INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

# An Evolutionary Random Policy Search Algorithm for Solving Markov Decision Processes

Jiaqiao Hu, Michael C. Fu, Vahid R. Ramezani, Steven I. Marcus,

To cite this article:
Jiaqiao Hu, Michael C. Fu, Vahid R. Ramezani, Steven I. Marcus, (2007) An Evolutionary Random Policy Search Algorithm for Solving Markov Decision Processes. INFORMS Journal on Computing 19(2):161-174. https://doi.org/10.1287/ijoc.1050.0155

*INFORMS Journal on Computing*
Vol. 19, No. 2, Spring 2007, pp. 161–174
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1902 | 0161

DOI 10.1287/ijoc.1050.0155


# An Evolutionary Random Policy Search Algorithm for Solving Markov Decision Processes

Jiaqiao Hu
Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, Stony Brook, New York 11794, USA, jqhu@ams.sunysb.edu

Michael C. Fu
Robert H. Smith School of Business and Institute for Systems Research, University of Maryland, College Park, Maryland 20742, USA, mfu@rhsmith.umd.edu

Vahid R. Ramezani
Institute for Systems Research, University of Maryland, College Park, Maryland 20742, USA, rvahid@isr.umd.edu

Steven I. Marcus
Department of Electrical and Computer Engineering, and Institute for Systems Research, University of Maryland, College Park, Maryland 20742, USA, marcus@umd.edu

This paper presents a new randomized search method called *evolutionary random policy search* (ERPS) for solving infinite-horizon discounted-cost Markov-decision-process (MDP) problems. The algorithm is particularly targeted at problems with large or uncountable action spaces. ERPS approaches a given MDP by iteratively dividing it into a sequence of smaller, random, sub-MDP problems based on information obtained from random sampling of the entire action space and local search. Each sub-MDP is then solved approximately by using a variant of the standard policy-improvement technique, where an *elite policy* is obtained. We show that the sequence of elite policies converges to an optimal policy with probability one. Some numerical studies are carried out to illustrate the algorithm and compare it with existing procedures.

*Key words*: dynamic programming, Markov, finite state; analysis of algorithms; programming, nonlinear; queues
*History*: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received April 2004; revised September 2004, January 2005, June 2005; accepted June 2005.

## 1. Introduction

From operations research to artificial intelligence, a broad range of problems can be formulated and described by Markov-decision-process (MDP) models. Most solution methods have concentrated on reducing the size of the state space to address the well-known "curse of dimensionality" (i.e., the typical exponential growth in the state-space size with the parameters of the problem). Some well-established approaches are state aggregation (Bertsekas and Castañon 1989), feature extraction (Tsitsiklis and Van Roy 1996), random successive approximations and random multigrid algorithms (Rust 1997), and various value-function approximations via basis functions (de Farias and Van Roy 2003, Trick and Zin 1997), to name just a few. The key idea throughout is to avoid enumerating the entire state space. However, most of the above approaches generally require searching the entire action space, so problems with very large action spaces may still pose a computational challenge.

The approach proposed in this paper is meant to complement these highly successful techniques. In particular, we focus on MDPs where the state space is relatively small but the action space is very large. For example, consider the problem of controlling the service rate of a single-server queue with a finite buffer size $M$ to minimize the average number of jobs in queue and the service cost. The state space of this problem is the possible number of jobs in the queue $\{0, 1, \ldots, M\}$, so the size of the state space is $M+1$, whereas the possible actions might be all values from a large set or an interval (e.g., representing a service rate), in which case the action space is very large or even uncountably infinite. This example can be generalized to high-dimensional cases where multiple servers with different service rates are used to serve a single queue, which typically arise in, e.g., banking services or call-center management. From a more general point of view, if one of the aforementioned state-space-reduction techniques is considered, e.g. state aggregation, then MDPs with small state

spaces and large action spaces can also be regarded as the outcomes resulting from the aggregation of MDPs with large state and action spaces. For these problems, algorithms like policy iteration (PI) and value iteration (VI) that require enumerating/searching the entire action space may become practically inefficient. Furthermore, when the problem is non-convex and has multiple local optima, local search-based algorithms cannot guarantee that a global optimal solution is obtained.

The issue of large action spaces was addressed in early work by MacQueen (1966), who used some inequality forms of Bellman's equation together with bounds on the optimal-value function to identify and eliminate nonoptimal actions to reduce the size of the action sets to be searched at each iteration of the algorithm. Since then, the procedure has been applied to several standard methods like PI, VI, and modified PI (see Puterman 1994 for a review). All these algorithms generally require that the admissible set of actions at each state is finite. We approach this problem in an entirely different manner, by using an evolutionary, population-based approach, and directly searching the policy space to avoid carrying out an optimization over the entire action space. Such an approach has proven effective in attacking many traditional optimization problems, both deterministic (combinatorial) and stochastic. Our work applies this approach to infinite-horizon discounted-cost MDP models and results in a novel algorithm we call *evolutionary random policy search* (ERPS).

For a given MDP problem, ERPS proceeds iteratively by constructing and solving a sequence of sub-MDP problems, which are MDPs defined on smaller policy spaces. At each iteration of the algorithm, two steps are fundamental: (1) The sub-MDP problem constructed in the previous iteration is approximately solved by using a variant of the policy-improvement technique based on parallel rollout (Chang et al. 2004) called *policy improvement with cost swapping* (PICS), and a policy called an *elite policy* is generated. (2) Based on the elite policy, a group of policies is then obtained by using a "nearest-neighbor" heuristic and random sampling of the entire action space, from which a new sub-MDP is created by restricting the original MDP problem (e.g., cost structure, transition probabilities) on the current available subsets of actions. Under some appropriate assumptions, we show that the sequence of elite policies converges with probability one to an optimal policy.

We briefly review the relatively sparse research literature applying evolutionary search methods such as genetic algorithms (GAs) and simulated-annealing algorithms (SAs) for solving MDPs. Wells et al. (1999) use GAs to find good finite-horizon policies for partially observable MDPs and discuss the effects of different GA parameters. Barash (1999) proposes a genetic search in policy space for solving infinite-horizon discounted MDPs, and by comparing with standard PI, concludes that it is unlikely that policy search based on GAs can offer a competitive approach in cases where PI is implementable. More recently, Chang et al. (2005) propose an algorithm called *evolutionary policy iteration* (EPI) for solving infinite-horizon discrete MDPs with large action spaces by imitating the standard procedures of GAs. Although the algorithm is guaranteed to converge with probability one, no performance comparisons with existing techniques are provided, and the theoretical convergence requires the action space to be finite.

ERPS shares some similarities with the EPI algorithm introduced in Chang et al. (2005), where a sequence of "elite" policies is also produced at successive iterations of the algorithm. However, the fundamental differences are that in EPI, policies are treated as the most essential elements in optimization, and each "elite" policy is directly generated from a group of policies. On the other hand, in our approach, policies are regarded as intermediate constructions from which sub-MDP problems are then constructed and solved. EPI follows the general framework of GAs, and thus operates only at the global level, which usually results in slow convergence. In contrast, ERPS combines global search with a local enhancement step (the "nearest-neighbor" heuristic) that leads to rapid convergence once a policy is found in a small neighborhood of an optimal policy. We argue that our approach substantially improves the performance of the EPI algorithm while maintaining the computational complexity at relatively the same level.

Perhaps the most straightforward and the most commonly used numerical approach in dealing with MDPs with uncountable action spaces is via discretization (Rust 1997). In practice, this could lead to computational difficulties, resulting in an action space that is too large or a solution that is not accurate enough. In contrast, our approach works directly on the action space, requiring no explicit discretization. As in standard approaches such as PI and VI, the computational complexity of each iteration of ERPS is polynomial in the size of the state space, but unlike these procedures, it is insensitive to the size of the action space, making the algorithm a promising candidate for problems with relatively small state spaces but uncountable action spaces.

In Section 2, we begin with the problem setting. In Sections 3 and 4, we give a detailed description of the ERPS algorithm and present some convergence results. Numerical studies and comparisons with EPI and PI are reported in Section 5. Finally some future research topics are outlined in Section 6.

## 2. Problem Setting

We consider the infinite-horizon discounted cost MDP problem $G = (X, A, P, R, \alpha)$ with finite state space $X$, a general action space $A$, a bounded nonnegative cost function $R$: $X \times A \to \Re^+ \cup \{0\}$, a fixed discount factor $\alpha \in (0, 1)$, and a transition function $P$ that maps a state-action pair to a probability distribution over $X$. The probability of transitioning to state $y \in X$ given that we are in state $x$ taking action $a \in A$, is denoted by $P_{x,y}(a)$.

Unless otherwise specified, $\Pi$ is the set of all stationary deterministic policies $\pi$: $X \to A$ and $|X|$ is the size of the state space. And without loss of generality, we assume that all actions $a \in A$ are admissible for all states $x \in X$.

Define the optimal value associated with an initial state $x$ as

$$J^*(x) = \inf_{\pi \in \Pi} J^\pi(x), \quad \text{where}$$

$$J^\pi(x) = E\left[\sum_{t=0}^{\infty} \alpha^t R(x_t, \pi(x_t)) \,\middle|\, x_0 = x\right], \quad x \in X,\ \alpha \in (0, 1), \quad (1)$$

$x_t$ is the state of the MDP at time $t$, and $E(\cdot)$ is understood with respect to the probability law induced by the transition probabilities. Assume there exists a stationary optimal policy $\pi^* \in \Pi$ that achieves the optimal value $J^*(x)$ for all initial states $x \in X$. The problem is to find such a policy $\pi^*$.

## 3. Evolutionary Random Policy Search

A high-level description of the ERPS algorithm is summarized in Figure 1. We provide a detailed discussion in the following subsections.

### 3.1. Initialization

We start by specifying an *action-selection distribution* $\mathcal{P}$, the exploitation probability $q_0 \in [0, 1]$, the population size $n$, and a search range $r_i$ for each state $x_i \in X$. Once chosen, these parameters are fixed throughout the algorithm. We then select an initial group of policies; however, because of the exploration step used in ERPS, performance of the algorithm is relatively insensitive to this choice. One simple method is to choose each individual policy uniformly from $\Pi$.

