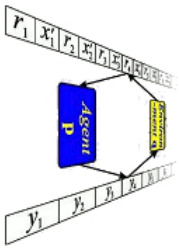
Proceedings of the 10th European Workshop on Reinforcement Learning

Volume 24:

10th European Workshop on Reinforcement Learning



European Workshop On Reinforcement Learning

Marc Peter Deisenroth, Csaba Szepesvári, and Jan Peters

Preface

The 10th European Workshop on Reinforcement Learning (EWRL), held June 30–July 1, 2012 in Edinburgh, Scotland, served as a forum to discuss the current state-of-the-art and future research directions in the continuously growing field of reinforcement learning (RL). We made EWRL an exciting and well-received event for the international RL community. Therefore, we appreciate that we could attract a wide spectrum of researchers by co-locating EWRL 2012 with the International Conference on Machine Learning.

We were very excited about our outstanding invited speakers Martin Riedmiller (University of Freiburg), Drew Bagnell (Carnegie Mellon University), Shie Mannor (Technion), and Richard Sutton (University of Alberta), who gave great overviews of current research and diverse application areas of reinforcement learning. EWRL 2012 had 63 submissions from which 43 (68%) were accepted for presentation at the workshop. These post-proceedings contain 12 (19%) selected revised papers submitted to EWRL 2012.

We thank our program committee for a fantastic job: Abdeslam Boularias, Adam White, Alborz Geramifard, Alessandro Lazaric, Amir-massoud Farahmand, Andre Damotta Salles Barreto, Andrew McHutchon, Bert Kappen, Bradley Knox, Byron Boots, Carlos Diuk Wasser, Christian Daniel, Christian Igel, Damien Ernst, David Silver, Doina Precup, Dvijotham Krishnamurthy, Emma Brunskill, Evangelos Theodorou, Fernand Fernandez, Francisco Melo, Gerhard Neumann, Hado van Hasselt, Jan Peters, Jens Kober, Jose Antonio Martin H., Jun Morimoto, Katharina Mülling, Kristian Kersting, Manuel Lopes, Marco Wiering, Martijn van Otterlo, Martin Riedmiller, Masashi Sugiyama, Matthew Hoffman, Matthew Robards, Matthieu Geist, Michal Valko, Mohammad Ghavamzadeh, Nikos Vlassis, Odalric-Ambrym Maillard, Oliver Kroemer, Olivier Pietquin, Pedro Ortega, Peter Auer, Peter Dayan, Peter Sunehag, Philipp Hennig, Philippe Preux, Remi Munos, Ronald Ortner, Shivaram Kalyanakrishnan, Stephane Ross, Teodor Moldovan, Thomas Furmston, Thomas J. Walsh, Thomas Rückstieß, Tobias Jung, Tobias Lang, Todd Hester, Tom Erez, Tom Schaul, Verena Heidrich-Meisner, Yuri Grinberg, Zhikun Wang, and Zico Kolter.

December 2012

The Editorial Team:

Marc Deisenroth
Technische Universität Darmstadt

Csaba Szepesvári
University of Alberta

Jan Peters
Technische Universität Darmstadt

Table of Contents

JMLR: Workshop and Conference Proceedings 24:1–9, 2012 10th European Workshop on Reinforcement Learning

Learning Exploration/Exploitation Strategies for Single Trajectory Reinforcement Learning

Michael Castronovo MICHAEL.CASTRONOVO@STUDENT.ULG.AC.BE
Francis Maes FRANCIS.MAES@ULG.AC.BE
Raphael Fonteneau RAPHAEL.FONTENEAU@ULG.AC.BE
Damien Ernst DERNST@ULG.AC.BE
Department of Electrical Engineering and Computer Science, University of Liège, BELGIUM

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

We consider the problem of learning high-performance Exploration/Exploitation (E/E) strategies for finite Markov Decision Processes (MDPs) when the MDP to be controlled is supposed to be drawn from a known probability distribution $p_{\mathcal{M}}(\cdot)$. The performance criterion is the sum of discounted rewards collected by the E/E strategy over an infinite length trajectory. We propose an approach for solving this problem that works by considering a rich set of candidate E/E strategies and by looking for the one that gives the best average performances on MDPs drawn according to $p_{\mathcal{M}}(\cdot)$. As candidate E/E strategies, we consider index-based strategies parametrized by small formulas combining variables that include the estimated reward function, the number of times each transition has occurred and the optimal value functions $\hat{V}$ and $\hat{Q}$ of the estimated MDP (obtained through value iteration). The search for the best formula is formalized as a multi-armed bandit problem, each arm being associated with a formula. We experimentally compare the performances of the approach with R-MAX as well as with ϵ-GREEDY strategies and the results are promising.

Keywords: Reinforcement Learning, Exploration/Exploitation dilemma, Formula discovery

1. Introduction

Most Reinforcement Learning (RL) techniques focus on determining high-performance policies maximizing the expected discounted sum of rewards to come using several episodes. The quality of such a learning process is often evaluated through the performances of the final policy regardless of rewards that have been gathered during learning. Some approaches have been proposed to take these rewards into account by minimizing the undiscounted regret (Kearns and Singh (2002); Brafman and Tennenholtz (2002); Auer and Ortner (2007); Jaksch et al. (2010)), but RL algorithms have troubles solving the *original RL problem* of maximizing the expected discounted return over a single trajectory. This problem is almost intractable in the general case because the discounted nature of the regret makes early mistakes - often due to hazardous exploration - almost impossible to recover from. Roughly speaking, the agent needs to learn very fast in one pass. One of the best solutions to face this Exploration/Exploitation (E/E) dilemma is the R-MAX algorithm (Brafman and Tennenholtz (2002)) which combines model learning and dynamic programming with

the "optimism in the face of uncertainty" principle. However, except in the case where the underlying Markov Decision Problem (MDP) comes with a small number of states and a discount factor very close to 1 (which corresponds to giving more chance to recover from bad initial decisions), the performance of R-max is still very far from the optimal (more details in Section 5).

In this paper, we assume some prior knowledge about the targeted class of MDPs, expressed in the form of a probability distribution over a set of MDPs. We propose a scheme for learning E/E strategies that makes use of this probability distribution to sample training MDPs. Note that this assumption is quite realistic, since before truly interacting with the MDP, it is often possible to have some prior knowledge concerning the number of states and actions of the MDP and/or the way rewards and transitions are distributed.

To instantiate our learning approach, we consider a rich set of candidate E/E strategies built around parametrized index-functions. Given the current state, such index-functions rely on all transitions observed so far to compute E/E scores associated to each possible action. The corresponding E/E strategies work by selecting actions that maximize these scores. Since most previous RL algorithms make use of small formulas to solve the E/E dilemma, we focus on the class of index-functions that can be described by a large set of such small formulas. We construct our E/E formulas with variables including the estimated reward function of the MDP (obtained from observations), the number of times each transition has occurred and the estimated optimal value functions $\hat{V}$ and $\hat{Q}$ (computed through off-line value iteration) associated with the estimated MDP. We then formalize the search for an optimal formula within that space as a multi-armed bandit problem, each formula being associated to an arm.

Since it assumes some prior knowledge given in the form of a probability distribution over possible underlying MDPs, our approach is related to Bayesian RL (BRL) approaches (Poupart et al. (2006); Asmuth et al. (2009)) that address the E/E trade-off by (i) assuming a prior over possible MDP models and (ii) maintaining - from observations - a posterior probability distribution (i.e., "refining the prior"). In other words, the prior is used to reduce the number of samples required to construct a good estimate of the underlying MDP and the E/E strategy itself is chosen a priori following Bayesian principles and does not depend on the targeted class of MDPs. Our approach is specific in the sense that the prior is not used for better estimating the underlying MDP but rather for identifying the best E/E strategy for a given class of targeted MDPs, among a large class of diverse strategies. We therefore follow the work of Maes et al. (2012), which already proposed to learn E/E strategies in the context of multi-armed bandit problems, which can be seen as state-less MDPs.

