Downloaded from UvA-DARE, the institutional repository of the University of Amsterdam (UvA)
http://hdl.handle.net/11245/2.59678

File ID uvapub:59678
Filename 289644.pdf
Version unknown

SOURCE (OR PART OF THE FOLLOWING SOURCE):
Type article
Title Rollout sampling approximate policy iteration
Author(s) C. Dimitrakakis, M.G. Lagoudakis
Faculty FNWI: Informatics Institute (II)
Year 2008

FULL BIBLIOGRAPHIC DETAILS:
http://hdl.handle.net/11245/1.289644

Copyright

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content licence (like Creative Commons).

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
(pagedate: 2014-11-17)

Mach Learn (2008) 72: 157–171
DOI 10.1007/s10994-008-5069-3

OpenAccess

Rollout sampling approximate policy iteration

Christos Dimitrakakis · Michail G. Lagoudakis

Received: 22 June 2008 / Revised: 22 June 2008 / Accepted: 23 June 2008 / Published online: 10 July 2008
The Author(s) 2008

Abstract Several researchers have recently investigated the connection between reinforcement learning and classification. We are motivated by proposals of approximate policy iteration schemes without value functions, which focus on policy representation using classifiers and address policy learning as a supervised learning problem. This paper proposes variants of an improved policy iteration scheme which addresses the core sampling problem in evaluating a policy through simulation as a multi-armed bandit machine. The resulting algorithm offers comparable performance to the previous algorithm achieved, however, with significantly less computational effort. An order of magnitude improvement is demonstrated experimentally in two standard reinforcement learning domains: inverted pendulum and mountain-car.

Keywords Reinforcement learning · Approximate policy iteration · Rollouts · Bandit problems · Classification · Sample complexity

1 Introduction

Supervised and reinforcement learning are two well-known learning paradigms, which have been researched mostly independently. Recent studies have investigated the use of supervised learning methods for reinforcement learning, either for value function (Lagoudakis and Parr 2003a; Riedmiller 2005) or policy representation (Lagoudakis and Parr 2003b; Fern et al. 2004; Langford and Zadrozny 2005). Initial results have shown that policies can

Editors: Walter Daelemans, Bart Goethals, Katharina Morik.

C. Dimitrakakis (✉)
Informatics Institute, University of Amsterdam, Kruislaan 403, 1098SJ Amsterdam, The Netherlands
e-mail: dimitrak@science.uva.nl

M.G. Lagoudakis
Department of Electronic and Computer Engineering, Technical University of Crete, Chania 73100, Crete, Greece
e-mail: lagoudakis@intelligence.tuc.gr

be approximately represented using either multi-class classifiers or combinations of binary classifiers (Rexakis and Lagoudakis 2008) and, therefore, it is possible to incorporate classification algorithms within the inner loops of several reinforcement learning algorithms (Lagoudakis and Parr 2003b; Fern et al. 2004). This viewpoint allows the quantification of the performance of reinforcement learning algorithms in terms of the performance of classification algorithms (Langford and Zadrozny 2005). While a variety of promising combinations become possible through this synergy, heretofore there have been limited practical and widely-applicable algorithms.

Our work builds on the work of Lagoudakis and Parr (2003b) who suggested an approximate policy iteration algorithm for learning a good policy represented as a classifier, avoiding representations of any kind of value function. At each iteration, a new policy/classifier is produced using training data obtained through extensive simulation (rollouts) of the previous policy on a generative model of the process. These rollouts aim at identifying better action choices over a subset of states in order to form a set of data for training the classifier representing the improved policy. A similar algorithm was proposed by Fern et al. (2004) at around the same time. The key differences between the two algorithms are related to the types of learning problems they are suitable for, the choice of the underlying classifier type, and the exact form of classifier training. Nevertheless, the main ideas of producing training data using rollouts and iterating over policies remain the same. Even though both of these studies look carefully into the distribution of training states over the state space, their major limitation remains the large amount of sampling employed at each training state. It is hinted (Lagoudakis 2003), however, that great improvement could be achieved with sophisticated management of rollout sampling.

Our paper suggests managing the rollout sampling procedure within the above algorithm with the goal of obtaining comparable training sets (and therefore policies of similar quality), but with significantly less effort in terms of number of rollouts and computation effort. This is done by viewing the setting as akin to a bandit problem over the rollout states (states sampled using rollouts). Well-known algorithms for bandit problems, such as Upper Confidence Bounds (Auer et al. 2002) and Successive Elimination (Even-Dar et al. 2006), allow optimal allocation of resources (rollouts) to trials (states). Our contribution is twofold: (a) we suitably adapt bandit techniques for rollout management, and (b) we suggest an improved statistical test for identifying early with high confidence states with dominating actions. In return, we obtain up to an order of magnitude improvement over the original algorithm in terms of the effort needed to collect the training data for each classifier. This makes the resulting algorithm attractive to practitioners who need to address large real-world problems.

The remainder of the paper is organized as follows. Section 2 provides the necessary background and Sect. 3 reviews the original algorithm we are based on. Subsequently, our approach is presented in detail in Sect. 4. Finally, Sect. 5 includes experimental results obtained from well-known learning domains.

2 Preliminaries

A *Markov Decision Process* (MDP) is a 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, D)$, where $\mathcal{S}$ is the state space of the process, $\mathcal{A}$ is a finite set of actions, P is a Markovian transition model ($P(s, a, s')$ denotes the probability of a transition to state s' when taking action a in state s), R is a reward function ($R(s, a)$ is the expected reward for taking action a in state s), $\gamma \in (0, 1]$ is the discount factor for future rewards, and D is the initial state distribution.

