# Evolutionary Policy Iteration for Solving Markov Decision Processes

Hyeong Soo Chang, Hong-Gi Lee, Michael C. Fu, and Steven I. Marcus

***Abstract*— We propose a novel algorithm called Evolutionary Policy Iteration (EPI) for solving infinite horizon discounted reward Markov decision processes. EPI inherits the spirit of policy iteration but eliminates the need to maximize over the entire action space in the policy improvement step, so it should be most effective for problems with very large action spaces. EPI iteratively generates a "population" or a set of policies such that the performance of the "elite policy" for a population monotonically improves with respect to a defined fitness function. EPI converges with probability one to a population whose elite policy is an optimal policy. EPI is naturally parallelizable and along this discussion, a distributed variant of PI is also studied.**

***Index Terms*— (Distributed) policy iteration, Markov decision process, genetic algorithm, evolutionary algorithm, parallelization**

## I. Introduction

In this note, we propose a novel algorithm called Evolutionary Policy Iteration (EPI) to solve Markov decision processes (MDPs) for an infinite horizon discounted reward criterion. The algorithm is especially targeted to problems where the state space is small but the action space is extremely large so that the policy improvement step in Policy Iteration (PI) becomes computationally impractical. For example, consider a simple problem of balancing the loads in $N$ parallel queues by early discard or migration of jobs to optimize a cost function of jobs' waiting time and throughput with a certain tradeoff parameter. Each queue can hold at most $K$ jobs, and is associated with a stochastic arrival sequence of jobs generated by a simple On/Off Markov modulated process with two states, along with a single server. A state consists of the traffic generation state of the Markov modulated process and the number of jobs in each queue, so that there are $(2K)^N$ states in total. An action is to migrate each job in a queue to another queue or to stay/drop at/from its own queue so that there are $O(N^K)$ possible actions in total. For $N = 2$ and $K = 50$, this leads to a state space size of $10^4$, whereas the number of possible actions is on the order of $10^{15}$.

EPI eliminates the operation of maximization over the entire action space in the policy improvement step by directly manipulating policies via a method called "policy switching" [2] that generates an improved policy from a set of given policies. The computation time for generating such an improved policy is on the order of the state space size. The basic algorithmic procedure imitates that of standard Genetic Algorithm (GAs) (see, e.g., [6] [11] [10]) with appropriate modifications and extensions required for the MDP setting, based on an idea similar to the "*elitism*" concepts introduced by De Jong [4]. In our setting, the elite policy for a population is a policy that improves the performances of all policies in the population. EPI starts with a set of policies or "population" and converges with probability one to a population of which the elite policy is an optimal policy, while maintaining a certain monotonicity property for elite policies over generations with respect to a fitness value.

The literature applying evolutionary algorithms such as GAs for solving MDPs is relatively sparse. The recent work of Lin, Bean, and White [8] uses a GA approach to construct the minimal set of affine functions that describes the value function in partially observable MDPs, yielding a variant of value iteration (VI). Chin and Jafari [3] propose an approach that maps heuristically "simple" GA [11] into the framework of PI. Unfortunately, the convergence to an optimal policy is not always guaranteed.

As noted earlier, the main motivation for the proposed EPI algorithm is the setting where the action space is finite but extremely large. In this case, it could be computationally impractical to apply exact PI or VI, due to the requirements of maximization over the entire action space via e.g., enumeration or random search methods. On the other hand, local search cannot guarantee that a global maximum has been found. Thus, the monotonicity in the policy improvement step is not preserved. The proposed EPI algorithm preserves an analogous monotonicity property over the elite policies in the populations. In some sense, this is similar to distributing "action improvement" over time, as in William and Baird's work [12] [13] in the context of actor-critic architecture, and asynchronous dynamic programming in [7] [1], with a communication protocol. However, all of those works consider operators that improve a *single* policy rather than improving upon a *set* of policies.

