Non-Parametric Policy Gradients: A Unified Treatment of Propositional and Relational Domains

Kristian Kersting KRISTIAN.KERSTING@IAIS.FRAUNHOFER.DE
Dept. of Knowledge Discovery, Fraunhofer IAIS, Schloss Birlinghoven, 53754 Sankt Augustin, Germany

Kurt Driessens KURT.DRIESSENS@CS.KULEUVEN.BE
Dept. of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract

Policy gradient approaches are a powerful instrument for learning how to interact with the environment. Existing approaches have focused on propositional and continuous domains only. Without extensive feature engineering, it is difficult – if not impossible – to apply them within structured domains, in which e.g. there is a varying number of objects and relations among them. In this paper, we describe a non-parametric policy gradient approach – called NPPG – that overcomes this limitation. The key idea is to apply Friedmann's gradient boosting: policies are represented as a weighted sum of regression models grown in an stage-wise optimization. Employing off-the-shelf regression learners, NPPG can deal with propositional, continuous, and relational domains in a unified way. Our experimental results show that it can even improve on established results.

1. Introduction

Acting optimally under uncertainty is a central problem of artificial intelligence. If an agents learns to act solely on the basis of the rewards associated with actions taken, this is called reinforcement learning (Sutton & Barto, 1998). More precisely, the agent's learning task is to find a policy for action selection that maximizes its reward over the long run.

The dominant reinforcement learning (RL) approach for the last decade has been the value-function approach. An agent uses the reward it occasionally receives to estimate a value-function indicating the expected value of being in a state or of taking an action in a state. The policy is represented only implicitly, for instance as the policy that selects in each state the action with highest estimated value. As Sutton et al. (2000) point out, the value function approach, however, has several limitations. First, it seeks to find deterministic policies, whereas in real world applications the optimal policy is often stochastic, selecting different actions with specific probabilities. Second, a small change in the value-function parameter can push the value of one action over that of another, causing a discontinuous change in the policy, the states visited, and overall performance. Such discontinuous changes have been identified as a key obstacle to establishing convergence assurances for algorithms following the value-function approach (Bertsekas & Tsitsiklis, 1996). Finally, value-functions can often be much more complex to represent than the corresponding policy as they encode information about both the size and distance to the appropriate rewards. Therefore, it is not surprising that so called policy gradient methods have been developed that attempt to avoid learning a value function explicitly (Williams, 1992; Baxter et al., 2001; Konda & Tsitsiklis, 2003). Given a space of parameterized policies, they compute the gradient of the expected reward with respect to the policy's parameters, move the parameters into the direction of the gradient, and repeat this until they reach a local optimum. This direct approximation of the policy overcomes the limitations of the value function approach stated above. For instance, Sutton et al. (2000) show convergence even when using function approximation.

Current policy gradient methods have focused on propositional and continuous domains assuming the environment of the learning agent to be representable as a vector-space. Nowadays, the role of structure and relations in the data, however, becomes more and

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

more important (Getoor & Taskar, 2007): information about one object can help the agent to reach conclusions about other objects. Such domains are hard to represent meaningfully using a fixed set of features. Therefore, relational RL approaches have been developed (Džeroski et al., 2001), which seek to avoid explicit state and action enumeration as – in principle – traditionally done in RL through a symbolic representation of states and actions. Existing relational RL approaches, however, have focused on value functions and, hence, suffer from the same problems as their propositional counterparts as listed above.

In this paper, we present the first model-free policy gradient approach that deals with relational and propositional domains. Specifically, we present a non-parametric approach to policy gradients, called NPPG. Triggered by the observation that finding many rough rules of thumb of how to change the way to act can be a lot easier than finding a single, highly accurate policy, we apply Friedmann's (2001) gradient boosting. That is, we represent policies as weighted sums of regression models grown in a stage-wise optimization. Such a functional gradient approach has recently been used to efficiently train conditional random fields for labeling (relational) sequences using boosting (Dietterich et al., 2004; Gutmann & Kersting, 2006) and for policy search in continuous domains (Bagnell & Schneider, 2003). In contrast to the supervised learning setting of the sequence labeling task, feedback on the performance is received only at the end of an action sequence in the policy search setting. The benefits of a boosting approach to functional policy gradients are twofold. First, interactions among states and actions are introduced only as needed, so that the potentially infinite search space is not explicitly considered. Second, existing off-the-shelf regression learners can be used to deal with propositional and relational domains in a unified way. To the best of the authors' knowledge, this is the first time that such a unified treatment is established. As our experimental results show, NPPG can even significantly improve upon established results in relational domains.