The *action-selection distribution* $\mathcal{P}$ is a probability distribution over the action space used to generate sub-MDPs (Section 3.3). Note that $\mathcal{P}$ could be state-dependent in general, i.e., we could prescribe for each state $x \in X$ a different action-selection distribution according to some prior knowledge of the problem structure. One simple choice of $\mathcal{P}$ is the uniform distribution. The exploitation probability $q_0$ and the search range $r_i$ will be used to construct sub-MDPs (Section 3.3).

**Evolutionary Random Policy Search (ERPS)**

- **Initialization:** Specify an *action selection distribution* $\mathcal{P}$, the population size $n > 1$, and the exploitation probability $q_0 \in [0, 1]$. Specify a search range $r_i$ for each state $x_i \in X$, $i = 1, \ldots, |X|$. Select an initial population of policies $\Lambda_0 = \{\pi_1^0, \pi_2^0, \ldots, \pi_n^0\}$. Construct the initial sub-MDP as $\mathcal{G}_{\Lambda_0} := (X, \Gamma_0, P, R, \alpha)$, where $\Gamma_0 = \bigcup_x \Lambda_0(x)$. Set $\pi_*^{-1} := \pi_1^0$, $k = 0$.
- **Repeat until a specified stopping rule is satisfied:**
  - **—Policy Improvement with Cost Swapping (PICS):**
    - Obtain the value function $J^{\pi_j^k}$ for each $\pi_j^k \in \Lambda_k$.
    - Generate the elite policy for $\mathcal{G}_{\Lambda_k}$ as

$$\pi_*^k(x) = \arg\min_{u \in \Lambda_k(x)} \left\{ R(x, u) + \alpha \sum_y P_{x,y}(u) \left[ \min_{\pi_j^k \in \Lambda_k} J^{\pi_j^k}(y) \right] \right\}, \quad \forall x \in X. \quad (2)$$

  - **—Sub-MDP Generation:**

```
* for j = 2 to n
    for i = 1 to |X|
      generate a random sample u from the uniform
        distribution over [0, 1],
      if u ≤ q0 (exploitation)
        choose the action π_j^{k+1}(x_i) in the neighborhood
          of π_*^k(x_i) by using the "nearest neighbor"
          heuristic.
      elseif u > q0 (exploration)
        choose the action π_j^{k+1}(x_i) ∈ A according to P.
      endif
    endfor
  endfor
```

    - Set the next population generation as $\Lambda_{k+1} = \{\pi_*^k, \pi_2^{k+1}, \ldots, \pi_n^{k+1}\}$.
    - Construct a new sub-MDP problem as $\mathcal{G}_{\Lambda_{k+1}} := (X, \Gamma_{k+1}, P, R, \alpha)$, where $\Gamma_{k+1} = \bigcup_x \Lambda_{k+1}(x)$.
    - $k \leftarrow k + 1$.

**Figure 1 Evolutionary Random Policy Search**

### 3.2. Policy Improvement with Cost Swapping via Parallel Rollout

As mentioned earlier, the idea behind ERPS is to split a large MDP problem randomly into a sequence of smaller, manageable MDPs, and to extract a possibly-convergent sequence of policies via solving these smaller problems. For a given policy population $\Lambda = \{\pi_1, \pi_2, \ldots, \pi_n\}$, if we restrict the original MDP (e.g., actions, costs, transition probabilities) on the subsets of actions $\Lambda(x) := \{\pi_1(x), \pi_2(x), \ldots, \pi_n(x)\}$ $\forall x \in X$, then a sub-MDP problem is induced from $\Lambda$ as $\mathcal{G}_\Lambda := (X, \Gamma, P, R, \alpha)$, where $\Gamma := \bigcup_x \Lambda(x)$ and the feasible action set is now state-dependent, given by $\Gamma(x) := \Lambda(x)$ $\forall x \in X$. Note that in general $\Lambda(x)$ is a multi-set, which means that the set may contain repeated elements; however, we can always discard the redundant members and view $\Lambda(x)$ as the set of admissible actions at state $x$. Since ERPS is an iterative random-search algorithm, rather than attempting to solve $\mathcal{G}_\Lambda$ exactly, it is more efficient to use approximation schemes and obtain an improved policy or

good candidate policies with worst-case performance guarantees.

Here we adopt a variant of the policy-improvement technique based on the parallel rollout idea in Chang et al. (2004) to find an "elite" policy, one that is superior to all policies in the current population, by executing the following two steps:

*Step* 1. Obtain the value functions $J^{\pi_j}$, $j = 1, \ldots, n$, by solving the equations:

$$J^{\pi_j}(x) = R(x, \pi_j(x)) + \alpha \sum_y P_{x,y}(\pi_j(x)) J^{\pi_j}(y), \quad \forall x \in X. \quad (3)$$

*Step* 2. Compute the elite policy $\pi_*$ by

$$\pi_*(x) = \arg\min_{u \in \Lambda(x)} \left\{ R(x, u) + \alpha \sum_y P_{x,y}(u) \left[ \min_{\pi_j \in \Lambda} J^{\pi_j}(y) \right] \right\}, \quad \forall x \in X. \quad (4)$$

Since in (4), we are basically performing the policy improvement on the "swapped cost" $\min_{\pi_j \in \Lambda} J^{\pi_j}(x)$, we call this procedure *policy improvement with cost swapping* (PICS). The "swapped cost" $\min_{\pi_j \in \Lambda} J^{\pi_j}(x)$ may not be the value function corresponding to any policy. We now show that the elite policy generated by PICS improves any policy in $\Lambda$.

Theorem 1. *Given* $\Lambda = \{\pi_1, \pi_2, \ldots, \pi_n\}$, *let* $\bar{J}(x) = \min_{\pi_j \in \Lambda} J^{\pi_j}(x)$ $\forall x \in X$, *and let*

$$\mu(x) = \arg\min_{u \in \Lambda(x)} \left\{ R(x, u) + \alpha \sum_y P_{x,y}(u(x)) \bar{J}(y) \right\}. \quad (5)$$

*Then* $J^\mu(x) \le \bar{J}(x)$, $\forall x \in X$. *Furthermore, if* $\mu$ *is not optimal for* $\mathcal{G}_\Lambda$, *then* $J^\mu(x) < \bar{J}(x)$ *for at least one* $x \in X$.

Proof. The proof idea is an extension of Chang et al. (2004), presented here for completeness. Let $J_0(x) = R(x, \mu(x)) + \alpha \sum_y P_{x,y}(\mu(x)) \bar{J}(y)$, and consider the sequence $J_1(x), J_2(x), \ldots$ generated by the recursion $J_{i+1}(x) = R(x, \mu(x)) + \alpha \sum_y P_{x,y}(\mu(x)) J_i(y)$, $\forall i = 0, 1, 2, \ldots$. At state $x$, by the definition of $\bar{J}(x)$, there exists $\pi_j$ such that $\bar{J}(x) = J^{\pi_j}(x)$. It follows that

$$\begin{aligned} J_0(x) &\le R(x, \pi_j(x)) + \alpha \sum_y P_{x,y}(\pi_j(x)) \bar{J}(y) \\ &\le R(x, \pi_j(x)) + \alpha \sum_y P_{x,y}(\pi_j(x)) J^{\pi_j}(y) \\ &= J^{\pi_j}(x) \\ &= \bar{J}(x), \end{aligned}$$

and since $x$ is arbitrary, we have

$$\begin{aligned} J_1(x) &= R(x, \mu(x)) + \alpha \sum_y P_{x,y}(\mu(x)) J_0(y) \\ &\le R(x, \mu(x)) + \alpha \sum_y P_{x,y}(\mu(x)) \bar{J}(y) \\ &= J_0(x). \end{aligned}$$

By induction $J_{i+1}(x) \le J_i(x)$, $\forall x \in X$ and $\forall i = 0, 1, 2, \ldots$. It is well known (Bertsekas 1995) that the sequence $J_0(x), J_1(x), J_2(x), \ldots$ generated above converges to $J^\mu(x)$, $\forall x \in X$. Therefore $J^\mu(x) \le \bar{J}(x)$, $\forall x$. If $J^\mu(x) = \bar{J}(x)$, $\forall x \in X$, then PICS reduces to the standard policy improvement on policy $\mu$, and it follows that $\mu$ satisfies Bellman's optimality equation and is thus optimal for $\mathcal{G}_\Lambda$. Hence we must have $J^\mu(x) < \bar{J}(x)$ for some $x \in X$ whenever $\mu$ is not optimal. □

Now at the $k$th iteration, given the current policy population $\Lambda_k$, we compute the $k$th elite policy $\pi_*^k$ via PICS. According to Theorem 1, the elite policy improves any policy in $\Lambda_k$, and since $\pi_*^k$ is directly used to generate the $(k+1)$th sub-MDP (see Figure 1 and Section 3.3), the following monotonicity property follows by induction.

Corollary 1. *For all* $k \ge 0$,

$$J^{\pi_*^{k+1}}(x) \le J^{\pi_*^k}(x), \quad \forall x \in X. \quad (6)$$

In EPI, an "elite" policy is also obtained at each iteration by a method called "policy switching." Unlike PICS, policy switching constructs an elite policy by directly manipulating each individual policy in the population. To be precise, for the given policy population $\Lambda = \{\pi_1, \pi_2, \ldots, \pi_n\}$, the elite policy is constructed as

$$\pi_*(x) \in \left\{ \arg\min_{\pi_i \in \Lambda} (J^{\pi_i}(x))(x) \right\}, \quad \forall x \in X, \quad (7)$$

where the value functions $J^{\pi_i}$, $\forall \pi_i \in \Lambda$ are obtained by solving (3). Chang et al. (2005) show that the elite policy $\pi_*$ generated by policy switching also improves any policy in the population $\Lambda$. Note that the computational complexity of executing (7) is $O(n|X|)$.

We now provide a heuristic comparison between PICS and policy switching. For a given group of policies $\Lambda$, let $\Omega$ be the policy space for the sub-MDP $\mathcal{G}_\Lambda$; it is clear that the size of $\Omega$ is on the order of $n^{|X|}$. Policy switching takes into account only each individual policy in $\Lambda$, while PICS tends to search the entire space $\Omega$, which is a much larger set than $\Lambda$. Although it is not true in general that the elite policy generated by PICS improves the elite policy generated by policy switching, since the policy-improvement step is quite fast and focuses on the best policy-updating directions, we believe this will be the case in many situations. For example, consider the case where one particular policy, say $\bar{\pi}$, dominates all other policies in $\Lambda$. It is obvious that policy switching will choose $\bar{\pi}$ as the elite policy; thus no further improvement can be achieved. In contrast, PICS considers the sub-MDP $\mathcal{G}_\Lambda$; as long as $\bar{\pi}$ is not optimal for $\mathcal{G}_\Lambda$, a better policy can always be obtained.