This paper is organized as follows. Section 2 formalizes the E/E strategy learning problem. Section 3 describes the space of formula-based E/E strategies that we consider in this paper. Section 4 details our algorithm for efficiently learning formula-based E/E strategies. Our approach is illustrated and empirically compared with R-max as well as with ϵ-Greedy strategies in Section 5. Finally, Section 6 concludes.

2. Background

Let $M = (\mathcal{S}, \mathcal{A}, p_{M,f}(\cdot), \rho_M, p_{M,0}(\cdot), \gamma)$ be a MDP. $\mathcal{S} = \left\{s^{(1)}, \ldots, s^{(n_\mathcal{S})}\right\}$ is its state space and $\mathcal{A} = \left\{a^{(1)}, \ldots, a^{(n_\mathcal{A})}\right\}$ its action space. When the MDP is in state s_t at time t and action a_t is selected, the MDP moves to a next state s_{t+1} drawn according to the probability distribution $p_{M,f}(\cdot|s_t, a_t)$. A deterministic instantaneous scalar reward $r_t = \rho_M(s_t, a_t, s_{t+1})$ is associated with the stochastic transition (s_t, a_t, s_{t+1}).

$H_t = [s_0, a_0, r_0, \ldots, s_t, a_t, r_t]$ is a vector that gathers the history over the first t steps and we denote by $\mathcal{H}$ the set of all possible histories of any length. An exploration / exploitation (E/E) strategy is a stochastic algorithm π that, given the current state s_t, processes at time t the vector H_{t-1} to select an action $a_t \in \mathcal{A}$: $a_t \sim \pi(H_{t-1}, s_t)$. Given the probability distribution over initial states $p_{M,0}(\cdot)$, the performance/return of a given E/E strategy π with respect to the MDP M can be defined as: $J^\pi_M = \mathop{\mathbb{E}}_{p_{M,0}(\cdot), p_{M,f}(\cdot)}[\mathcal{R}^\pi_M(s_0)]$ where $\mathcal{R}^\pi_M(s_0)$ is the stochastic discounted return of the E/E strategy π when starting from the state s_0. This return is defined as:

$$\mathcal{R}^\pi_M(s_0) = \sum_{t=0}^{\infty} \gamma^t r_t \ ,$$

where $r_t = \rho_M(s_t, \pi(H_{t-1}, s_t), s_{t+1})$ and $s_{t+1} \sim p_{M,f}(.|s_t, \pi(H_{t-1}, s_t))$ $\forall t \in \mathbb{N}$ and where the discount factor γ belongs to $[0, 1)$. Let $p_\mathcal{M}(\cdot)$ be a probability distribution over MDPs, from which we assume that the actual underlying MDP M is drawn. Our goal is to learn a high performance finite E/E strategy π given the prior $p_\mathcal{M}(\cdot)$, i.e. an E/E strategy that maximizes the following criterion:

$$J^\pi = \mathop{\mathbb{E}}_{M' \sim p_\mathcal{M}(\cdot)}[J^\pi_{M'}] \ . \tag{1}$$

3. Formula-based E/E strategies

In this section, we describe the set of E/E strategies that are considered in this paper.

3.1. Index-based E/E strategies

Index-based E/E strategies are implicitly defined by maximizing history-dependent state-action index functions. Formally, we call a history-dependent state-action index function any mapping $I : \mathcal{H} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Given such an index function I, a decision can be taken at time t in the state $s_t \in \mathcal{S}$ by drawing an optimal action according to I: $\pi(H_{t-1}, s_t) \in \mathop{\arg\max}_{a \in \mathcal{A}} I(H_{t-1}, s_t, a)$[1]. Such a procedure has already been vastly used in the particular case where the index function is an estimate of the action-value function, eventually randomized using $\epsilon-$greedy or Boltzmann exploration, as in Q-Learning (Watkins and Dayan (1992)).

1. Ties are broken randomly in our experiments.

3.2. Formula-based E/E strategies

We consider in this paper index functions that are given in the form of small, closed-form formulas. This leads to a very rich set of candidate E/E strategies that have the advantage of being easily interpretable by humans. Formally, a formula $F \in \mathbb{F}$ is:

• either a binary expression $F = B(F', F'')$, where B belongs to a set of binary operators $\mathbb{B}$ and F' and F'' are also formulas from $\mathbb{F}$,

• or a unary expression $F = U(F')$ where U belongs to a set of unary operators $\mathbb{U}$ and $F' \in \mathbb{F}$,

• or an atomic variable $F = V$, where V belongs to a set of variables $\mathbb{V}$ depending on the history H_{t-1}, the state s_t and the action a,

• or a constant $F = C$, where C belongs to a set of constants $\mathbb{C}$.

Since it is high dimensional data of variable length, the history H_{t-1} is non-trivial to use directly inside E/E index-functions. We proceed as follows to transform the information contained in H_{t-1} into a small set of relevant variables. We first compute an estimated model of the MDP $\hat{M}$ that differs from the original M due to the fact that the transition probabilities and the reward function are not known and need to be learned from the history H_{t-1}. Let $\hat{P}(s, a, s')$ and $\hat{\rho}(s, a)$ be the transition probabilities and the reward function of this estimated model. $\hat{P}(s, a, s')$ is learned by computing the empirical frequency of jumping to state s' when taking action a in state s and $\hat{\rho}(s, a)$ is learned by computing the empirical mean reward associated to all transitions originating from (s, a)[2]. Given the estimated MDP, we run a value iteration algorithm to compute the estimated optimal value functions $\hat{V}(\cdot)$ and $\hat{Q}(\cdot, \cdot)$. Our set of variables is then defined as: $\mathbb{V} = \left\{\hat{\rho}(s_t, a), N(s_t, a), \hat{Q}(s_t, a), \hat{V}(s_t), t, \gamma^t\right\}$ where $N(s, a)$ is the number of times a transition starting from (s, a) has been observed in H_{t-1}.

We consider a set of operators and constants that provides a good compromise between high expressiveness and low cardinality of $\mathbb{F}$. The set of binary operators $\mathbb{B}$ includes the four elementary mathematical operations and the min and max operators: $\mathbb{B} = \{+, -, \times, \div, \min, \max\}$. The set of unary operators $\mathbb{U}$ contains the square root, the logarithm and the absolute value: $\mathbb{U} = \left\{\sqrt{\cdot}, \ln(\cdot), |\cdot|\right\}$. The set of constants is: $\mathbb{C} = \{1, 2, 3, 5, 7\}$.

In the following, we denote by π^F the E/E strategy induced by formula F:

$$\pi^F(H_{t-1}, s_t) \in \underset{a \in \mathcal{A}}{\arg\max}\ F\left(\hat{\rho}(s_t, a), N(s_t, a), \hat{Q}(s_t, a), \hat{V}(s_t), t, \gamma^t\right)$$

We denote by $|F|$ the description length of the formula F, i.e. the total number of operators, constants and variables occurring in F. Let K be a maximal formula length. We denote by $\mathbb{F}^K$ the set of formulas whose length is not greater than K. This defines our so-called set of small formulas.

2. If a pair (s, a) has not been visited, we consider the following default values: $\hat{\rho}(s, a) = 0$, $\hat{P}(s, a, s) = 1$ and $\hat{P}(s, a, s') = 0, \forall s' \neq s$.