We proceed as follows. We will start off by reviewing policy gradients and their mathematical background. Afterwards, we will develop NPPG in Section 3. In Section 4, we will present our experimental results. Before concluding, we will touch upon related work.

2. Policy Gradients

Policy gradient algorithms find a locally optimal policy starting from an arbitrary initial policy using a gradient ascent search through an explicit policy space. Consider the standard RL framework (Sutton & Barto, 1998), where an agent interacts with a Markov Decision Process (MDP). The MDP is defined by a number of states $s \in \mathcal{S}$, a number of actions $a \in \mathcal{A}$, state-transition probabilities $\delta(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ that represent the probability that taking action a in state s will result in a transition to state s' and a reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. When the reward function is nondeterministic, we will use the expected rewards $R(s, a) = E_{s,a}[r(s, a)]$, where $E_{s,a}$ denotes the expectation over all states s and actions a. The state, action, and reward at time t are denoted as $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ and $r_t (or\ R_t) \in \mathbb{R}$.

The agent selects which action to execute in a state following a policy function $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Current policy gradient approaches assume that the policy function π is parameterized by a (weight) vector $\theta \in \mathbb{R}^n$ and that this policy function is differentiable with respect to its parameters, i.e., that $\frac{\partial \pi(s,a,\theta)}{\partial \theta}$ exists. A common choice is a Gibbs distribution based on a linear combination of features:

$$\pi(s, a) = e^{\Psi(s,a)} / \left(\sum_b e^{\Psi(s,b)} \right) , \qquad (1)$$

where the potential function $\Psi(s, a) = \theta^T \phi_{sa}$ with ϕ_{sa} the feature vector describing state s and action a. This representation guarantees that the policy specifies a probability distribution independent of the exact form of the function Ψ. Choosing a parameterization, creates an explicit policy space $\mathbb{R}^n$ with n equal to size of the parameter vector θ. This space can be traversed by an appropriate search algorithm.

Policy gradients are computed w.r.t. a function ρ that expresses the value of a policy in an environment, $\rho(\pi) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi(s, a) \cdot Q^\pi(s, a)$, where Q^π is the usual (possibly discounted) state-action value function, e.g., $Q^\pi(s, a) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i} | s_t = s, a_t = a \right]$, and $d^\pi(s)$ is the (possibly discounted) stationary distribution of states under policy π. We assume a fully ergodic environment so that $d^\pi(s)$ exists and is independent of any starting state s_0 for all policies.

A property of ρ for both average reward reinforcement learning and episodic tasks with a fixed starting state s_0 is that (Sutton et al., 2000, Theorem 1)

$$\frac{\partial \rho}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s, a)}{\partial \theta} \cdot Q^\pi(s, a) . \qquad (2)$$

Important to note is that this gradient does not include the term $\frac{\partial d^\pi(s)}{\partial \theta}$. Eq. (2) allows the computation of an approximate policy gradient through exploration. By sampling states s through exploration of the environment following policy π, the distribution $d^\pi(s)$ is

automatically represented in the generated sample of encountered states. The sum $\sum_a \frac{\partial \pi(s,a)}{\partial \theta} Q^\pi(s,a)$ then becomes an unbiased estimate of $\frac{\partial \rho}{\partial \theta}$ and can be used in a gradient ascent algorithm.

Of course, the value Q^π is unknown and must be estimated for each visited state-action pair either by using a Monte Carlo approach or by building an explicit representation of Q, although in this latter case care must be taken when choosing the parameterization of the Q-function (Sutton et al., 2000).

3. Non-Parametric Policy Gradients

A drawback of a fixed, finite parameterization of a policy such as in Eq. (1) is that it assumes each feature makes an independent contribution to the policy. Of course it is possible to define more features to capture combinations of the basic features, but this leads to a combinatorial explosion in the number of features, and hence, in the dimensionality of the optimization problem. Moreover, in continuous and in relational environments it is not clear at all which features to choose as there are infinitely many possibilities.