The main contribution of our work is the introduction of a new (randomized) evolutionary (population-based) search algorithm for solving MDPs that (i) will be superior to PI and VI in settings where the size of the action space is sufficiently large so as to render the maximization step over all actions computationally impractical, and (ii) is guaranteed to converge (with probability 1). The use of policy switching in a population-based approach is novel. Furthermore, the algorithm can easily be parallelized by partitioning the policy space and applying policy switching to (convergent) elite policies for the subsets, obtaining an optimal policy for the original policy space.

This note is organized as follows. We start with some background on MDPs in Section II. In Section III, we formally describe EPI with detailed discussion and the convergence proof. In Section IV, we study a distributed variant of PI and discuss how to speed up EPI by parallelization. We conclude with some remarks in Section V.

## II. Background

Consider an MDP $M = (X, A, P, R)$ with finite state space $X$, finite action space $A$, reward function $R : X \times A \to \mathcal{R}$, and transition function $P$ that maps a state and action pair to a probability distribution over $X$. We denote the probability of transitioning to state $y \in X$ when taking action $a$ in state $x \in X$ by $P(x, a)(y)$. For simplicity, we assume that every action is admissible in every state.

Let $\Pi$ be the set of all stationary policies $\pi : X \to A$. Define the *optimal value* associated with an initial state $x \in X$: $V^*(x) = \max_{\pi\in\Pi} V^\pi(x), x \in X$, where for $x \in X, 0 < \gamma < 1, \pi \in \Pi$,

$$V^\pi(x) = E\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, \pi(x_t)) \; x_0 = x\right],$$

where $x_t$ is a random variable denoting state at time $t$ and $\gamma$ is the discount factor. Throughout the paper, we assume that $\gamma$ is fixed. The problem is to find an optimal policy $\pi^*$ that achieves the optimal

This work was supported by Institute for Applied Science and Technology at Sogang University, Seoul, Korea, and was supported in part by the National Science Foundation under Grants DMI-9988867 and DMI-0323220, by the Air Force Office of Scientific Research under Grants F496200110161 and FA95500410210, and by the Department of Defense under Contract MDA 90499C2521.

H. S. Chang is with the Department of Computer Science and Engineering and also affiliated with Program of Integrated Biotechnology at Sogang University, Seoul, Korea. (e-mail:hschang@sogang.ac.kr).

H-G. Lee is with the School of Electrical and Electronics Engineering at Chung Ang University, Seoul, Korea. (email:hglee@cau.ac.kr).

M.C. Fu is with the Robert H. Smith School of Business and Institute for Systems Research at the University of Maryland, College Park. (e-mail:mfu@rhsmith.umd.edu).

S.I. Marcus is with the Department of Electrical & Computer Engineering and Institute for Systems Research at the University of Maryland, College Park. (e-mail:marcus@eng.umd.edu).

value with an initial state, where the initial state is distributed with a probability distribution $\delta$ defined over $X$.

*Policy Iteration* (PI) computes $\pi^*$ in a finite number of steps, because there are a finite number of policies in $\Pi$ and each iteration preserves monotonicity in terms of the policy performance. The PI algorithm consists of two parts: policy evaluation and policy improvement. We provide a formal description of PI below.

Let $B(X)$ be the space of real-valued functions on $X$. We define an operator $T : B(X) \to B(X)$ as

$$T(\Phi)(x) = \max_{a \in A} \left\{ R(x,a) + \gamma \sum_{y \in X} P(x,a)(y)\Phi(y) \right\}$$

for $\Phi \in B(X), x \in X$, and similarly, an operator $T_\pi : B(X) \to B(X)$ for $\pi \in \Pi$ as