4. Finding a high-performance formula-based E/E strategy for a given class of MDPs

We look for a formula F^* whose corresponding E/E strategy is specifically efficient for the subclass of MDPs implicitly defined by the probability distribution $p_{\mathcal{M}}(\cdot)$. We first describe a procedure for accelerating the search in the space $\mathbb{F}^K$ by eliminating equivalent formulas in Section 4.1. We then describe our optimization scheme for finding a high-performance E/E strategy in Section 4.2.

4.1. Reducing $\mathbb{F}^K$

Notice first that several formulas $\mathbb{F}^K$ can lead to the same policy. All formulas that rank all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ in the same order define the same policy. We partition the set $\mathbb{F}^K$ into equivalence classes, two formulas being equivalent if and only if they lead to the same policy. For each equivalence class, we then consider one member of minimal length, and we gather all those minimal members into a set $\bar{\mathbb{F}}^K$.

Computing the set $\bar{\mathbb{F}}^K$ is not trivial: given a formula, equivalent formulas can be obtained through commutativity, associativity, operator-specific rules and through any increasing transformation. We thus propose to approximately discriminate between formulas by comparing how they rank (in terms of values returned by the formula) a set of d random samples of the variables $\hat{\rho}(\cdot,\cdot), N(\cdot,\cdot), \hat{Q}(\cdot,\cdot), \hat{V}(\cdot), t, \gamma^t$. More formally, the procedure is the following:

- we first build $\mathbb{F}^K$, the space of all formulas such that $|F| \leq K$;
- for $i = 1 \dots d$, we uniformly draw (within their respective domains) some random realizations of the variables $\hat{\rho}(\cdot,\cdot), N(\cdot,\cdot), \hat{Q}(\cdot,\cdot), \hat{V}(\cdot), t, \gamma^t$ that we concatenate into a vector Θ_i;
- we cluster all formulas from $\mathbb{F}^K$ according to the following rule: two formulas F and F' belong to the same cluster if and only if they rank all the Θ_i points in the same order, i.e.: $\forall i, j \in \{1, \dots, d\}, i \neq j, F(\Theta_i) \geq F(\Theta_j) \iff F'(\Theta_i) \geq F'(\Theta_j)$. Formulas leading to invalid index functions (caused for instance by division by zero or logarithm of negative values) are discarded;
- among each cluster, we select one formula of minimal length;
- we gather all the selected minimal length formulas into an approximated reduced set of formulas $\tilde{\mathbb{F}}^K$.

In the following, we denote by N the cardinality of the approximate set of formulas $\tilde{\mathbb{F}}^K = \{F_1, \dots, F_N\}$.

4.2. Finding a high-performance formula

A naive approach for determining a high-performance formula $F^* \in \tilde{\mathbb{F}}^K$ would be to perform Monte-Carlo simulations for all candidate formulas in $\tilde{\mathbb{F}}^K$. Such an approach could reveal itself to be time-inefficient in case of spaces $\tilde{\mathbb{F}}^K$ of large cardinality.

We propose instead to formalize the problem of finding a high-performance formula-based E/E strategy in $\tilde{\mathbb{F}}^K$ as a $N-$armed bandit problem. To each formula $F_n \in \tilde{\mathbb{F}}^K$ ($n \in \{1, \dots, N\}$), we associate an arm. Pulling the arm n consists first in randomly drawing a MDP M according to $p_{\mathcal{M}}(\cdot)$ and an initial state s_0 for this MDP according to $p_{M,0}(\cdot)$.

Afterwards, an episode starting from this initial state is generated with the E/E strategy π^{F_n} until a truncated time horizon T. This leads to a reward associated to arm n whose value is the discounted return $\mathcal{R}^{\pi}_{M}(s_0)$ observed during the episode. The purpose of multi-armed bandit algorithms is here to process the sequence of such observed rewards to select in a smart way the next arm to be played so that when the budget of pulls has been exhausted, one (or several) high-quality formula(s) can be identified.

Multi-armed bandit problems have been vastly studied, and several algorithms have been proposed, such as for instance all UCB-type algorithms (Auer et al. (2002); Audibert et al. (2007)). New approaches have also recently been proposed for identifying automatically empirically efficient algorithms for playing multi-armed bandit problems (Maes et al. (2011)).

5. Experimental results

In this section, we empirically analyze our approach on a specific class of random MDPs defined hereafter.

Random MDPs. MDPs generated by our prior $p_{\mathcal{M}}(\cdot)$ have $n_{\mathcal{S}} = 20$ states and $n_{\mathcal{A}} = 5$ actions. When drawing a MDP according to this prior, the following procedure is called for generating $p_{M,f}(\cdot)$ and $\rho_M(\cdot,\cdot,\cdot)$. For every state-action pair (s,a) : (i) it randomly selects 10% of the states to form a set of successor states $Succ(s,a) \subset \mathcal{S}$ (ii) it sets $p_{M,f}(s'|s,a) = 0$ for each $s' \in \mathcal{S} \setminus Succ(s,a)$ (iii) for each $s' \in Succ(s,a)$, it draws a number $N(s')$ at random in $[0,1]$ and sets $p_{M,f}(s'|s,a) = \frac{N(s')}{\sum_{s'' \in Succ(s,a)} N(s'')}$ (iv) for each $s' \in Succ(s,a)$, it sets $\rho_M(s,a,s')$ equal to a number chosen at random in $[0,1]$ with a 0.1 probability and to zero otherwise. The distribution $p_{M,0}(\cdot)$ of initial states is chosen uniform over $\mathcal{S}$. The value of γ is equal to 0.995.

Learning protocol. In our experiments, we consider a maximal formula length of $K = 5$ and use $d = 1000$ samples to discriminate between formulas, which leads to a total number of candidate E/E strategies $N = 3834$. For solving the multi-armed bandit problem described in Section 4.2, we use an Upper Confidence Bound (UCB) algorithm (Auer et al. (2002)). The total budget allocated to the search of a high-performance policy is set to $T_b = 10^6$. We use a truncated optimization horizon $T = \log_\gamma((1-\gamma)\delta)$ for estimating the stochastic discounted return of an E/E strategy where $\delta = 0.001$ is the chosen precision (which is also used as stopping condition in the off-line value iteration algorithm for computing $\hat{Q}$ and $\hat{V}$). At the end of the T_b plays, the five E/E strategies that have the highest empirical return mean are returned.

Baselines. Our first baseline, the Optimal strategy, consists in using for each test MDP, a corresponding optimal policy. The next baselines, the Random and Greedy strategies perform pure exploration and pure exploitation, respectively. The Greedy strategy is equivalent to an index-based E/E strategy with formula $\hat{Q}(s,a)$. The last two baselines are classical E/E strategies whose parameters have been tuned so as to give the best performances on MDPs drawn from $p_{\mathcal{M}}(\cdot)$: ϵ-Greedy and R-max. For ϵ-Greedy, the best value we found was $\epsilon = 0$ in which case it behaves as the Greedy strategy. This confirms that hazardous exploration is particularly harmful in the context of single trajectory RL with discounted return. Consistently with this result, we observed that R-max works at its best when it performs the least exploration ($m = 1$).

Baselines		Learned strategies	
Name	J^π	Formula	J^π
Optimal	65.3	$(N(s,a) \times \hat{Q}(s,a)) - N(s,a)$	30.3
Random	10.1	$\max(1, (N(s,a) \times \hat{Q}(s,a)))$	22.6
Greedy	20.0	$\hat{Q}(s,a)$ (= Greedy)	20.0
ϵ-Greedy($\epsilon = 0$)	20.0	$\min(\gamma^t, (\hat{Q}(s,a) - \hat{V}(s)))$	19.4
R-max ($m = 1$)	27.7	$\min(\hat{\rho}(s,a), (\hat{Q}(s,a) - \hat{V}(s)))$	19.4

Table 1: Performance of the top-5 learned strategies with respect to baseline strategies.