The computational complexity of each iteration of PICS is approximately the same as that of policy

switching, because step 1 of PICS, i.e. (3), which is also used by policy switching, requires solution of $n$ systems of linear equations, and the number of operations required by using a direct method (e.g., Gaussian elimination) is $O(n|X|^3)$, and this dominates the cost of step 2, which is at most $O(n|X|^2)$.

### 3.3. Sub-MDP Generation

The description of the "sub-MDP-generation" step in Figure 1 is only at a conceptual level. To elaborate, we need to distinguish between two cases. We first consider the discrete-action-space case; then we discuss the setting where the action space is continuous.

**3.3.1. Discrete Action Spaces.** By Corollary 1, performance of the elite policy at the current iteration is no worse than the performances of the elite policies generated at previous iterations. Our concern now is how to achieve continuous improvements among the elite policies found at consecutive iterations. One possibility is to use unbiased random sampling and choose at each iteration a sub-MDP problem by making use of the action-selection distribution $\mathcal{P}$. The sub-MDPs at successive iterations are then independent of one another, and it is intuitively clear that we may obtain improved elite policies after a sufficient number of iterations. Such an unbiased sampling scheme is very effective in escaping local optima and is often useful in finding a good candidate solution. However, in practice persistent improvements will be increasingly difficult to achieve as the number of iterations (sampling instances) increases, since the probability of finding better elite policies becomes smaller. See Lourenco et al. (2002) for a more insightful discussion in a global-optimization context. Thus, it appears that a biased sampling scheme could be more helpful, which can be accomplished by using a "nearest-neighbor" heuristic.

To achieve a biased sampling configuration, ERPS combines exploitation ("nearest-neighbor" heuristic) with exploration (unbiased sampling). The key to balance these two types of searches is use of the exploitation probability $q_0$. For a given elite policy $\pi$, we construct a new policy $\hat{\pi}$ in the next population generation as follows: At each state $x \in X$, with probability $q_0$, $\hat{\pi}(x)$ is selected from a small neighborhood of $\pi(x)$; and with probability $1-q_0$, $\hat{\pi}(x)$ is chosen using unbiased random sampling. The preceding procedure is performed repeatedly until we have obtained $n-1$ new policies, and the next population generation is simply formed by the elite policy $\pi$ and the $n-1$ newly generated policies. Intuitively, on the one hand, use of exploitation will introduce more robustness into the algorithm and helps locate the exact optimal policy, while on the other hand, the exploration step will help the algorithm escape local optima and find attractive policies quickly. In effect, this idea is equivalent to altering the underlying action-selection distribution, in that $\mathcal{P}$ is artificially made more peaked around the action $\pi(x)$.

If we assume that $A$ is a nonempty metric space with a defined metric $d(\cdot,\cdot)$, then the "nearest-neighbor" heuristic in Figure 1 could be implemented as follows:

Let $r_i$, a positive integer, be the search range for state $x_i$, $i = 1, 2, \ldots, |X|$. We assume that $r_i < |A|$ for all $i$, where $|A|$ is the size of the action space.

- Generate a random variable $l \sim DU(1, r_i)$, where $DU(1, r_i)$ represents the discrete uniform distribution between 1 and $r_i$. Set $\pi_j^{k+1}(x_i) = a \in A$ such that $a$ is the $l$th closest action to $\pi_*^k(x_i)$ (measured by $d(\cdot,\cdot)$).

Remark 1. Sometimes the above procedure is not easy to implement. It is often necessary to index a possibly high-dimensional metric space, whose complexity will depend on the dimension of the problem and the cost in evaluating the distance functions. However, the action spaces of many MDP problems are subsets of $\Re^N$, where a lot of efficient methods can be applied, such as Kd-trees (Bentley 1979) and R-trees (Guttman 1984). Such settings constitute the most favorable situation, since the action space is "naturally ordered."

Remark 2. In EPI, policies in a new generation are generated by a policy-mutation procedure where two types of mutations are considered: "global mutation" and "local mutation." For each policy, the algorithm first decides the mutation type, where the probability of the global type is $P_m$; else the mutation type is local (i.e., with probability $1-P_m$). Then, for each state $x$ of a given policy $\pi$, the action $\pi(x)$ is mutated with probability $P_g$ for global mutation or $P_l$ for local mutation, where $P_g \gg P_l$, the idea being that "global mutation" helps the algorithm escape local optima, whereas "local mutation" helps the algorithm fine-tune the solution. The mutated action is generated using the action-selection distribution $\mathcal{P}$. As a result, each action in a new policy generated by "policy mutation" either remains unchanged or is altered by pure random sampling. Since no local-search element is actually involved in the so-called "local mutation," this is essentially equivalent to setting the exploitation probability $q_0 = 0$ in our approach.

**3.3.2. Continuous Action Spaces.** The biased-sampling idea in the previous section can be naturally extended to MDPs with continuous action spaces. Let $\mathcal{B}_A$ be the smallest $\sigma$-algebra containing all the open sets in $A$, and choose the action-selection distribution $\mathcal{P}$ as a probability measure defined on $(A, \mathcal{B}_A)$. Again, denote the metric defined on $A$ by $d(\cdot,\cdot)$.

By following the "nearest-neighbor" heuristic, we now give a general implementation of the exploitation step in Figure 1.

Let $r_i > 0$ denote the search range for state $x_i$, $i = 1, 2, \ldots, |X|$.

• Choose an action uniformly from the set of neighbors $\{a: d(a, \pi_*^k(x_i)) \le r_i, a \in A\}$.

Note the difference in the search range $r_i$ between the discrete-action-space case and the continuous-action-space case. In the former case, $r_i$ is a positive integer indicating the number of candidate actions that are the closest to the current elite action $\pi_*^k(x_i)$, whereas in the latter case, $r_i$ is the distance from the current elite action, which may take any positive real value.

If we further assume that $A$ is a nonempty open connected subset of $\Re^N$ with some metric (e.g., the infinity norm), then a detailed implementation of the above exploitation step is as follows:

• Generate a random vector $\lambda^i = (\lambda_1^i, \ldots, \lambda_N^i)^T$ with each $\lambda_h^i \sim U[-1, 1]$ independent for all $h = 1, 2, \ldots, N$, where $U[-1, 1]$ represents the uniform distribution over $[-1, 1]$. Choose the action $\pi_j^{k+1}(x_i) = \pi_*^k(x_i) + \lambda^i r_i$.

• *If* $\pi_j^{k+1}(x_i) \notin A$, then repeat the above step.

In this specific implementation, the same search range $r_i$ is used along all directions of the action space. However, in practice, it may often be useful to generalize $r_i$ to an $N$-dimensional vector, where each component controls the search range in a particular direction of the action space.

Remark 3. Note that the action space does not need to have any structure other than being a metric space. The metric $d(\cdot, \cdot)$ used in the "nearest-neighbor" heuristic implicitly imposes a structure on the action space. It follows that the efficiency of the algorithm depends on how the metric is actually defined. Like most of the random-search methods for global optimization, our approach is designed to explore structures where good policies tend to be clustered together. Thus, in our context, a good metric should have a good potential in representing this structure. For example, the discrete metric (i.e., $d(a, a) = 0 \ \forall a \in A$ and $d(a, b) = 1 \ \forall a, b \in A, \ a \ne b$) should never be considered as a good choice, since it provides no useful information about the action space. For a given action space, a good metric always exists but may not be known a priori. In the special case where the action space is a subset of $\Re^N$, we take the Euclidean metric as the default metric, in accord with most optimization techniques in $\Re^N$.

### 3.4. Stopping Rule

Different stopping criteria can be used. The simplest one is to stop the algorithm when a predefined maximum number of iterations is reached. In the numerical experiments reported in Section 5, we use one of the most common stopping rules in standard GAs (Srinivas and Patnaik 1994, Wells et al. 1999, Chang et al. 2005): The algorithm is stopped when no further improvement in the value function is obtained for several, say $K$, consecutive iterations. To be precise, we stop the algorithm if $\exists k > 0$, such that $\|J^{\pi_*^{k+m}} - J^{\pi_*^k}\| = 0 \ \forall m = 1, 2, \ldots, K$.

## 4. Convergence of ERPS

As before, denote by $d(\cdot, \cdot)$ the metric on the action space $A$. We define the distance between two policies $\pi^1$ and $\pi^2$ by

$$d_\infty(\pi^1, \pi^2) := \max_{1 \le i \le |X|} d(\pi^1(x_i), \pi^2(x_i)). \tag{8}$$

For a given policy $\hat{\pi} \in \Pi$ and any $\sigma > 0$, we further define the $\sigma$-neighborhood of $\hat{\pi}$ by

$$\mathcal{N}(\hat{\pi}, \sigma) := \{\pi \in \Pi \mid d_\infty(\hat{\pi}, \pi) \le \sigma\}. \tag{9}$$

For each policy $\pi \in \Pi$, we also define $P_\pi$ as the transition matrix whose $(x, y)$th entry is $P_{x,y}(\pi(x))$ and $R_\pi$ as the one-stage cost vector whose $x$th entry is $R(x, \pi(x))$.

Since the ERPS method is randomized, different runs of the algorithm will give different sequences of elite policies (i.e., sample paths); thus the algorithm induces a probability distribution over the set of all sequences of elite policies. Let $\hat{\mathcal{P}}(\cdot)$ and $\hat{E}(\cdot)$ be the probability and expectation taken with respect to this distribution.

Define the infinity norm of $V$ as

$$\|V\|_\infty := \max_{1 \le i \le N} |V_i| \quad \text{if } V \in \Re^N,$$

and

$$\|V\|_\infty := \max_{1 \le i \le N} \sum_{j=1}^{M} |V_{i,j}| \quad \text{if } V \in \Re^{N \times M}.$$

We have the following convergence result for the ERPS algorithm.