A *deterministic policy* π for an MDP is a mapping $\pi : \mathcal{S} \mapsto \mathcal{A}$ from states to actions; $\pi(s)$ denotes the action choice at state s. The value $V^\pi(s)$ of a state s under a policy π is the expected, total, discounted reward when the process begins in state s and all decisions at all steps are made according to policy π:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \;\middle|\; s_0 = s, s_t \sim P\right].$$

The goal of the decision maker is to find an optimal policy π^* that maximizes the expected, total, discounted reward from the initial state distribution D:

$$\pi^* = \arg\max_\pi E_{s\sim D}\left[V^\pi(s)\right].$$

It is well-known that for every MDP, there exists at least one optimal deterministic policy.

Policy iteration (PI) (Howard 1960) is an efficient method for deriving an optimal policy. It generates a sequence $\pi_1, \pi_2, \ldots, \pi_k$ of gradually improving policies and terminates when there is no change in the policy ($\pi_k = \pi_{k-1}$); π_k is an optimal policy. Improvement is achieved by computing V^{π_i} analytically (solving the linear Bellman equations) and the action values:

$$Q^{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s,a,s')V^{\pi_i}(s'),$$

and then determining the improved policy as:

$$\pi_{i+1}(s) = \arg\max_{a\in\mathcal{A}} Q^{\pi_i}(s,a).$$

Policy iteration typically terminates in a small number of steps. However, it relies on knowledge of the full MDP model, exact computation and representation of the value function of each policy, and exact representation of each policy. *Approximate policy iteration* (API) is a family of methods, which have been suggested to address the "curse of dimensionality", that is, the huge growth in complexity as the problem grows. In API, value functions and policies are represented approximately in some compact form, but the iterative improvement process remains the same. Apparently, the guarantees for monotonic improvement, optimality, and convergence are compromised. API may never converge, however in practice it reaches good policies in only a few iterations.

In reinforcement learning, the learner interacts with the process and typically observes the state and the immediate reward at every step, however P and R are not accessible. The goal is to gradually learn an optimal policy through interaction with the process. At each step of interaction, the learner observes the current state s, chooses an action a, and observes the resulting next state s' and the reward received r. In many cases, it is further assumed that the learner has the ability to reset the process in any arbitrary state s. This amounts to having access to a generative model of the process (a simulator) from where the learner can draw arbitrarily many times a next state s' and a reward r for performing any given action a in any given state s. Several algorithms have been proposed for learning good or even optimal policies (Sutton and Barto 1998).

3 Rollout classification policy iteration

The *Rollout Classification Policy Iteration* (RCPI) algorithm (Lagoudakis and Parr 2003b; Lagoudakis 2003) belongs to the API family and focuses on direct policy learning and rep-

resentation bypassing the need for an explicit value function. The key idea in RCPI is to cast the problem of policy learning as a classification problem. Thinking of states as examples and of actions as class labels, any deterministic policy can be thought of as a classifier that maps states to actions. Therefore, policies in RCPI are represented (approximately) as generic multi-class classifiers that assign states (examples) to actions (classes). The problem of finding a good policy is equivalent to the problem of finding a classifier that maps states to "good" actions, where the goodness of an action is measured in terms of its contribution to the long term goal of the agent. The state-action value function Q^π in the context of a fixed policy π provides such a measure; the action that maximizes Q^π in state s is a "good" action, whereas any action with smaller value of Q^π is a "bad" one. A training set could be easily formed if the Q^π values for all actions were available for a subset of states.

The Monte-Carlo estimation technique of *rollouts* provides a way of accurately estimating Q^π at any given state-action pair (s, a) without requiring an explicit representation of the value function. A rollout for (s, a) amounts to simulating a trajectory of the process beginning from state s, choosing action a for the first step, and choosing actions according to the policy π thereafter up to a certain horizon T. The observed total discounted reward is averaged over a number of rollouts to yield an estimate. Thus, using a sufficient amount of rollouts it is possible to form a valid training set for the improved policy over any base policy. More specifically, if we denote the sequence of collected rewards during the i-th simulated trajectory as $r_t^{(i)}$, $t = 0, 1, 2, \ldots, T-1$, then the rollout estimate $\hat{Q}_K^{\pi,T}(s, a)$ of the true state-action value function $Q^\pi(s, a)$ is the observed total discounted reward, averaged over all K trajectories:

$$\hat{Q}_K^{\pi,T}(s,a) \triangleq \frac{1}{K}\sum_{i=1}^{K} \tilde{Q}_{(i)}^{\pi,T}(s,a), \qquad \tilde{Q}_{(i)}^{\pi,T}(s,a) \triangleq \sum_{t=0}^{T} \gamma^t r_t^{(i)}.$$

With a sufficient amount of rollouts and a large T, we can create an improved policy π' from π at any state s, without requiring a model of the MDP.

Algorithm 1 describes RCPI step-by-step. Beginning with any initial policy π_0, a training set over a subset of states S_R is formed by querying the rollout procedure for the state-action values of all actions in each state $s \in S_R$ with the purpose of identifying the "best" action and the "bad" actions in s. An action is said to be *dominating* if its empirical value is significantly greater than those of all other actions. In RCPI this is measured in a statistical sense using a pairwise t-test, to factor out estimation errors. Notice that the training set contains both positive and negative examples for each state where a clear domination is found. A new classifier is trained using these examples to yield an approximate representation of the improved policy over the previous one. This cycle is then repeated until a termination condition is met. Given the approximate nature of this policy iteration, the termination condition cannot rely on convergence to a single optimal policy. Rather, it terminates when the performance of the new policy (measured via simulation) does not exceed that of the previous policy.