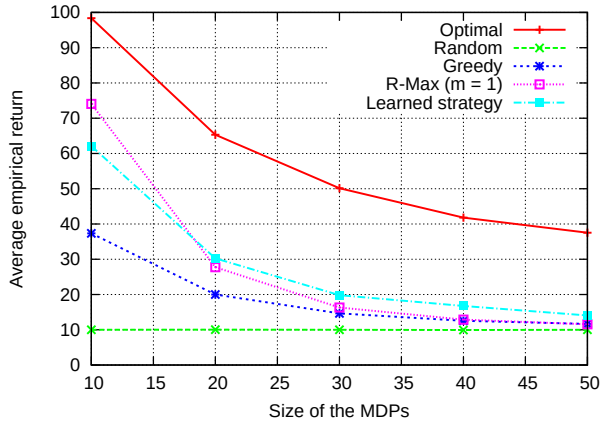


Figure 1: Performances of the learned and the baseline strategies for different distributions of MDPs that differ by the size of the MDPs belonging to their support.

Results. Table 1 reports the mean empirical returns obtained by the E/E strategies on a set of 2000 test MDPs drawn from $p_{\mathcal{M}}(\cdot)$. Note that these MDPs are different from those used during learning and tuning. As we can see, the best E/E strategy that has been learned performs better than all baselines (except the Optimal), including the state-of-the-art approach R-max.

We may wonder to what extent the E/E strategies found by our learning procedure would perform well on MDPs which are not generated by $p_{\mathcal{M}}(\cdot)$. As a preliminary step to answer this question, we have evaluated the mean return of our policies on sets of 2000 MDPs drawn from slightly different distributions as the one used for learning: we changed the number of states $n_{\mathcal{S}}$ to different values in $\{10, 20, \ldots, 50\}$. The results are reported in Figure 1. We observe that, except in the case $n_{\mathcal{S}} = 10$, our best E/E strategy still performs better than the R-max and the ϵ-Greedy strategies tuned on the original distribution $p_{\mathcal{M}}(\cdot)$ that generates MDPs with 20 states. We also observe that for larger values of $n_{\mathcal{S}}$, the performances of R-max become very close to those of Greedy, whereas the performances of our best E/E strategy remain clearly above. Investigating why this formula performs well is left for future work, but we notice that it is analog to the formula $t_k(r_k - C)$ that was automatically discovered as being well-performing in the context of multi-armed bandit problems (Maes et al. (2011)).

6. Conclusions

In this paper, we have proposed an approach for learning E/E strategies for MDPs when the MDP to be controlled is supposed to be drawn from a known probability distribution $p_{\mathcal{M}}(\cdot)$. The strategies are learned from a set of training MDPs (drawn from $p_{\mathcal{M}}(\cdot)$) whose size depends on the computational budget allocated to the learning phase. Our results show that the learned strategies perform very well on test problems generated from the same distribution. In particular, they outperform on these problems R-max and ϵ-Greedy policies. Interestingly, the strategies also generalize well to MDPs that do not belong to the support of $p_{\mathcal{M}}(\cdot)$. This is demonstrated by the good results obtained on MDPs having a larger number of states than those belonging to $p_{\mathcal{M}}(\cdot)$'s support.

These encouraging results suggest several future research direction. First, it would be interesting to better study the generalization performances of our approach either theoretically or empirically. Second, we believe that our approach could still be improved by considering richer sets of formulas w.r.t. the length of the formulas and the number of variables extracted from the history. Finally, it would be worth investigating ways to improve the optimization procedure upon which our learning approach is based so as to be able to deal with spaces of candidate E/E strategies that are so large that even running once every strategy on a single training problem would be impossible.

References

J. Asmuth, L. Li, M.L. Littman, A. Nouri, and D. Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press, 2009.

J.Y. Audibert, R. Munos, and C. Szepesvári. Tuning bandit algorithms in stochastic environments. In *Algorithmic Learning Theory*, pages 150–165. Springer, 2007.

P. Auer and R. Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, volume 19, page 49. The MIT Press, 2007.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

R.I. Brafman and M. Tennenholtz. R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2002.

T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *The Journal of Machine Learning Research*, 11:1563–1600, 2010.

M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

F. Maes, L. Wehenkel, and D. Ernst. Automatic discovery of ranking formulas for playing with multi-armed bandits. In *9th European workshop on reinforcement learning*, Athens, Greece, September 2011.

F. Maes, L. Wehenkel, and D. Ernst. Learning to play K-armed bandit problems. In *International Conference on Agents and Artificial Intelligence*, Vilamoura, Algarve, Portugal, February 2012.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 697–704. ACM, 2006.

C.J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):179–192, 1992.

JMLR: Workshop and Conference Proceedings 24:11–22, 2012 10th European Workshop on Reinforcement Learning

Feature Reinforcement Learning using Looping Suffix Trees

Mayank Daswani mayank.daswani@anu.edu.au
Peter Sunehag Peter.Sunehag@anu.edu.au
Marcus Hutter Marcus.Hutter@anu.edu.au
Research School of Computer Science,
Australian National University,
Canberra, ACT, 0200, Australia.

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

There has recently been much interest in history-based methods using suffix trees to solve POMDPs. However, these suffix trees cannot efficiently represent environments that have long-term dependencies. We extend the recently introduced CTΦMDP algorithm to the space of looping suffix trees which have previously only been used in solving deterministic POMDPs. The resulting algorithm replicates results from CTΦMDP for environments with short term dependencies, while it outperforms LSTM-based methods on TMaze, a deep memory environment.

1. Introduction

In reinforcement learning (RL) an agent must learn a policy (behaviour) that performs well in a given (dynamic) environment through interactions with the environment itself [Kaelbling et al., 1996]. Traditional RL methods maximise the reward in a given unknown finite Markov Decision Process (MDP). In the general RL problem there are no states, the agent instead receives observations that are not necessarily Markov. Feature Reinforcement Learning (ΦMDP) Hutter [2009] automates the extraction of a good state representation in the form of an MDP from the agent's observation-reward-action history. More clearly, this means that given a class of maps Φ we consider a class of environments that at least approximately reduce to an MDP through a map $\phi \in \Phi$. Nguyen et al. [2011] recently showed that this is viable in practice using a map class of history suffix trees.

The problem with using suffix trees as a map class is that they cannot efficiently remember events over a long period of time. The depth of the suffix tree is proportional to the length of history that it has to remember. In order to deal with long-term dependencies we make use of the class of looping suffix trees (looping STs) within the ΦMDP framework. Holmes and Jr. [2006] first proposed looping STs in the deterministic case, we also consider stochastic models which can be crucial even in deterministic environments. We show that looping suffix trees in conjunction with the ΦMDP framework can be used to successfully find compact representations of environments that require long-term memory in order to perform optimally.

1.1. Related Work

Our looping suffix tree method learns a finite state automaton that is well suited to long-term memory tasks. While tree-based methods such as USM [McCallum, 1995], MC-AIXI-CTW [Veness et al., 2011], Active LZ [Farias et al., 2010], CTΦMDP [Nguyen et al., 2011] and many others can in principle handle long-term memory tasks, they require excessively large trees to represent such environments. These large trees can result in large state spaces, which then promote the exacerbated exploration-exploitation problem. More related to our work, Mahmud [2010] aims at searching the very large space of probabilistic deterministic finite automata (with some restrictions). In a similar vein, but restricted to deterministic observations [Haghighi et al., 2007] also construct finite automata that aim at being the minimal predicting machine.