Theorem 2. *Let $\pi^*$ be an optimal policy with corresponding value function $J^{\pi^*}$, and let the sequence of elite policies generated by ERPS together with their corresponding value functions be denoted by $\{\pi_*^k, k = 1, 2, \ldots\}$ and $\{J^{\pi_*^k}, k = 1, 2, \ldots\}$, respectively. Assume that*:

1. $q_0 < 1$.
2. *For any given $\ell > 0$,*

$$\mathcal{P}(\{a \mid d(a, \pi^*(x)) \le \ell, a \in A\}) > 0, \quad \forall x \in X,$$

(*recall that $\mathcal{P}(\cdot)$ is a probability measure on the action space $A$*).

3. *There exist constants $\sigma > 0$, $\phi > 0$, $L_1 < \infty$, and $L_2 < \infty$, such that for all $\pi \in \mathcal{N}(\pi^*, \sigma)$ we have $\|P_\pi - P_{\pi^*}\|_\infty \le \min\{L_1 d_\infty(\pi, \pi^*), ((1-\alpha)/\alpha) - \phi\}$ and $\|R_\pi - R_{\pi^*}\|_\infty \le L_2 d_\infty(\pi, \pi^*)$.*

*Then for any given $\varepsilon > 0$, there exists a random variable $\mathcal{M}_\varepsilon > 0$ with $\hat{E}(\mathcal{M}_\varepsilon) < \infty$ such that $\|J^{\pi_*^k} - J^{\pi^*}\|_\infty \le \varepsilon \ \forall k \ge \mathcal{M}_\varepsilon$.*

Remark 4. Assumption 1 restricts the exploitation probability from pure local search. Assumption 2 simply requires that any "ball" that contains the optimal policy will have a strictly positive probability measure. It is trivially satisfied if the set $\{a \mid d(a, \pi^*(x)) \le \ell, a \in A\}$ has a positive (Borel) measure $\forall x \in X$ and the action-selection distribution $\mathcal{P}$ has infinite tails (e.g., Gaussian). Assumption 3 imposes Lipschitz conditions on $P_\pi$ and $R_\pi$; as we will see, it formalizes the notion that near-optimal policies are clustered together (see Remark 3). The assumption can be verified if $P_\pi$ and $R_\pi$ are explicit functions of $\pi$ (which is the case in our numerical examples in Section 5). For a given $\varepsilon > 0$, a policy $\pi$ satisfying $\|J^\pi - J^{\pi^*}\|_\infty \le \varepsilon$ is referred to as an *$\varepsilon$-optimal policy* (Bertsekas 1995).

Remark 5. Theorem 2 implies almost-sure convergence of $\{J^{\pi_*^k}, k = 0, 1, \ldots\}$ to the optimal-value function $J^{\pi^*}$. To see this, note that Theorem 2 implies that $\widehat{\mathcal{P}}(\|J^{\pi_*^k} - J^{\pi^*}\|_\infty > \varepsilon) \to 0$ as $k \to \infty$ for every given $\varepsilon$, which means that the sequence converges in probability. Furthermore, since $\|J^{\pi_*^k} - J^{\pi^*}\|_\infty \le \varepsilon \ \forall k \ge \mathcal{M}_\varepsilon$ is equivalent to $\sup_{\bar{k}\ge k} \|J^{\pi_*^{\bar{k}}} - J^{\pi^*}\|_\infty \le \varepsilon \ \forall k \ge \mathcal{M}_\varepsilon$, we will also have $\widehat{\mathcal{P}}(\sup_{\bar{k}\ge k} \|J^{\pi_*^{\bar{k}}} - J^{\pi^*}\|_\infty > \varepsilon) \to 0$ as $k \to \infty$, and almost-sure convergence thus follows.

Proof of Theorem 2: First derive an upper bound for $\|J^\pi - J^{\pi^*}\|_\infty$ in terms of the distance $d_\infty(\pi, \pi^*)$. For policy $\pi^*$ and policy $\pi$ we have

$$J^{\pi^*} = R_{\pi^*} + \alpha P_{\pi^*} J^{\pi^*}, \tag{10}$$

$$J^\pi = R_\pi + \alpha P_\pi J^\pi. \tag{11}$$

Now subtract the above two equations and define $\Delta J^{\pi^*} = J^\pi - J^{\pi^*}$, $\Delta P_{\pi^*} = P_\pi - P_{\pi^*}$, and $\Delta R_{\pi^*} = R_\pi - R_{\pi^*}$. We have

$$\Delta J^{\pi^*} = [I - (I - \alpha P_{\pi^*})^{-1} \alpha \Delta P_{\pi^*}]^{-1} (I - \alpha P_{\pi^*})^{-1} \cdot (\alpha \Delta P_{\pi^*} J^{\pi^*} + \Delta R_{\pi^*}). \tag{12}$$

Taking the norm of both sides of (12) and using the consistency property of the operator norm (i.e., $\|AB\| \le \|A\| \cdot \|B\|$), it follows that

$$\|\Delta J^{\pi^*}\|_\infty \le \|[I - (I - \alpha P_{\pi^*})^{-1} \alpha \Delta P_{\pi^*}]^{-1}\|_\infty \|(I - \alpha P_{\pi^*})^{-1}\|_\infty \cdot (\alpha \|\Delta P_{\pi^*}\|_\infty \|J^{\pi^*}\|_\infty + \|\Delta R_{\pi^*}\|_\infty). \tag{13}$$

By assumption 3, we have $\|\Delta P_{\pi^*}\|_\infty < (1 - \alpha)/\alpha$ for all $\pi \in \mathcal{N}(\pi^*, \sigma)$, thus

$$\begin{aligned} \|(I - \alpha P_{\pi^*})^{-1} \alpha \Delta P_{\pi^*}\|_\infty &\le \|(I - \alpha P_{\pi^*})^{-1}\|_\infty \alpha \|\Delta P_{\pi^*}\|_\infty \\ &< \|(I - \alpha P_{\pi^*})^{-1}\|_\infty (1 - \alpha) \\ &< 1, \quad \forall \pi \in \mathcal{N}(\pi^*, \sigma). \end{aligned}$$

We now try to divide both sides of equation (13) by $\|J^{\pi^*}\|_\infty$. Before we proceed, we need to distinguish between two cases, $\|J^{\pi^*}\|_\infty = 0$ and $\|J^{\pi^*}\|_\infty \ne 0$.

*Case* 1. If $R_{\pi^*} = 0$ (i.e., $R(x, \pi^*(x)) = 0$ for all $x \in X$), then we have $J^{\pi^*} = 0$. Thus $\Delta J^{\pi^*} = J^\pi$ and $\Delta R_{\pi^*} = R_\pi$. By noting $\|P_\pi\|_\infty = 1$, it follows from (11) that

$$\|\Delta J^{\pi^*}\|_\infty = \|J^\pi\|_\infty \le \frac{1}{1 - \alpha \|P_\pi\|_\infty} \|R_\pi\|_\infty = \frac{1}{1 - \alpha} \|\Delta R_{\pi^*}\|_\infty. \tag{14}$$

Then by assumption 3,

$$\|\Delta J^{\pi^*}\|_\infty \le \frac{L_2}{1 - \alpha} d_\infty(\pi, \pi^*), \quad \forall \pi \in \mathcal{N}(\pi^*, \sigma). \tag{15}$$

*Case* 2. If $R_{\pi^*} \ne 0$ (i.e., $R(x, \pi^*(x)) \ne 0$ for some $x \in X$), then from (10), $J^{\pi^*} \ne 0$. Divide both sides of (13) by $\|J^{\pi^*}\|_\infty$, and use the relation that $\|(I - B)^{-1}\| \le 1/(1 - \|B\|)$ whenever $\|B\| < 1$ and the consistency property; it immediately follows that

$$\begin{aligned} \frac{\|\Delta J^{\pi^*}\|_\infty}{\|J^{\pi^*}\|_\infty} &\le \frac{\|(I - \alpha P_{\pi^*})^{-1}\|_\infty}{1 - \|(I - \alpha P_{\pi^*})^{-1}\|_\infty \alpha \|\Delta P_{\pi^*}\|_\infty} \cdot \left\{ \alpha \|\Delta P_{\pi^*}\|_\infty + \frac{\|\Delta R_{\pi^*}\|_\infty}{\|J^{\pi^*}\|_\infty} \right\} \\ &= \frac{\|(I - \alpha P_{\pi^*})^{-1}\|_\infty \|I - \alpha P_{\pi^*}\|_\infty}{1 - \|(I - \alpha P_{\pi^*})^{-1}\|_\infty \alpha \|\Delta P_{\pi^*}\|_\infty} \cdot \left\{ \frac{\alpha \|\Delta P_{\pi^*}\|_\infty}{\|I - \alpha P_{\pi^*}\|_\infty} + \frac{\|\Delta R_{\pi^*}\|_\infty}{\|I - \alpha P_{\pi^*}\|_\infty \|J^{\pi^*}\|_\infty} \right\} \\ &\le \frac{\mathcal{K}}{1 - \mathcal{K} \alpha \|\Delta P_{\pi^*}\|_\infty / \|I - \alpha P_{\pi^*}\|_\infty} \cdot \left\{ \frac{\alpha \|\Delta P_{\pi^*}\|_\infty}{\|I - \alpha P_{\pi^*}\|_\infty} + \frac{\|\Delta R_{\pi^*}\|_\infty}{\|R_{\pi^*}\|_\infty} \right\} \\ &\le \frac{\mathcal{K}}{1 - \mathcal{K} \alpha \|\Delta P_{\pi^*}\|_\infty / \|I - \alpha P_{\pi^*}\|_\infty} \cdot \left\{ \frac{\alpha L_1}{\|I - \alpha P_{\pi^*}\|_\infty} + \frac{L_2}{\|R_{\pi^*}\|_\infty} \right\} d_\infty(\pi, \pi^*), \\ &\qquad \forall \pi \in \mathcal{N}(\pi^*, \sigma), \end{aligned} \tag{16}$$

where $\mathcal{K} = \|(I - \alpha P_{\pi^*})^{-1}\|_\infty \|I - \alpha P_{\pi^*}\|_\infty$.