To overcome these problems, we introduce a different policy gradient approach based on Friedmann's (2001) gradient boosting. In our case, the potential function Ψ in the Gibbs distribution[1] of Eq. (1) is represented as a weighted sum of regression models grown in an stage-wise optimization. Each regression model can be viewed as defining several new feature combinations. The resulting policy is still a linear combination of features, but the features can be quite complex. Formally, gradient boosting is based on the idea of functional gradient ascent, which we will now describe.

3.1. Functional Gradient Ascent

Traditional gradient ascent estimates the parameters θ of a policy iteratively as follows. Starting with some initial parameters θ_0, the parameters θ_m in the next iteration are set to the current parameters plus the gradient of ρ w.r.t. to θ, i.e., $\theta_m = \theta_0 + \delta_1 + \ldots + \delta_m$ where $\delta_m = \eta_m \cdot \partial\rho/\partial\theta_{m-1}$ is the gradient multiplied by a constant η_m, which is obtained by doing a line search along the gradient. Functional gradient ascent is a more general approach, see e.g. (Friedman, 2001; Dietterich et al., 2004). Instead of assuming a linear parameterization for Ψ, it just assumes that Ψ will be represented by a linear combination of functions. Specifically, one starts with some initial function Ψ_0, e.g. based on the zero potential, and iteratively adds corrections $\Psi_m = \Psi_0 + \Delta_1 + \ldots + \Delta_m$. In contrast to the standard gradient approach, Δ_m here denotes the so-called functional gradient, i.e., $\Delta_m = \eta_m \cdot E_{s,a}\left[\partial\rho/\partial\Psi_{m-1}\right]$. Interestingly, this functional gradient coincides with what traditional policy gradient approach estimate, namely (2), as the following theorem says.

Theorem 3.1 (Functional Policy Gradient) *For any MDP, in either the average-reward or start-state formulation,*

$$E_{s,a}\left[\frac{\partial \rho}{\partial \Psi}\right] = \sum\nolimits_{s,a} d^\pi(s) \cdot \frac{\partial \pi(s,a)}{\partial \Psi} \cdot Q(s,a)\,. \quad (3)$$

This is a straightforward adaptation of Theorem 1 in (Sutton et al., 2000) and is also quite intuitive: the functional policy gradient indicates how we would like the policy to change in all states and actions in order to increase the performance measure ρ.

Unfortunately, we do not know the distribution $d^\pi(s)$ of how often we visit each state s under policy π. This is, however, easy to approximate from the empirical distribution of states visited when following π:

$$E_{s,a}\left[\frac{\partial \rho}{\partial \Psi}\right] \approx E_{s\sim\pi}\left[\sum\nolimits_a \underbrace{\frac{\partial \pi(s,a)}{\partial \Psi} \cdot Q(s,a)}_{=:f_m(s,a)}\right]\,, \quad (4)$$

where the state s is sampled according to π, denoted as $s \sim \pi$. We now have a set of traning examples from the distribution $d^\pi(s)$, so we can compute the value $f_m(s,a)$ of the functional gradient at each of the training data points for all[2] actions a applicable in s. We can then use these point-wise functional gradients to define a set of training examples $\{(s,a), f_m(s,a)\}$ and then train a function $f_m : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ so that it minimizes the squared error over the training examples. Although the fitted function f_m is not exactly the same as the desired functional gradient in Eq. (3), it will point in the same general direction assuming there are enough training examples. So, taking a step $\Delta_m = \eta_m \cdot f_m$ will approximate the true functional policy gradient ascent.

[1] Other distributions are possible but are subject to future research. For instance, it would be interesting to investigate modeling joint actions of multiple agents in relatioanl domains a long the lines of Guestrin et al. (2002).

[2] Here, an update is made for all actions possible in each state encountered irrespective of which action was actually taken. Alternatively, we can only make an update for the one action actually taken (Baxter et al., 2001). To compensate for the fact that some actions are selected more frequently than others, we divide by the probability of choosing the action, i.e., we use $\Delta_m(s,a)/\pi(s,a)$ as functional gradient training examples and do not run over all actions.

As an example, we will now derive the point-wise functional gradient of a policy parameterized as a Gibbs distribution (1). For the sake of readability, we denote $\Psi(s,a)$ as Ψ_a and $\Psi(s,b)$ as Ψ_b.