$$T_\pi(\Phi)(x) = R(x,\pi(x)) + \gamma \sum_{y \in X} P(x,\pi(x))(y)\Phi(y)$$

for $\Phi \in B(X), x \in X$. It is well known (see, e.g., [9]) that for each policy $\pi \in \Pi$, there exists a corresponding unique $\Phi \in B(X)$ such that for $x \in X$, $T_\pi(\Phi)(x) = \Phi(x)$ and $\Phi(x) = V^\pi(x)$. The policy evaluation step obtains $V^\pi$ for a given $\pi$, and the policy improvement step obtains $\hat{\pi} \in \Pi$ such that $T(V^\pi)(x) = T_{\hat{\pi}}(V^\pi)(x), x \in X$. The policy $\hat{\pi}$ improves $\pi$ in that $V^{\hat{\pi}}(x) \geq V^\pi(x)$ for all $x \in X$.

## III. Evolutionary Policy Iteration

### A. Algorithm description

As with all evolutionary/GA algorithms, we define the $k$th generation population, ($k = 0, 1, 2, ...$), denoted by $\Lambda(k)$, which is a set of policies in $\Pi$, and $n = |\Lambda(k)| \geq 2$ is the population size, which we take to be constant in each generation. Given the fixed initial state probability distribution $\delta$ defined over $X$, we define the *average value of* $\pi$ *for* $\delta$ or *fitness value of* $\pi$: $J_\delta^\pi = \sum_{x \in X} V^\pi(x)\delta(x)$. Note that an *optimal policy* $\pi^*$ satisfies for any $\pi \in \Pi$, $J_\delta^{\pi^*} \geq J_\delta^\pi$. We denote $P_m$ as the mutation selection probability, $P_g$ the global mutation probability, and $P_l$ the local mutation probability. We also define *action selection distribution* $\mu$ as a probability distribution over $A$ such that $\sum_{a \in A} \mu(a) = 1$ and $\mu(a) > 0$ for all $a \in A$. The mutation probability determines whether or not $\pi(x)$ is changed (mutated) for each state $x$, and the action selection distribution $\mu$ is used to change $\pi(x)$ (see Section III-D for details). A high-level description of EPI is shown in Figure 1, where some steps (e.g., mutation) are described at a conceptual level, with details provided in the following subsections.

### B. Initialization and Policy Selection

The EPI algorithm's convergence is independent of the initial population $\Lambda(0)$ (to be shown later) mainly due to the **Policy Mutation** step. We can randomly generate an initial population or start with a set of heuristic policies. A simple example initialization is to set $\Lambda(0)$ such that for each policy in $\Lambda(0)$, the same action is prescribed for all states, but for each policy in the initial population prescribes a different action.

### C. Policy Switching

One of the basic procedural steps in GA is to select members from the current population to create a "mating pool" to which "crossover" is applied; this step is called "parent selection". Similarly, we can design a "policy selection" step to create a mating pool; there are many ways of doing this. The **Policy Switching** step includes this selection step implicitly.

**Evolutionary Policy Iteration (EPI)**

- **Initialization:**
  Select population size $n$ and $K > 0$. $\Lambda(0) = \{\pi_1, ..., \pi_n\}$, where $\pi_i \in \Pi$. Set $N = k = 0$, and $P_m$, $P_g$ and $P_l$ in $(0, 1]$, and $\pi^*(-1) = \pi_1$.
- **Repeat:**
  **Policy Switching:**
  - Obtain $V^\pi$ for each $\pi \in \Lambda(k)$.
  - Generate the elite policy of $\Lambda(k)$ defined as
    $$\pi^*(k)(x) \in \{\arg\max_{\pi \in \Lambda(k)}(V^\pi(x))(x)\}, x \in X.$$
  - **Stopping Rule:**
    - If $J_\delta^{\pi^*(k)} \neq J_\delta^{\pi^*(k-1)}$, $N = 0$.
    - If $J_\delta^{\pi^*(k)} = J_\delta^{\pi^*(k-1)}$ and $N = K$, terminate EPI.
    - If $J_\delta^{\pi^*(k)} = J_\delta^{\pi^*(k-1)}$ and $N < K$, $N \leftarrow N + 1$.
  - Generate $n-1$ random subsets $S_i, i = 1, ..., n-1$ of $\Lambda(k)$ by selecting $m \in \{2, ..., n-1\}$ with equal probability and selecting $m$ policies in $\Lambda(k)$ with equal probability.
  - Generate $n-1$ policies $\pi(S_i)$ defined as:
    $$\pi(S_i)(x) \in \{\arg\max_{\pi \in S_i}(V^\pi(x))(x)\}, x \in X.$$