In view of (15) and (16), we conclude that for any given $\varepsilon > 0$, there exists a $\theta > 0$ such that for any $\pi \in \mathcal{N}(\pi^*, \sigma)$ where

$$d_\infty(\pi, \pi^*) := \max_{1 \le i \le |X|} d(\pi(x_i), \pi^*(x_i)) \le \theta, \tag{17}$$

we have $\|J^\pi - J^{\pi^*}\|_\infty = \|\Delta J^{\pi^*}\|_\infty \le \varepsilon$. Note that

$$\max_{1 \le i \le |X|} d(\pi(x_i), \pi^*(x_i)) \le \theta$$

is equivalent to

$$d(\pi(x_i), \pi^*(x_i)) \le \theta, \quad \forall i = 1, 2, \ldots, |X|. \tag{18}$$

By assumption 2, the set of actions that satisfies (18) will have a strictly positive probability measure, and

since $q_0 < 1$, it follows that the probability that a population generation does not contain a policy in the neighborhood $\mathcal{N}(\pi^*, \min\{\theta, \sigma\})$ of the optimal policy is strictly less than 1. Let $\psi$ be the probability that a randomly constructed policy is in $\mathcal{N}(\pi^*, \min\{\theta, \sigma\})$. Then at each iteration, the probability that at least one policy is obtained in $\mathcal{N}(\pi^*, \min\{\theta, \sigma\})$ is $1-(1-\psi)^{n-1}$, where $n$ is the population size. Assume that, at iteration $k$, we obtain a policy $\pi_j^{k+1}$ in $\mathcal{N}(\pi^*, \min\{\theta, \sigma\})$. Then, it is guaranteed that $\|J^{\pi_j^{k+1}} - J^{\pi^*}\|_\infty \le \varepsilon$ (by the initial part of the proof). The elite policy obtained at the next iteration improves all the available policies in $\Lambda_{k+1}$ (by Theorem 1). Therefore, if $\pi_*^{k+1}$ is the elite policy obtained in the next iteration, we have $\|J^{\pi_*^{k+1}} - J^{\pi^*}\|_\infty \le \|J^{\pi_j^{k+1}} - J^{\pi^*}\|_\infty \le \varepsilon$. Since we now have an elite policy $\pi_*^{k+1}$ that satisfies $\|J^{\pi_*^{k+1}} - J^{\pi^*}\|_\infty \le \varepsilon$, then in subsequent iterations of the algorithm we will always have an elite policy in $\Lambda_m$ such that $\|J^{\pi_*^m} - J^{\pi^*}\|_\infty \le \varepsilon$, for $m = k+1, k+2, \ldots$ (see Corollary 1). Let $\mathcal{M}_\varepsilon$ denote the number of iterations required to generate such an elite policy for the first time. We clearly have $\|J^{\pi_*^k} - J^{\pi^*}\|_\infty \le \varepsilon \ \forall k \ge \mathcal{M}_\varepsilon$. Now consider a random variable $\bar{\mathcal{M}}$ that is geometrically distributed with a success probability of $1-(1-\psi)^{n-1}$. It is not difficult to see that $\bar{\mathcal{M}}$ dominates $\mathcal{M}_\varepsilon$ stochastically (i.e., $\bar{\mathcal{M}} \ge_{st} \mathcal{M}_\varepsilon$), and because $\psi > 0$, it follows that $\hat{E}(\mathcal{M}_\varepsilon) \le \hat{E}(\bar{\mathcal{M}}) = 1/(1-(1-\psi)^{n-1}) < \infty$. □

Remark 6. In the above proof, we used the infinity norm. Since in finite dimensional spaces all norms are equivalent (Demmel 1997), similar results can also be easily established by using different norms, e.g., the Euclidean norm.

Remark 7. The result in Theorem 2 is rather theoretical, because nothing can be said about the convergence rate of the algorithm as well as how much improvement can be achieved at each iteration. As a consequence, the random variable $\mathcal{M}_\varepsilon$ could be extremely large in practice.

Note that for a discrete finite action space, assumption 3 in Theorem 2 is automatically satisfied, and assumption 2 also holds trivially if we take $\mathcal{P}(a) > 0$ for all actions $a \in A$. Furthermore, when the action space is finite, there always exists an $\varepsilon > 0$ such that the only $\varepsilon$-optimal policy is the optimal policy itself. We have the following stronger convergence result for ERPS when the action space is finite.

Corollary 2 (Discrete Finite Action Space). *If the action space is finite, $q_0 < 1$, and the action-selection distribution $\mathcal{P}(a) > 0 \ \forall a \in A$, then there exists a random variable $\mathcal{M} > 0$ with $\hat{E}(\mathcal{M}) < \infty$ such that $J^{\pi_*^k} = J^{\pi^*}$ $\forall k \ge \mathcal{M}$.*

## 5. Numerical Examples

In this section, we apply ERPS to two discrete-time controlled queueing problems and compare its performance with that of EPI (Chang et al. 2005) and standard PI. For ERPS, the same search-range parameter is prescribed for all states, denoted by a single variable $r$, and the action-selection distribution $\mathcal{P}$ is uniform. All computations were performed on an IBM PC with a 2.4 GHz Pentium 4 processor and 512 MB memory, and the computation time units are in seconds.

### 5.1. A One-Dimensional Queueing Example

The following example is adapted from de Farias and Van Roy (2003) (see also Bertsekas 1995). A finite-capacity single-server queue has controlled service-completion probabilities. A server can serve only one customer in a period, and the service of a customer begins/ends only at the beginning/end of a period. Customers arrive independently with probability $p = 0.2$, and there is at most one arrival per period (so no arrival with probability 0.8). The maximum queue length is $\mathcal{L}$, and an arrival that finds $\mathcal{L}$ customers in queue is lost. We let $x_t$, the state variable, be the number of customers in the system at the beginning of period $t$. The action to be chosen at each state is the service-completion probability $a$, which takes value in a set $A$. In period $t$, a possible service completion is generated with probability $a(x_t)$, a cost of $R(x_t, a(x_t))$ is incurred, and resulting in a transition to state $x_{t+1}$. The goal is to choose the optimal service-completion probability for each state such that the total discounted cost $E[\sum_{t=0}^{\infty} \alpha^t R(x_t, a(x_t))]$ is minimized.

**5.1.1. Discrete Action Space.** Two different choices of one-stage cost functions are considered: (i) a simple cost function that is convex in both state and action; (ii) a complicated non-convex cost function. The MDP problem resulting from case (i) may possess some nice properties (e.g., free of multiple local optimal solutions), so finding an optimal solution should be a relatively easy task, but case (ii) introduces computational difficulties (e.g., multiple local minima), intended to test more fully the effectiveness of a global algorithm like ERPS. For both cases, unless otherwise specified, the following parameter settings are used: maximum queue length $\mathcal{L} = 49$; state space $X = \{0, 1, 2, \ldots, 49\}$; discount factor $\alpha = 0.98$; action set $A = \{10^{-4}k\colon k = 0, 1, \ldots, 10^4\}$; and in ERPS, population size $n = 10$, search range $r = 10$, and the standard Euclidean distance is used to define the neighborhood. All results for ERPS are based on 30 independent replications.

For *Case* (i), the one-stage cost at any period for being in state $x$ and taking action $a$ is given by

$$R(x, a) = x + 50a^2. \tag{19}$$

We test convergence of ERPS by varying the values of the exploitation probability. Table 1 shows performance of the algorithm, where we take the performance measure

$$\text{reldev} := \max_{x \in X} \frac{|J(x) - J^*(x)|}{|J^*(x)|}, \tag{20}$$

**Table 1 Convergence Results for ERPS ($n = 10$, $r = 10$) Based on 30 Independent Replications**

| $q_0$ | Stop rule ($K$) | Avg. time (std. err.) | Mean reldev. (std. err.) |
|---|---|---|---|
| 0.0 | 2 | 0.84 (0.03) | 3.98e−05 (8.20e−06) |
| | 4 | 1.42 (0.05) | 1.34e−05 (2.43e−06) |
| | 8 | 2.63 (0.10) | 4.14e−06 (8.58e−07) |
| | 16 | 5.20 (0.16) | 7.64e−07 (1.07e−07) |
| | 32 | 8.96 (0.38) | 2.72e−07 (3.17e−08) |
| 0.25 | 2 | 0.94 (0.02) | 1.19e−08 (4.46e−09) |
| | 4 | 1.09 (0.02) | 4.09e−09 (2.04e−09) |
| | 8 | 1.24 (0.02) | 7.94e−10 (2.88e−10) |
| | 16 | 1.54 (0.03) | 4.87e−11 (3.91e−11) |
| | 32 | 1.85 (0.04) | 0.00e−00 (0.00e−00) |
| 0.50 | 2 | 0.92 (0.02) | 2.10e−08 (1.51e−08) |
| | 4 | 1.02 (0.02) | 1.50e−09 (8.52e−10) |
| | 8 | 1.13 (0.02) | 5.95e−10 (5.03e−10) |
| | 16 | 1.27 (0.03) | 0.00e−00 (0.00e−00) |
| 0.75 | 2 | 1.14 (0.02) | 2.79e−09 (2.53e−09) |
| | 4 | 1.20 (0.02) | 5.59e−11 (3.97e−11) |
| | 8 | 1.27 (0.02) | 3.38e−11 (3.38e−11) |
| | 16 | 1.43 (0.03) | 0.00e−00 (0.00e−00) |
| 1.0 | 2 | 12.13 (0.02) | 1.92e−10 (5.97e−11) |
| | 4 | 12.17 (0.02) | 5.60e−11 (4.00e−11) |
| | 8 | 12.27 (0.01) | 0.00e−00 (0.00e−00) |

*Note.* The standard errors are in parentheses.

which signifies the maximum relative deviation of the value function $J$ from the optimal-value function $J^*$. The computation time required for PI to find $J^*$ was 15 seconds. Test results indicate superior performance of ERPS over PI; in particular, for the cases ($q_0 = 0.25$, $K = 32$), ($q_0 = 0.5$, $K = 16$), and ($q_0 = 0.75$, $K = 16$), ERPS attains the optimal solutions in all 30 independent trials within 2 seconds. Moreover, we see that the algorithm performs quite well even when $q_0 = 0$, which corresponds to pure random-search from the action-selection point of view. We believe this is because ERPS (under $q_0 = 0$) will differ from a pure random-search algorithm in the space of policies, in that ERPS is a population-based approach and it contains a PICS step that tends to search the policy space induced by the population of policies, whereas a pure random-search algorithm merely compares the performances of all sampled policies and then simply takes the best one.