Proposition 3.1 *The point-wise functional gradient of ρ with respect to a policy parameterized as a Gibbs distribution* (1) *equals to $Q(s,a) \cdot \frac{\partial \pi(s,a)}{\partial \Psi}$ with*

$$\frac{\partial \pi(s,a)}{\partial \Psi(s',a')} = \begin{cases} \pi(s,a)(1-\pi(s,a)) & \text{if } s = s' \wedge a = a', \\ -\pi(s,a)\pi(s,a') & \text{if } s = s' \wedge a \neq a' \\ 0 & \text{otherwise,} \end{cases}.$$

To see this, consider the first case; the other cases can be derived in a similar way. Due to the Gibbs distribution, we can write $(\partial \pi(s,a))/(\partial \Psi(s,a)) =$

$$\frac{\partial}{\partial \Psi_a} \frac{e^{\Psi_a}}{\sum_b e^{\Psi_b}} = \frac{e^{\Psi_a} \cdot \sum_b e^{\Psi_b} - e^{\Psi_a} \cdot \sum_b \partial e^{\Psi_b}/\partial \Psi_a}{\left[\sum_b e^{\Psi_b}\right]^2} .$$

Assuming $(\partial \Psi_b)/(\partial \Psi_a) = 0$, i.e., Ψ_a and Ψ_b are independent, it holds $\sum_b \partial e^{\Psi_b}/\partial \Psi_a = e^{\Psi_a}$ and we can rewrite the state-action gradient as

$$= e^{\Psi_a} \frac{\left(\sum_b e^{\Psi_b} - e^{\Psi_a} \cdot \frac{\sum_b e^{\Psi_b}}{\sum_b e^{\Psi_b}}\right)}{\left[\sum_b e^{\Psi_b}\right]^2} = e^{\Psi_a} \frac{\left(1 - \frac{e^{\Psi_a}}{\sum_b e^{\Psi_b}}\right)}{\sum_b e^{\Psi_b}}$$

which simplifies – due to the definition of $\pi(s,a)$ – to

$$\frac{\partial \pi(s,a)}{\partial \Psi(s,a)} = \pi(s,a)(1-\pi(s,a)) .$$

The key point is the assumption $(\partial \Psi_b)/(\partial \Psi_a) = 0$. This actually means that we model Ψ with k functions f_k, one for each action a, i.e., $\Psi(s,a) = f_a(s)$. In turn, we estimate k regression models h_m^a. In the experiments, however, learning a single regression model h_m did not decrease performance so that we stick here to the conceptually easier variant of learning a single regression model h_m.

We call policy gradient methods that follow the outlined functional gradient approach *non-parametric policy gradients*, or NPPG for short. They are non-parametric because the number of parameters can grow with the number of episodes.

3.2. Gradient Tree Boosting

NPPG as summarized in Alg. 1 describes actually a family of approaches. In the following, we will develop a particular instance, called TreeNPPG, which uses regression tree learners to estimate h_m in line 6.

In TreeNPPG, the policy is represented by sums of regression *trees*. Each regression tree can be viewed as defining several new feature combinations, one corresponding to each path in the tree from the root to a leaf. The resulting policies still have the form of a linear combination of features, but the features can be quite complex. The trees are grown using a regression tree learner such as CART (Breiman et al., 1984), which in principle runs as follows. It starts with the empty tree and repeatedly searches for the best test for a node according to some splitting criterion such as weighted variance. Next, the examples $\mathcal{R}$ in the node are split into $\mathcal{R}_s$ (success) and $\mathcal{R}_f$ (failure) according to the test. For each split, the procedure is recursively applied, obtaining subtrees for the respective splits. As splitting criterion, we use the weighted variance on $\mathcal{R}_s$ and $\mathcal{R}_f$. We stop splitting if the variance in one node is small enough or a depth limit was reached. In leaves, the average regression value is predicted.

We propose to use regression tree learners because a rich variety of variants exists that can deal with finite, continuous and even relational data. Depending on the type of the problem domain at hand, one can instantiate the TreeNPPG algorithm by choosing the appropriate regression tree learner. We will give several examples in the following experimental section.