A popular alternative to finite state automaton learning is a class of algorithms based on recurrent neural networks (RNNs) particularly those based on the Long Short-Term Memory (LSTM) framework by Hochreiter and Schmidhuber [1997]. The LSTM framework was first proposed to predict time-series data with long-term dependencies. This was introduced to the RL context by Bakker [2002] and more recently a new model-free variant based on policy gradients by Wierstra et al. [2007]. These methods are more often used in the continuous case, but were also tested in the discrete setting. Recently, Echo State Networks [Szita et al., 2006] which are also RNN-based have also been tested on long-term memory tasks.

2. Preliminaries

Agent-Environment Framework. We use the notation and framework from [Hutter, 2005]. An agent acts in an Environment Env. It chooses actions $a \in \mathcal{A}$, and receives observations $o \in \mathcal{O}$ and real-valued rewards $r \in \mathcal{R}$ where $\mathcal{A}, \mathcal{O}$ and $\mathcal{R}$ are all finite. This observation-reward-action sequence happens in cycles indexed by $t = 1, 2, 3, ...$ We use $x_{1:n}$ throughout to represent the sequence $x_1...x_n$. We define the space of histories as $\mathcal{H} := (\mathcal{O} \times \mathcal{R} \times \mathcal{A})^* \times \mathcal{O} \times \mathcal{R}$. The history at time t is given by $h_t = o_1 r_1 a_1 ... o_{t-1} r_{t-1} a_{t-1} o_t r_t$. Using this definition of history we formally define the agent to be a function $Agent : \mathcal{H} \rightsquigarrow \mathcal{A}$ where $Agent(h_t) := a_t$. Similarly, the environment can be viewed as a stochastic function of the history, $Env : \mathcal{H} \times \mathcal{A} \rightsquigarrow \mathcal{O} \times \mathcal{R}$, where $Env(h_{t-1}, a_{t-1}) := o_t r_t$. The symbol $\rightsquigarrow$ indicates a possibly stochastic function. A policy is defined as a map $\pi : \mathcal{H} \rightsquigarrow A$.

If $Pr(o_t r_t | h_t, a_t) = Pr(o_t r_t | o_{t-1} a_t)$, the environment is said to be a discrete **Markov Decision Process (MDP)** [Puterman, 1994]. In this case, the observations form the state space of the MDP. Formally an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions and $R : \mathcal{S} \times \mathcal{A} \rightsquigarrow \mathcal{R}$ is the (possibly stochastic) reward function which gives the (real-valued) reward gained by the agent after taking action a in state s. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *state-transition function*. The agent's goal is to maximise its future discounted expected reward, where a geometric discount function with rate γ is used. The value of a state according to a stationary policy is given by $V^\pi(s) = E_\pi\{R_t | s_t = s\}$ where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is the *return*. We want to find the optimal value function V^* such that $V^*(s) = \max_\pi V^\pi(s)$. If an MDP is known then this can be done by value iteration [Sutton and Barto, 1998]; in the unknown case the agent must deal with the problem of exploration vs exploitation, which only has efficient approximate solutions.

ΦMDP. Hutter [2009] proposes a framework that extracts relevant features from the history for reward prediction. This framework gives us a method to find a map $\phi : \mathcal{H} \to \mathcal{S}$ such that the state at any time step $s_t = \phi(h_t)$ is approximately a sufficient statistic of the history. It uses a global cost function that is inspired by the minimum description length principle [Rissanen, 1978]. The cost is the sum of the code lengths of state and reward sequences given actions. This cost is combined with a global stochastic search technique (such as simulated annealing [Liu, 2008]) to find the optimal map. The standard cost is defined as follows

$$Cost(\phi|h_n) := CL(s_{1:n}|a_{1:n}) + CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi)$$

$CL(s_{1:n}|a_{1:n})$ is the code length of the state sequence given the action sequence. The subsequence of states reached from a given state s via action a is i.i.d as it is sampled from an MDP. We form a frequency estimate of the model of this MDP. The code length is then the length of the arithmetic code with respect to the model plus a penalty for coding parameters. The coding is optimal by construction. $CL(r_{1:n}|s_{1:n}, a_{1:n})$ follows similarly.

The consistency of this cost criterion was proven by Sunehag and Hutter [2010]. The modified cost by Nguyen et al. [2011] adds a parameter α to control the balance between reward coding and state coding,

$$Cost_\alpha(\phi|h_n) := \alpha CL(s_{1:n}|a_{1:n}) + (1-\alpha)CL(r_{1:n}|s_{1:n}, a_{1:n}) + CL(\phi).$$

We primarily care about reward prediction. However, since rewards depend on states we also need to code the states. The *Cost* is well-motivated since it balances between coding states and coding rewards. A state space that is too large results in poor learning and a long state coding, while a state space that is too small can obscure structure in the reward sequence resulting in a long code for the rewards.

Nguyen et al. [2011] search the map space of suffix trees (explained below). Our method extends this to looping suffix trees. The generic ΦMDP algorithm is given in Algorithm 1 in the Appendix. The agent is first initialised with some history based on random actions. Then it alternates between finding a "best" ϕ using the *simulated annealing* algorithm and performing actions based on the optimal policy for that ϕ found via the *FindPolicy()* function. The *FindPolicy()* function can be any standard reinforcement learning algorithm that finds the optimal policy in an unknown MDP, and should perform some amount of exploration, generally via an optimistic initialisation. In this paper, we use the model-based method as specified by Hutter [2009] which is based on Szita and Lörincz [2008]. This method adds an additional "garden of eden" state (s_e) to the estimated MDP, which is an absorbing state with a high reward. The agent is told that it has been to s_e once from every other state, however the agent cannot actually transition to this state. Then we simply perform value iteration on this augmented MDP. Initially the agent will explore in a systematic manner to try and visit s_e, but as it accumulates more transitions from a particular state, the estimated transition probability to s_e decreases, and the agent eventually settles on the optimal policy.

Definition 1 (Suffix Tree) *Let $\mathcal{O} = \{o^1, o^2, o^3, ..., o^d\}$ be a d-ary alphabet. A suffix tree is a d-ary tree in which the outgoing edges from each internal node are labelled by the elements of $\mathcal{O}$. Every suffix tree has a corresponding suffix set which is the set of strings*

$\mathcal{S} = \{s^1, s^2, ..., s^n\}$ *generated by listing the labels on the path from each leaf node to the root of the tree.*

The suffix set has the property that no string is a suffix of any other string and any sufficiently long string must have a suffix in the set. Each string in the suffix set is called a state, and hence this is also called a suffix state set. The *l-th* level of the tree corresponds to the *l-th* last observation in the history. By the above properties, any history of sufficient length must be mapped to one and only one state based on its suffix.

Definition 2 (Looping Tree) *A looping tree is a tree which may have loops from any leaf node to an ancestor.*

Definition 3 (Looping suffix Tree) *A looping suffix tree based on a d-ary alphabet* $\mathcal{O} = \{o^1, o^2, o^3, ..., o^d\}$ *is a d-ary looping tree in which edges coming from each internal node are labelled by the elements of* $\mathcal{O}$*. The loops in the tree are unlabelled. The non-looping leaf nodes in the looping ST form the state set along with an additional state* s^{empty} *known as the empty state.*

In order to map a history sequence to a state in a looping ST we simply follow the edges in the tree until we get to a state (Algorithm 2, Appendix). If we reach the beginning of the history sequence without reaching a state, we map the sequence to the empty state.