  **Policy Mutation:** For each policy $\pi(S_i), i = 1, ..., n-1$,
  - Generate a "globally" mutated policy $\pi^m(S_i)$ with $P_m$ using $P_g$ and $\mu$ or a "locally" mutated policy $\pi^m(S_i)$ with $1 - P_m$ using $P_l$ and $\mu$.

  **Population Generation:**
  - $\Lambda(k+1) = \{\pi^*(k), \pi^m(S_i)\}$, $i = 1, ..., n-1$.
  - $k \leftarrow k + 1$.

Fig. 1. Evolutionary Policy Iteration (EPI)

Given a nonempty subset $\Delta$ of $\Pi$, we define a policy $\bar{\pi}$ generated by *policy switching* ([2]) with respect to $\Delta$ as

$$\bar{\pi}(x) \in \{\arg\max_{\pi \in \Delta}(V^\pi(x))(x)\}, x \in X. \quad (1)$$

It has been shown that the policy generated by policy switching improves any policy in $\Delta$ (see Theorem 3 in [2]):

*Theorem 3.1:* Consider a nonempty subset $\Delta$ of $\Pi$ and the policy $\bar{\pi}$ generated by policy switching with respect to $\Delta$ given in (1). Then, for all $x \in X$, $V^{\bar{\pi}}(x) \geq \max_{\pi \in \Delta} V^\pi(x)$.

This theorem immediately can be used to obtain the following result relevant for the EPI algorithm.

*Corollary 3.1:* Consider a nonempty subset $\Delta$ of $\Pi$ and the policy $\bar{\pi}$ generated by policy switching with respect to $\Delta$ given in (1). Then, for any initial state distribution $\delta$, $J_\delta^{\bar{\pi}} \geq \max_{\pi \in \Delta} J_\delta^\pi$.

*Proof:* By the definition of $\bar{\pi}$ (see (1)), for any fixed $\pi \in \Delta$ and $x \in X$, $V^{\bar{\pi}}(x) \geq V^\pi(x)$. Therefore,

$$J_\delta^{\bar{\pi}} = \sum_{x \in X} V^{\bar{\pi}}(x)\delta(x) \geq \sum_{x \in X} V^\pi(x)\delta(x) = J_\delta^\pi,$$

which implies that $J_\delta^{\bar{\pi}} \geq \max_{\pi \in \Delta} J_\delta^\pi$. ■

We first generate a policy $\pi^*(k)$, called the elite policy with respect to the current population $\Lambda(k)$, which improves any policy in $\Lambda(k)$ via policy switching. Note that this is different from the elitist concept of De Jong [4], where the elitist is a best policy in the current population $\Lambda(k)$. In contrast, the elite policy $\pi^*(k)$ may not be a member of $\Lambda(k)$, and it may not be a best policy in $\Lambda(k+1)$, where it is included unmutated. Thus, the new population $\Lambda(k+1)$ contains a policy that improves any policy in the previous population. Therefore, the following monotonicity property holds:

*Lemma 3.1:* For any $\delta$ and for all $k \geq 0$, $J_\delta^{\pi^*(k)} \geq J_\delta^{\pi^*(k-1)}$.

*Proof:* The proof is by induction. The base step is obvious from the definition of $\pi^*(0)$ and $\pi^*(-1)$ by Corollary 3.1. Assume that

$J_\delta^{\pi^*(i)} \geq J_\delta^{\pi^*(i-1)}$ for all $i \leq k$. Because EPI includes $\pi^*(k)$ in $\Lambda(k+1)$, the elite policy at $k+1$ is generated over a population that contains $\pi^*(k)$. It implies that $J_\delta^{\pi^*(k+1)} \geq J_\delta^{\pi^*(k)}$. ■

We then generate $n-1$ random subsets $S_i (i = 1, ..., n-1)$ of $\Lambda(k)$ as follows. We first select $m \in \{2, ..., n-1\}$ with equal probability and then select $m$ policies from $\Lambda(k)$ with equal probability. By applying policy switching, we generate $n-1$ policies defined as

$$\pi(S_i)(x) \in \{\arg\max_{\pi \in S_i}(V^\pi(x))(x)\}, x \in X.$$

These policies will be mutated to generate a new population (see the next subsection).

Because policy switching directly manipulates policies, eliminating the operation of maximization over the entire action space, its computational time-complexity is $O(m|X|)$, where $m = |S_i|$, independent of the action space size, leading to $O(nm|X|)$ complexity in the **Policy Switching** step, and hence $O(nm|X|^3)$ overall when including the $O(|X|^2)$ complexity for the policy evaluation needed to compute $V^\pi$. On the other hand, applying a single-policy improvement step of PI directly to each policy in $\Lambda(k)$, instead of generating $\pi(S_i), i = 1, ..., n-1$, is of complexity $O(n|X|^2|A|)$.

Based on the above analysis, we now make a brief comparison of the EPI approach with Williams and Baird's asynchronous policy iteration [13][14]. As noted already, the main advantage of the EPI approach is its independence of the size of the action space. Because the action space is assumed to be quite large, applying the single-policy improvement step over the full action space is impractical. Thus, to apply the Williams/Baird asynchronous PI approach, one would have to choose a subset of the action space; furthermore, this subset may have to change from iteration to iteration. An obvious advantage of the EPI algorithm is that this choice does not have to be made. On the other hand, the EPI algorithm requires the choice of population size and requires solving the state space equations for every member (policy) of the population, whereas the Williams/Baird approach would only require this for a single policy, so for large state spaces the EPI algorithm would definitely be at a disadvantage. Preserving theoretical convergence under the PI operation over a changing action space subset does not appear to be straightforward for the Williams/Baird approach, whereas convergence of the EPI algorithm is independent of the population size, though it does require the mutation step described in the next section.

### D. Policy Mutation

Policy mutation takes a given policy, and for each state, alters the specified action probabilistically. The main reason to generate mutated policies is to avoid being caught in local optima, making a probabilistic convergence guarantee possible.

We distinguish between two types of mutation – "local" and "global" – which are differentiated by how much of the policy is likely be changed (mutated). Local mutation is intended to guide the algorithm in obtaining an exact optimal policy through local search of "nearby" policies, whereas the global mutation allows EPI to escape from local optima. A high mutation probability indicates that many components of the policy vector are likely to be mutated, representing a more global change, whereas a low mutation probability implies that very little mutation is likely to occur, meaning a more localized perturbation. For this reason, we assume that $P_l \ll P_g$, with $P_l$ being very close to zero and $P_g$ being very close to one. The **Policy Mutation** step first determines whether the mutation will be global or local, with probability $P_m$. If the policy $\pi$ is globally (locally) mutated, for each state $x$, $\pi(x)$ is changed with probability $P_g(P_l)$. If a mutation does occur, it is carried out according to the action selection distribution $\mu$, i.e., if the mutated policy is dented by $\pi'$, then the new policy is generated according to $P(\pi'(x) = a) = \mu(a)$, for all mutated states $x$ (the actions for all other states remain unchanged). For example, one simple $\mu$ is the uniform action selection distribution, in which case the new (mutated) policy would randomly select a new action for each mutated state (independently) with equal probability over the entire action space.

### E. Population Generation and Stopping Rule

At each $k$th generation, the new population $\Lambda(k+1)$ is simply given by the elite policy generated from $\Lambda(k)$ and $n-1$ mutated policies from $\pi(S_i), i = 1, ..., n-1$. This population generation method allows a policy that is poor in terms of the performance, but might be in the neighborhood of an optimal value located at the top of the very narrow hill, to be kept in the population so that a new search region can be started from the policy. This helps EPI avoid from being caught in the region of local optima.

Once we have a new population, we need to test whether EPI should terminate. Even if the fitness values for the two consecutive elite policies are identical, this does not necessarily mean that the elite policy is an optimal policy as in PI; thus, we run the EPI algorithm $K$ more times so that these random jumps by the mutation step will eventually bring EPI to a neighborhood of the optimum. As the value of $K$ gets larger, the probability of being in a neighborhood of the optimum increases. Therefore, the elite policy at termination is optimal with more confidence as $K$ increases.

### F. Convergence

*Theorem 3.2:* Given $P_m > 0$, $P_g > 0$, and $P_l > 0$ and a action selection distribution $\mu$ such that $\sum_{a \in A} \mu(a) = 1$ and $\mu(a) > 0$ for all $a \in A$, as $K \to \infty$, $V^{\pi^*(k)}(x) \to V^{\pi^*}(x), x \in X$ with probability one uniformly over $X$, regardless of $\Lambda(0)$.

*Proof:* The proof is very simple. Observe first that as $K \to \infty$, $k \to \infty$. This is because EPI terminates when $N = K$ and if $N \neq K$, the value of $k$ increases by one.

From the assumption, the probability of generating an optimal policy by the **Policy Mutation** step is positive. To see this, let $\alpha$ be the probability of generating one of the optimal policies by local mutation and let $\beta$ the probability of generating one of the optimal policies by global mutation. Then,

$$\begin{aligned} \alpha &\geq \prod_{x \in X} P_l \mu(\pi^*(x)) = (P_l)^{|X|} \cdot \prod_{x \in X} \mu(\pi^*(x)) > 0 \\ \beta &\geq \prod_{x \in X} P_g \mu(\pi^*(x)) = (P_g)^{|X|} \cdot \prod_{x \in X} \mu(\pi^*(x)) > 0, \end{aligned}$$

where $\pi^*$ is a particular optimal policy in $\Pi$. Therefore, the probability of generating an optimal policy by the **Policy Mutation** step is positive and this probability is independent of $\Lambda(0)$.

Therefore, the probability that $\Lambda(k)$ does not contain an optimal policy (starting from an arbitrary $\Lambda(0)$) is at most $((1-\alpha)P_m))^{(n-1)k}((1-\beta)(1-P_m))^{(n-1)k}$, which goes to zero as $k \to \infty$. By Lemma 3.1, once $\Lambda(k)$ contains an optimal policy, $\Lambda(k+m)$ contains an optimal policy for any $m \geq 1$ because the fitness value of an optimal policy is the maximum among all policies in $\Pi$. This proves the claim. ■

## IV. Parallelization

The EPI algorithm can be naturally parallelized and by doing so, we can improve the running rate. Basically, we partition the policy space $\Pi$ into subsets of $\{\Pi_i\}$ such that $\bigcup_i \Pi_i = \Pi$ and $\Pi_i \cap \Pi_j = \emptyset$ for all $i \neq j$. We then apply EPI into each $\Pi_i$ in parallel and then once each part terminates, the best policy $\pi_i^*$ from each part is taken.

We apply then policy switching to the set of best policies $\{\pi_i^*\}$. We state a general result regarding parallelization of an algorithm that solves an MDP.

*Theorem 4.1:* Given a partition of $\Pi$ such that $\bigcup_i \Pi_i = \Pi$ and $\Pi_i \cap \Pi_j = \emptyset$ for all $i \neq j$, consider an algorithm $\mathcal{A}$ that generates the best policy $\pi_i^*$ for $\Pi_i$ such that for all $x \in X$, $V^{\pi_i^*}(x) \geq \max_{\pi \in \Pi_i} V^{\pi}(x)$. Then, the policy $\bar{\pi}$ defined as

$$\bar{\pi}(x) \in \{\arg\max_{\pi_i^*}(V^{\pi_i^*}(x))(x)\}, x \in X,$$

is an optimal policy for $\Pi$.

*Proof:* Via policy switching, $\bar{\pi}$ improves the performance of each $\pi_i^*$, i.e., $V^{\bar{\pi}}(x) \geq \max_{\pi_i^*} V^{\pi_i^*}(x), x \in X$, implying that $\bar{\pi}$ is an optimal policy for $\Pi$, since the partition covers the entire policy space. ∎

Note that we cannot just pick the best policy among $\pi_i^*$ in terms of the fitness value $J_\delta^\pi$. The condition that $J_\delta^\pi \geq J_\delta^{\pi'}$ for $\pi \neq \pi'$ does not always imply that $V^\pi(x) \geq V^{\pi'}(x)$ for all $x \in X$ even though the converse is true. In other words, we need a policy that improves all policies $\pi_i^*$. Picking the best policy among such policies does not necessarily guarantee an optimal policy for $\Pi$.

If the number of subsets in the partition is $N$, the overall convergence of the algorithm $\mathcal{A}$ is faster by a factor of $N$. For example, if at state $x$, the action $a$ or $b$ can be taken, let $\Pi_1 = \{\pi | \pi(x) = a, \pi \in \Pi\}$ and $\Pi_2 = \{\pi | \pi(x) = b, \pi \in \Pi\}$. By using this partition, the convergence rate of the algorithm $\mathcal{A}$ will be twice as fast.

By Theorem 4.1, this idea can be applied to PI via policy switching, yielding a "*distributed*" PI. Apply PI to each $\Pi_i$; once PI for each part terminates, combine the resulting policy for each part by policy switching. The combined policy is an optimal policy, so that this method will speed up the original PI by a factor of $N$ if the number of subsets in the partition is $N$. However, this distributed variant of PI requires the maximization operation over the action space in the policy improvement step. The result of Theorem 4.1 also naturally extends to a *dynamic programming version* of PI, similarly to EPI. For example, we can partition $\Pi$ into $\Pi_1$ and $\Pi_2$, and further partition $\Pi_1$ into $\Pi_{11}$ and $\Pi_{12}$, and $\Pi_2$ into $\Pi_{21}$ and $\Pi_{22}$. The optimal substructure property is preserved by policy switching. If the number of subsets generated in this way is $\beta$, then the overall computation time of an optimal policy is $O(\beta \cdot |X| \cdot C)$, where $C$ is the maximum size of the subsets in terms of the number of policies, because policy switching is applied $N$ times with $O(|X|)$ complexity and $C$ is an upper bound on PI-complexity.

The discussion in this section motivates a very interesting future research topic. How do we partition the policy space so that PI or EPI converges faster? For some partitions, we can even obtain the best policies for some subsets *analytically*. However, in general, partitioning the policy space to speed up a convergence would be a difficult problem and would require a structural analysis on the problem or the policy space. Here, we just briefly speculate on three possible ways of partitioning the policy space, as a more in-depth development of this topic requires further research. Let $N$ be the number of subspaces. The simplest way of partitioning the policy space is a random partition, where each policy is assigned to a particular subspace according to some probability distribution (e.g., uniformly). Another possible approach is to build a "decision tree" on the policy space based on identification of suboptimal actions. Assume that there exist a lower bound function $V^L(x)$ and an upper bound function $V^U(x)$ such that $V^L(x) \leq V^*(x) \leq V^U(x)$ for all $x \in X$. Then, if for $x \in X$ and $b \in A$,