The RCPI algorithm has yielded promising results in several learning domains, however, as stated also by Lagoudakis (2003), it is sensitive to the distribution of states in S_R over the state space. For this reason it is suggested to draw states from the γ-discounted future state distribution of the improved policy. This tricky-to-sample distribution, also suggested by Fern et al. (2004), yields better results and resolves any potential mismatch between the training and testing distributions of the classifier. However, the main drawback is still the excessive computational cost due to the need for lengthy and repeated rollouts to reach a good level of accuracy. In our experiments with RCPI, it has been observed that most of the

Algorithm 1 Rollout Classification Policy Iteration

Input: rollout states S_R, initial policy π_0, trajectories K, horizon T, discount factor γ

$\pi' = \pi_0$ (default: uniformly random)
repeat
  $\pi = \pi'$
  TrainingSet $= \varnothing$
  for (each $s \in S_R$) **do**
    for (each $a \in \mathcal{A}$) **do**
      estimate $Q^{\pi}(s, a)$ using K rollouts of length T
    end for
    if (a dominating action a^* exists in s) **then**
      TrainingSet = TrainingSet $\cup \{(s, a^*)^+\}$
      TrainingSet = TrainingSet $\cup \{(s, a)^-\}, \forall\, a \neq a^*$
    end if
  end for
  $\pi' =$ TRAINCLASSIFIER(TrainingSet)
until ($\pi \approx \pi'$)
return π

effort is wasted on states where action value differences are either non-existent or so fine that they require one to use a prohibitive number of rollouts to identify them. Significant effort is also wasted on sampling states where a dominating action could be easily identified without exhausting all rollouts allocated to it. In this paper, we propose rollout sampling methods to remove this performance bottle-neck.

4 Rollout sampling policy iteration

The excessive sampling cost mentioned above can be reduced by careful management of resources. The scheme suggested by RCPI, also used by Fern et al. (2004), is somewhat naïve; the same number of $K|\mathcal{A}|$ rollouts is allocated to each state in the subset S_R and all K rollouts dedicated to a single action are exhausted before moving on to the next action. Intuitively, if the desired outcome (domination of a single action) in some state can be confidently determined early, there is no need to exhaust all $K|\mathcal{A}|$ rollouts available in that state; the training data could be stored and the state could be removed from the pool without further examination. Similarly, if we can confidently determine that all actions are indifferent in some state, we can simply reject it without wasting any more rollouts; such rejected states could be replaced by fresh ones which might yield meaningful results. These ideas lead to the following question: can we examine all states in the subset S_R collectively in some interleaved manner by choosing each time a single state to focus on, allocating rollouts only as needed?

A similar resource allocation setting in the context of reinforcement learning are bandit problems. Therein, the learner is faced with a choice between n bandits, each one having an unknown reward function. The task is to allocate plays such as to discover the bandit with the highest expected reward without wasting too many resources in either cumulative

Algorithm 2 SAMPLESTATE

Input: state s, policy π, horizon T, discount factor γ

for (each $a \in \mathcal{A}$) **do**
  $(s', r) = \text{SIMULATE}(s, a)$
  $\tilde{Q}^{\pi}(s, a) = r$
  $x = s'$
  for $t = 1$ **to** $T - 1$ **do**
    $(x', r) = \text{SIMULATE}(x, \pi(x))$
    $\tilde{Q}^{\pi}(s, a) = \tilde{Q}^{\pi}(s, a) + \gamma^t r$
    $x = x'$
  end for
end for
return $\tilde{Q}^{\pi}(s, \cdot)$

reward, or in number of plays required.[1] Taking inspiration from such problems, we view the set of rollout states as a multi-armed bandit, where each state corresponds to a single lever/arm. Pulling a lever corresponds to sampling the corresponding state once. By *sampling a state* we mean that we perform a single rollout for each action in that state as shown in Algorithm 2. This is the minimum amount of information we can request from a single state.[2] Thus, the problem is transformed to a *variant* of the classic multi-armed bandit problem. Several methods have been proposed for various versions of this problem, which could potentially be used in this context. In this paper, we focus on three of them: simple counting, upper confidence bounds (Auer et al. 2002), and successive elimination (Even-Dar et al. 2006).

Our goal at this point is to collect good training data for the classifier with as little computational effort as possible. We can quantify the notion of goodness for the training data in terms of three guarantees: (a) that states will be sampled only as needed to produce training data without wasting rollouts, (b) that with high probability, the discovered action labels in the training data indicate dominating actions, and (c) that the training data cover the state space sufficiently to produce a good representation of the entire policy. We look at each one of these objectives in turn.

4.1 Rollout management

As mentioned previously, our algorithm maintains a pool of states S_R from which sampling is performed. In this paper, states $s \in S_R$ are drawn from a uniformly random distribution to cover the state space evenly, however other, more sophisticated, distributions may also be used. In order to allocate rollouts wisely, we need to decide which state to sample from at every step. We also need to determine criteria to decide when to stop sampling from a state, when to add new states to the pool, and finally when to stop sampling completely.

[1] The precise definition of the task depends on the specific problem formulation and is beyond the scope of this article.

[2] It is possible to also manage sampling of the actions within a state, but our preliminary experiments showed that managing action sampling alone saved little effort compared to managing state sampling. We are currently working on managing sampling at both levels.

The general form of the state selection rule for all algorithms is:

$$s = \arg\max_{s' \in S_R} U(s'),$$

where $U(s)$ represents the utility associated with sampling state s. The presented algorithms use one of the following variants:

1. COUNT, SUCCE: $U(s) \triangleq -c(s)$
2. SUCB1: $U(s) \triangleq \hat{\Delta}^{\pi}(s) + \sqrt{1/(1+c(s))}$
3. SUCB2: $U(s) \triangleq \hat{\Delta}^{\pi}(s) + \sqrt{\ln m/(1+c(s))}$

where $c(s)$ is a counter recording the number of times state s has been sampled, m is the total number of state samples, and $\hat{\Delta}^{\pi}(s)$ is the empirical counterpart of the marginal difference $\Delta^{\pi}(s)$ in Q^{π} values in state s defined as

$$\Delta^{\pi}(s) \triangleq Q^{\pi}(s, a^*_{s,\pi}) - \max_{a \neq a^*_{s,\pi}} Q^{\pi}(s, a),$$

where $a^*_{s,\pi}$ is the action[3] that maximizes Q^{π} in state s:

$$a^*_{s,\pi} = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a).$$

Similarly, the empirical difference $\hat{\Delta}^{\pi}(s)$ is defined in terms of the empirical Q values:

$$\hat{\Delta}^{\pi}(s) \triangleq \hat{Q}^{\pi,T}_K(s, \hat{a}^*_{s,\pi}) - \max_{a \neq \hat{a}^*_{s,\pi}} \hat{Q}^{\pi,T}_K(s, a),$$

where $\hat{a}^*_{s,\pi}$ is the action that maximizes $\hat{Q}^{\pi,T}_K$ in state s:

$$\hat{a}^*_{s,\pi} = \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi,T}_K(s, a),$$

with $K = c(s)$ and some fixed T independent of s.

The COUNT variant is a simple counting criterion, where the state that has been sampled least has higher priority for being sampled next. Since we stop sampling a state as soon as we have a sufficiently good estimate, this criterion should result in less sampling compared to RCPI, which continues sampling even after an estimate is deemed sufficiently good.

The SUCCE variant uses the same criterion as COUNT to sample states, but features an additional mechanism for removing apparently hopeless states from S_R. This is based on the Successive Elimination algorithm (Algorithm 3 in Even-Dar et al. 2006). We expect this criterion to be useful in problems with many states where all actions are indifferent. However, it might also result in the continual rejection of small-difference states until a high-difference state is sampled, effectively limiting the amount of state space covered by the final gathered examples.

The SUCB1 variant is based on the UCB algorithm (Auer et al. 2002) and gives higher priority to states with a high empirical difference and high uncertainty as to what the difference is. Thus, states can take priority for two reasons. Firstly, because they have been sampled less, and secondly because they are more likely to result in acceptance quickly.

[3]The case of multiple equivalent maximizing actions can be easily handled by generalising to sets of actions in the manner of Fern et al. (2006). Here we discuss only the single best action case to simplify the exposition.

The SUCB2 variant is based on the original UCB1 algorithm by Auer et al. (2002), in that it uses a shrinking error bound for calculating the upper confidence interval. Since in our setting we stop sampling states where the difference in actions is sufficiently large, this will be similar to simple counting as the process continues. However, intuitively it will focus on those states that are most likely to result in a positive identification of a dominating action quickly towards the end.

In all cases, new states are added to the pool as soon as a state has been removed, so S_R has a constant size. The criterion for selecting examples is described in the following section.

4.2 Statistical significance

Sampling of states proceeds according to one of these rules at each step. Once a state is identified as "good", it is removed from the state pool and is added to the training data to prevent further "wasted" sampling on that state.[4] In order to terminate sampling and accept a state as good, we rely on the following well-known lemma.

Lemma 1 (Hoeffding inequality) *Let X be a random variable in $[b_1, b_2]$ with $\bar{X} \triangleq \mathbf{E}[X]$, observed values $x_1, x_2, \ldots, x_n$ of X, and $\hat{X}_n \triangleq \frac{1}{n}\sum_{i=1}^{n} x_i$. Then $\mathbf{P}(\hat{X}_n \geq \bar{X} + \epsilon) = \mathbf{P}(\hat{X}_n \leq \bar{X} - \epsilon) \leq \exp(-2n\epsilon^2/(b_1 - b_2)^2)$.*

Consider two random variables X, Y, their true means $\bar{X}, \bar{Y}$, and their empirical means $\hat{X}_n, \hat{Y}_n$, as well as a random variable $\Delta \triangleq X - Y$ representing their difference, its true mean $\bar{\Delta} \triangleq \bar{X} - \bar{Y}$, and its empirical mean $\hat{\Delta}_n \triangleq \hat{X}_n - \hat{Y}_n$. If $\Delta \in [b_1, b_2]$, it follows from Lemma 1 that

$$\mathbf{P}(\hat{\Delta}_n \geq \bar{\Delta} + \epsilon) \leq \exp\left(-\frac{2n\epsilon^2}{(b_2 - b_1)^2}\right). \tag{1}$$

We now consider applying this for determining the best action at any state s where we have taken $c(s)$ samples from every action. As previously, let $\hat{a}^*_{s,\pi}$ be the empirically optimal action in that state. If $\Delta^\pi(s) \in [b_1, b_2]$, then for any $a' \neq \hat{a}^*_{s,\pi}$, we can set $\bar{X} = Q^\pi(s, \hat{a}^*_{s,\pi})$, $\bar{Y} = Q^\pi(s, a')$, and correspondingly $\hat{X}_n, \hat{Y}_n$ to obtain:

$$\mathbf{P}\left(\hat{Q}^\pi(s, \hat{a}^*_{s,\pi}) - \hat{Q}^\pi(s, a') \geq Q^\pi(s, \hat{a}^*_{s,\pi}) - Q^\pi(s, a') + \epsilon\right) \leq \exp\left(-\frac{2c(s)\epsilon^2}{(b_2 - b_1)^2}\right). \tag{2}$$

Corollary 1 *For any state s where the following condition holds*

$$\hat{\Delta}^\pi(s) \geq \sqrt{\frac{(b_2 - b_1)^2}{2c(s)} \ln\left(\frac{|\mathcal{A}| - 1}{\delta}\right)}, \tag{3}$$

the probability of incorrectly identifying $a^*_{s,\pi}$ is bounded by δ.

[4] Of course, if we wanted to continuously shrink the probability of error we could continue sampling from those states.

Proof We can set ϵ equal to the right hand side of (3), to obtain:

$$\mathbf{P}\left(\hat{Q}^{\pi}(s,\hat{a}^{*}_{s,\pi}) - \hat{Q}^{\pi}(s,a') \geq Q^{\pi}(s,\hat{a}^{*}_{s,\pi}) - Q^{\pi}(s,a') + \sqrt{\frac{(b_2-b_1)^2}{2c(s)}\ln\left(\frac{|\mathcal{A}|-1}{\delta}\right)}\right) \leq \delta/(|\mathcal{A}|-1). \tag{4}$$

Incorrectly identifying $a^{*}_{s,\pi}$ implies that there exists some a' such that $Q^{\pi}(s,\hat{a}^{*}_{s,\pi}) - Q^{\pi}(s,a') \leq 0$, while $\hat{Q}^{\pi}(s,\hat{a}^{*}_{s,\pi}) - \hat{Q}^{\pi}(s,a') > 0$. However, due to our stopping condition,

$$\hat{Q}^{\pi}(s,\hat{a}^{*}_{s,\pi}) - \hat{Q}^{\pi}(s,a') \geq \hat{\Delta}^{\pi}(s) \geq \sqrt{\frac{(b_2-b_1)^2}{2c(s)}\ln[(|\mathcal{A}|-1)/\delta]},$$

so in order to make a mistake concerning the ordering of the two actions, the estimation error must be larger than the right side of (3). Thus, this probability is also bounded by $\delta/(|\mathcal{A}|-1)$. Given that the number of actions $a' \neq \hat{a}^{*}_{s,\pi}$ is $|\mathcal{A}|-1$, an application of the union bound implies that the total probability of making a mistake in state s must be bounded by δ. □

In summary, every time s is sampled, both $c(s)$ and $\hat{\Delta}^{\pi}(s)$ change. Whenever the stopping condition in (3) is satisfied, state s can be safely removed from S_R; with high probability $(1-\delta)$ the current empirical difference value will not change sign with further sampling and confidently the resulting action label is indeed a dominating action.[5] Finally note that in practice, we might not be able to obtain full trajectories—in this case, the estimates and true value functions should be replaced with their T-horizon versions.

4.3 State space coverage

For each policy improvement step, the algorithm terminates when we have succeeded in collecting $n_{\max}$ examples, or when we have performed $m_{\max}$ rollouts. Initially, $|S_R| = n_{\max}$. In order to make sure that training data are not restricted to a static subset S_R, every time a state is characterized good and removed from S_R, we add a new state to S_R drawn from some fixed distribution $\mathcal{D}_R$ that serves as a source of rollout states. The simplest choice for $\mathcal{D}_R$ would be a uniform distribution over the state space, however other choices are possible, especially if domain knowledge about the structure of good policies is known. A sophisticated choice of $\mathcal{D}_R$ is a difficult problem itself and we do not investigate it here; it has been conjectured that a good choice is the γ-discounted future state distribution of the improved policy being learned (Lagoudakis and Parr 2003b; Fern et al. 2004).

We have also toyed with the idea of rejecting states which seem hopeless to produce training data, replacing them with fresh states sampled from some distribution $\mathcal{D}_R$. The SUCCE rule incorporates such a rejection criterion by default (Even-Dar et al. 2006). For the other variants, if rejection is adopted, we reject all states $s \in S_R$ with $U(s) < \sqrt{\ln m}$, which suits SUCB2 particularly well.

The complete algorithm, called *Rollout Sampling Policy Iteration* (RSPI), is described in detail in Algorithm 3. The call to SELECTSTATE refers to one of the four selection rules described above. Note that a call to SUCCE might also eliminate some states from S_R replacing them with fresh ones drawn from $\mathcal{D}_R$.

[5] The original RCPI algorithm employed a pairwise t-test. This choice is flawed, since it assumes a normal distribution of errors, whereas the Hoeffding bound simply assumes that the variables are bounded.

Algorithm 3 Rollout Sampling Policy Iteration

Input: distribution $\mathcal{D}_R$, initial policy π_0, horizon T, discount factor γ, max data $n_{\max}$, max samples $m_{\max}$, probability δ, number of rollout states N, Boolean Rejection, range $[a, b]$

$\pi' = \pi_0$ (default: random), $n = 0$, $m = 0$
$S_R \sim \mathcal{D}_R^N$ (default: $N = n_{\max}$)
for all $s \in S_R$, $a \in \mathcal{A}$: $\hat{Q}^\pi(s,a) = 0$, $\hat{\Delta}^\pi(s) = 0$, $U(s) = 0$, $c(s) = 0$
repeat
  $\pi = \pi'$
  TrainingSet $= \varnothing$
  while ($n \le n_{\max}$ and $m \le m_{\max}$) **do**
    $s =$ SELECTSTATE($S_R, \hat{\Delta}^\pi, c, m$)
    $\tilde{Q}^\pi =$ SAMPLESTATE(s, π, T, γ)
    update $\hat{Q}^\pi(s,a)$, $\hat{\Delta}^\pi(s)$, and $U(s)$ using $\tilde{Q}^\pi(s,a)$
    $c(s) = c(s) + 1$
    $m = m + 1$
    if $\left(2c(s)\left(\hat{\Delta}^\pi(s)\right)^2 \ge (b_2 - b_1)^2 \ln\left(\frac{|\mathcal{A}|-1}{\delta}\right)\right)$ **then**
      $n = n + 1$
      TrainingSet $=$ TrainingSet $\cup \{(s, \hat{a}^*_{s,\pi})^+\}$
      TrainingSet $=$ TrainingSet $\cup \{(s,a)^-\}, \forall a \ne \hat{a}^*_{s,\pi}$
      $S_R = S_R - \{s\}$
      $S_R = S_R \cup \{s' \sim \mathcal{D}_R\}$
    end if
    if (Rejection) **then**
      for (each $s \in S_R$) **do**
        if ($U(s) < \sqrt{\ln m}$) **then**
          $S_R = S_R - \{s\}$
          $S_R = S_R \cup \{s' \sim \mathcal{D}_R\}$
        end if
      end for
    end if
  end while
  $\pi' =$ TRAINCLASSIFIER(TrainingSet)
until ($\pi \approx \pi'$)
return π

5 Experiments

To demonstrate the performance of the proposed algorithm in practice and to set the basis for comparison with RCPI, we present experimental results on two standard reinforcement learning domains, namely the inverted pendulum and the mountain car. In both domains, we tried several settings of the various parameters related to state sampling. However, we kept the learning parameters of the classifier constant and used the new statistical test even for RCPI to filter out their influence. In all cases, we measured the performance of the resulting policies against the effort needed to derive them in terms of number of samples. Sections 5.1 and 5.2 describe the learning domains, while the exact evaluation method used and results are described in Sect. 5.3.

5.1 Inverted pendulum

The *inverted pendulum* problem is to balance a pendulum of unknown length and mass at the upright position by applying forces to the cart it is attached to. Three actions are allowed: left force (LF), right force (RF), or no force (NF), applying $-50N$, $+50N$, $0N$ respectively, with uniform noise in $[-10, 10]$ added to the chosen action. Due to the noise in the problem, the return from any single state-action pair is stochastic even though we are only employing deterministic policies. Had this not been the case, we would have needed but a single sample from each state. The state space is continuous and consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The transitions are governed by the nonlinear dynamics of the system (Wang et al. 1996) and depend on the current state and the current control u:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)u}{4l/3 - \alpha ml\cos^2(\theta)},$$

where g is the gravity constant ($g = 9.8$ m/s^2), m is the mass of the pendulum ($m = 2.0$ kg), M is the mass of the cart ($M = 8.0$ kg), l is the length of the pendulum ($l = 0.5$ m), and $\alpha = 1/(m + M)$. The simulation step is 0.1 seconds, while the control input is changed only at the beginning of each time step, and is kept constant for its duration.

A reward of 0 is given as long as the angle of the pendulum does not exceed $\pi/2$ in absolute value (the pendulum is above the horizontal line). An angle greater than $\pi/2$ signals the end of the episode and a reward (penalty) of -1. The discount factor of the process is set to 0.95. This forces the Q value function to lie in $[-1, 0]$, so we can set $b_1 = -1, b_2 = 0$ for this problem.

5.2 Mountain-car

The *mountain-car* problem is to drive an underpowered car from the bottom of a valley between two mountains to the top of the mountain on the right. The car is not powerful enough to climb any of the hills directly from the bottom of the valley even at full throttle; it must build some momentum by climbing first to the left (moving away from the goal) and then to the right. Three actions are allowed: forward throttle FT ($+1$), reverse throttle RT (-1), or no throttle NT (0). The original specification assumes a deterministic transition model. To make the problem a little more challenging we have added noise to all three actions; uniform noise in $[-0.2, 0.2]$ is added to the chosen action's effect. Again, due to the noise in this problem, the returns are stochastic, thus necessitating the use of multiple samples at each state. The state space of the problem is continuous and consists of the position x and the velocity $\dot{x}$ of the car along the horizontal axis. The transitions are governed by the simplified nonlinear dynamics of the system (Sutton and Barto 1998) and depend on the current state $(x(t), \dot{x}(t))$ and the current (noisy) control $u(t)$:

$$x(t+1) = \textsc{Bound}_x[x(t) + \dot{x}(t+1)]$$
$$\dot{x}(t+1) = \textsc{Bound}_{\dot{x}}[\dot{x}(t) + 0.001u(t) - 0.0025\cos(3x(t))],$$

where $\textsc{Bound}_x$ is a function that keeps x within $[-1.2, 0.5]$, while $\textsc{Bound}_{\dot{x}}$ keeps $\dot{x}$ within $[-0.07, 0.07]$. If the car hits the left bound of the position x, the velocity $\dot{x}$ is set to zero.

For this problem, a penalty of -1 is given at each step as long as the position of the car is below the right bound (0.5). As soon as the car position hits the right bound, the episode ends successfully and a reward of 0 is given. The discount factor of the process is set to 0.99.

Choosing $[b_1, b_2]$ for this problem is trickier, since without any further conditions, the value function lies in $(-100, 0]$. However, the difference between Q values for any state does not vary much in practice. That is, for most state and policy combinations the initial action does not alter the final reward by more than 1. For this reason, we used $|b_1 - b_2| = 1$.

5.3 Evaluation

After a preliminary investigation we selected a multi-layer perceptron with 10 hidden units as the classifier for representing policies and stochastic gradient descent with a learning rate of 0.5 for 25 iterations of training. Note that this is only one of numerous choices.

The main problem was to devise an experiment to determine the computational effort that would be required by each method to find an optimal policy in practice. This meant that for each method we would have to simulate the process of manual tuning that a practitioner would perform in order to discover optimal solutions. A usual practice is to perform a grid search in the space of hyper-parameters, with multiple runs per grid point. Assuming that the experimenter can perform a number of such runs in parallel, we can then use the number of solutions found after a certain number of samples taken by each method as a practical metric of the sample complexity of the algorithms.