Algorithm 1: Non-Parametric Policy Gradient

1 Let Ψ_0 be the zero potential (the empty tree)
2 **for** $m = 1$ *to* N **do**
3 Choose a starting state s
4 Generate episodes $\mathcal{E}_m$ starting in s following the policy π_{m-1} with $\pi_{m-1}(s,a) = e^{\Psi_{m-1}(s,a)} / \left(\sum_b e^{\Psi_{m-1}(s,b)}\right)$
5 Generate functional gradient examples $\mathcal{R}_m = \{(s_i, a_{ij}), f_m(s_i, a_{ij})\}$ based on $\mathcal{E}_m$
6 Induce regression model f_m based on $\mathcal{R}_m$
7 Set potential to $\Psi_m = \Psi_{m-1} + \Delta_m$ where $\Delta_m = \eta_m \cdot h_m$ with local step size η_m
8 **return** *final potential* $\Psi = \Psi_0 + \Delta_1 + \ldots + \Delta_m$

4. Experimental Evaluation

Our intention is to investigate how well NPPG works. To this aim, we implemented it and investigated the following questions:

(Q1) Does TreeNPPG work and, if so, are there cases where it yields better results than current state-of-the-art methods? **(Q2)** Is TreeNPPG applicable across finite, continuous, and relational domains?

In the following, we will describe the experiments carried out to investigate the questions and their results.

(Q1) Blocks World: A Relational Domain

As a complex domain for empirical evaluation of TreeNPPG , we consider the well-known *blocks world* (Slaney & Thiébaux, 2001). To be able to compare the performance of TreeNPPG to other relational RL (RRL) systems, we adopt the same experimental environment as used for RRL by e.g. Driessens and Džeroski (2005). We learn in a world with 10 blocks and try to accomplish the $on(A, B)$ goal. This goal is parameterized and appropriate values for A and B are chosen at same the time as a starting state is generated. Thus, the reinforcement learn learns a single policy that stacks any two blocks. Although this is a relatively simple goal in a planning context, both RRL and our TreeNPPG algorithm use a model free approach and only learn from interactions with the environment. For the given setup this means that there are approximately 55.7 million reachable states[3] of which 1.44 million are goal states. The minimal number of steps to the goal is 3.9 on average. The percentage of states for other minimal solution sizes are given in Fig. 2. The agent only receives a reward of 1 if it reaches the goal in the minimal number of steps. The probability of receiving a reward using a random strategy is approximately 1.3%. To counter this difficulty of reaching any reward, we adopted the active guidance approach as proposed by Driessens and Dzeroski (2004), presenting the RL agent with 10% of expert traces during exploration in all experiments.

We apply TreeNPPG to this relational domain by simply employing the relational regression tree learner Tilde (Blockeel & De Raedt, 1998). Rather than using attribute-value or threshold tests in node of the tree, Tilde employs logical queries. Furthermore, a placeholder for domain elements (such as blocks) can occur in different nodes meaning that all occurrences denote the same domain element. Indeed, this slightly complicates the induction process, for example when generating the possible tests to be placed in a node. To this aim, it employs a classical *refinement operator* under θ-subsumption. The operator basically adds a literal, unifies variables, and grounds variables. When a node is to be splitted, the set of all refinements are computed and evaluated according to the chosen heuristic. Except for the representational differences, Tilde uses the same approach to tree-building as the generic tree learner. Because single episodes are too short and thus generate too few examples for Tilde to learn meaningful trees, we postpone calling Tilde until an episode brings the cumulated number of examples over 100. As a language bias for Tilde we employ the same language bias as used in published RRL experiments, including predicates such as *clear*, *on*, *above* and *compheight*[4]. Each learned model Δ_m, updates the potential function Ψ using a step-size $\eta_m = 1$. We count on the self-correcting property of tree boosting to correct over- or under-stepping the target on the next iteration.

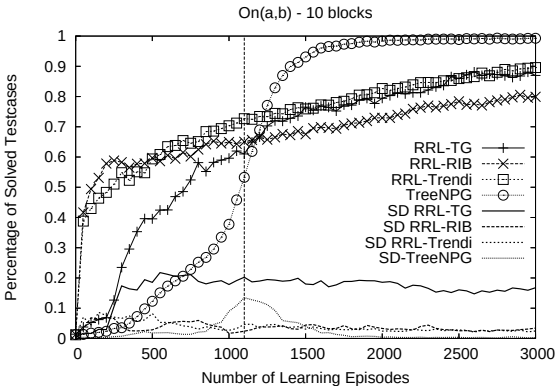

Figure 1. Comparison of the learning curves for Non-Parametric Policy Gradient and various RRL implementations on the on(A,B) task in a world with 10 blocks.

We ran experiments using three versions of the RRL system, i.e, TG (Driessens et al., 2001), RIB (Driessens & Ramon, 2003), and Trendi (Driessens & Džeroski, 2005), which represent the current state-of-the-art of RRL systems, and our TreeNPPG algorithm. After every 50 learning episodes, we fixed the strategy learned by RRL and tested the performance on 1000 randomly generated starting states. For TreeNPPG fixing the strategy is not required as it uses the learned strategy for exploration directly. Fig. 1 shows the resulting learning curves averaged over 10 test-runs as well as the standard deviations of these curves. As shown, TreeNPPG outperforms all tested versions of the RRL system. After approximately 2000 learning episodes, TreeNPPG solves 99% of all presented testcases in the minimal number of steps. With less than 4 steps per learning episode on average, this means that TreeNPPG generalizes the knowledge it collected by visiting less than 8000 states to solve 99% of 54.3 million states. Around this time, TreeNPPG has generated a list of 500 trees on average. One observation that can not be made directly on the shown graph is the stability of the TreeNPPG algorithm. Where the performance of the RRL system using TG

[3]We do not consider states in which the goal has already been satisfied as starting states. Therefore, a number of states where the goal is satisfied are not reachable, i.e., the states where $on(A, B)$ is satisfied and there are extra blocks on top of A.

[4]For a full overview and the semantics of these predicates, we refer to (Driessens & Džeroski, 2005).

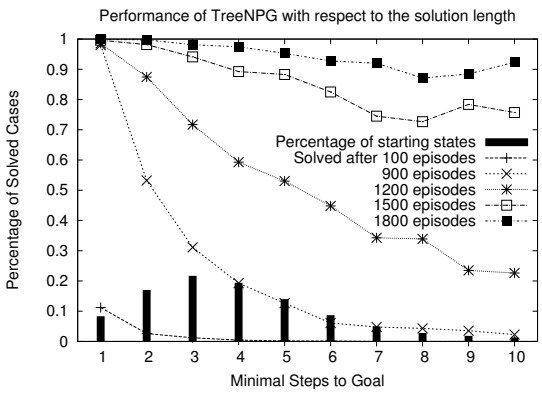

Figure 2. Detailed view of the learning performance of TreeNPPG showing the percentage of solved test-cases with respect to the minimal solution size for varying amounts of learning experience. The graph also shows the percentage of test-cases for each solution length.

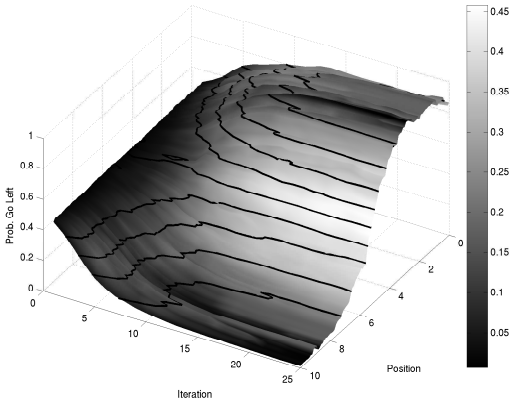

Figure 3. Learning performance of TreeNPPG on the continuous *corridor* task. Shown is how the probability of going *left* at all corridor positions evolves with the number of iterations. The shading indicates the standard deviation over the 30 runs of the experiment.

for regression can vary substantially between experiments, TreeNPPG follows an extremely similar path in each iteration of the experiment. The only variation between experiments is the exact time-frame of the phase transition in the results, as shown by the peak in TreeNPPG's standard deviation curve. Fig. 2 shows the results in more detail, plotting the percentage of solved test-cases with respect to the minimal solution length. As one can see, TreeNPPG gradually learns to solve problems with growing solution sizes. The graph also shows the percentage of test-cases for each solution size.