To explore the computational complexity of ERPS, tests were performed on MDPs with increasing numbers of actions; for each problem, the foregoing setting is used except that the action space now takes the form $A = \{hk: k = 0, 1, \ldots, 1/h\}$, where $h$ is the mesh size, selected sequentially (one for each problem) from {1/100, 1/250, 1/500, 1/1,000, 1/2,500, 1/5,000, 1/10,000, 1/25,000, 1/50,000, 1/100,000, 1/200,000}.

In Figure 2, we plot the running time required for PI and ERPS to find the optimal solutions as a function of the number of actions of each MDP considered, where the results for ERPS are the averaged time over 30 independent replications. The time for PI increases linearly in the number of actions (note the log-scale used in Figure 2), while the time for ERPS does so in an asymptotic sense. We see that ERPS delivers very competitive performance even when the action space is small; when the action space is relatively large (number of actions greater than $10^4$), ERPS reduces the computational efforts of PI by a factor of roughly 14. In the experiments, we used a search range $r = 10$ in ERPS, regardless of the size of the action space; we believe the performance of the algorithm could be enhanced by using a search range that is proportional to the size of the action space. Moreover, the computational effort of ERPS can be reduced considerably if we are merely seeking solutions within some required accuracy rather than insisting on the optimal solution.

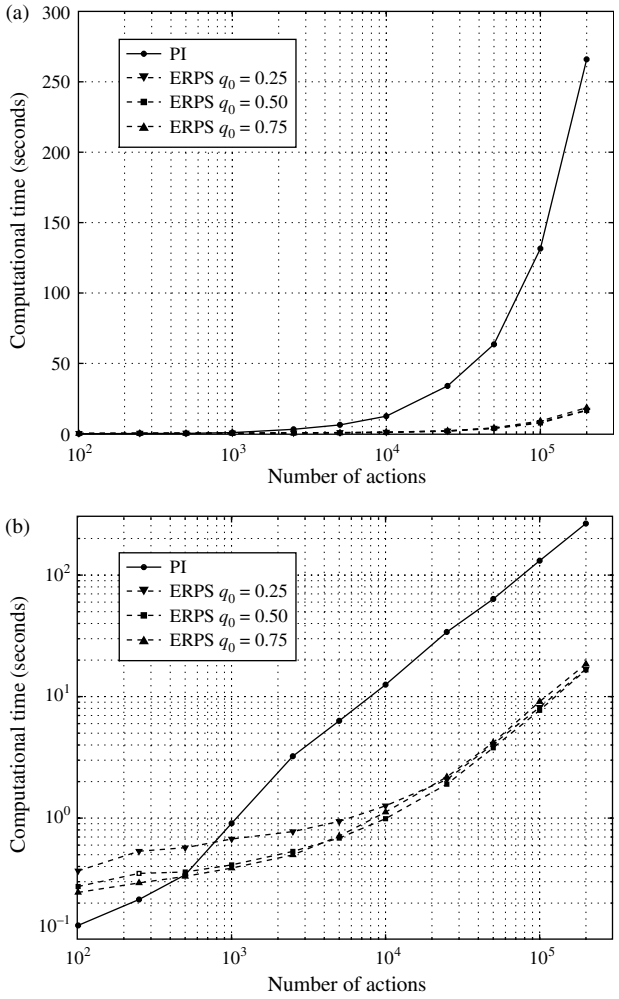



**Figure 2 Running Time Required for PI and ERPS ($n = 10, r = 10$) Based on 30 Independent Replications to Find the Optimal Solutions to MDPs with Different Numbers of Actions**

*Note.* (a) Using log-scale for horizontal axis. (b) Using log-log plot.

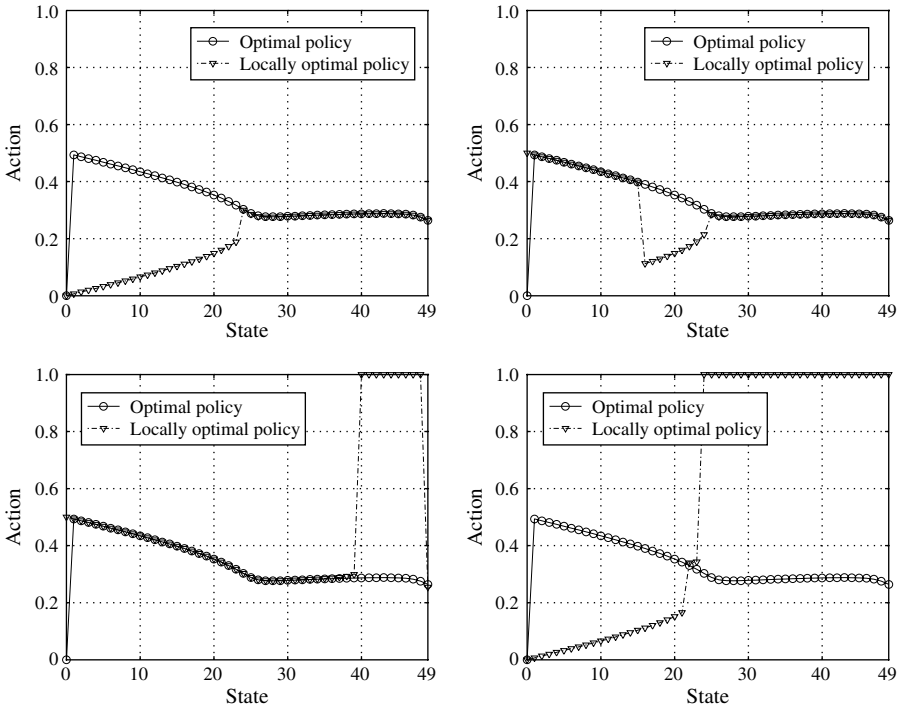



**Figure 3** **Four Typical Locally Optimal Solutions to the Test Problem**

For *Case* (ii), we used the following one-stage cost function

$$R(x, a) = x + 5\left[\frac{|X|}{2}\sin(2\pi a) - x\right]^2, \qquad (21)$$

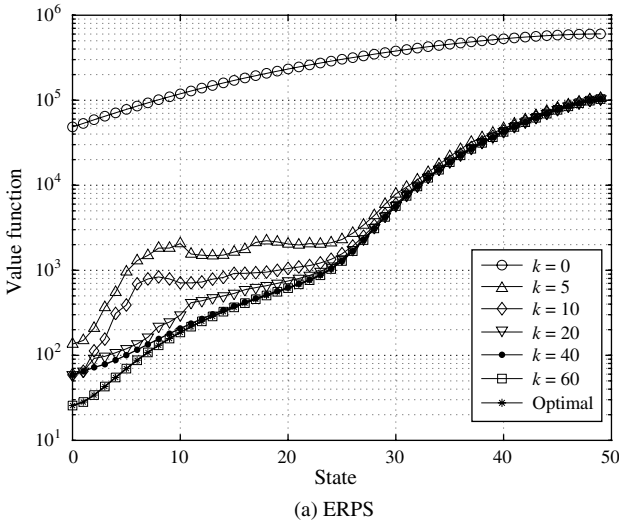
which induces a tradeoff in choosing between large values of $a$ to reduce the state $x$ and appropriate values of $a$ to make the squared term small. Moreover, since the sine function is not monotone, the resultant MDP problem has a very high number of local minima; some typical locally optimal policies are shown in Figure 3.

Table 2 shows the convergence properties of EPI and ERPS, where both algorithms start with the same initial population. The computation time required for PI to find the optimal-value function $J^*$ was 14 seconds. For EPI, we have tested different sets of parameters (recall from Section 3.3.1, Remark 2, that $P_m$ is the mutation probability; $P_g$ and $P_l$ are the predefined global and local mutation probabilities); the results reported in Table 2 are the best obtained. Also note that because of the slow convergence of EPI, the values for the stopping-control parameter $K$ are chosen much larger than those for ERPS. The typical performances of ERPS and EPI are in Figure 4, where we plot the corresponding value functions of the generated elite policies for some particular iterations.

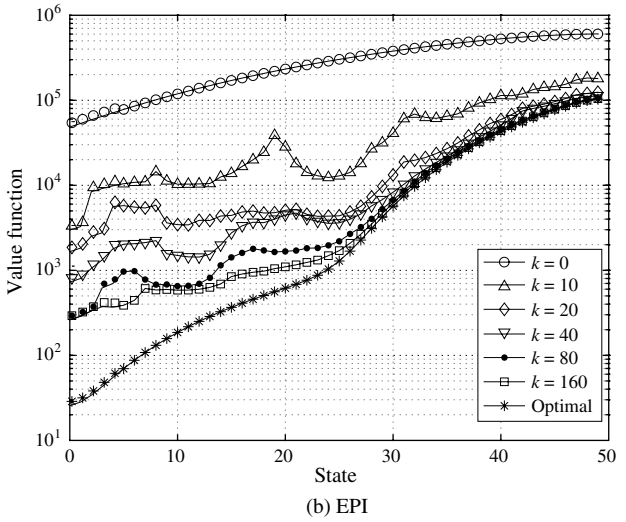
**Table 2** **Convergence Results for EPI** ($n = 10$) **and ERPS** ($n = 10$, $r = 10$) **Based on 30 Independent Replications**

| Algorithms | Stop rule ($K$) | Avg. time (std. err.) | Mean reldev. (std. err.) |
|---|---|---|---|
| EPI | 20 | 2.13 (0.11) | 3.48e−00 (3.16e−01) |
| $P_m = 0.1$ | 40 | 3.82 (0.17) | 1.55e−00 (1.73e−01) |
| $P_g = 0.9$ | 80 | 6.83 (0.35) | 8.34e−01 (8.57e−02) |
| $P_l = 0.1$ | 160 | 17.03 (0.61) | 1.65e−01 (1.83e−02) |
| ERPS | 2 | 1.03 (0.02) | 1.42e−01 (7.95e−02) |
| $q_0 = 0.5$ | 4 | 1.12 (0.03) | 8.64e−02 (6.01e−02) |
| $r = 10$ | 8 | 1.29 (0.03) | 4.32e−02 (4.32e−02) |
| | 16 | 1.49 (0.03) | 2.25e−07 (1.36e−07) |
| | 32 | 1.86 (0.04) | 0.00e−00 (0.00e−00) |

To demonstrate the role of the exploitation probability $q_0$ in the ERPS algorithm, we fix the stopping-control parameter $K = 10$ and vary $q_0$. The results are in Table 3, where $N_{opt}$ indicates the number of times an optimal solution was found out of 30 trials. The $q_0 = 1.0$ case corresponds to pure local search. Obviously in this case, the algorithm gets trapped into a local minimum, which has a mean maximum relative deviation of 1.35e+1. However, note that the standard error is very small, which means that the local minimum is estimated with very high precision. This shows that the "nearest-neighbor" heuristic is indeed useful in fine tuning the solutions. In contrast, the pure-random-search ($q_0 = 0$) case is helpful in avoiding the local minima, yielding a lower mean relative deviation of 2.42e−2, but it is not very good in locating the exact optimal solutions, as none was found out of 30 trials. Roughly, increasing $q_0$ between 0 and 0.5 leads to more accurate estimation of the optimal solution; however, increasing $q_0$ on the range 0.6 to 1.0 decreases the quality of the solution, because the local-search part gradually begins to dominate, so that




(a) ERPS




(b) EPI

**Figure 4** **Convergence of the Value Function for Case (ii), Where $k$ Is the Iteration Counter**

*Note.* (a) ERPS ($n = 10$, $r = 10$, $K = 16$, $q_0 = 0.5$). (b) EPI ($n = 10$, $K = 160$, $P_m = 0.1$, $P_g = 0.9$, $P_l = 0.1$).

the algorithm is more easily trapped in local minima. This also explains why we have larger variances when $q_0 = 0.6$, 0.7, 0.8, and 0.9 in Table 3. Notice that the algorithm is very slow in the pure-local-search case; setting $q_0 < 1$ speeds up the algorithm substantially.

To provide a numerical comparison between the "nearest-neighbor" heuristic (biased sampling) and the policy-mutation procedure (unbiased sampling), we construct a new algorithm that uses the PICS step to generate the elite policy from the current population of policies but the policy-mutation procedure (as in EPI) to generate the remaining policies in the next population. Denote this new algorithm by PICS + PM. In both ERPS and PICS + PM, we fix the population size $n = 10$, and stop the algorithms only when a desired accuracy is reached.

**Table 3** **Performance of ERPS with Different Exploitation Probabilities ($n = 10$, $K = 10$, $r = 10$) Based on 30 Independent Replications**

| $q_0$ | Avg. time (std. err.) | $N_{opt}$ | Mean reldev. (std. err.) |
|---|---|---|---|
| 0.0 | 3.47 (0.14) | 0 | 2.42e−02 (6.04e−03) |
| 0.1 | 2.04 (0.04) | 6 | 5.82e−05 (4.42e−05) |
| 0.2 | 1.48 (0.03) | 14 | 8.75e−06 (4.76e−06) |
| 0.3 | 1.36 (0.02) | 23 | 1.78e−07 (1.08e−07) |
| 0.4 | 1.28 (0.03) | 22 | 3.25e−06 (2.56e−06) |
| 0.5 | 1.32 (0.03) | 26 | 2.44e−06 (2.32e−06) |
| 0.6 | 1.43 (0.04) | 26 | 1.67e−01 (9.77e−02) |
| 0.7 | 1.47 (0.04) | 24 | 2.08e−01 (8.97e−02) |
| 0.8 | 1.80 (0.04) | 20 | 5.49e−01 (1.51e−01) |
| 0.9 | 2.28 (0.08) | 8 | 1.19e−00 (1.89e−01) |
| 1.0 | 8.90 (0.02) | 0 | 1.35e+01 (3.30e−16) |

In Table 4, we record the length of time required for different algorithms to reach a relative deviation of at least 1.0e−3. Indeed, we see that ERPS uses far less time to reach a required accuracy than does PICS + PM.

**5.1.2. Continuous Action Space.** We test the algorithm when the action space $A$ is continuous, where the service-completion probability can be any value between 0 and 1. Again, two cost functions are considered, corresponding to cases (i) and (ii) in Section 5.1.1. In both cases, the maximum queue length $\mathscr{L}$, state space $X$, and the discount factor $\alpha$ are all taken to be the same as before.

In the numerical experiments, we approximated the optimal-value functions $J_1^*$ and $J_2^*$ for each of the respective cases (i) and (ii) by two value functions $\hat{J}_1^*$ and $\hat{J}_2^*$, which were computed by using a discretization-based PI algorithm, where we first uniformly discretize the action space into evenly spaced points by using a mesh size $h$, and then apply the standard PI algorithm on the discretized problem. In both cases, we take $h = 1e{-}8$. A brute-force calculation of $\hat{J}_1^*$ or $\hat{J}_2^*$ requires more than 40 hours of CPU time.

We set the population size $n = 10$, termination control parameter $K = 10$, and test the ERPS algorithm by using different values of the search range $r$. The performance of the algorithm is also compared with that of the discretization-based PI algorithm. Tables 5 and 6 show performances of both algorithms for cases (i) and (ii), respectively. Note that the relative deviations are actually computed by replacing the optimal-value functions $J_1^*$ and $J_2^*$ with their corresponding approximations $\hat{J}_1^*$ and $\hat{J}_2^*$ in (20).

Test results indicate that ERPS outperforms the discretization-based PI algorithm in both cases, not only in computation time but also in solution quality. The computation time for PI increases by a factor of 2

**Table 4 Average Time Required to Reach a Relative Deviation of at Least 1.0e−3 for Different Algorithms**

| Algorithms | Parameters | Avg. time (std. err.) | Actual reldev. (std. err.) |
|---|---|---|---|
| ERPS | $q_0 = 0.0$ | 14.34 (1.68) | 5.01e−04 (4.59e−05) |
| $r = 10$ | $q_0 = 0.1$ | 1.05 (0.02) | 4.28e−04 (5.29e−05) |
| | $q_0 = 0.3$ | 0.91 (0.04) | 4.04e−04 (5.77e−05) |
| | $q_0 = 0.5$ | 0.94 (0.04) | 4.36e−04 (6.01e−05) |
| | $q_0 = 0.7$ | 1.63 (0.18) | 3.06e−04 (5.59e−05) |
| | $q_0 = 0.9$ | 4.10 (0.64) | 2.12e−04 (4.27e−05) |
| PICS + PM | $P_m = 0.1, P_g = 0.9, P_l = 0.1$ | 66.6 (9.8) | 5.19e−04 (5.30e−05) |
| | $P_m = 0.3, P_g = 0.9, P_l = 0.1$ | 39.1 (6.6) | 5.60e−04 (5.19e−05) |
| | $P_m = 0.5, P_g = 0.9, P_l = 0.1$ | 21.7 (1.8) | 6.14e−04 (4.42e−05) |
| | $P_m = 0.7, P_g = 0.9, P_l = 0.1$ | 23.4 (3.1) | 4.85e−04 (5.77e−05) |
| | $P_m = 0.9, P_g = 0.9, P_l = 0.1$ | 21.1 (2.9) | 5.81e−04 (5.78e−05) |
| | $P_m = 1.0, P_g = 1.0, P_l = 0.0$ | 23.7 (2.7) | 4.49e−04 (5.71e−05) |

*Note.* All results are based on 30 independent replications.

for each halving of the mesh size, while the time for ERPS increases at a much slower rate.

### 5.2. A Two-Dimensional Queueing Example

The second example, shown in Figure 5, is a slight modification of the first one, with the difference being that now we have a single queue that feeds two independent servers with different service-completion probabilities $a_1$ and $a_2$. We consider only the continuous-action-space case. The action to be chosen at each state $x$ is $(a_1, a_2)^T$, which takes value from the set $A = [0, 1] \times [0, 1]$. We assume that an arrival that finds the system empty will always be served by the server with service-completion probability $a_1$. The state space of this problem is $X = \{0, 1_{S_1}, 1_{S_2}, 2, \ldots, 48\}$, where we have assumed that the maximum queue length (not including those in service) is 46, and $1_{S_1}$, $1_{S_2}$ are used to distinguish the situations whether server 1 or server 2 is busy when there is only one customer in the system. As before, the discount factor $\alpha = 0.98$.

The one-stage cost is

$$R(y, a_1, a_2) = y + \left[\frac{|X|}{2}\cos(\pi a_1) - y\right]^2 I_{\{S_1\}} + \left[\frac{|X|}{2}\sin(\pi a_2) - y\right]^2 I_{\{S_2\}}, \quad (22)$$

where

$$I_{\{S_i\}} = \begin{cases} 1 & \text{if server } i \text{ is busy,} \\ 0 & \text{otherwise,} \end{cases} \quad (i = 1, 2), \quad \text{and}$$

$$y = \begin{cases} 1 & \text{if } x \in \{1_{S_1}, 1_{S_2}\}, \\ x & \text{otherwise.} \end{cases}$$

**Table 5 Comparison of the ERPS Algorithm ($n = 10$, $K = 10$) with the Deterministic PI Algorithm for Case (i)**

| Algorithms | Parameters | Avg. time (std. err.) | Mean reldev. (std. err.) |
|---|---|---|---|
| ERPS | $q_0 = 0.25$ | 2.66 (0.10) | 1.12e−11 (3.72e−12) |
| ($r = 1/4{,}000$) | $q_0 = 0.50$ | 2.27 (0.09) | 2.86e−12 (4.20e−13) |
| | $q_0 = 0.75$ | 2.94 (0.08) | 1.11e−12 (2.51e−13) |
| ERPS | $q_0 = 0.25$ | 2.63 (0.10) | 2.87e−12 (5.62e−13) |
| ($r = 1/8{,}000$) | $q_0 = 0.50$ | 2.93 (0.10) | 6.12e−13 (1.49e−13) |
| | $q_0 = 0.75$ | 3.10 (0.11) | 3.94e−13 (7.02e−14) |
| ERPS | $q_0 = 0.25$ | 2.85 (0.09) | 8.80e−13 (2.45e−13) |
| ($r = 1/16{,}000$) | $q_0 = 0.50$ | 3.27 (0.10) | 1.87e−13 (3.85e−14) |
| | $q_0 = 0.75$ | 3.72 (0.10) | 9.91e−14 (2.34e−14) |
| PI | $h = 1/4{,}000$ | 6 (N/A) | 2.55e−08 (N/A) |
| | $h = 1/8{,}000$ | 12 (N/A) | 1.35e−08 (N/A) |
| | $h = 1/16{,}000$ | 23 (N/A) | 5.04e−09 (N/A) |
| | $h = 1/32{,}000$ | 46 (N/A) | 5.84e−10 (N/A) |
| | $h = 1/128{,}000$ | 188 (N/A) | 3.90e−11 (N/A) |
| | $h = 1/512{,}000$ | 793 (N/A) | 3.83e−12 (N/A) |

*Note.* The results of ERPS are based on 30 independent replications.

**Table 6 Comparison of the ERPS Algorithm ($n = 10$, $K = 10$) with the Deterministic PI Algorithm for Case (ii)**

| Algorithms | Parameters | Avg. time (std. err.) | Mean reldev. (std. err.) |
|---|---|---|---|
| ERPS | $q_0 = 0.25$ | 2.74 (0.10) | 1.09e−07 (3.24e−08) |
| ($r = 1/4{,}000$) | $q_0 = 0.50$ | 2.86 (0.08) | 2.19e−08 (6.15e−09) |
| | $q_0 = 0.75$ | 3.13 (0.09) | 7.69e−09 (1.36e−09) |
| ERPS | $q_0 = 0.25$ | 3.06 (0.12) | 1.47e−08 (3.61e−09) |
| ($r = 1/8{,}000$) | $q_0 = 0.50$ | 2.98 (0.13) | 4.55e−09 (9.77e−10) |
| | $q_0 = 0.75$ | 3.57 (0.08) | 1.76e−09 (4.21e−10) |
| ERPS | $q_0 = 0.25$ | 3.17 (0.09) | 9.50e−09 (3.55e−09) |
| ($r = 1/16{,}000$) | $q_0 = 0.50$ | 3.26 (0.11) | 1.42e−09 (2.44e−10) |
| | $q_0 = 0.75$ | 4.17 (0.12) | 3.49e−10 (7.70e−11) |
| PI | $h = 1/4{,}000$ | 5 (N/A) | 8.35e−04 (N/A) |
| | $h = 1/8{,}000$ | 11 (N/A) | 4.51e−05 (N/A) |
| | $h = 1/16{,}000$ | 21 (N/A) | 4.50e−05 (N/A) |
| | $h = 1/32{,}000$ | 42 (N/A) | 9.66e−06 (N/A) |
| | $h = 1/128{,}000$ | 175 (N/A) | 8.96e−07 (N/A) |
| | $h = 1/512{,}000$ | 734 (N/A) | 2.34e−08 (N/A) |

*Note.* The results of ERPS are based on 30 independent replications.

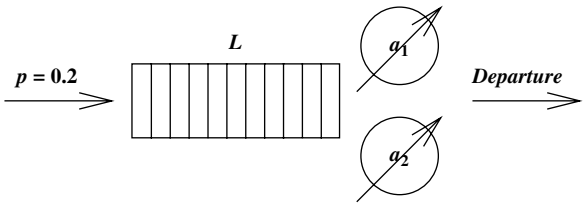



**Figure 5 A Two-Dimensional Queueing Example**

The performances of the ERPS and the discretization-based PI are in Table 7. In ERPS, both the population size $n$ and the stopping-control parameter $K$ are set to 10. In PI, we adopt a uniform discretization, where the same mesh size $h$ is used in both coordinates of the action space. Again, in computing the relative deviation, we approximated $J^*$ by $\hat{J}^*$, which was computed by using the discretization-based PI algorithm with a mesh size $h = 1/15{,}000$. Notice that the computation time for PI increases by a factor of 4 for each halving of the mesh size, whereas the time required by ERPS increases much more slowly.

## 6. Conclusions and Future Work

We have presented an evolutionary, population-based method called ERPS for solving infinite-horizon discounted-cost MDP problems. We showed that the algorithm converges almost surely to an optimal policy. We also illustrated the algorithm by applying it to two controlled queueing examples with large or uncountable action spaces. Numerical experiments on these small examples indicate that the ERPS algorithm is a promising approach, outperforming some existing methods (including the standard policy-iteration algorithm).

Many challenges remain to be addressed before the algorithm can be applied to realistic-sized problems. The motivation behind ERPS is the setting where the action space is extremely large so that enumerating the entire action space becomes computationally impractical; however, the approach still requires enumerating the entire state space. To make it applicable to large-state-space problems, the algorithm will probably need to be used in conjunction with some other state-space-reduction techniques such as state aggregation or value-function approximation. This avenue of investigation clearly merits further research.

Another important issue is the dependence of ERPS on the underlying distance metric, as determining a good metric could be challenging for problems that do not have a natural metric already available. One possible way to get around this is to update/change the action-selection distribution $\mathcal{P}$ adaptively at each iteration of the algorithm based on the sampling information obtained during the previous iterations. This actually constitutes a learning process; the hope is that more promising actions will have larger chances of being selected so that the future search will be biased toward the region containing high-quality solutions (policies).

Another practical issue is the choice of the exploitation probability $q_0$. As noted earlier, the parameter $q_0$ serves as a tradeoff between exploitation and exploration in action selections. Preliminary experimental results indicate some robustness with respect to the value of this parameter, in that values between 0.25 and 0.75 all seem to work well; however, this may not hold for larger problems or other settings, so further investigation is required. One approach is to design a similar strategy as in simulated-annealing algorithms and study the behavior of the algorithm when the value of $q_0$ is gradually increasing from 0 to 1, which corresponds to the transitioning of the search mechanism from pure random sampling to pure local search.

**Table 7 A Two-Dimensional Test Example**

| Algorithms | Parameters | Avg. time (std. err.) | Mean reldev. (std. err.) |
|---|---|---|---|
| ERPS ($r = 1/100$) | $q_0 = 0.25$ | 3.18 (0.15) | 3.86e−04 (3.18e−05) |
| | $q_0 = 0.50$ | 3.16 (0.16) | 7.48e−03 (7.25e−03) |
| | $q_0 = 0.75$ | 3.54 (0.14) | 5.83e−02 (1.78e−02) |
| ERPS ($r = 1/200$) | $q_0 = 0.25$ | 3.31 (0.12) | 9.44e−05 (8.25e−06) |
| | $q_0 = 0.50$ | 3.26 (0.12) | 7.31e−03 (7.27e−03) |
| | $q_0 = 0.75$ | 3.88 (0.17) | 5.48e−02 (1.83e−02) |
| ERPS ($r = 1/400$) | $q_0 = 0.25$ | 3.53 (0.12) | 2.06e−05 (1.97e−06) |
| | $q_0 = 0.50$ | 3.74 (0.12) | 7.27e−03 (7.26e−03) |
| | $q_0 = 0.75$ | 4.36 (0.14) | 3.55e−02 (1.48e−02) |
| PI | $h = 1/100$ | 14 (N/A) | 6.23e−02 (N/A) |
| | $h = 1/200$ | 55 (N/A) | 2.98e−02 (N/A) |
| | $h = 1/400$ | 226 (N/A) | 1.24e−03 (N/A) |

*Note.* The results of ERPS are based on 30 independent replications ($n = 10$, $K = 10$).

### Acknowledgments

This work was supported in part by the National Science Foundation under Grants DMI-9988867 and DMI-0323220, and by the Air Force Office of Scientific Research under Grants F496200110161 and FA95500410210. The authors thank the three referees for their detailed comments and suggestions, which have led to a substantially improved paper, and Hyeong Soo Chang for pointing out the relationship between the PICS step and parallel rollout.

### References

Barash, D. 1999. A genetic search in policy space for solving Markov decision processes. *AAAI Spring Sympos. Search Techniques Problem Solving Under Uncertainty and Incomplete Inform.* Stanford University, Stanford, CA.

Bentley, J. 1979. Multidimensional binary search trees in database applications. *IEEE Trans. Software Engrg.* **5** 333–340.

Bertsekas, D. P. 1995. *Dynamic Programming and Optimal Control*, Vols. 1 and 2. Athena Scientific, Belmont, MA.

Bertsekas, D. P., D. A. Castañon. 1989. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Trans. Automatic Control* **34** 589–598.

Chang, H. S., R. L. Givan, E. K. P. Chong. 2004. Parallel rollout for online solution of partially observable Markov decision processes. *Discrete Event Dynamic Systems: Theory Application* **14** 309–341.

Chang, H. S., H. G. Lee, M. C. Fu, S. I. Marcus. 2005. Evolutionary policy iteration for solving Markov decision processes. *IEEE Trans. Automatic Control.* **50** 1804–1808.

de Farias, D. P., B. Van Roy. 2003. The linear programming approach to approximate dynamic programming. *Oper. Res.* **51** 850–865.

Demmel, J. W. 1997. *Applied Numerical Linear Algebra*. Soc. Indust. Appl. Math., Philadelphia, PA.

Guttman, A. 1984. R-trees: A dynamic index structure for spatial searching. *Proc. 1984 Association for Computing Machinery Special Interest Group on Management of Data*, ACM Press, New York, 47–57.

Lourenco, H. R., O. C. Martin, T. Stützle. 2002. Iterated local search. F. Glover, G. Kochenberger, eds. *Handbook on MetaHeuristics*. Kluwer Academic Publishers, Boston, MA, 321–353.

MacQueen, J. 1966. A modified dynamic programming method for Markovian decision problems. *J. Math. Anal. Appl.* **14** 38–43.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York.

Rust, J. 1997. Using randomization to break the curse of dimensionality. *Econometrica* **65** 487–516.

Srinivas, M., L. M. Patnaik. 1994. Genetic algorithms: A survey. *IEEE Comput.* **27** 17–26.

Trick, M., S. Zin. 1997. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynam.* **1** 255-277.

Tsitsiklis, J. N., B. Van Roy. 1996. Feature-based methods for large-scale dynamic programming. *Machine Learning* **22** 59–94.

Wells, C., C. Lusena, J. Goldsmith. 1999. Genetic algorithms for approximating solutions to POMDPs. Department of Computer Science Technical Report TR-290-99, University of Kentucky, Lexington, KY, http://citeseer.ist.psu.edu/277136.html