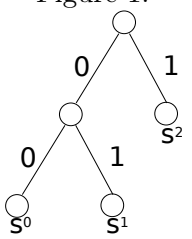
Looping suffix trees have the effect of giving Kleene-star like representational ability to the standard suffix set. For example, Figure 1 shows a looping suffix tree which has the suffix set $\{0, 00(10)^*1, 1(10)^*1\}$. Let $h = [0, 0, 1, 1, 0, 1]$. We can map this history sequence to the state sequence $stateSeq = [s^0, s^0, s^1, s^2, s^0, s^2]$. The last state is mapped by following 1,0,1 down the tree, then following the loop back up the tree to finally take another 1 to end in s^2.

3. Looping Suffix Trees in ΦMDP

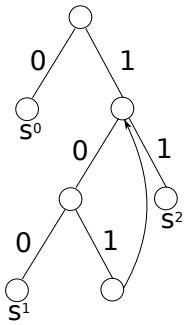
The benefit of using looping STs comes from the ability to remember relevant past events by 'forgetting' or looping over irrelevant details. Holmes and Jr. [2006] restrict their discussion to the deterministic case without rewards. Unfortunately their loopability criterion cannot work in the stochastic case since a loop can change not only the possible transitions but also the transition probabilities.

The cost function of the ΦMDP framework immediately gives us a well-motivated criterion for evaluating looping suffix trees. Using looping suffix trees as the map class in this framework allows us to extend them to stochastic environments. While we have not yet shown theoretical guarantees, experimental results show that ΦMDP works well in the space of looping suffix trees. The extension to stochastic tree sources is also useful in deterministic environments, where in some cases a smaller stochastic tree source can sufficiently capture a deterministic environment.

Figure 1:



(*a*) Suffix Tree



(*b*) Looping ST

Definition 4 *A history is said to be* consistent *with respect to a particular looping suffix tree if it can be mapped to a state sequence that does not include the empty state. The definition of inconsistent follows in the obvious manner.*

Algorithm. The algorithm consists of a specification of $CL(\phi)$ and the neighbourhood method which is needed for the simulated annealing algorithm in the generic ΦMDP algorithm (Algorithm 1, Appendix). We call our algorithm LSTΦMDP. A tree with num_nodes can be coded in num_nodes bits [Veness et al., 2011, Sec.5] and the starting and ending nodes of all loops can be coded in $2\log(num\_nodes)num\_loops$, so we define the model cost of the map $CL(\phi)$ as

$$CL(\phi) = num\_nodes + 2\log(num\_nodes)num\_loops.$$

The getNeighbour() method (Algorithm 3, Appendix) first selects a state randomly and then with equal probability subject to certain conditions, it selects between one of 4 operations. Note that the simulated annealing procedure that we use is a very simple generic method. However this can be extended to more sophisticated annealing schemes such as parallel tempering as done by Nguyen et al. [2011].

merge : In order to merge a state, all sibling nodes must also be states. From the definition of a suffix set, we know that every state corresponds to a unique suffix. The merge is simply the shortening of a context for those states. If s^i is the state being merged and $s^i = o^j n'$ where $o^j \in \mathcal{O}$ and n' is the remainder of the suffix corresponding to that state, then the siblings of s^i are $o^k n'$ where $k \neq i$. If these siblings are also states then the merge operator removes $o^i n'$ for all i from the suffix set and adds a new state n'.

split : Analogously, we can split any state s^i by adding a depth one context to the state i.e. by constructing $|\mathcal{O}|$ new states of the form $o^j s^i$ for all $o^j \in \mathcal{O}$ and removing the state s^i.

addLoop : The addLoop function has two cases. Either we add a loop from an existing state to it's parent (thereby removing it from the state set and adding it to the loop set) or we extend an existing loop to the parent of the existing node looped to.

removeLoop : The removeLoop function is simply a reverse of the addLoop function allowing us to decrease the length of a loop, or if it is a length one loop create a new state from the node.

Loops introduce a few problems to the ΦMDP procedure. A looped tree can be inconsistent with the current history. This can be problematic if, for instance, the optimal tree is inconsistent with the current history. One solution is to always provide a reasonable pre-history that the optimal tree should be consistent with. For example in the TMaze case (see Section 4), we ensure that the first observation is in fact the start of an episode, which is a reasonable assumption. Then any trees that are inconsistent can be discarded. In fact to make the search quicker, we can mark nodes where loops make the tree inconsistent and no longer add those loops. The initial map is always set to be the depth one tree (i.e. one split). Non-looping suffix trees were shown to be statistically consistent with reference to the **Cost** function (under the restrictions of a fixed policy and ergodicity) only by neglecting the root tree by Sunehag and Hutter [2010].

The space of looping STs includes the space of ordinary STs. Therefore, results from the non-looping case [Nguyen et al., 2011] should be reproducible, as long as the simulated annealing procedure is not adversely affected by the enlargement of the search space. Ex-

perimental results show that some care must be taken in choosing α for this to be the case. This is further discussed in Section 4.

4. Experiments

In this section we describe our experimental setup and the domains that we used to evaluate our algorithm. Each domain was used to test a different ability of the algorithm. Every experiment was run 50 times. The agent is given an initial history produced by taking random actions. Each run of an experiment was conducted over some number of epochs with each epoch containing 100 iterations of the agent performing actions according to its current policy, based on the current map with a constant ϵ-exploration of 0.1 until a point where it stops exploration. After every epoch, the agent was given a chance to change its optimal map via a simulated annealing procedure. The annealing procedure used an exponential cooling function with constants chosen so that the first few maps had an initial acceptance probability in the range $[0.6, 0.7]$. Plots show every 10th point with 2 standard error on either side. The exact constants used for all the experiments can be found in Table 1 in the Appendix.

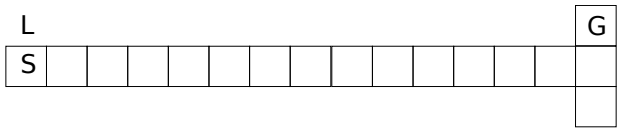


Figure 2: TMaze environment showing goal at left

TMaze. The TMaze problem is a classic non-Markovian problem in RL. It nicely demonstrates the need for long-term memory as well as the exploration vs exploitation problem. We use the formulation as described by Bakker [2002]. The environment is a T-shaped maze (see Figure 2) with the length of the neck of the T (the corridor) being adjustable. The observation space is $\mathcal{O} = \{0, 1, 2, 3\}$, the rewards are $\mathcal{R} = \{-0.1, 4\}$ and there are four actions denoted by up, right, left, down. The agent needs to remember the observation it receives at the start of the maze, which tells it whether to turn left or right at the end.

The agent receives an observation (either 1 or 2) at the start of the maze that it must remember until it reaches the decision node (observation 3), at which point it must turn left(1) or right(2) according to the initial observation in order to receive a reward of 4. If it chooses any other action it gets reset into the decision state and gets another observation of 0 and a reward of -0.1.

