Journal of Heuristics, 5, 89–108 (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Rollout Algorithms for Stochastic Scheduling Problems

DIMITRI P. BERTSEKAS*
Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass., 02139.

DAVID A. CASTAÑON
Department of Electrical Engineering, Boston University, and ALPHATECH, Inc., Burlington, Mass., 01803.

Abstract. Stochastic scheduling problems are difficult stochastic control problems with combinatorial decision spaces. In this paper we focus on a class of stochastic scheduling problems, the quiz problem and its variations. We discuss the use of heuristics for their solution, and we propose rollout algorithms based on these heuristics which approximate the stochastic dynamic programming algorithm. We show how the rollout algorithms can be implemented efficiently, with considerable savings in computation over optimal algorithms. We delineate circumstances under which the rollout algorithms are guaranteed to perform better than the heuristics on which they are based. We also show computational results which suggest that the performance of the rollout policies is near-optimal, and is substantially better than the performance of their underlying heuristics.

Keywords: rollout algorithms, scheduling, neuro-dynamic programming

1. Introduction

Consider the following variation of a planning problem: There is a finite set of K locations which contain tasks of interest, of differing value. There is a single processor on which the tasks are to be scheduled. Associated with each task is a task-dependent risk that, while executing that task, the processor will be damaged and no further tasks will be processed. The objective is to find the optimal task schedule in order to maximize the expected value of the completed tasks.

The above is an example of a class of stochastic scheduling problems known in the literature as *quiz problems* (see Bertsekas (1995), Ross (1983), or Whittle (1982)). The simplest form of this problem involves a quiz contest where a person is given a list of N questions and can answer these questions in any order he/she chooses. Question i will be answered correctly with probability p_i, and the person will then receive a reward v_i. At the first incorrect answer, the quiz terminates and the person is allowed to keep his or her previous rewards. The problem is to choose the ordering of questions so as to maximize expected rewards.

The problem can be viewed in terms of dynamic programming (DP for short), but can more simply be viewed as a deterministic combinatorial problem, whereby we are seeking an optimal sequence in which to answer the questions. It is well-known that for the simple form of the quiz problem described above, the optimal sequence is deterministic, and can be obtained using an interchange argument; questions should be answered in decreasing order

* This work was supported in part by the Air Force Office of Scientific Research under grant no. F49620-97-C-0013 and the Air Force Research Laboratory under grant F33615-96-C-1930.

of $p_i v_i/(1-p_i)$. Thus, this quiz problem belongs to the class of scheduling problems that admit an optimal policy, which is of the index type. This particular policy can also be used in variants of the quiz problem, where it is not necessarily optimal, and will be referred to as the *index policy*. Another interesting policy for quiz problems is the *greedy policy*, which answers questions in decreasing order of their expected reward $p_i v_i$. A greedy policy is suboptimal for the simple form of the quiz problem described above, essentially because it does not consider the future opportunity loss resulting from an incorrect answer.

Unfortunately, with only minor changes in the structure of the problem, the optimal solution becomes much more complicated (although DP and interchange arguments are still relevant). Examples of interesting and difficult variations of the problem involve one or more of the following characteristics:

(a) A limit on the maximum number of questions that can be answered, which is smaller than the number of questions N. To see that the index policy is not optimal anymore, consider the case where there are two questions, only one of which may be answered. Then it is optimal to use the greedy policy rather than the index policy.

(b) A time window for each question, which constrains the set of time slots when each question may be answered. Time windows may also be combined with the option to refuse answering a question at a given period, when either no question is available during the period, or answering any one of the available questions involves excessive risk.

(c) Precedence constraints, whereby the set of questions that can be answered in a given time slot depends on the immediately preceding question, and possibly on some earlier answered questions.

(d) Sequence-dependent rewards, whereby the reward from answering correctly a given question depends on the immediately preceding question, and possibly on some questions answered earlier.

It is clear that the quiz problem variants listed above encompass a very large collection of practical scheduling problems. The version of the problem with time windows and precedence constraints relates to vehicle routing problems (involving a single vehicle). The version of the problem with sequence-dependent rewards, and a number of questions that is equal to the maximum number of answers relates to the traveling salesman problem. Thus, in general, it is very difficult to solve the variants described above exactly.

An important feature of the quiz problem, which is absent in the classical versions of vehicle routing and traveling salesman problems is that *there is a random mechanism for termination of the quiz*. Despite the randomness in the problem, however, in all of the preceding variants, there is an *optimal open-loop policy*, i.e., an optimal order for the questions that does not depend on the random outcome of the earlier questions. The reason is that we do not need to plan the answer sequence following the event of an incorrect answer, because the quiz terminates when this event occurs. Thus, we refer to the above variations of the quiz problem as *deterministic quiz problems*.

There are variants of the quiz problem where the optimal order to answer questions depends on random events. Examples of these are:

(e) There is a random mechanism by which the quiz taker may miss a turn, i.e., be denied the opportunity to answer a question at a given period, but may continue answering questions at future time periods.

(f) New questions can appear and/or old questions can disappear in the course of the quiz according to some random mechanism. A similar case arises when the start and end of the time windows can change randomly during the quiz.

(g) There may be multiple quiz takers that answer questions individually, and drop out of the quiz upon their own first error, while the remaining quiz takers continue to answer questions.

(h) The quiz taker may be allowed multiple chances, i.e., may continue answering questions up to a given number of errors.

(i) The reward for answering a given question may be random and may be revealed to the quiz taker at various points during the course of the quiz.

The variants (e)–(i) of the quiz problem described above require a genuinely stochastic formulation as Markovian decision problems. We refer to these variations in the paper as *stochastic quiz problems*. They can be solved exactly only with DP, but their optimal solution is prohibitively difficult. This is because the states over which DP must be executed are subsets of questions, and the number of these subsets increases exponentially with the number of questions.

In this paper, we develop suboptimal solution approaches for deterministic and stochastic quiz problems that are computationally tractable. In particular, we focus on rollout algorithms, a class of suboptimal solution methods inspired by the policy iteration methodology of DP and the approximate policy iteration methodology of neuro-dynamic programming (NDP for short). One may view a rollout algorithm as a single step of the classical policy iteration method, starting from some given easily implementable policy. Algorithms of this type have been sporadically proposed in several DP application contexts. They have also been proposed by Tesauro and Galperin (1996) in the context of simulation-based computer backgammon. (The name “rollout” was introduced by Tesauro as a synonym for repeatedly playing out a given backgammon position to calculate by Monte Carlo averaging the expected game score starting from that position.)

Rollout algorithms were first proposed for the approximate solution of discrete optimization problems by Bertsekas and Tsitsiklis (1996), and by Bertsekas, Tsitsiklis, and Wu (1997), and the methodology developed here for the quiz problem strongly relates to the ideas in these sources. Generally, rollout algorithms are capable of magnifying the effectiveness of any given heuristic algorithm through sequential application. This is due to the policy improvement mechanism of the underlying policy iteration process.

In the next section, we introduce rollout algorithms for deterministic quiz problems, where the optimal order for the questions from a given period onward does not depend on earlier random events. In Section 3, we provide computational results indicating that rollout algorithms can improve impressively on the performance of their underlying heuristics. In Sections 4 and 5, we extend the rollout methodology to stochastic quiz problems [cf. variants (e)–(i) above], that require the use of stochastic DP for their optimal solution. Here we introduce the new idea of using *multiple scenarios* for the future uncertainty starting from a given state, and we show how these scenarios can be used to construct an approximation to the optimal value function of the problem using NDP techniques and a process of *scenario aggregation*. In Section 6, we provide computational results using rollout algorithms for stochastic quiz problems. Finally, in Section 7, we provide computational results using rollout algorithms for quiz problems that involve graph-based precedence constraints. Our results indicate consistent and substantial improvement of rollout algorithms over their underlying heuristics.

2. Rollout Algorithms for Deterministic Quiz Problems

Consider a variation of a quiz problem of the type described in (a)–(c) above. Let N denote the number of questions available, and let M denote the maximum number of questions which may be attempted. Associated with each question i is a value v_i, and a probability of successfully answering that question p_i. Assume that there are constraints such as time windows or precedence constraints which restrict the possible question orders. Denote by $V(i_1, \ldots, i_M)$ the expected reward of a feasible question order $(i_1, \ldots, i_M)$:

$$V(i_1, \ldots, i_M) = p_{i_1}\left(v_{i_1} + p_{i_2}(v_{i_2} + p_{i_3}(\cdots + p_{i_M} v_{i_M})\cdots)\right). \tag{2.1}$$

For an infeasible question order $(i_1, \ldots, i_M)$, we use the convention

$$V(i_1, \ldots, i_M) = -\infty.$$

The classical quiz problem is the case where $M = N$, and all question orders are feasible. In this case, the optimal solution is simply obtained by using an interchange argument. Let i and j be the kth and $(k+1)$st questions in an optimally ordered list

$$L = (i_1, \ldots, i_{k-1}, i, j, i_{k+2}, \ldots, i_N).$$

Consider the list

$$L' = (i_1, \ldots, i_{k-1}, j, i, i_{k+2}, \ldots, i_N)$$

obtained from L by interchanging the order of questions i and j. We compare the expected rewards of L and L'. We have

$$\begin{aligned} E\{\text{reward of } L\} &= E\left\{\text{reward of } \{i_1, \ldots, i_{k-1}\}\right\} \\ &\quad + p_{i_1}\cdots p_{i_{k-1}}(p_i v_i + p_i p_j v_j) \\ &\quad + p_{i_1}\cdots p_{i_{k-1}} p_i p_j E\left\{\text{reward of } \{i_{k+2}, \ldots, i_N\}\right\} \end{aligned}$$

$$\begin{aligned} E\{\text{reward of } L'\} &= E\left\{\text{reward of } \{i_1, \ldots, i_{k-1}\}\right\} \\ &\quad + p_{i_1}\cdots p_{i_{k-1}}(p_j v_j + p_j p_i v_i) \\ &\quad + p_{i_1}\cdots p_{i_{k-1}} p_j p_i E\left\{\text{reward of } \{i_{k+2}, \ldots, i_N\}\right\}. \end{aligned}$$

Since L is optimally ordered, we have

$$E\{\text{reward of } L\} \geq E\{\text{reward of } L'\},$$

so it follows from these equations that

$$p_i v_i + p_i p_j v_j \geq p_j v_j + p_j p_i v_i$$

or equivalently

$$\frac{p_i v_i}{1 - p_i} \geq \frac{p_j v_j}{1 - p_j}.$$

It follows that to maximize expected rewards, questions should be answered in decreasing order of $p_i v_i/(1 - p_i)$, which yields the index policy.

Unfortunately, the above argument breaks down when either $M < N$, or there are constraints on the admissibility of sequences due to time windows or precedence constraints. For these cases, we can still use heuristics such as the index policy or the greedy policy, but they will not perform optimally.

Consider a heuristic algorithm, which given a *partial schedule* $P = (i_1, \ldots, i_k)$ of distinct questions constructs a *complementary schedule* $\bar{P} = (i_{k+1}, \ldots, i_M)$ of distinct questions such that $P \cap \bar{P} = \emptyset$. The heuristic algorithm is referred to as the *base heuristic*. We define the *heuristic reward* of the partial schedule P as

$$H(P) = V(i_1, \ldots, i_k, i_{k+1} \ldots, i_M). \tag{2.2}$$

If $P = (i_1, \ldots, i_M)$ is a complete solution, by convention the heuristic reward of P is the true expected reward $V(i_1, \ldots, i_M)$.

Given the base heuristic, the corresponding *rollout algorithm* constructs a complete schedule in M iterations, one question per iteration. The rollout algorithm can be described as follows:

At the 1st iteration it selects question i_1 according to

$$i_1 = \arg \max_{i=1,\ldots,N} H(i), \tag{2.3}$$

and at the kth iteration ($k > 1$) it selects i_k according to

$$i_k = \arg \max_{\{i \mid i \neq i_1,\ldots,i_{k-1}\}} H(i_1, \ldots, i_{k-1}, i), \quad k = 2, \ldots, M. \tag{2.4}$$

Thus a rollout policy involves $N + (N - 1) + \cdots + (N - M) = O(MN)$ applications of the base heuristic and corresponding calculations of expected reward of the form (2.1). While this is a significant increase over the calculations required to apply the base heuristic and compute its expected reward, the rollout policy is still computationally tractable. In particular, if the running time of the base heuristic is polynomial, so is the running time of the corresponding rollout algorithm. On the other hand, it will be shown shortly that the expected reward of the rollout policy is at least as large as the one of the base heuristic.

As an example of a rollout algorithm, consider the special variant (a) of the quiz problem in the preceding section, where at most M out of N questions may be answered and there are no time windows or other complications. Let us use as base heuristic the index heuristic, which given a partial schedule $(i_1, \ldots, i_k)$, attempts the remaining questions according to the index policy, in decreasing order of $p_i v_i/(1 - p_i)$. The calculation of $H(i_1, \ldots, i_k)$ is done using Eq. (2.1), once the questions have been sorted in decreasing order of index. The corresponding rollout algorithm, given $(i_1, \ldots, i_{k-1})$ selects i, calculates $H(i_1, \ldots, i_{k-1}, i)$ for all $i \neq i_1, \ldots, i_{k-1}$, using Eq. (2.1), and then optimizes this expression over i to select i_k.

Note that one may use a different heuristic, such as the greedy heuristic, in place of the index heuristic. There are also other possibilities for base heuristics. For example, one may first construct a complementary schedule using the index heuristic, and then try to improve this schedule by using a 2-OPT local search heuristic, that involves exchanges of positions of pairs of questions. One may also use multiple heuristics, which produce heuristic values $H_j(i_1, \ldots, i_k)$, $j = 1, \ldots, J$, of a generic partial schedule $(i_1, \ldots, i_k)$, and then combine them into a "superheuristic" that gives the maximal value

$$H(i_1, \ldots, i_k) = \max_{j=1,\ldots,J} H_j(i_1, \ldots, i_k).$$

An important question is whether the rollout algorithm performs at least as well as its base heuristic when started from the initial partial schedule. This can be guaranteed if the base heuristic is *sequentially consistent*. By this we mean that the heuristic has the following property:

Suppose that starting from a partial schedule

$$P = (i_1, \ldots, i_{k-1}),$$

the heuristic produces the complementary schedule

$$\bar{P} = (i_k, \ldots, i_M).$$

Then starting from the partial schedule

$$P^+ = (i_1, \ldots, i_{k-1}, i_k),$$

the heuristic produces the complementary schedule

$$\bar{P}^+ = (i_{k+1}, \ldots, i_M).$$

As an example, it can be seen that the index and the greedy heuristics, discussed earlier, are sequentially consistent. This is a manifestation of a more general property: many common base heuristics of the greedy type are by nature sequentially consistent. It may be verified, based on Eq. (2.4), that a sequentially consistent rollout algorithm keeps generating the same schedule $P \cup \bar{P}$, up to the point where by examining the alternatives in Eq. (2.4) and by calculating their heuristic rewards, it discovers a better schedule. As a result, sequential consistency guarantees that the reward of the schedules $P \cup \bar{P}$ produced by the rollout algorithm is monotonically nonincreasing; that is, we have

$$H(P^+) \leq H(P)$$

at every iteration. For further elaboration of the sequential consistency property, we refer to the paper by Bertsekas, Tsitsiklis, and Wu (1997), which also discusses some underlying connections with the policy iteration method of dynamic programming.

A condition that is more general than sequential consistency is that the algorithm be *sequentially improving,* in the sense that at each iteration there holds

$$H(P^+) \leq H(P).$$

This property also guarantees that the rewards of the schedules produces by the rollout algorithm are monotonically nonincreasing. The paper by Bertsekas, Tsitsiklis, and Wu (1997) discusses situations where this property holds, and shows that with fairly simple modification, a rollout algorithm can be made sequentially improving.

There are a number of variations of the basic rollout algorithm described above. In particular, we may incorporate *multistep lookahead* or *selective depth lookahead* into the rollout framework. An example of a rollout algorithm with m-step lookahead operates as follows: at the kth iteration we augment the current partial schedule $P = (i_1, \ldots, i_{k-1})$ with all possible sequences of m questions $i \neq i_1, \ldots, i_{k-1}$. We run the base heuristic from each of the corresponding augmented partial schedules, we select the m-question sequence with maximum heuristic reward, and then augment the current partial schedule P with the first question in this sequence. An example of a rollout algorithm with *selective* two-step lookahead operates as follows: at the kth iteration we start with the current partial schedule $P = (i_1, \ldots, i_{k-1})$, and we run the base heuristic starting from each partial schedule $(i_1, \ldots, i_{k-1}, i)$ with $i \neq I_1, \ldots, i_{k-1}$. We then form the subset $\bar{I}$ consisting of the n questions $i \neq i_1, \ldots, i_k$ that correspond to the n best complete schedules thus obtained. We run the base heuristic starting from each of the partial schedules $(i_1, \ldots, i_{k-1}, i, j)$ with $i \in \bar{I}$ and $j \neq i_1, \ldots, i_{k-1}, i$, and obtain a corresponding complete schedule. We then select as next question i_k of the rollout schedule the question $i \in \bar{I}$ that corresponds to a maximal reward schedule. Note that by choosing the number n to be smaller than the maximum possible, $N - k + 1$, we can reduce substantially the computational requirements of the two-step lookahead.

3. Computational Experiments with Deterministic Quiz Problems

In order to explore the performance of rollout algorithms for deterministic scheduling, we conducted a series of computational experiments involving the following seven algorithms:

(1) The optimal stochastic dynamic programming algorithm.

(2) The greedy heuristic, where questions are ranked in order of decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

(3) The index heuristic, where questions are ranked in order of decreasing $p_i v_i/(1 - p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

(4) The one-step rollout policy based on the greedy heuristic, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k-1}$, the question is chosen according to the rollout rule (2.4), where the function H uses the greedy heuristic as the base policy.

(5) The one-step rollout policy based on the index heuristic, where the function H in (2.4) uses the index heuristic as the base policy,

(6) The selective two-step lookahead rollout policy based on the greedy heuristic. At the k-th stage, the base heuristic is used in a one-step rollout to select the best four choices for the current question among the admissible choices. For each of these choices at stage k, the feasible continuations at stage $k+1$ are evaluated using the greedy heuristic to complete the schedule. The choice at stage k is then selected from the sequence with the highest evaluation.

(7) The selective two-step lookahead rollout policy that is similar to the one in (6) above, but is based on the index heuristic rather than the greedy heuristic.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which questions were answered correctly.

Our experiments focused on the effects of two factors on the relative performance of the different algorithms:

(a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8.

(b) The average percent of questions which can be answered at any one stage, which ranged from 10% to 50%.

The first set of experiments fixed the average percentage of questions which can be answered at a single stage to 10%, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6, and 0.8. For each experimental condition, we generated 30 independent problems and solved them, and evaluated the corresponding performance using 10,000 Monte Carlo runs. We computed the average performance across the 30 problems, and compared this performance with the performance obtained using the stochastic dynamic programming algorithm.

Table 1. Performance of the different algorithms as the minimum probability of success of answering a question varies. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Minimum Prob. of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	41%	50%	61%	76%
One-Step Rollout	75%	82%	88%	90%
Two-Step Rollout	81%	84%	88%	90%
Index Heuristic	43%	53%	66%	80%
One-Step Rollout	77%	83%	89%	90%
Two-Step Rollout	81%	86%	90%	91%

Table 1 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed in terms of the percentage of the optimal performance. The results indicate that rollout algorithms significantly improve the performance of both the greedy and the index heuristics in these difficult stochastic combinatorial problems, and achieve close to optimal performance. In particular, the rollouts recovered in all cases at least 50% of the loss of optimality due to the use of heuristic. Loss recovery of this order or better was typical in all of the experiments with rollout algorithms reported in this paper. The results also illustrate that the performance of the simple heuristics improves as the average probability of success increases, thereby reducing the potential advantage of rollout strategies. Even in these unfavorable cases, the rollout strategies improved performance levels by at least 10% of the optimal performance.

For the size of problems tested in these experiments, the advantages of using a two-step selective lookahead rollout were small. In many cases, the performances of the one-step rollout and the two-step lookahead rollout were identical. Nevertheless, for selected difficult individual problems, the two-step lookahead rollout improved performance by as much as 40% of the optimal strategy over the level achieved by the one-step rollout with the same base heuristic.

The second set of experiments fixed the lower bound on the probability of successfully answering a question to 0.2, and varied the average percent of questions which can be answered at any one stage across 3 levels: 10%, 30%, and 50%. As before, we generated 30 independent problems and evaluated the performance of each algorithm on each problem instance. The results of these experiments are summarized in Table 2. The performance of the greedy and index heuristics improves as the experimental condition approaches the standard conditions of the quiz problem, where 100% of the questions can be answered at any time. The results confirm the trend first seen in Table 1: even in cases where the heuristics achieve good performance, rollout strategies offer significant performance gains.

The results in Tables 1 and 2 suggest that the advantages of rollout strategies over the greedy and index heuristics increase proportionately with the risk involved in the problem. By constructing a feasible strategy for the entire horizon for evaluating the current decision,

Table 2. Performance of the different algorithms as the average number of questions per period increases. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Problem Density	0.1	0.3	0.5
Greedy Heuristic	41%	58%	76%
One-Step Rollout	75%	86%	91%
Two-Step Rollout	81%	90%	92%
Index Heuristic	43%	68%	85%
One-Step Rollout	77%	90%	93%
Two-Step Rollout	81%	92%	94%

rollout strategies account for the limited future accessibility of questions, and compute tradeoffs between future accessibility and the risk of the current choice. In contrast, myopic strategies such as the greedy and index heuristics do not account for future access to questions, and thus are forced to make risky choices when no other alternatives are present. Thus, as the risk of missing a question increases and the average accessibility of questions decreases, rollout strategies achieve nearly double the performance of the corresponding myopic heuristics. However, even in cases where the heuristics perform quite well, the rollouts resulted in significant recovery (at least 50%) of the loss of optimality due to the use of the heuristic.

4. Rollout Algorithms for Stochastic Quiz Problems

We now consider variants of the quiz problem where there is no optimal policy that is open-loop. The situations (e)–(i) given in Section 1 provide examples of quiz problems of this type. We can view such problems as stochastic DP problems. Their exact solution, however, is prohibitively expensive.

Let us state a quiz problem in the basic form of a dynamic programming problem, where we have the stationary discrete-time dynamic system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, T-1, \tag{4.1}$$

that evolves over T time periods. Here x_k is the state taking values in some set, u_k is the control to be selected from a finite set $U_k(x_k)$, w_k is a random disturbance, and f_k is a given function. We assume that the disturbance w_k, $k = 0, 1, \ldots$, has a given probability distribution that depends explicitly only on the current state and control. The one-stage cost function is denoted by $g_k(x, u, w)$.

To apply the rollout framework, we need to have a base policy for making a decision at each state-time pair (x_k, k). We view this policy as a sequence of feedback functions

$\pi = \{\mu_0, \mu_1, \ldots, \mu_T\}$, which at time k maps a state x_k to a control $\mu_k(x_k) \in U_k(x_k)$. The cost-to-go of π starting from a state-time pair (x_k, k) will be denoted by

$$J_k(x_k) = E\left\{\sum_{i=k}^{T-1} g_i(x_i, \mu_i(x_i), w_i)\right\}. \tag{4.2}$$

The cost-to-go functions J_k satisfy the following recursion of dynamic programming (DP for short)

$$J_k(x) = E\left\{g(x, \mu_k(x), w) + J_{k+1}(f(x, \mu_k(x), w))\right\}, \qquad k = 0, 1, \ldots \tag{4.3}$$

with the initial condition

$$J_T(x) = 0.$$

The *rollout policy based on* π is denoted by $\bar{\pi} = \{\bar{\mu}_0, \bar{\mu}_1, \ldots\}$, and is defined through the operation

$$\bar{\mu}_k(x) = \arg\min_{u \in U(x)} E\{g(x, u, w) + J_{k+1}(f(x, u, w))\}, \qquad \forall x, k = 0, 1, \ldots. \tag{4.4}$$

Thus the rollout policy is a one step-lookahead policy, with the optimal cost-to-go approximated by the cost-to-go of the base policy. This amounts essentially to a single step of the method of policy iteration. Indeed using standard policy iteration arguments, one can show that the rollout policy $\bar{\pi}$ is an improved policy over the base policy π.

In practice, one typically has a method or algorithm to compute the control $\mu_k(x)$ of the base policy, given the state x, but the corresponding cost-to-go functions J_k may not be known in closed form. Then the exact or approximate computation of the rollout control $\bar{\mu}_k(x)$ using Eq. (4.4) becomes an important and nontrivial issue, since we need for all $u \in U(x)$ the value of

$$Q_k(x, u) = E\{g(x, u, w) + J_{k+1}(f(x, u, w))\}, \tag{4.5}$$

known as the *Q-factor* at time k. Alternatively, for the computation of $\bar{\mu}_k(x)$ we need the value of the cost-to-go

$$J_{k+1}(f(x, u, w))$$

at all possible next states $f(x, u, w)$.

In favorable cases, it is possible to compute the cost-to-go $J_k(x)$ of the base policy π for any time k and state x. An example is the variant of the quiz problem discussed in Sections 2 and 3, where the base policy is an open-loop policy that consists of the schedule generated by the index policy or the greedy policy. The corresponding cost-to-go can then be computed using Eq. (2.1). In general, however, the computation of the cost-to-go of the base policy may be much more difficult. In particular, when the number of states is very large, the DP recursion (4.3) may be infeasible.

A conceptually straightforward approach for computing the rollout control at a given state x and time k is to use Monte Carlo simulation. This was suggested by Tesauro (1996) in the context of backgammon. To implement this approach, we consider all possible controls $u \in U(x)$ and we generate a "large" number of simulated trajectories of the system starting from x, using u as the first control, and using the policy π thereafter. Thus a simulated trajectory has the form

$$x_{i+1} = f(x_i, \mu_i(x_i), w_i), \qquad i = k+1, \ldots, T-1,$$

where the first generated state is

$$x_{k+1} = f(x, u, w_k),$$

and each of the disturbances $w_k, \ldots, w_{T-1}$ is an independent random sample from the given distribution. The costs corresponding to these trajectories are averaged to compute an approximation $\tilde{Q}_k(x, u)$ to the Q-factor $Q_k(x, u)$. The approximation becomes increasingly accurate as the number of simulated trajectories increases. Once the approximate Q-factor $\tilde{Q}_k(x, u)$ corresponding to each control $u \in U(x)$ is computed, we can obtain the (approximate) rollout control $\tilde{\mu}_k(x)$ by the minimization

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} \tilde{Q}_k(x, u).$$

Unfortunately, this method suffers from the excessive computational overhead of the Monte Carlo simulation. We are thus motivated to consider approximations that involve reduced overhead, and yet capture the essence of the basic rollout idea. We describe next an approximation approach of this type, and in the following section, we discuss its application to stochastic scheduling problems.

Approximation Using Scenarios

Let us suppose that we approximate the cost-to-go of the base policy π using *certainty equivalence*. In particular, given a state x_k at time k, we fix the remaining disturbances at some nominal values $\bar{w}_k, \bar{w}_{k+1}, \ldots, \bar{w}_{T-1}$, and we generate a state and control trajectory of the system using the base policy π starting from x_k and time k. The corresponding cost is denoted by $\tilde{J}_k(x_k)$, and is used as an approximation to the true cost $J_k(x_k)$. The approximate rollout control at state-time (x_k, k) based on π is given by

$$\tilde{\mu}_k(x_k) = \arg \min_{u \in U(X_k)} E\{g(x_k, u, w) + \tilde{J}_{k+1}(f(x_k, u, w))\}.$$

We thus need to run π from all possible next states $f(x_k, u, w)$ and evaluate the corresponding approximate cost-to-go $\tilde{J}_{k+1}(f(x_k, u, w))$ using a single state-control trajectory calculation based on the nominal values of the uncertainty.

The certainty equivalent approximation involves a single nominal trajectory of the remaining uncertainty. To strengthen this approach, it is natural to consider multiple trajectories of the uncertainty, called *scenarios,* and to construct an approximation to the relevant Q-factors that involves, for every one of the scenarios, the cost of the base policy π. Mathematically, we assume that we have a method, which at each state x_k, generates M uncertainty sequences

$$w^m(x_k) = (w_k^m, w_{k+1}^m, \ldots, w_{T-1}^m), \qquad m = 1, \ldots, M.$$

The sequences $w^m(x_k)$ are the scenarios at state x_k. The cost $J_k(x_k)$ of the base policy is approximated by

$$\tilde{J}_k(x_k, r) = r_0 + \sum_{m=1}^{M} r_m C_m(x_k), \tag{4.6}$$

where $r = (r_0, r_1, \ldots, r_M)$ is a vector of parameters to be determined, and $C_m(x_k)$ is the cost corresponding to an occurrence of the scenario $w^m(x_k)$, when starting at state x_k and using the base policy. We may interpret the parameter r_m as an "aggregate weight" that encodes the aggregate effect on the cost-to-go function of the base policy of uncertainty sequences that are similar to the scenario $w^m(x_k)$. We will assume for simplicity that r does not depend on the time index k or the state x_k. However, there are interesting possibilities for allowing a dependence of r on k or x_k (or both), with straightforward changes in the following methodology. Note that, if $r_0 = 0$, the approximation (4.6) may be also be viewed as *limited simulation approach*, based on just the M scenarios $w^m(x_k)$, and using the weights r_m as "aggregate probabilities."

Given the parameter vector r, and the corresponding approximation $\tilde{J}_k(x_k, r)$ to the cost of the base policy, as defined above, a corresponding approximate rollout policy is determined by

$$\tilde{\mu}_k(x) = \arg \min_{u \in U(x)} \tilde{Q}_k(x, u, r), \tag{4.7}$$

where

$$\tilde{Q}_k(x, u, r) = E\{g(x, u, w) + \tilde{J}_{k+1}(f(x, u, w), r)\} \tag{4.8}$$

is the approximate Q-factor. We envision here that the parameter r will be determined by an off-line "training" process and it will then be used for calculating on-line the approximate rollout policy as above.

One may use standard methods of Neuro-Dynamic Programming (NDP for short) to train the parameter vector r. In particular, we may view the approximating function $\tilde{J}_k(x_k, r)$ of Eq. (4.6) as a linear feature-based architecture where the scenario costs $C_m(x_k)$ are the features at state x_k. One possibility is to use a straightforward least squares fit of $\tilde{J}_k(x_k, r)$ to random sample values of the cost-to-go $J_k(x_k)$. These sample values may be obtained by Monte-Carlo simulation, starting from a representative subset of states. Another possibility is to use Sutton's TD(λ). We refer to the books by Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998), and the survey by Barto et al. (1995) for extensive accounts of training methods and relating techniques.

We finally mention a variation of the scenario-based approximation method, whereby only a portion of the future uncertain quantities are fixed at nominal scenario values, while the remaining uncertain quantities are explicitly viewed as random. The cost of scenario m at state x_k is now a random variable, and the quantity $C_m(x_k)$ used in Eq. (4.6) should be the *expected* cost of this random variable. This variation is appropriate and makes practical sense as long as the computation of the corresponding expected scenario costs $C_m(x_k)$ is convenient.

5. Rollout Algorithms for Stochastic Quiz Problems

We now apply the rollout approach based on certainty equivalence and scenarios to variants of the quiz problem where there is no optimal policy that is open-loop, such as the situations (e)–(i) given in Section 1. The state after questions $i_1, \ldots, i_k$ have been successfully answered, is the current partial schedule $(i_1, \ldots, i_k)$, and possibly the list of surviving quiz takers [in the case where there are multiple quiz takers, as in variant (g) of Section 1]. A scenario at this state corresponds to a (deterministic) sequence of realizations of some of the future random quantities, such as:

(1) The list of turns that will be missed in answer attempts from time k onward; this is for the case of variant (e) in Section 1.

(2) The list of new questions that will appear and old questions that will disappear from time k onward; this is for the case of variant (f) in Section 1.

(3) The specific future times at which the surviving quiz takers will drop out of the quiz; this is for the case of variant (g) in Section 1.

Given any scenario of this type at a given state, and a base heuristic such as an index or a greedy policy, the corresponding value of the heuristic [cf. the cost $C_m(x_k)$ in Eq. (4.6)] can be easily calculated. The approximate value of the heuristic at the given state can be computed by weighing the values of all the scenarios using a weight vector r, as in Eq. (4.6). In the case of a single scenario, a form of certainty equivalence is used, whereby the value of the scenario at a given state is used as the (approximate) value of the heuristic starting from that state. In the next section we present computational results for the case of a problem, which is identical to the one tested in Section 3, but a turn may be missed with a certain probability.

6. Computational Experiments with Stochastic Quiz Problems

The class of quiz problems which we used in our computational experiments is similar to the class of problems used in Section 3, with the additional feature that an attempt to answer a question can be blocked with a prespecified probability, corresponding to the case of variant (e) in Section 1. The problems involve 20 questions and 20 time periods, where each question has a prescribed set of times where it can be attempted. The result of a blocking event is a loss of opportunity to answer any question at that state. Unanswered questions can be attempted in future stages, until a wrong answer is obtained.

In order to evaluate the performance of the base policy for rollout algorithms, we use a particular version of the scenario approach described previously. Assume that the blocking probability is denoted by P_b. At any stage k, given M stages remaining and this blocking probability, the equivalent scenario duration is computed as the smallest integer greater than or equal to the expected number of stages remaining:

$$T_e = \lceil P_b * (M - k) \rceil$$

Using this horizon, the expected cost of a base heuristic is computed as the cost incurred for an equivalent deterministic quiz problem starting with the current state, with remaining duration T_e. The cost of the strategy obtained by the base policy is approximated using the resulting value function for this horizon, as computed by Eq. (2.1).

As in Section 4, we used seven algorithms in our experiments:

1. The optimal stochastic dynamic programming algorithm.

2. The greedy heuristic, where questions are ranked in decreasing $p_i v_i$, and, for each stage k, the feasible unanswered question with the highest ranking is selected.

3. The index heuristic, where questions are ranked by decreasing $p_i v_i/(1 - p_i v_i)$, and for each stage k, the feasible unanswered question with the highest ranking is selected.

4. The one-step rollout policy based on the greedy heuristic and certainty equivalence policy evaluation, where, at each stage k, for every feasible unanswered question i_k and prior sequence $i_1, \ldots, i_{k-1}$, the question is chosen according to the rollout rule (2.4). The function H uses the greedy heuristic as the base policy, and its performance is approximated by the performance of an equivalent non-blocking quiz problem as described above.

5. The one-step rollout policy based on the index heuristic and certainty equivalence policy evaluation, where the function H in (2.4) uses the index heuristic as the base policy, and is approximated using the certainty equivalence approach described previously.

6. The selective two-step lookahead rollout policy based on the greedy heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

7. The selective two-step lookahead rollout policy based on the index heuristic, with certainty equivalence policy evaluation corresponding to an equivalent non-blocking quiz problem with horizon described as above.

The problems selected for evaluation involve 20 possible questions and 20 stages, which are small enough so that exact solution using dynamic programming is possible. Associated with each question is a sequence of times, determined randomly for each experiment, when that question can be attempted. Floating point values were assigned randomly to each question from 1 to 10 in each problem instance. The probabilities of successfully answering each question were also chosen randomly, between a specified lower bound and 1.0. In order to evaluate the performance of the last six algorithms, each suboptimal algorithm was simulated 10,000 times, using independent event sequences determining which question attempts were blocked and which questions were answered correctly.

Our experiments focused on the effects of three factors on the relative performance of the different algorithms:

a) The lower bound on the probability of successfully answering a question, which varied from 0.2 to 0.8.

Table 3. Performance of the different algorithms as the minimum probability of success of answering a question varies. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Minimum Prob. of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	54%	63%	73%	82%
One-Step Rollout	85%	89%	90%	88%
Two-Step Rollout	87%	89%	90%	88%
Index Heuristic	56%	67%	78%	84%
One-Step Rollout	86%	89%	90%	88%
Two-Step Rollout	87%	90%	90%	88%

b) The average percent of questions which can be answered at any one stage, which ranged from 10% to 50%.

c) The probability that individual question attempts will not be blocked, ranging from 0.3 to 1.0.

As in Section 4, for each experimental condition, we generated 30 independent problems and solved them with each of the 7 algorithms, and evaluate the corresponding performance using 10,000 Monte Carlo runs. The average performance is reported for each condition.

The first set of experiments fixed the average percentage of questions which can be answered at a single stage to 10%, the probability that question attempts will not be blocked to 0.6, and varied the lower bound on the probability of successfully answering a question across four conditions: 0.2, 0.4, 0.6 and 0.8. Table 3 shows the results of our experiments. The average performance of the greedy and index heuristics in each condition are expressed as a percentage of the optimal performance. The results for this experiment are very similar to the results we obtained earlier for deterministic quiz problems. Without rollouts, the performance of either heuristic is poor, whereas the use of one-step rollouts can recover a significant percentage of the optimal performance. As the risk associated with answering questions decreases, the performance of the heuristics improves, and the resulting improvement offered by the use of rollouts decreases. On average, the advantage of using selective two-step rollouts is small, but this advantage can be large for selected difficult problems.

The second set of experiments fixed the lower bound on the probability of successfully answering a question to 0.2, and varied the average percent of questions which can be answered at any one stage across 3 levels: 10%, 30%, and 50%. The results of these experiments are summarized in Table 4. As in the deterministic quiz problems, the performance of the greedy and index heuristics improves as the number of questions which can be answered at any one time approaches 100%. The results show that, even in cases where the heuristics achieve good performance, rollout strategies offer significant performance gains.

The last set of experiments focused on varying the blocking probability that an attempt to answer a question at any one time is not blocked over 3 conditions: 0.3, 0.6, and 1.0. The last condition corresponds to the deterministic quiz problems of Section 3. Table 5 contains the results of these experiments. As the blocking probability increases, there is increased

Table 4. Performance of the different algorithms as the average number of questions per period increases. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Problem Density	0.1	0.3	0.5
Greedy Heuristic	54%	65%	78%
One-Step Rollout	85%	88%	91%
Two-Step Rollout	87%	89%	91%
Index Heuristic	56%	74%	87%
One-Step Rollout	86%	89%	92%
Two-Step Rollout	87%	90%	92%

Table 5. Performance of the different algorithms on stochastic quiz problems as the probability of non-blocking increases. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Probability of Nonblocking	0.3	0.6	1.0
Greedy Heuristic	73%	54%	41%
One-Step Rollout	90%	85%	75%
Two-Step Rollout	91%	87%	81%
Index Heuristic	75%	56%	43%
One-Step Rollout	91%	86%	77%
Two-Step Rollout	91%	87%	81%

randomness as to whether the questions may be available in the future. This increased randomness leads to improved performance of myopic strategies, as shown in Table 5. Again, the advantages of the rollout strategies are evident even in this favorable case.

The results in Tables 3, 4, and 5 provide ample evidence that rollout strategies offer superior performance for stochastic quiz problems, while maintaining polynomial solution complexity.

7. Quiz Problems with Graph Precedence Constraints

The previous set of experiments focused on quiz problems where questions could be attempted during specific time periods, with no constraints imposed on the questions which had been attempted previously. In order to study the effectiveness of rollout strategies for problems with precedence constraints, we defined a class of quiz problems where the sequence of questions to be attempted must form a connected path in a graph. In these problems, a question attempt cannot be blocked as in the problems of Section 6, so there exists an optimal open-loop policy.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph where the nodes $\mathcal{N}$ represent questions in a quiz problem. Associated with each node n is a value for answering the question correctly, v_n, and a probability of correctly answering the question, p_n. Once a question has been answered correctly at node n, the value of subsequent visits to node n is reduced to zero, and there is no risk of failure on subsequent visits to node n.

The graph constrains the quiz problem as follows: a question n_1 may be attempted at stage k only if there is an arc $(n, n_1) \in \mathcal{A}$, where n is the question attempted at stage $k - 1$. The graph-constrained quiz problem of duration N consists of finding a path $n_0, n_1, \ldots, n_N$ in the graph $\mathcal{G}$ such that n_0 is the fixed starting node, $(n_k, n_{k+1}) \in \mathcal{A}$ for all $k = 0, \ldots, N-1$, and the path maximizes the expected value of the questions answered correctly before the first erroneous answer.

The previous heuristic algorithms can be extended to the graph constrained case. The greedy heuristic can be described as follows: Given that the current attempted question was n, determine the feasible questions i such that $(n, i) \in \mathcal{A}$. Select the feasible question which has the highest expected value for the next attempt $p_i v_i$. In the graph-constrained problem, it is possible that there are no feasible questions with positive value, and the path is forced to revisit a question already answered. If no feasible question has positive value, the greedy heuristic is modified to select a feasible node which has been visited the least number of times among the feasible nodes from node n. The index heuristic is defined similarly, except that the index $p_i v_i/(1 - p_i v_i)$ is used to rank the feasible questions.

One-step rollout policies can be based on the greedy or index heuristics, as before. Since the class of problems is similar to the deterministic quiz problems discussed before, it is straightforward to determine the expected value associated with a given policy. The rollout policies are based on exact evaluation of these expected values.

In the experiments below, we compare the following five algorithms:

(1) The optimal dynamic programming algorithm.

(2) The greedy policy.

(3) The index policy.

(4) The one-step rollout policy based on the greedy heuristic.

(5) The one-step rollout policy based on the index heuristic.

The first set of experiments involves problems with 16 questions and 16 stages. This problem size is small enough to permit exact solution using the dynamic programming algorithm. The questions were valued from 1 to 10, selected randomly. On average, each node was connected to 5 other nodes, corresponding to 30% density. In these experiments, the probability of successfully answering a question was randomly selected between a lower bound and 1.0, and the lower bound was varied from 0.2 to 0.8, thereby varying the average risk associated with a problem.

Table 6 summarizes the results of these experiments. The first observation is that the performance of the heuristics in graph-constrained problems is relatively superior to the

Table 6. Performance of the different algorithms on graph-constrained quiz problems as the minimum probability of success of answering a question increases. The probability of successfully answering a question was randomly selected between a lower bound and 1.0, and the lower bound was varied from 0.2 to 0.8. The numbers reported are percentage of the performance of the optimal, averaged across 30 independent problems.

Minimum Prob. of Success	0.2	0.4	0.6	0.8
Greedy Heuristic	74%	77%	77%	84%
One-Step Rollout	94%	94%	91%	94%
Index Heuristic	84%	87%	89%	90%
One-Step Rollout	95%	96%	96%	95%

performance obtained in the experiments in Section 4. This is due in part to the lack of structure concerning when questions could be attempted in the problems tested in Section 4. In contrast, the graph structure in this section provides a time-invariant set of constraints, leading to better performance. In spite of this improved performance, the results show that rollout algorithms can improve the performance of the heuristics, to levels where the achieved performance is roughly 95% of the performance of the optimal dynamic programming algorithm, with a significant reduction in computation cost compared with the optimal algorithm.

To illustrate the performance of rollout algorithms on larger problems, we ran experiments on graphs involving 100 questions and 100 stages. For problems of this size, exact solution via dynamic programming is computationally infeasible. The problems involved graphs with 10% density and varying risks as before. The results are summarized in Table 7. Since there is no optimal solution for reference, the results provide the average improvement by the rollout strategies over the corresponding heuristics, expressed as a percentage of the performance achieved by the rollout strategies. The average improvement achieved by the rollout algorithms, as shown in Table 7, is consistent with the corresponding improvement shown in Table 6. The results indicate that rollout strategies continue to offer significant performance advantages over the corresponding heuristics. In contrast with the optimal dynamic programming algorithm, the average computation time of the rollout algorithms for these problems is a fraction of a second on a Sun HyperSparc workstation.

8. Conclusion

In this paper, we studied stochastic scheduling problems arising from variations of a classical search problem known as the quiz problem. We grouped these variations into two classes: the deterministic quiz problems, for which optimal strategies can be expressed as deterministic sequences, and the stochastic quiz problems, for which optimal strategies

Table 7. Performance improvement achieved by rollout algorithms over the corresponding heuristics on 100 question graph-constrained quiz problems as the minimum probability of success of answering a question increases. The numbers reported are percentage of the performance of the rollout algorithms, averaged across 30 independent problems.

Minimum Prob. of Success	0.2	0.4	0.6	0.8
Improvement over Greedy by One-Step Rollout	28%	29%	31%	24%
Improvement over Index by One-Step Rollout	13%	12%	10%	6%

are feedback functions of the problem state. For either of these classes, the computational complexity of obtaining exact optimal solutions grows exponentially with the size of the scheduling problem, limiting the applicability of exact techniques such as stochastic dynamic programming.

In this paper, we develop near-optimal solution approaches for deterministic and stochastic quiz problems that are computationally tractable based on the use of rollout algorithms. For stochastic quiz problems, we introduced a novel approach to policy evaluation, based on the use of scenarios, which resulted in polynomial complexity algorithms for obtaining near-optimal strategies. Our computational experiments show that these rollout algorithms can substantially improve the performance of index-based and greedy algorithms for both deterministic and stochastic quiz problems. The performance achieved by the rollout algorithms is quite close to the optimal, and appears insensitive to the quality of the corresponding heuristic performance.

References

Barto, A. G., S. J. Bradtke, and S. P. Singh. (1995). “Learning to Act Using Real-Time Dynamic Programming,” *Artificial Intelligence* 72, 81–138.

Bertsekas, D. P., J. N. Tsitsiklis, and C. Wu. (1997). “Rollout Algorithms for Combinatorial Optimization,” *Heuristics* 3, 245–262.

Bertsekas, D. P., and J. N. Tsitsiklis. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Ross, S. M. (1983). *Introduction to Stochastic Dynamic Programming*. N.Y.: Academic Press.

Sutton, R., and A. G. Barto. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Tesauro, G., and G. R. Galperin. (1996). “On-Line Policy Improvement Using Monte Carlo Search,” presented at the 1996 Neural Information Processing Systems Conference, Denver, CO.

Whittle, P. (1982). *Optimization over Time*, vol. I. N.Y.: Wiley.