Directed Exploration of Policy Space using Support Vector Classifiers

Ioannis Rexakis and Michail G. Lagoudakis
Department of Electronic and Computer Engineering
Technical University of Crete
Chania, Crete 73100, Greece
Email: {yr, lagoudakis}@intelligence.tuc.gr

***Abstract*—Good policies in reinforcement learning problems typically exhibit significant structure. Several recent learning approaches based on the approximate policy iteration scheme suggest the use of classifiers for capturing this structure and representing policies compactly. Nevertheless, the space of possible policies, even under such structured representations, is huge and needs to be explored carefully to avoid computationally expensive simulations (rollouts) needed to probe the improved policy and obtain training samples at various points over the state space. Regarding rollouts as a scarce resource, we propose a method for directed exploration of policy space using support vector classifiers. We use a collection of binary support vector classifiers to represent policies, whereby each of these classifiers corresponds to a single action and captures the parts of the state space where this action dominates over the other actions. After an initial training phase with rollouts uniformly distributed over the entire state space, we use the support vectors of the classifiers to identify the critical parts of the state space with boundaries between different action choices in the represented policy. The policy is subsequently improved by probing the state space only at points around the support vectors that are distributed perpendicularly to the separating border. This directed focus on critical parts of the state space iteratively leads to the gradual refinement and improvement of the underlying policy and delivers excellent control policies in only a few iterations with a conservative use of rollouts. We demonstrate the proposed approach on three standard reinforcement learning domains: inverted pendulum, mountain car, and acrobot.**

I. Introduction

The goal in sequential decision making is to find good policies for selecting actions in order to achieve a long term reward; such problems are typically modeled as Markov Decision Processes (MDPs). In reinforcement learning, the goal of finding a good policy must be accomplished through interaction with an unknown process typically modeled as an MDP. In either case, good deterministic action policies over the state space of the process can be approximately represented using (multi-class) classifiers; each action is viewed as a distinct class and the states are the instances to be classified. In addition, such policies for common domains are not arbitrary, but rather exhibit significant structure. Several recent learning approaches based on the approximate policy iteration framework suggest the use of classifiers for capturing this structure and representing policies compactly [1]–[7]. Such classifiers can be learned using appropriate training data sets that reveal the desired action choice over a finite set of states. The most attractive benefit of this approach is the elimination of value function representation and the focus instead directly on policy learning. While it is known [8] that it is easier to represent a policy, rather than a value function, the full potential of this reinforcement-learning-through-classification approach for various learning problems has not been studied in depth.

The space of possible policies, even under such structured representations, is huge and needs to be explored carefully to avoid computationally expensive simulations (rollouts). Our work aims at providing guidance on how to select the subset of the state space where the improved policy is probed to form the training set for the classification problem. This aspect has not been studied in depth in the past, nevertheless it plays a crucial role, considering that each probe requires a significant amount of resources (simulation) and therefore they better be focused on critical states that can potentially lead to policy improvement. We propose a method for directed exploration of policy space using support vector classifiers. We use a collection of binary support vector classifiers to represent policies, whereby each of these classifiers corresponds to a single action and captures the parts of the state space where this action dominates over the other actions. We exploit the ability of support vector classifiers to reveal analytically their decision boundary. After an initial training phase with uniform rollouts over the entire state space, we use the support vectors of the classifiers to identify the critical parts of the state space with boundaries between different action choices in the represented policy. The policy is subsequently improved by probing the state space only at a randomized selection of points around the support vectors that are distributed perpendicularly to the separating border. This directed focus on critical parts of the state space iteratively leads to the gradual refinement and improvement of the underlying policy, while making conservative use of rollouts. The randomized nature of each iteration allows us to make several improvement attempts at each iteration to guarantee monotonicity in policy quality until no improvement is possible. We demonstrate our approach on three standard reinforcement learning domains: inverted pendulum, mountain car, and acrobot.

The remainder of the paper is organized as follows. Section II provides the necessary background and Section III reviews reinforcement learning as classification. In Section IV we present our approach on how to direct the exploration of policy space to critical parts by careful selection of the probes in the state space. Section V describes in detail the three benchmarks domains on which we validate our proposed algorithm. Section VI presents experimental results demonstrating the effectiveness of our approach on those three domains and finally Section VII discusses our results and concludes.

II. Background

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 ($P(s'|s,a)$ 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 π:

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

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:

$$\pi^* = \arg\max_\pi E_{s \sim D; s_t \sim P}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big) \Big| s_0 = s\right] .$$

It is well-known that for every MDP, there exists at least one optimal deterministic policy. Value iteration, policy iteration, and linear programming are well-known methods for deriving an optimal policy given the full MDP model.

In reinforcement learning, the learner interacts with an unknown process assumed to be an MDP and typically observes the state of the process and the immediate reward at every step, however P and R are not accessible. The goal is to gradually learn an optimal policy using the experience collected 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, thus experience comes in the form of (s, a, r, s') tuples. In many cases, it is further assumed that the learner has the ability to reset the process to any arbitrary state s [9]. 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 policies from samples [9], [10].