We conducted experiments on three variants of TMaze. In the first variant, the observation it receives at the start determines where the goal lies every time. In the second variant, the agent receives two different observations in the corridor with equal probability. This means that the looping ST needs to loop over both observations in any possible order. The third variant adds uncertainty to the accuracy of the starting observation along with the stochasticity in the corridor, it predicts the position of the reward with 0.8 probability. In each variant

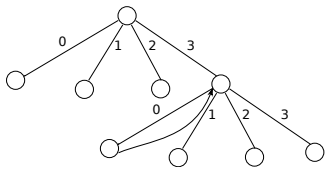


Figure 3: A reward optimal LST for the TMaze problem

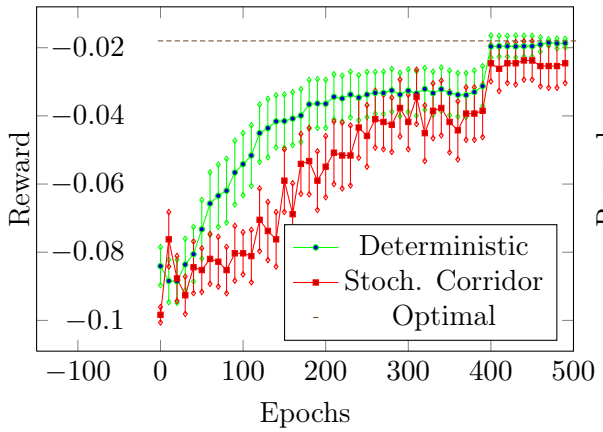


Figure 4: $LST\Phi MDP$ on TMaze length 50

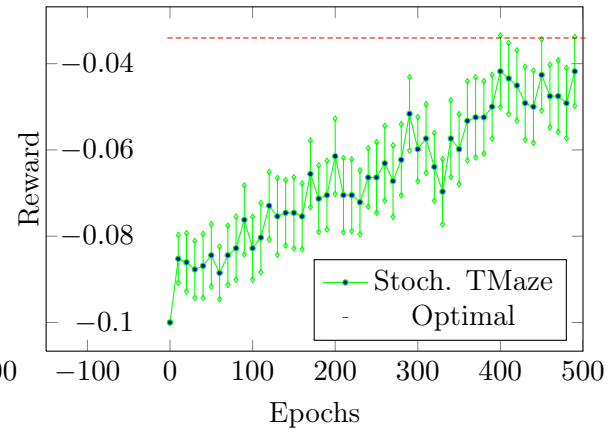


Figure 5: Stochastic TMaze length 50

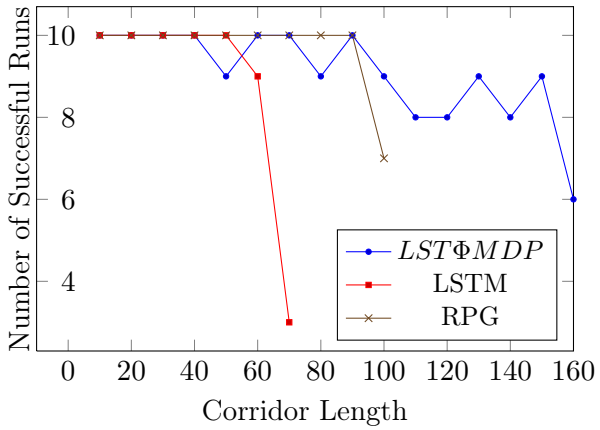


Figure 6: #optimal runs, varying corridor lengths for LSTΦ, RPG and RL-LSTM on Det. TMaze

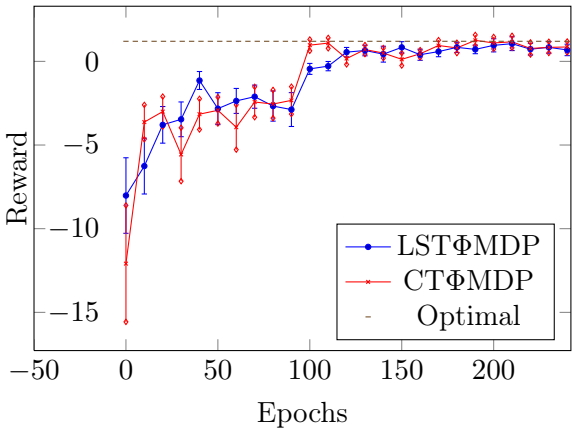


Figure 7: Comparison on Tiger with prob(listen)=0.85 for LSTΦMDP and CTΦMDP

we can adjust the length of the corridor. Note that the first variant is deterministic within a given episode, however the history itself is not deterministic since the observation received at the start of the episode is selected randomly, which is enough to prohibit deterministic approaches.

We compare our LSTΦMDP to RL-LSTM [Bakker, 2002] and Recurrent Policy Gradients (RPG) [Wierstra et al., 2007] on the deterministic TMaze. Note that we use the results from the corresponding papers; we did not implement the methods ourselves. Following the experiments in those papers, we increase the length of the corridor systematically from 10 to 100, in increments of 10. In this case each experiment was run 10 times, and we measured the number of successful runs per length. A run is said to be successful if the agent achieves the optimal policy and hence the optimal reward in at least the last 10 epochs. We used this metric to compare with other methods. All the successful runs had optimal policies from 400 epochs onward i.e. once there was no longer any ϵ-exploration. We continued to increase the corridor length until the performance of our algorithm was worse than the performance of the RPG method at length 100, which happened at corridor length 160 (6 successful runs).

Locked door. In order to show that our algorithm was useful in solving other long-term dependency problems we tested on a new domain we call the "locked door". The agent is in a room (represented by a grid). The room has a locked door and in order to leave, the agent must collect a key from a particular location. In our experiment we use a 7x7 grid with the door in the top-left corner, the key in the top-right corner and the agent starting in the location one square below the door. The agent has actions up, down, left and right and receives observations that are a binary coding of the adjacent walls. This means that states with the same wall configuration have the same observation. Bumping into a wall, collecting the key, and visiting the door have their own unique observations. The agent gets a reward -5 for bumping into a wall, +10 for visiting the door after obtaining the key and and -1 for every other timestep. The agent was given a history of 1000 random actions at the start and every run of the experiment was 1000 epochs long with each epoch being a 100 iterations as usual.

5. Analysis

In this section we analyse the results from our experiments, and explain characteristic behaviours and parameter settings. The neighbourhood function was chosen to traverse the state space slowly through the looping trees linked to a particular suffix tree, after a few experiments with larger jumps failed. Loops make smaller representations of large environments possible. The difference in cost between two adjacent trees can be quite large, since a loop can suddenly explain a very large amount of data by ignoring irrelevant sequences.

Deterministic TMaze. In the case of corridor length 50, the optimal policy has a value of -0.018. The agent reaches the optimal policy in every run once the ϵ-exploration has been turned off at 400 epochs. See Figure 4 for details. The results of the separate experiment comparing the algorithms performance on varying corridor lengths are displayed in Figure 6. Up to length 100 the agent reaches the optimal policy, with a few corridor lengths having one run stuck on traversing the corridor without every having seen the goal. Note that the algorithm does not necessarily reach the optimal tree, but finds a reward-optimal tree that contains it. In comparison, RL-LSTM [Bakker, 2002] has increasingly many suboptimal runs as the length of the corridor increases past 50. RPG [Wierstra et al., 2007] has optimal results up to length 90 but has 3 unsuccessful runs at length 100. We continue increasing corridor length until we have more than 3 unsuccessful runs at length 160. Additionally, our algorithm uses 50000 iterations (500 epochs) in all cases, while RPG uses around 2 million iterations for corridor length 100. We also tested CTΦMDP but it was not successful for corridor lengths>5. We would need a depth n suffix tree to represent a TMaze with length n. However, a looping suffix tree with optimal reward prediction is much easier to find, as shown in Figure 3 and also much smaller, leading to greater data efficiency. We did not test Echo State Networks, however from [Szita et al., 2006] we note that the method was not successful on corridor lengths greater than 25. In this environment, the optimal looping suffix tree (Figure 3) is the same regardless of the length of the corridor, since the tree simply loops over the corridor observations. Of course the exploration-exploitation problem gets harder as the corridor length increases. Despite this the systematic exploration of the agent appears to work well. We also note that in comparison to Recurrent Neural Networks (i.e. the LSTM based methods) it is relatively much simpler to interpret a looping suffix tree.