$$R(x,b) + \gamma \sum_{y \in X} P(x,b)(y) V^U(y) < V^L(x),$$

any stationary policy that uses action $b$ in state $x$ is nonoptimal (see, Proposition 6.7.3 in [9]). Based on this fact, we start with a particular state $x \in X$ and identify nonoptimal action set $\varphi(x)$ at $x$ and build subsets $\Pi_{x,a}$ of the policy space with $a \in A - \varphi(x)$. Effectively, we are building a $|A - \varphi(x)|$-ary tree with $x$ being a root. We then repeat this with another state $y \in X$ at each child $\Pi_{x,a}$ of the tree. We continue building a tree in this way until the number of children at the leaf level is $N$. Note that for some problems, nonoptimal actions are directly observable. For example, for a simple multiclass deadline job scheduling problem of minimizing the weighted loss, if a job's deadline is about to expire, all actions selecting a pending job from a less important class than that of the dying job are nonoptimal. A third approach is to select some features on policies, and then use the features for building the decision tree to partition the policy space.

## V. Concluding Remarks

Much of the work in the MDP literature considers aggregation in the state space (see, e.g., [5]) for an approximate solution for a given MDP. Our discussion on the parallelization of PI and EPI can be viewed in some sense as an aggregation in the policy space, where the distributed version of PI can be used to generate an approximate solution of a given MDP.

In our setting, mutation of a specified action in a state is carried out using a given action selection distribution. If the action space is continuous, say $[0, 1]$, a straightforward implementation would only change the least significant digit for local mutation and the most significant digit for global mutation, if the numbers in $[0, 1]$ are represented by a certain number of significant digits.

GAs are known to work well for many continuous domain problems but to face difficulties of a different kind for problems where the decision variables are discrete [10]. However, EPI circumvents this problem via policy switching, an idea that has not been exploited in the GA literature previously.

## References


[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[2] H. S. Chang, R. Givan, and E. K. P. Chong, "Parallel rollout for on-line solution of partially observable Markov decision processes," *Discrete Event Dynamic Systems: Theory and Application*, vol. 15, no. 3, pp. 309–341, 2004.

[3] H. Chin and A. Jafari, "Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes," in *Proc. of the 30th Southeastern Symposium on System Theory*, 1998, pp. 538–543.

[4] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, U. Michigan, Ann Arbor, MI, 1975.

[5] R. Givan, S. Leach, and T. Dean, "Bounded Markov decision processes," *Artificial Intelligence*, vol. 122, pp. 71–109, 2000.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,*, Addison-Wesley, 1989.

[7] A. Jalali and M. J. Ferguson, "On distributed dynamic programming," *IEEE Trans. Automat. Control*, vol. 37, no. 5, pp. 685–689, 1992.

[8] A. Z.-Z. Lin, J. Bean, and C. White, III, "A hybrid genetic/optimization algorithm for finite horizon partially observed Markov decision processes," Tech. Rep. 98-25, Dept. Ind. and Oper. Eng., U. Michigan, Ann Arbor, 1998.

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

[10] C. R. Reeves, "Genetic algorithms for the operations researcher," *INFORMS J. on Computing*, Vol. 9, No. 3, pp. 231–250, 1997.

[11] M. Srinivas, and L. M. Patnaik, "Genetic algorithms: a survey," *IEEE Computer*, Vol. 27. No. 6, pp. 17-26, 1994.

[12] R. J. Williams and L. C. Baird, "Analysis of some incremental variants of policy iteration: first steps toward understanding actor-critic learning systems," Tech. Rep. NU-CCS-93-11, 1993.

[13] R. J. Williams and L. C. Baird, "A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming,". in *Proc. of the 6th Yale Workshop on Adaptive and Learning Systems*, Yale University, 15-17 August, 1990.