To summarize, the results clearly show that question **Q1** can be answered affirmatively.

(Q2) Corridor World: A Continuous Domain

To qualitatively test whether TreeNPPG is also applicable in continuous domains, we considered a simple *continuous corridor* domain. The task is to navigate a robot from any position $pos_0 \in [0, 10]$ in a one-dimensional corridor $[0, 10]$ to one of the exists at both ends (0 and 10). At each time $t = 0, 1, 2, \ldots$, the robot can go either one step to the *left* ($a = -1$) or one step to the *right* ($a = 1$); the outcome, however, is uncertain with $pos_{t+1} = pos_t + a + \mathcal{N}(1, 1)$. The robot is given a reward after each step equal to -1 if $pos \in (0, 10)$ and equal to 20 otherwise, i.e., it reaches one of the exists.

Fig. 3 shows how the stochastic policy evolves with the number of iterations, i.e., calls to the tree learner. These results are averaged over 30 reruns. In each run, we selected a random starting position pos_0 uniformly in $(0, 10)$, gathered learning examples from 30 episodes in each iteration, and used a step size $\eta_m = 0.7$. As one can see, the robot gradually learns to go *left* in the left section of the corridor and *right* in the right section. Quite intuitively, the uncertainty is highest in the middle part of the corridor.

We also ran experiments in a grid-world version of this problem. We do not report on the successful results here because finite domains are a special case of the relational and continuous cases. To summarize, question **Q2** can also be answered affirmatively.

5. Related Work

Within reinforcement learning (RL), there are two main classes of solution methods: value-function methods seek to estimate the value of states and actions whereas policy-based methods search for a good policy within some class of policies.

Within policy-based methods, policy gradients have received increased attention, see e.g. (Williams, 1992; Baxter et al., 2001; Guestrin et al., 2002; Konda & Tsitsiklis, 2003; Munos, 2006) as a non-exhausting list. Most closely related to NPPG is the work of Bagnell and Schneider (2003), see also (Bagnell, 2004). They proposed a functional gradient policy algorithms in reproducing kernel hilbert spaces for continuous domains. So far, however, this line of work has not considered (relationally) structured domains and the connection to gradient boosting was not employed.

Within value function approaches, the situation is slightly different. NPPG can be viewed as automatically generating a "variable" propositionalization or discretization of the domain at hand. In this sense, it is akin to tree-based state discretization RL ap-

proaches such as (Chapman & Kaelbling, 1991; McCallum, 1996; Uther & Veloso, 1998) and related approaches. Within this line of research, there have been some boosting methods proposed. Ernst et al. (2005) showed how to estimate Q functions with ensemble methods based on regression trees. Riedmiller (2005) keeps all regression examples and re-weights them according to some heuristic. Both neither consider policy gradients nor relational domains.

Recently, there have been some exciting new developments in combining the rich relational representations of classical knowledge representation with RL. While traditional RL requires (in principle) explicit state and action enumeration, these symbolic approaches seek to avoid explicit state and action enumeration through a symbolic representation of states and actions. Most work in this context, however, has focused on value function approaches. Basically, a number of relational regression algorithms have been developed for use in this RL system that employ relational regression trees (Driessens et al., 2001), relational instance based regression (Driessens & Ramon, 2003), graph kernels and Gaussian processes (Gärtner et al., 2003) and relational model-trees (Driessens & Džeroski, 2005). Finally, there is an increasing number of dynamic programming approaches for solving relational MDPs (Kersting et al., 2004; Sanner & Boutilier, 2005; Wang et al., 2007). In contrast to NPPG, they assume a model of the domain. It would be interesting, however, to combine NPPG with these approaches along the line of (Wang & Dietterich, 2003). A parametric step into this direction has been already taken by Aberdeen (2006).

6. Conclusions

We have introduced the framework of non-parametric policy gradient (NPPG) methods. It seeks to leverage the policy selection problem by approaching it from a gradient boosting perspective. NPPG is fast and straightforward to implement, combines the expressive power of relational RL with the benefits of policy gradient methods, and can deal with finite, continuous, and relational domains in a unified way. Moreover, the experimental results show a significant improvement over established results; for the first time, a (model-free) relational RL approach learns to solve $on(A, B)$ in a world with 10 blocks.

NPPG suggests several interesting directions for future research such as using more advanced regression models, developing actor-critic versions of NPPG estimating a value function in parallel to reduce the variance of the gradient estimates, and exploiting NPPG's ability to learn in hybrid domains with both discrete and continuous variables within real-world domains such as as robotics and network routing. Most interesting, however, is to address the more general problem of learning how to interact with (relationally) structured environments in the presence observation noise. NPPG is naturally applicable in this case and, hence, paves the way towards (model-free) solutions of what can be called relational POMDPs. This is a topic of high current interest since it combines the expressive representations of classical AI with the decision-theoretic emphasis of modern AI.

Acknowledgments The authors would like to thank the anonymous reviewers for their valuable comments. The material is based upon work performed while KK was with CSAIL@MIT and is supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010, and by a Fraunhofer ATTRACT fellowship. KD is a post-doctoral research fellow of the Research Fund - Flanders (FWO).

References

Aberdeen, D. (2006). Policy-gradient methods for planning. *Advances in Neural Information Processing Systems 18* (pp. 9–17).

Bagnell, J. (2004). *Learning decisions: Robustness, uncertainty, and approximation.* Doctoral dissertation, Robotics Institute, Carnegie Mellon University, Pittsburg, Pa, USA.

Bagnell, J., & Schneider, J. (2003). *Policy search in reproducing kernel hilbert space* (Technical Report CMU-RI-TR-03-45). Robotics Institute, Carnegie Mellon University.

Baxter, J., Bartlett, P., & Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Arificial Intellifence Research (JAIR)*, *15*, 351–381.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming.* Belmont, MA: Athena Scientific.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, *101*, 285–297.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees.* Belmont: Wadsworth.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 726–731).

Dietterich, T., Ashenfelter, A., & Bulatov, Y. (2004). Training conditional random fields via gradient tree boosting. *Proc. 21st International Conf. on Machine Learning* (pp. 217–224). ACM.

Driessens, K., & Džeroski, S. (2005). Combining model-based and instance-based learning for first order regression. *Proceedings of the 22nd International Conference on Machine Learning* (pp. 193–200).

Driessens, K., & Dzeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, *57*, 271–304.

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. *Proceedings of the 20th International Conference on Machine Learning* (pp. 123–130).

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proceedings of the 12th European Conference on Machine Learning* (pp. 97–108).

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)*, *6*, 503–556.

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Gärtner, T., Driessens, K., & Ramon, J. (2003). Graph kernels and gaussian processes for relational reinforcement learning. *International Conference on Inductive Logic Programming*.

Getoor, L., & Taskar, B. (2007). *An introduction to statistical relational learning*. MIT Press.

Guestrin, C., Lagoudakis, M., & R.Parr (2002). Coordinated reinforcement learning. *Proceedings of the 19th International Conference on Machine Learning* (pp. 227–234).

Gutmann, B., & Kersting, K. (2006). TildeCRF: Conditional random fields for logical sequences. *Proceedings of the 17th European Conference on Machine Learning*. Berlin, Germany.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)* (pp. 465–472). Banff, Canada.

Konda, V. R., & Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM J. Control Optim.*, *42*, 1143–1166.

McCallum, A. (1996). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, University of Rochester, New York.

Munos, R. (2006). Policy gradient in continuous time. *Journal of Machine Learning Research (JMLR)*, *7*, 771–791.

Riedmiller, M. (2005). Neural fitted Q iteration - First experiences with a data efficient neural reinforcement learning method. *Proceedings of the 16th European Conference on Machine Learning* (pp. 317–328).

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. *Proceedings of the 21st conference on Uncertainty in AI (UAI)*.

Slaney, J., & Thiébaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, *125*, 119–153.

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

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems 12* (pp. 1057–1063). MIT Press.

Uther, W., & Veloso, M. (1998). Tree based discretization for continuous state space reinforcement learning. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-96)* (pp. 769–774).

Wang, C., Joshi, S., & Khardon, R. (2007). First order decision diagrams for relational mdps. *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (pp. 1095–1100). Hyderabad, India: AAAI press.

Wang, X., & Dietterich, T. (2003). Model-based policy gradient reinforcement learning. *Proceedings of the 20th International Conference on Machine Learning*.

Williams, R. (1992). Simple statistical gradient following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.