Stochastic TMaze (corridor length 50). The optimal policy in the stochastic corridor TMaze case has a value of -0.018 the same as the deterministic case. However, the agent has to loop over a new observation, and hence needs a larger tree. The task is made hard by the stochastic nature of the corridor observations. Failures occur mainly due to exploration issues (agent not finding a reward often enough) rather than problems with simulated annealing. This means that the average reward is a little lower than the optimal, however in most cases the agent did reach the optimal. In the case where the accuracy of the initial observation is 0.8, the expected reward is -0.03404. The results have more variability at each point as seen in the higher error bars, but overall the agent reaches nearly optimal reward in every run with the average of the final point being -0.04178.

Tiger. The Tiger example is interesting since it shows that the agent can still reproduce results from the regular non-looping suffix tree case. The agent achieves the optimal reward when the parameter α is set to a lower value of 0.01. Figure 7 shows that LSTΦMDP and CTΦMDP perform nearly identically on this problem.

Locked door. When the agent visits the door location there are two contexts, it either has the key or it doesn't. Remembering that it has a key is much easier with loops, since it can simply loop over observations once it has collected the key. The LSTΦMDP agent with $\alpha = 0.1$ succeeds in finding a near-optimal policy in about half the runs. CTΦMDP succeeds in learning how to avoid walls but never improves further in 1000 epochs. See Figure 8 for the graph of the near-optimal runs of LSTΦMDP.

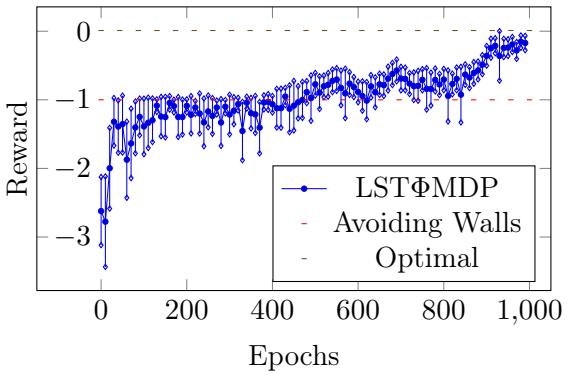


Figure 8: Locked door

General Problems. The cost function needed mild tuning of the parameter α for the experiments, generally relying on low values (especially in Tiger). This emphasises reward prediction over state prediction. Looping STs can reduce the cost of coding state sequences dramatically by looping over several observations and substantially reducing the number of states. Obviously this can lead to a bad reward coding, which should eventually cause the tree to be rejected. However, if the agent has not seen enough of the various available rewards then the reward cost may not be particularly high. This can be self-reinforcing. Bad models of the environment can result in policies that only rarely experience critical events, for example opening the door in the Tiger or Locked door problem. This means that the reward cost changes very slowly and may not ever dominate the total cost. Note that this often means that if the agent does not find the reward early on in the run, then it has not much chance of finding it later. Inspecting the failed runs for the deterministic TMaze, we see that the agent never experiences the reward or only sees it once or twice. Particularly, as the length of the maze increases both the optimism and the ϵ-exploration are insufficient to fully explore the maze.

Computational Complexity. The most time consuming part of ΦMDP is the calculation of the **Cost** of a new map. The calculation of **Cost** from the statistics is $O(|S|^2|A|) + O(|S||A||R|)$. However, since the state space changes the statistics must be recomputed. In

CTΦMDP this can be done using a pass over the history (of length n) with backtracking limited to the depth d of the tree, making the worst-case complexity $O(dn)$. However, loops can require backtracking to the start of the history making the worst-case complexity $O(n^2)$. Note that if $n < |S|^2|A|$ there are some transitions that have not been seen and can thus be ignored when calculating **Cost**, so the complexity is dominated by the $O(dn)$ or $O(n^2)$ term. In practice, the execution times are competitive to (non-looping) suffix tree based methods on environments that do not require loops. For example on Tiger, the average execution time for LSTΦMDP is 11.49s and for CTΦMDP it is 11.27s. In environments where loops matter, LSTΦMDP is much slower, for example on TMaze (length 50) an average run for LSTΦMDP is 216.93s while for CTΦMDP it is 38s. The large speed difference is because CTΦMDP remains on the (sub-optimal) minimal tree of 4 states, which results in less time spent in the annealing procedure.

6. Conclusion

We introduced looping suffix trees to the feature reinforcement learning framework [Hutter, 2009] to create an algorithm called LSTΦMDP. The experimental results show that looping suffix trees are particularly useful in representing long-term dependencies by looping over unnecessary observations. Loops allow for smaller representations leading to greater data efficiency. We outperform LSTM-based algorithms [Bakker, 2002; Wierstra et al., 2007] on TMaze. LSTΦMDP was also able to perform well on stochastic environments, which is a handicap of previous methods using looping suffix trees [Holmes and Jr., 2006; Haghighi et al., 2007]. We also replicated results of CTΦMDP [Nguyen et al., 2011] on short-term environments.

Scaling to larger domains is the future direction of our research. Nguyen et al. [2012] deal with large observation spaces by adding an unseen context state to the suffix tree, which can be problematic with regard to data efficiency. A potentially better way is function approximation techniques. An interesting idea is to use loops in instead of an unseen symbol, e.g. requiring all things unseen to be looped over. In order to deal with the added time complexity of loops, we aim to develop a neighbourhood function over *Markov* looping suffix trees only, which would reduce the complexity of finding **Cost** of a new tree to $O(n)$ by allowing us to incrementally update the state based on the new observation.

Acknowledgements. The research was partly supported by the Australian Research Council Discovery Project DP120100950.

References

B. Bakker. Reinforcement Learning with long short-term Memory. *Advances in Neural Information Processing Systems*, 2(14):1475–1482, 2002.

V. F. Farias, C. C. Moallemi, T. Weissman, and B. Van Roy. Universal Reinforcement Learning. *IEEE Transactions on Information Theory*, 56(5):2441–2454, 2010.

D. K. Haghighi, Supervised D. Precup, J. Pineau, and P. Panangaden. Learning Algorithms for Automata with Observations. Technical report, School of Computer Science, McGill University, Canada, 2007.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

M.P. Holmes and C. L. Isbell , Jr. Looping Suffix Tree-Based Inference of Partially Observable Hidden State. In *Proceedings of ICML. 2006*, pages 409–416. Apple Deve1oper Press, 2006.

M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, Berlin, 2005.

M. Hutter. Feature Reinforcement Learning: Part I: Unstructured MDPs. *Journal of Artificial General Intelligence*, 1:3–24, 2009.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

J. S. Liu. *Monte Carlo Strategies in Scientific Computing.* Springer, 2nd edition, 2008.

M. M. H. Mahmud. Constructing States for Reinforcement Learning. In J. Fürnkranz and T. Joachims, editors, *International Conference on Machine Learning*, pages 727–734. Omnipress, 2010.

R. A. McCallum. Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395. Morgan Kaufmann, 1995.

P. Nguyen, P. Sunehag, and M. Hutter. Feature Reinforcement Learning in Practice. In *Proc. 9th European Workshop on Reinforcement Learning (EWRL-9)*, volume 7188 of *LNAI*, pages 66–77. Springer, September 2011.

P. Nguyen, P. Sunehag, and M. Hutter. Context Tree Maximizing Reinforcement Learning. In Jörg Hoffmann and Bart Selman, editors, *Proc. of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1075–1082. AAAI Press, 2012.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

J. J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14(5):465–471, 1978.

P. Sunehag and M. Hutter. Consistency of Feature Markov Processes. In *Proc. 21st International Conf. on Algorithmic Learning Theory (ALT'10)*, volume 6331 of *LNAI*, pages 360–374, