3n}$. All other probabilities of failure are 0.

What is the best strategy for traversing G from x_1 to C_{m+1}? When in x_1, if the arc to C_{m+1} has not failed, we go directly to C_{m+1}. Otherwise, we go through the variables and clauses, trying to pick the best values for the existential variables, and available values for the stochastic ones, in order to satisfy all clauses, and therefore be able to reach C_{m+1}. We succeed with probability $1 - p((x_1, C_{m+1}), x_1)$ plus the probability of satisfying the formula. The overall probability of success is above $\frac{1}{2}$ iff this latter probability is b or more. ▮

The three-machine *Poisson-tree scheduling* problem remains a very interesting open problem; we do not know whether it is PSPACE-complete or polynomially solvable (or what). However, we can exhibit a generalization of the tractable two-

machine problem in another direction, which is PSPACE-complete. We have a tree-structured task system with two processors and exponentially distributed execution times with identical parameters. The new feature that makes the problem complex is a *dominance* relation D between the tasks. If a task a dominates another task b, this means that, if a is completed, then b becomes unnecessary, and need not be executed. The tasks it precedes may start immediately, if they are not preceeded by other tasks. This problem may sound complex and artificial, but in fact is a very special case of GERT, a stochastic generalization of critical-path scheduling.

THEOREM 5. *Poisson-tree scheduling with two processors and a dominance relation is* PSPACE-complete.

Proof. We are given an instance of SSAT with variables $x_1,..., x_{2n}$ and with clauses $C_1,..., C_m$. We have to construct a tree and a dominance relation connecting a set of exponentially distributed tasks with unit mean as well as a bound B, so that there is a scheduling strategy that achieves expected timespan B or less iff the instance of SSAT is satisfiable.

For each existential variable x of the formula, we have four tasks in our system, x^{T1}, x^{T2}, x^{F1}, x^{F2}. The first two dominate each other, and so do the latter two. The idea is that, once we reach this stage, we will have to make a decision between $x =$ **true** and $x =$ **false**, and this entails choosing one of the two pairs for execution. Choosing one element from each pair would be suboptimal, because it would mean yielding a move to Nature. For each stochastic variable x we have two tasks x^T, x^F, with the intention of scheduling them both for execution, and having Nature decide which will terminate. Also, we have a task for each clause C_j and a root r.

For each variable, existential or stochastic, we have the nodes with superscript T dominate all clauses in which the variable appears positively, and the nodes with superscript F dominate all clauses in which the variable appears negated. Thus, if a clause is satisfied by the truth assignment implied by the outcome of the schedule, then it need not be executed. Finally, all clauses dominate each other, so we will never have to execute more than one clause task.

We need some mechanism for enforcing the above order in the scheduling, that is, to forbid, for example, task x_6^T from executing before task x_3^{F2}. This is easy to do. All nodes at the level of the variable x_i dominate a set of n "synchronization" nodes $\{s_i^k\}$, which precede the nodes at level x_{i+1}. Therefore, it is suboptimal to do anything else but go from one variable to the next. Finally, the clauses precede the root r, and the latter dominates all variable nodes. The precedence relation is obviously a forest. The bound is $n + \frac{3}{2}$.

We claim that there is a scheduling strategy that has expected timespan less than B iff there is a strategy satisfying the formula with probability more than half. Suppose such a satisfying strategy exists. Then we proceed by scheduling the appropriate pairs at the levels corresponding to the existential variables, the only pair at the stochastic ones, notice that after 0.5 time units on the average we shall be done, and the task that has finished is going to dominate the other one (in the

case of existential variables) and the synchronization nodes, so we can proceed to the next level. If the truth assignment resulting from the tasks that have completed satisfies the formula, then we can immediately proceed to the root, at an expected cost of $2n$ (number of stages) $\times 0.5$ (expected time per stage) $+ 1$ (the root) $= n + 1$. Otherwise, an extra time unit must be spent, on the average, over some clause (expected timespan $= n + 2$). Thus, if more than half the truth assignments in our strategy are satisfying, then the expected timespan will be less than B. Otherwise, it will be B or more. Conversely, a scheduling strategy is clearly suboptimal if it does not proceed in the manner suggested above, that is, to each variable in its turn, to one clause, if any, and the roots. The only possible outcomes from such a policy are expected timespans of $n + 1$ or $n + 2$. B can be beaten only if the corresponding satisfying strategy is successful more than half the times. ∎

THEOREM 6. *Optimal Control with R nonpositive-definite is* PSPACE-*hard, even if the dimension is one.*

Proof. We first transform SSAT to a dynamic version of exact cover [7]. In this version, the decision-maker and Nature alternate picking one set from a pair of sets presented to them (we allow also *empty* sets in these choices). The goal is to pick a collection of sets that are disjoint and cover the graph. We accept such sequences of pairs of sets iff there is a strategy that achieves probability of success more than half. To show completeness, we use a careful variant of the standard reduction from 3-SAT to 3-dimensional matching. In that reduction, the choice of truth value is represented by a choice between two sets T and F of triples. We simulate this in our reduction from SSAT to the stochastic variant of exact cover as follows: For an existential variable, there is an existential choice between an element of T and one of F. Subsequent existential choices will pick consistently the rest of the chosen set (inconsistent strategies are obviously suboptimal). We insert dummy stochastic choices, "choosing" between two empty sets. For each stochastic variable, we have a stochastic choice between an element of T and one of F, but the remaining elements of the chosen set are picked by existential choices. To make sure that all clauses are satisfied, we have a sequence of three choices between the empty set and a set consisting of a clause and a literal; one literal must be still free, otherwise the clause will never be picked up. Finally, to collect the remaining nodes, we simply have choices between the empty set and singletons corresponding to the nodes that possibly have not been picked up yet. All these deterministic steps are interleaved with dummy stochastic ones. There is a strategy that succeeds half of the time iff there was a strategy for the instance of SSAT.

From dynamic exact cover we can go to a dynamic version of the knapsack problem. We are given a sequence of pairs of integers $(a_1, b_1), \ldots, (a_{2n}, b_{2n})$, and a target K. Odd-numbered pairs are existential moves, even-numbered ones are stochastic moves. Is there a strategy of picking one number from each pair on the odd steps so that the probability that we achieve K in the end is more than half? This problem is PSPACE-complete, as the old idea of considering the characteristic

vectors of the sets as integers in a sufficiently large basis reduces the dynamic version of exact cover discussed above to dynamic knapsack. In fact, we can arrange things so that the sum achieved is either the target value, or $K+1$.

Suppose now that our *Optimal Control* instance is a one-dimensional one, such that the control u_t is bound to be between two values a_{2t-1}, b_{2t-1}, the desired control value is $\hat{u}_t=(a_{2t-1}+b_{2t-1})/2$, and the matrix R is (-1); this makes the extreme values a_{2t-1}, b_{2t-1} least costly. The desired final value is $\hat{x}_T=K$, and the matrix Q is (1) (positive-definite). Also, the random variable w_t takes the values a_{2t}, b_{2t}, with probability 0.5. That is, our instance of *Optimal Control* faithfully simulates dynamic knapsack, with the choices of the decision-maker embedded in the controls, and the choice of Nature in w_t. There is a control strategy $u_1,\ldots, u_{T-1}$ with expected cost $-\sum((a_{2t-1}-b_{2t-1})^2/4)+\frac{1}{2}$ or less iff there is a strategy in the dynamic knapsack instance achieving K with probability at least half. ▮

5. Conclusion

We have presented a new characterization of PSPACE, in terms of a classical area of Optimization Theory, the complexity of which had not been explored. We showed that several important problems in this area can be proved intractable using this characterization. For a direction of future research, we note that certain popular games of *skill and luck*—such as backgammon—can be formulated as computations alternating between an existential, a universal, and a stochastic quantifier, a simple extension of the case discussed so far. One might conjecture that (the obvious parametric generalization of) backgammon, for example, is PSPACE-complete.

Acknowledgments

I want to thank John Bruno who introduced me to the three-machine Poisson scheduling problem, which led to this paper (but still remains open). Many thanks to David S. Johnson for helping me improve and clarify the exposition. This research was supported by the National Science Foundation.

References

1. D. Bertsekas, "Dynamic Programming and Stochastic Control," Academic Press, New York, 1976.
2. A. Chandra and L. J. Stockmeyer, Alternation *in* "Proc. 17th Found. of Comput. Sci.," 1976.
3. M. Chandy and J. Robinson, unpublished manuscript, University of Texas, Austin, 1982.
4. S. A. Cook, The complexity of theorem proving procedures, *in* "Proc. 3rd ACM Symposium on the Theory of Computing," 1971, 151–158.
5. E. Denardo, "Dynamic Programming: Models and Applications," Prentice–Hall, Englewood Cliffs, N.J., 1982.
6. C. Derman, "Finite State Markov Decision Processes," Academic Press, New York, 1972.
7. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness," Freeman, San Francisco, 1979.

8. M. R. GAREY, D. S. JOHNSON, AND R. E. TARJAN, The planar Hamilton circuit problem is NP-complete, *SIAM J. Comput.* **5** (1976), 704–714.
9. J. GILL, The computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6** (1977), 675–695.
10. R. HOWARD, "Dynamic Programming and Markov Processes," MIT Press, Cambridge, Mass., 1960.
11. J. ORLIN, The complexity of periodic of periodic problems, *in* "Proc. 21st ACM Sympos. Theory of Comput.," 1980.
12. C. H. PAPADIMITRIOU AND K. STEIGLITZ, "Combinatorial Optimization: Algorithms and Complexity," Prentice-Hall, Englewood Cliffs, N.J., 1982.
13. J. REIF, Logics for probabilistic programming, *in* "Proc. 12th ACM Sympos. Theory of Comput.," 1980.
14. T. J. SCHAEFER, The complexity of some two-person perfect-information games, *J. Comput. System Sci.* **16** (1978), 185–225.
15. L. J. STOCKMEYER, The polynomial time hierarchy, *Theor. Comput. Sci.* **1** (1976), 1–22.
16. L. G. VALIANT, The complexity of computing the permanent, *Theor. Comput. Sci.* **00** (1979).
17. E. G. COFFMAN (Ed.), "Computer and Job-shop Scheduling Theory," Wiley, New York, 1976.

Printed in Belgium