More specifically, we tested all the proposed state selection methods (COUNT, SUCB1, SUCB2, SUCCE) with RSPI and RCPI for each problem. For all methods, we used the following sets of hyper-parameters: $m_{\max}, n_{\max} \in \{10, 20, 50, 100, 200\}$, and $\delta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ for the pendulum and $\delta \in \{0.5, 10^{-1}, 10^{-2}\}$ for the car.[6] We performed 5 runs with different random seeds for each hyper-parameter combination, for a total of 375 runs per method. After each run, the resulting policy was tested for quality; a policy that could balance the pendulum for at least 1000 steps or a policy that could drive the car to the goal in under 75 steps from the starting position were considered successful (practically optimal).

We report the cumulative distribution of successful policies found against the number of samples (rollouts) used by each method, summed over all runs. Formally, if x is the number of samples along the horizontal axis, we plot the measure $f(x) = \mu\{\pi_i : \pi_i \text{ is successful}, m_i \leq x\}$, where μ denotes the measure of a set, i.e. the horizontal axis shows the least number of samples required to obtain the number of successful runs shown in the vertical axis. Effectively, the figures show the number of samples x required to obtain $f(x)$ near-optimal policies, if the experimenter was fortuitous enough to select the appropriate hyper-parameters.

In more detail, Fig. 1 shows the results for the pendulum problem. While the COUNT, SUCB1, SUCB2, SUCCE methods have approximately the same total number of successful runs, SUCB1 clearly dominates, as after 4000 samples per run, it had already obtained 180 successful policies; at that point it has six times more chances of producing a successful policy compared to RCPI. In the contrary, RCPI only managed to produce less than half the total number of policies as the first method. More importantly, none of its runs had produced any successful policies at all with fewer than 2000 samples—a point at which all the other methods were already making significant progress.

Perhaps it is worthwhile noting at this point that the step-wise form of the RCPI plot is due to the fact that it was always terminating sampling when all its rollouts had been

[6] In exploratory runs, it appeared particularly hard to obtain any samples at all for the car problem with $\delta = 10^{-3}$ so we used 0.5 instead.

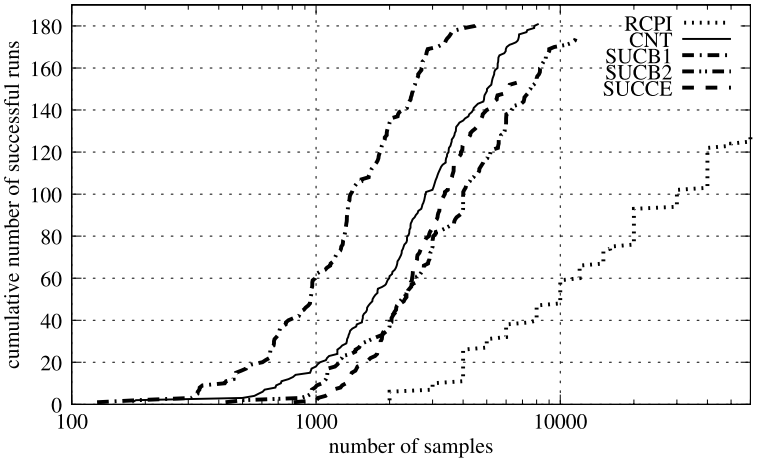

Fig. 1 The cumulative distribution of successful runs (at least 1000 steps of balancing) in the pendulum domain

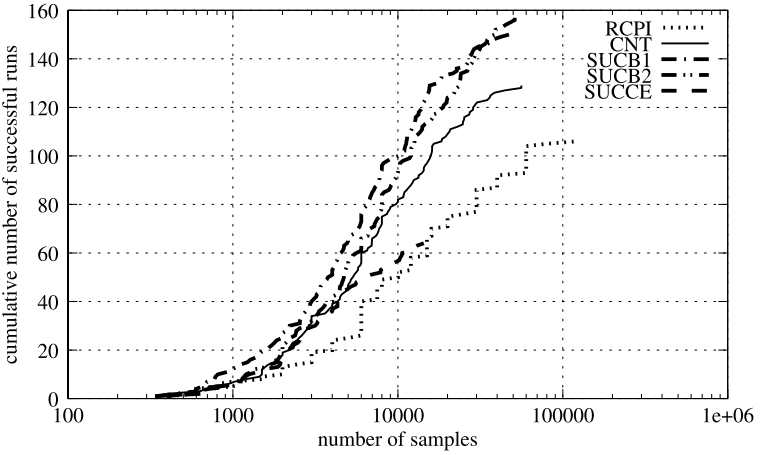

Fig. 2 The cumulative distribution of successful runs (less than 75 steps to reach the goal) in the mountain-car domain

exhausted. The other methods may also terminate whenever $n_{\max}$ good samples have been obtained. Due to this reason, the plots might terminate at an earlier stage.

Similarly, Fig. 2 shows the results for the mountain-car problem. This time, we consider runs where less than 75 steps have been taken to reach the goal as successful. Again, it is clear that the proposed methods perform better than RCPI as they have higher chances of producing good policies with fewer samples. Once again, SUCB1 exhibits an advantage over the other methods. However, the differences between methods are slightly finer in this domain.

It is interesting to note that the results were not very sensitive to the actual value of δ. In fact we were usually able to obtain good policies with quite large values (i.e. 0.5 in the mountain car domain). On the other hand, if one is working with a limited budget of

rollouts, a very small value of δ, might make convergence impossible, since there are not enough rollouts available to obtain the best actions with the necessary confidence. A similar thing occurs when $|b_2 - b_1|$ is very large, as we noticed with initial experiments with the mountain car where we had set them to $[-100, 0]$.

Perhaps predictably, the most important parameter appeared to be $n_{\max}$. Below a certain threshold, no good policies could be found by any algorithm. This in general occurred when the total number of good states at the end of an iteration were too few for the classifier to be able to create an improved policy.

Of course, when δ and $n_{\max}$ are very large, there is no guarantee for the performance of policy improvement, i.e. we cannot bound the probability that all of the states will use the correct action labels. However, this does not appear to be a problem in practice. We posit two factors that may explain this. Firstly, the relatively low stochasticity of the problems: if the environments and policies were deterministic, then a single sample would have been enough to determine the optimal action at each state. Secondly, the smoothing influence of the classifier may be sufficient for policy improvement even if some portion of the states sampled have incorrect labels.

Computational time does not give meaningful measurements in this setting as the time taken for each trajectory depends on how many steps pass until the episode terminates. For some problems (i.e. infinite-horizon problems with a finite horizon cutoff for the rollout estimate), this may be constant, but for others the length of time varies with the quality of the policy: in the pendulum domain, policies run for longer as they improve, while the opposite occurs in the mountain car problem. For this reason we decided to only report results of sample complexity.

We would finally like to note that our experiments with additional rejections and replacements of states failed to produce a further improvement. However, such methods might be of use in environments where the actions are indistinguishable in most states.

6 Discussion

The proposed approaches deliver equally good policies as those produced by RCPI, but with significantly less effort; in both problems, there is up to an order of magnitude reduction in the number of rollouts performed and thus in computational effort. We thus conclude that the selective sampling approach can make rollout algorithms much more practical, especially since similar approaches have already demonstrated their effectiveness in the planning domain (Kocsis and Szepesvári 2006). However, some practical obstacles remain—in particular, the choice of $\delta, n_{\max}, |b_1 - b_2|$ is not easy to determine a priori, especially when the choice of classifier needs to be taken into account as well. For example, a nearest-neighbour classifier may not tolerate as large a δ as a soft-margin support vector machine. Unfortunately, at this point, the choice of hyper-parameters can only be done via laborious experimentation. Even so, since the original algorithm suffered from the same problem, the experimenter is at least assured that not as much time will be spent until an optimal solution is found, as our results show.

Currently the bandit algorithm variants employed for state rollout selection are used in a heuristic manner. However, in a companion paper (Dimitrakakis and Lagoudakis 2008), we have analyzed the whole policy iteration process and proved PAC-style bounds on the progress that the COUNT method is guaranteed to make under certain assumptions on the underlying MDP model. We hope to extend this work in the future in order to produce bandit-like algorithms that are specifically tuned for this task. Furthermore, we plan to address rollout sampling both at the state and the action levels and focus our attention on

sophisticated state sampling distributions and on exploiting sampled states for which no clear negative or positive action examples are drawn, possibly by developing a variant of the upper bound on trees algorithm (Kocsis and Szepesvári 2006). A complementary research route would be to integrate sampling procedures with fitting algorithms that can use a single trajectory, such as (Antos et al. 2008).

In summary, we have presented an approximate policy iteration scheme for reinforcement learning, which relies on classification technology for policy representation and learning and clever management of resources for obtaining data. It is our belief that the synergy between these two learning paradigms has still a lot to reveal to machine learning researchers.

Acknowledgements We would like to thank the reviewers for providing valuable feedback and Katerina Mitrokotsa for additional proofreading. This work was partially supported by the ICIS-IAS project and the European Marie-Curie International Reintegration Grant MCIRG-CT-2006-044980 awarded to Michail G. Lagoudakis.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

Antos, A., Szepesvári, C., & Munos, R. (2008). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, *71*(1), 89–129. 10.1007/s10994-007-5038-2.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning Journal 47*(2–3), 235–256.

Dimitrakakis, C., & Lagoudakis, M. (2008). *Algorithms and bounds for sampling-based approximate policy iteration*. (To be presented at the 8th European Workshop on Reinforcement Learning).

Even-Dar, E., Mannor, S., & Mansour, Y. (2006). Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, *7*, 1079–1105. ISSN 1533-7928.

Fern, A., Yoon, S., & Givan, R. (2004). Approximate policy iteration with a policy language bias. *Advances in Neural Information Processing Systems*, *16*(3).

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, *25*, 75–118.

Howard, R. A. (1960). *Dynamic programming and Markov processes*. Cambridge: MIT Press.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European conference on machine learning*.

Lagoudakis, M. G. (2003). *Efficient approximate policy iteration methods for sequential decision making in reinforcement learning*. PhD thesis, Department of Computer Science, Duke University.

Lagoudakis, M. G., & Parr, R. (2003a). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*(6), 1107–1149.

Lagoudakis, M. G. & Parr, R. (2003b). Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the 20th international conference on machine learning (ICML)* (pp. 424–431). Washington, DC, USA.

Langford, J., & Zadrozny, B. (2005). Relating reinforcement learning performance to classification performance. In *Proceedings of the 22nd international conference on machine learning (ICML)* (pp. 473–480). Bonn, Germany, 2005. ISBN 1-59593-180-5. doi:10.1145/1102351.1102411.

Rexakis, I., & Lagoudakis, M. (2008). *Classifier-based policy representation*. (To be presented at the 8th European Workshop on Reinforcement Learning).

Riedmiller, M. (2005). Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In *16th European conference on machine learning* (pp. 317–328).

Sutton, R., & Barto, A. (1998). *Reinforcement learning: an introduction*. Cambridge: MIT Press.

Wang, H. O., Tanaka, K., & Griffin, M. F. (1996). An approach to fuzzy control of nonlinear systems: Stability and design issues. *IEEE Transactions on Fuzzy Systems*, *4*(1), 14–23.