III. Reinforcement Learning as Classification

A key distinction among reinforcement learning algorithms relies on whether they are based on representing value functions, policies, or both. Value-function-based algorithms have received much criticism in recent years due to difficulties associated with the estimation and representation of value functions. Many learning problems of interest exhibit non-linear and non-smooth value functions that can hardly be compactly represented. On the other hand, advocates of direct policy learning have employed parametric representations that differ little from their value-function-representation counterparts. Most of them rely on representations of stochastic policies which take the form of a softmax over a parametric real-valued function (similar to a value function).

A recent trend in policy learning [1]–[7] attempts to exploit the generalization abilities of modern classification technology under the assumption that optimal or good deterministic policies for real learning problems are not arbitrarily complex, but rather exhibit a high degree of structure. It should, therefore, be plausible to learn a good policy using only a small set of training data consisting of selected states over the state space labeled with the actions that are deemed to be best in those states. To illustrate the learning process under such representations, we briefly review the *Rollout Classification Policy Iteration* (RCPI) algorithm [1]. The key idea behind RCPI is to cast the problem of policy learning as a classification problem. Finding a good policy becomes equivalent to finding a classifier that generalizes well over the state space and 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

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s') ,$$

provides such a measure given a fixed base policy π; the action that maximizes Q_π in state s is a "good" action in that state, whereas any action with smaller Q_π value is a "bad" one. The policy π' formed in this manner

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s,a)$$

is guaranteed to be at least as good as π, if not better. A training set for π' 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* [9] 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 actions according to policy π thereafter up to a certain horizon H, and computing the total discounted reward along this trajectory. The observed total discounted reward is averaged over a number of rollouts to yield an accurate estimate of $Q_\pi(s,a)$. Thus, using a sufficient amount of rollouts, it is possible to create a training example of the improved policy in state s. A collection of such examples over a finite set of states forms a valid training set for the improved policy π' over any base policy π.

The goal of the learner is not only to improve a policy, but rather to find a good or optimal policy, therefore RCPI employs an approximate policy iteration scheme as described

Algorithm 1 Rollout Classification Policy Iteration (RCPI)

```
Input: policy $\pi_0$, trials $K$, horizon $H$

$k \leftarrow -1$
repeat
  $k \leftarrow k+1$
  select the rollout states $S_{k+1} \subseteq \mathcal{S}$
  $T_{k+1} = \varnothing$
  for (each $s \in S_{k+1}$) do
    for (each $a \in \mathcal{A}$) do
      estimate $Q_{\pi_k}(s,a)$ using $K$ rollouts of length $H$
    end for
    if (a dominating action $a^*$ exists in $s$) then
      $T_{k+1} = T_{k+1} \cup \{(s,a^*)^+\}$ for dominating actions
      $T_{k+1} = T_{k+1} \cup \{(s,a)^-\}$ for dominated actions
    end if
  end for
  $\pi_{k+1}$ = TRAINCLASSIFIER($T_{k+1}$)
until ($\pi_{k+1}$ is not better than $\pi_k$)
return $\pi_k$
```

in Algorithm 1. At each iteration, a new policy/classifier is produced using training data obtained by rolling out the previous policy on a generative model of the process. Beginning with any initial policy π_0, at each iteration k a training set over a subset of states S_k is formed by querying the rollout procedure to identify dominating actions in the states of S_k. 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 empirical expected total discounted reward from D obtained from multiple runs is used as the policy performance criterion.

The RCPI algorithm yielded promising results in the pendulum and the bicycle domains using Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs) as classifiers. A similar algorithm by Fern et al. [2], [3] yielded satisfying results in seven planning domains mainly from the AIPS-2000 planning competition using Decision Lists as the underlying classifier. Both studies have identified sensitivities related to the distribution of states in S_k, however, none of them have examined closely the structure of the learned policy π_k and the possibility of exploiting this structure for guiding the selection of states in S_{k+1}. The typical choice is to pick states uniformly from the state space. Our work takes a first step in this direction by considering classifiers that reveal the internal structure of policy π_k, in particular the boundaries between different action choices over the state space. This information can be used to guide the selection of states in S_{k+1}, given that in typical control learning problems each policy improvement step refines these boundaries and therefore it is important to probe for dominating actions in states around these boundaries.

IV. Directed Exploration of Policy Space

Modern classification technology has delivered impressive results in a variety of learning domains, nevertheless most classification approaches, such as decision trees, neural networks, and nearest neighbors, deliver classifiers that may perform superbly, but look like a black box to an external observer. The classification function they implement can hardly be described in a compact intuitive way and furthermore the boundaries between classes cannot be easily and analytically identified, unless one probes the entire input space of the classifier using a fine grid only to sample these boundaries. There is however a distinct exception to this rule; Support Vector Machines (SVM) deliver classifiers whose boundaries between classes are compactly described by the support vectors and the hyperplane that separates them is analytically defined. This observation gave rise to the idea that, if SVMs are used to represent policies in RCPI, then there should be a way to guide the selection of the next set of rollout states around the action boundaries in the current policy.

In most reinforcement learning problems, there are more than two action choices in each state and therefore the resulting problem in RCPI is a multi-class classification problem. It has been suggested that in this context, it is more effective to adopt a binary formulation, whereby the learned policy is represented by an ensemble of binary classifiers [11]; each classifier corresponds to a single action, is trained against all other actions, and captures the parts of the state space where this action dominates over the other actions. The final decision of the represented policy could be any action that is reported as positive by the corresponding binary classifier, since all such actions are considered good. Despite the requirement for manually resolving ties between actions, this scheme offers much better representational accuracy and matches the way SVMs address multi-class classification.

Our analysis focuses on the C-SVM classifier [12]–[14]. Given a set of training data $\big\{(x_1,y_1),(x_2,y_2),\ldots,(x_l,y_l)\big\}$, where $y_i \in \{-1,1\}$, a hyperplane is constructed in a high-dimensional feature space to separate the two classes. The n-dimensional input vector x is mapped onto an N-dimensional feature vector through the choice of an N-dimensional function $\phi : \mathbb{R}^n \mapsto \mathbb{R}^N$. There no need to know the $\phi(x)$ function explicitly. Instead, what is really needed to proceed is the kernel function $k(x_i,x_j)$, which is defined as the dot product between two N-dimensional feature vectors, that is $k(x_i,x_j) = \phi(x_i)\cdot\phi(x_j)$. The C-SVM primal problem formulation for deriving the classifier is the following:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}w^\top w + C\sum_{i=1}^{\ell}\xi_i \\ \text{subject to} \quad & y_i\big(w^\top\phi(x_i)+b\big) \ge 1-\xi_i,\ i=1,\ldots,\ell \\ & \xi_i \ge 0,\ i=1,\ldots,\ell \end{aligned}$$

where $C > 0$. The dual problem formulation is the following:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j y_i y_j k(x_i,x_j) - \sum_{i=1}^{\ell}\alpha_i \\ \text{subject to} \quad & 0 \le \alpha_i \le C,\ i=1,\dots,m \\ & \sum_{i=1}^{\ell}\alpha_i y_i = 0 \end{aligned}$$

The α_i coefficients in this dual quadratic optimization problem are nonzero only for certain x_i's (the support vectors), which also define the separating hyperplane equation for the classification problem:

$$f(x) = \sum_{i=1}^{\ell} y_i\alpha_i k(x_i,x) + b = 0\ .$$

Any point x in the input space is classified using this equation. If $f(x)$ yields a positive number, x is classified in the $+1$ class, otherwise it is classified in the -1 class.

When SVMs are used to represent policies, the input space is the state space of the process and the support vectors represent some key states found in the training set that essentially define the boundary between different action choices (different classes). As discussed above, an attempt to improve the currently represented policy will certainly have to probe the states around this boundary. One way to find such states is to look on the line that goes through a support vector and is perpendicular (normal) to the separating boundary in the input space. The normal to the boundary for points x in the input space lying on the boundary is given [15] by

$$g(x) = \nabla_x f(x) = \nabla_x\left(\sum_{i=1}^{\ell} y_i\alpha_i k(x_i,x) + b\right)$$

For the polynomial kernel

$$k_p(x',x) = (x'\cdot x + r)^d, \quad d \ge 1, \quad r \in \mathbb{R},$$

we have

$$g(x) = d\sum_{i=1}^{\ell} y_i\alpha_i (x_i\cdot x + r)^{d-1}x_i\ ,$$

whereas for the Gaussian (radial basis functions) kernel

$$k_g(x',x) = \exp\big(-\beta \parallel x' - x \parallel^2\big), \quad \beta > 0,$$

we have

$$g(x) = 2\beta\sum_{i=1}^{\ell} y_i\alpha_i (x_i - x)k_g(x_i,x)\ .$$

Now that we have the direction, we seek to find the projections of the support vectors on the separating boundary, given that the support vectors lie at critical areas along the boundary. In particular, for each support vector x_i, we seek a point u_i that satisfies the following two conditions:

$$\begin{aligned} \lambda_i g(u_i) + u_i - x_i &= 0 \\ \sum_{j=1}^{\ell} y_j\alpha_j k(x_j,u_i) + b &= 0 \end{aligned}$$

This is a system of $n+1$ non-linear equations with unknowns u_i (vector) and λ_i (scalar) [15]. An efficient arithmetic solution to this system can be given by Powell's hybrid algorithm [16] using $[x_i, 0]^\top$ as an initial guess for $[u_i, \lambda_i]^\top$. Solutions u_i that fall outside the input space are discarded. Given a valid u_i, the unit (normalized) vector g_i at point u_i which is perpendicular to the separating boundary is $g_i = g(u_i)/||g(u_i)||$. A new input point $z_i(d)$ along this line at distance d from u_i will be $z_i(d) = u_i + dg_i$, where $d \in \mathbb{R}$.

Given the above derivation and any policy π_k represented as an SVM with support vectors $s_1^k, s_2^k, \dots, s_{l_k}^k$, it is straightforward how to select a new set of states around the separating boundary covering a wide enough zone to accommodate for possible mistakes of the policy/classifier π_k in identifying correctly the boundary between different action choices. Our choice is to take m_i^k points normally distributed on the line $g_i^k(d)$ with the normal distribution being centered at the projection u_i^k of s_i^k on the boundary having variance $\sigma_i^k = |s_i^k - u_i^k|$. More specifically, the next set of rollout states (probes) is

$$S_{k+1} = \big\{u_i^k + d_{ij}g_i^k : i=1\dots l_k,\ j=1\dots m_i^k,\ d_{ij}\sim\mathcal{N}(0,\sigma_i^k)\big\}.$$

The actual value of each m_i^k is proportional to σ_i^k and is chosen so that the total number M of rollout states remains constant between iterations. Note that the first iteration begins with a purely random, but deterministic, policy and therefore S_1 cannot be formed in the way it is described here, since there is no SVM classifier that represents π_0. S_1 is simply a uniformly random selection of states from the entire state space to guarantee full coverage at the beginning[1].

Probing a particular state s for the improved policy over a base policy π boils down to estimating the Q_π values for all actions in that state using rollouts and identifying the dominating action(s) (if any). Say that these estimated values are $\hat{Q}_\pi(s,a)$, $a \in \mathcal{A}$. In order to include state s in the training set, we need to identify at least one dominating action, whose value exceeds significantly the value of some other action. To quantify this difference we use the ΔQ quantity:

$$\begin{aligned} \Delta Q(s,a) &= \frac{\max_{a'\in\mathcal{A}}\{\hat{Q}_\pi(s,a')\} - \hat{Q}_\pi(s,a)}{\big|\max_{a'\in\mathcal{A}}\{\hat{Q}_\pi(s,a')\}\big|} \\ \Delta Q(s) &= \max_{a\in\mathcal{A}} \Delta Q(s,a) \end{aligned}$$

If $\Delta Q(s) > \epsilon$, then any action $a^* = \arg\max_{a'\in\mathcal{A}}\{\hat{Q}_\pi(s,a')\}$ that maximizes $\hat{Q}_\pi$ in state s is considered *dominating* and a pair (s,a^*) is inserted in the training set as a positive $(+)$ example. Any action a with $\Delta Q(s,a) > \epsilon$ is considered *dominated* and a pair (s,a) is inserted in the training set as a negative $(-)$ example. Note that some actions may be neither dominating, nor dominated; no training data will be produced for such actions. States with $\Delta Q(s) \le \epsilon$ do not yield any training data at all. This simple dominance criterion is sufficient in most cases for dealing with estimation

[1]Without domain knowledge, uniform sampling is the only option to guarantee that no part will remain initially unexplored. In certain cases, domain knowledge could be used to perform a focused non-uniform sampling.

Algorithm 2 Directed RCPI (DRCPI)

```
Input: policy $\pi_0$, attempts $L$, trials $K$, horizon $H$, threshold $\epsilon$, size $U$, points $M$

$k \leftarrow -1$
repeat
  $k \leftarrow k+1$
  $l \leftarrow 0$
  repeat
    $l \leftarrow l+1$
    if ($k = 0$) then
      $S_{k+1}$ = a uniformly random subset of $\mathcal{S}$ of size $U$
    else
      $s_1^k, s_2^k, \ldots, s_{l_k}^k \leftarrow$ support vectors in $\pi_k$
      $u_1^k, u_2^k, \ldots, u_{l_k}^k \leftarrow$ projections of $s_i^k$ on boundary
      $g_1^k, g_2^k, \ldots, g_{l_k}^k \leftarrow$ perpendicular directions at $u_i^k$
      $\sigma_1^k, \sigma_2^k, \ldots, \sigma_{l_k}^k \leftarrow$ variances of sampling at $u_i^k$
      $m_1^k, m_2^k, \ldots, m_{l_k}^k \leftarrow$ num of samples, $\sum_i m_i^k = M$
      $S_{k+1} = \{u_i^k + d_{ij} g_i^k : i=1{:}l_k, j=1{:}m_i^k, d_{ij} \sim \mathcal{N}(0, \sigma_i^k)\}$
    end if
    $T_{k+1} = \varnothing$
    for (each $s \in S_{k+1}$) do
      for (each $a \in \mathcal{A}$) do
        estimate $Q_{\pi_k}(s,a)$ using $K$ rollouts of length $H$
      end for
      if ($\Delta Q(s) > \epsilon$) then
        $T_{k+1} = T_{k+1} \cup \{(s,a^*)^+\}$ for dominating actions
        $T_{k+1} = T_{k+1} \cup \{(s,a)^-\}$ for dominated actions
      end if
    end for
    $\pi_{k+1}$ = TRAINCLASSIFIER($T_{k+1}$)
  until ( ($\pi_{k+1}$ is better than $\pi_k$) or ($l = L$) )
until ($\pi_{k+1}$ is not better than $\pi_k$)
return $\pi_k$
```

noise. A more sophisticated criterion could be based on using Hoeffding's inequality, the way it is used by Dimitrakakis and Lagoudakis [5]; this criterion could be also used to terminate the rollout trials early, if domination has been established with high confidence. We should stress that our approach requires the estimation of action values at few isolated points (rollout states) only; nowhere does it require a full value function over the entire state-action space. Therefore, the known issues of value function approximation are not applicable. These required values are obtained as unbiased estimates from Monte-Carlo simulation (policy rollouts) and can be estimated to any desired accuracy provided sufficient simulation time.

To summarize, given a policy π_k at iteration k represented as an SVM, the support vectors in π_k are used to select the new subset of states S_{k+1} at which the improved policy π_{k+1} will be probed. Each probe will yield pairs of data in the training set T_{k+1}, if domination is detected. Once the training set is formed, a new classifier is trained to represent π_{k+1} and the process repeats from the beginning. Clearly, there is no guarantee for monotonic policy improvement in this iterative scheme, however the chances can be increased by repeating some iteration if no improvement was achieved; since each iteration is randomized, it is likely that another run may produce better results. Therefore, if policy π_{k+1} is not better than π_k (tested through simulation), iteration k is repeated for a maximum of L attempts until an improved policy is found. If all attempts are exhausted without improvement, the algorithm terminates. The entire Directed RCPI (DRCPI) algorithm is shown in Algorithm 2.

V. Benchmark Domains

We chose to study the proposed algorithm on three standard domains in reinforcement learning: inverted pendulum, mountain car, and acrobot. The first two problems are defined on two-dimensional continuous state spaces, therefore they are appropriate for visualization and inspection, whereas the third is defined on a four-dimensional space and there is appropriate for verifying that learning is indeed directed to important parts of the state space.

A. 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 (-50 Newtons), right force RF ($+50$ Newtons), or no force NF (0 Newtons). All three actions are noisy; Gaussian noise with $\mu = 0$ and $\sigma^2 = 10$ is added to the chosen action. The state space of the problem 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 [17] and depend on the current state and the current (noisy) control u:

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

where $g = 9.8\text{m/s}^2$ is the gravity constant, $m = 2.0$kg is the mass of the pendulum, $M = 8.0$kg is the mass of the cart, $l = 0.5$m is the length of the pendulum, and $\alpha = 1/(m+M)$. The control step is 0.1 seconds and the control input is kept constant between successive steps. A reward of 1 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 $\pi/2$ in absolute value signals the end of the episode and a reward of 0. The discount factor of the process is set to 0.95.

B. 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 enough energy 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). Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.2$ is added to the chosen action. 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 discrete-time nonlinear dynamics of the system [10] and depend on the current state $(x(t), \dot{x}(t))$ and the current control $u(t)$:

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

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 bounds of the position x, the velocity $\dot{x}$ is set to zero. Reward of 0 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 of position, it has reached the goal; the episode ends successfully and a reward of 1 is given. The discount factor of the process is set to 0.99.

C. Acrobot

The *acrobot* is two-link robot arm which is attached to a fixed point at one end. The goal is to swing the arm around the fixed point, so that the other end reaches a height which is one link's length higher than the fixed point. Torque τ can be applied to the elbow joint only and the shoulder joint is free to swing around the fixed point. Three levels of torque are allowed: positive $(+1)$, negative (-1), or no torque (0). Gaussian noise $(\mu = 0,\ \sigma^2 = 0.2)$ is added to the chosen action. The system is described by four state variables: shoulder angular position $\theta \in [-\pi, \pi]$, shoulder angular velocity $\dot{\theta} \in [-4\pi, 4\pi]$, elbow angular position $\psi \in [-\pi, \pi]$, elbow angular velocity $\dot{\psi} \in [-9\pi, 9\pi]$. The transitions are governed by the nonlinear dynamics of the system [10] and depend on the current state $(\theta, \dot{\theta}, \psi, \dot{\psi})$ and the current control τ:

$$\begin{aligned} \ddot{\theta} &= -\frac{(d_2\ddot{\psi} + \phi_1)}{d_1} \qquad \ddot{\psi} = \frac{(\tau + \frac{d_2}{d_1}\phi_1 - \phi_2)}{(m_2\ell_{c2}^2 + I_2 - \frac{d_2^2}{d_1})} \\ d_1 &= m_1\ell_{c1}^2 + m_2(\ell_1^2 + \ell_{c2}^2 + 2\ell_1\ell_{c2}\cos\psi) + I_1 + I_2 \\ d_2 &= m_2(\ell_{c2}^2 + \ell_1\ell_{c2}\cos\psi) + I_2 \\ \phi_1 &= -m_2\ell_1\ell_{c2}\dot{\psi}^2\sin\psi - 2m_2\ell_1\ell_{c2}\dot{\theta}\dot{\psi}\sin\psi + \\ &\qquad (m_1\ell_{c1} + m_2\ell_1)g\cos(\theta - \frac{\pi}{2}) + \phi_2 \\ \phi_2 &= m_2\ell_{c2}g\cos(\theta + \psi - \frac{\pi}{2}) \end{aligned}$$

where $g = 9.8\text{m/s}^2$ is the gravity constant, $m_1 = m_2 = 1\text{Kg}$ are the masses of the links, $l_1 = l_2 = 1\text{m}$ are the lengths of links, $l_{c1} = l_{c2} = 0.5\text{m}$ are the lengths to the center of mass of the links, and $I_1 = I_2 = 1$ are the moments of inertia of the links. The control step is 0.05 seconds and the control input is kept constant between successive steps. The angular velocities are limited so that they stay within their bounds. A reward of -1 is given at all time steps as long as the goal has not been reached. A reward of 0 is given for reaching the goal. The discount factor of the process is set to 0.98.

VI. Experimental Results

We applied our approach to the three benchmark domains described above to test its effectiveness. All policies were represented using the C-SVM classifier $(C = 10)$ either with the radial basis function (RBF) kernel $(\beta = 1)$ or with the polynomial (POLY) kernel ($d = 2$ for the inverted pendulum and $d = 4$ for the mountain car and the acrobot, $r = 1$) and were implemented using the libSVM package [18]. Note that we did not try to optimize the parameters of the classifiers, since our ultimate goal is to employ simple off-the-shelf classification solutions for the benefit of reinforcement learning. Powell's hybrid method for solving the non-linear system was taken from GSL [19]. The parameters of our algorithm were tested selectively within their range to find operational values. The values used in the experiments reported here were $L = 4$, $K = 50$, $H = 100$, and $\epsilon = 0.2$ for all domains; $U = 200$, $M = 50$ for the inverted pendulum and the mountain car; $U = 800$, $M = 100$ for the acrobot. The initial policy π_0 was a random deterministic policy in all domains. Each dimension of the state space in all domains was scaled to $[-1, +1]$ to account for magnitude differences in computing perpendicular directions.

Figure 1 shows a typical run with the RBF kernel on the inverted pendulum domain. Data are displayed over the 2-dimensional state space (the horizontal axis is the angle and the vertical axis is the angular velocity). A fully-balancing policy was found in just two iterations (individual improvement attempts are not shown). The first iteration delivered a rather poor policy, while the third iteration did not improve the already fully-balancing policy. Notice that, after the initial uniform distribution, the rollout states are positioned mostly around the boundary.

Figure 2 shows a typical run with the RBF kernel on the mountain car domain. Once again, data are displayed over the 2-dimensional state space (the horizontal axis is the position and the vertical axis is the velocity). A successful exiting policy is found in the first iteration, but the policy in the second iteration was better, as it could exit in fewer steps. The third iteration did not improve performance and the algorithm terminated.

Table I shows collective results and statistics from multiple runs. Each row represents averages from 100 different executions with the same settings, but with different random seeds. These data offer a comparison between RCPI and Directed RCPI (DRCPI). For fairness, we added the multiple improvement attempts in RCPI, so that the only difference between the two algorithms is the choice of the rollout states in terms of quantity and location. The Simulation tab shows the amount of simulation steps needed for each run, while the Rollouts tab shows the number of rollouts executed in each run. The Attempts tab shows the number of improvement attempts (the number of iterations is less than or equal to that) before termination and the Time tab shows the real time (seconds) taken by each run. Finally, the Return and Success tabs show the total expected discounted reward and the success

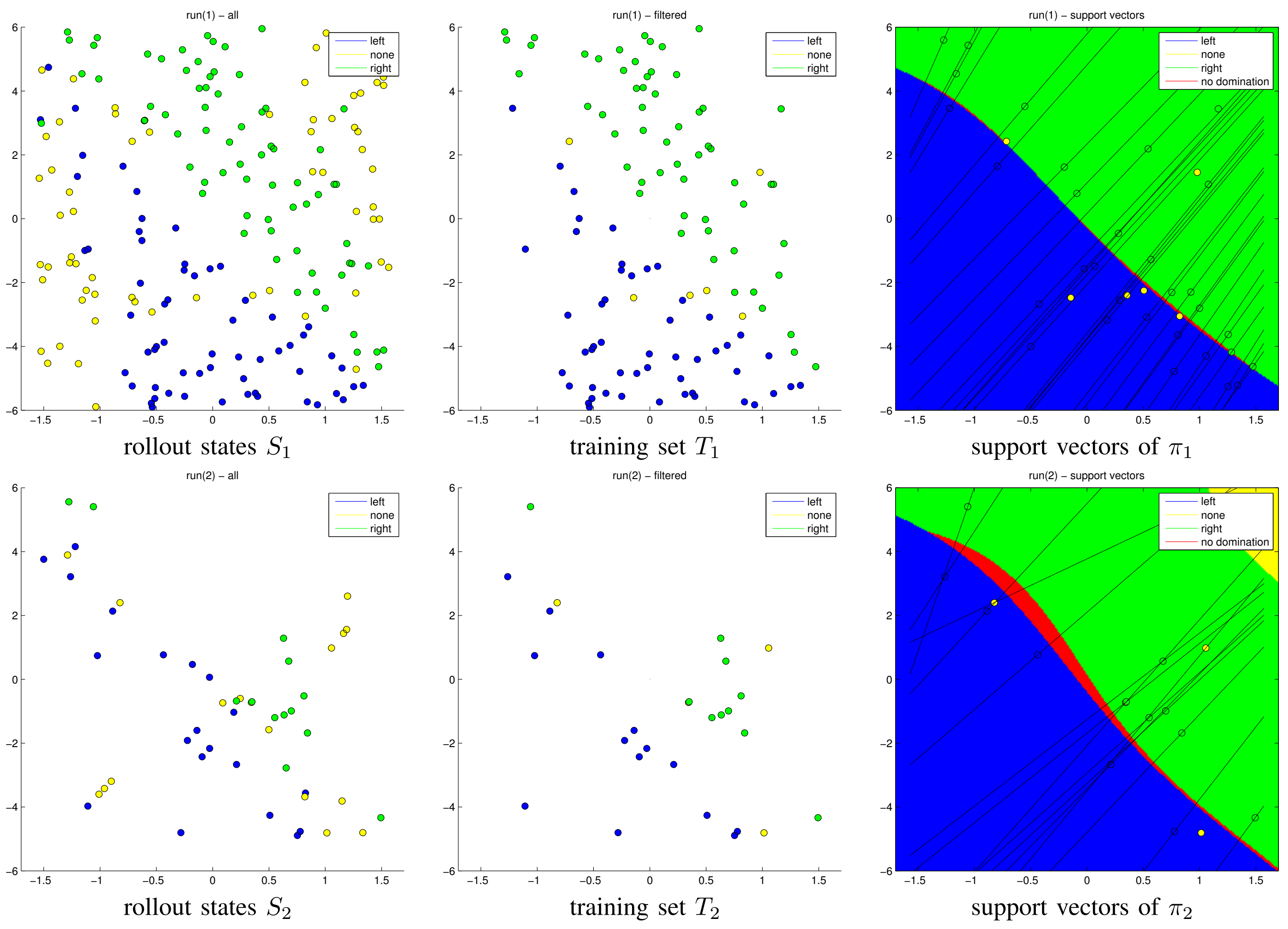

Fig. 1. Inverted Pendulum: two iterations of the proposed algorithm (policies π_1 and π_2 are shown in color underneath the support vectors).

rate respectively of the final learned policy. A successful run corresponds to 3000 steps (5 minutes) of balancing in the inverted pendulum domain, exiting from the valley in less than 3000 steps in the mountain car domain, and reaching the goal in less than 3000 steps in the acrobot domain. Clearly, DRCPI yields significant savings in terms of rollouts and simulation compared to RCPI, while delivering policies of almost equal performance, except in the mountain car domain where the DRCPI policies suffer a loss. Despite the extra computations required by DRCPI, there were also significant savings in real computation time, which implies that the amount of simulation is the main cost factor. These findings indicate that directed exploration of policy space can lead to significant computational performance improvements with little or no deterioration of learning performance.

VII. Conclusion

We have presented a scheme for directing classifier-based policy search. Our scheme exploits the properties of support vector classifiers to identify the critical parts of the state space where probing should focus to improve an existing policy. Our results indicate significant savings in simulation time (rollouts) and generation of very good policies in only a few iterations.

References

[1] M. G. Lagoudakis and R. Parr, “Reinforcement learning as classification: Leveraging modern classifiers,” in *Proceedings of the 20th International Conference on Machine Learning (ICML)*, Washington, DC, USA, 2003, pp. 424–431.

[2] A. Fern, S. Yoon, and R. Givan, “Approximate policy iteration with a policy language bias,” *Advances in Neural Information Processing Systems*, vol. 16, no. 3, 2004.

[3] ——, “Approximate policy iteration with a policy language bias: Solving relational Markov decision processes,” *Journal of Artificial Intelligence Research*, vol. 25, pp. 75–118, 2006.

[4] J. Langford and B. Zadrozny, “Relating reinforcement learning performance to classification performance,” in *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, Bonn, Germany, 2005, pp. 473–480.

[5] C. Dimitrakakis and M. Lagoudakis, “Rollout sampling approximate policy iteration,” *Machine Learning*, vol. 72, no. 3, pp. 157–171, 2008.

[6] E. Rachelson and M. G. Lagoudakis, “On the locality of action domination in sequential decision making,” in *Proceedings of the 11th International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2010.

[7] A. Lazaric, M. Ghavamzadeh, and R. Munos, “Analysis of a classification-based policy iteration algorithm,” in *International Conference on Machine Learning (ICML)*, 2010, pp. 607–614.

[8] C. W. Anderson, “Approximating a policy can be easier than approximating a value function,” Department of Computer Science, Colorado State University, Tech. Rep. CS-00-101, 2000.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

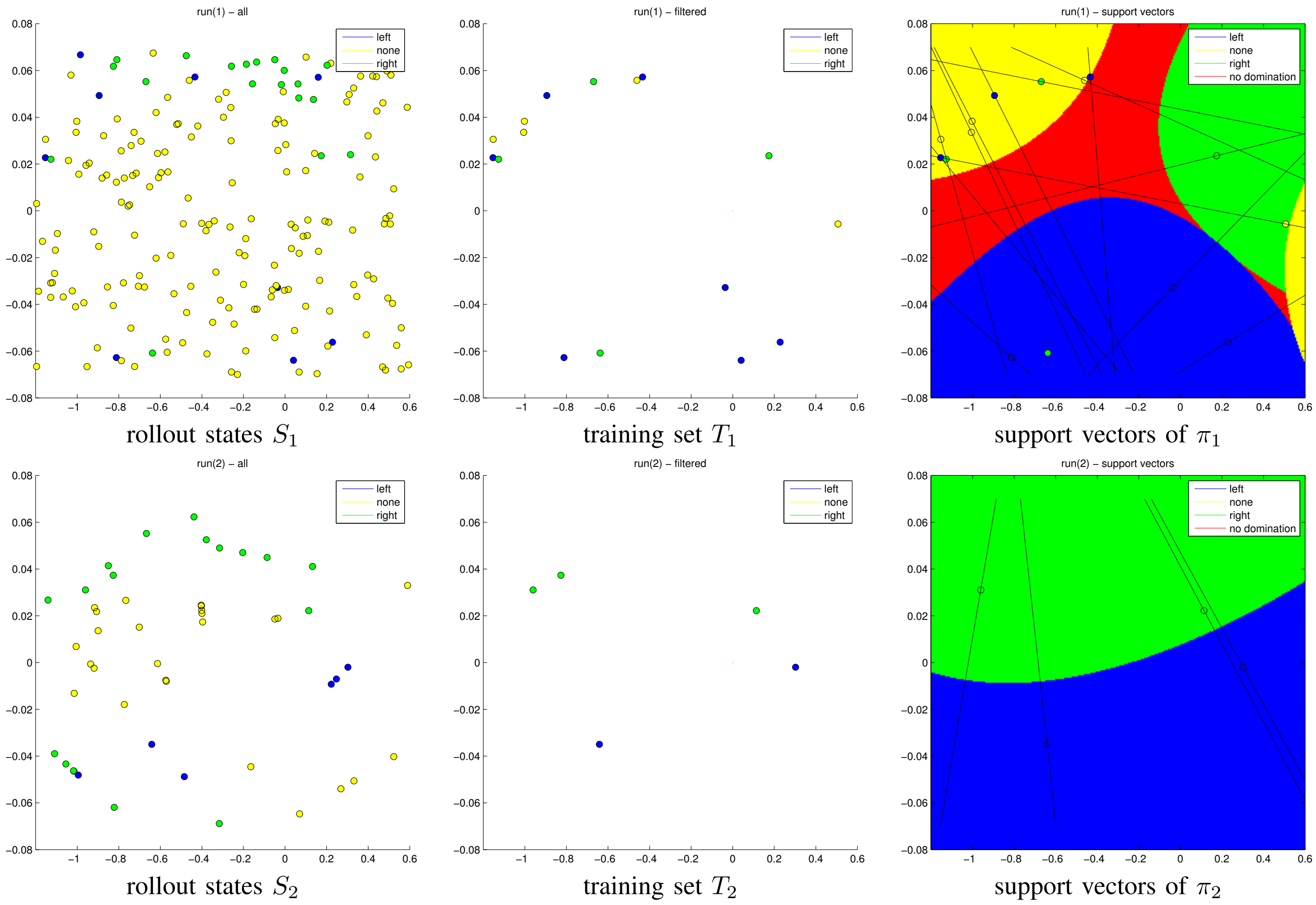

Fig. 2. Mountain Car: two iterations of the proposed algorithm (policies π_1 and π_2 are shown in color underneath the support vectors).

TABLE I
COLLECTIVE RESULTS (100 RUNS) AND COMPARISON OF DRCPI AND RCPI IN TERMS OF COMPUTATIONAL AND LEARNING PERFORMANCE.

	Kernel	Algorithm	States	Simulation	Rollouts	Attempts	Time	Return	Success
Pendulum	RBF	DRCPI	200/50	2122569	469	6.34	33.1	20	92.88%
	RBF	RCPI	200	4182303	1336	6.68	86.6	20	95.78%
	POLY	DRCPI	200/50	1867523	471	6.47	26.8	20	92.70%
	POLY	RCPI	200	4255602	1352	6.76	58.2	20	94.91%
Mountain	RBF	DRCPI	200/50	6541460	585	7.39	29.7	0.29	88.45%
	RBF	RCPI	200	14714292	1690	8.45	65.1	0.42	98.89%
	POLY	DRCPI	200/50	6993685	621	7.55	26.2	0.28	83.68%
	POLY	RCPI	200	15087372	1734	8.67	58.9	0.42	97.97%
Acrobot	RBF	DRCPI	800/100	15706067	2339	6.25	144.1	-49.54	47.40%
	RBF	RCPI	800	30294939	4800	6.00	452.5	-49.60	46.38%
	POLY	DRCPI	800/100	15738436	2314	6.80	133.6	-49.59	45.42%
	POLY	RCPI	800	35965883	5720	7.15	443.7	-49.47	39.60%

[10] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[11] I. Rexakis and M. G. Lagoudakis, “Classifier-based policy representation,” in *Proceedings of the 7th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2008, pp. 91–98.

[12] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT)*, 1992, pp. 144–152.

[13] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.

[15] K. F. Riley, M. P. Hobson, and S. J. Bence, *Mathematical Methods for Physics and Engineering*. Cambridge University Press, 2006.

[16] M. Powell, “A Fortran subroutine for solving systems of nonlinear algebraic equations,” in *Numerical Methods for Nonlinear Algebraic Equations*, P. Rabinowitz, Ed. Gordon and Breach, Science Publishers Ltd., 1970, ch. 7, pp. 115–161.

[17] H. O. Wang, K. Tanaka, and M. F. Griffin, “An approach to fuzzy control of nonlinear systems: Stability and design issues,” *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 1, pp. 14–23, 1996.

[18] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[19] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *Gnu Scientific Library: Reference Manual*, 2003.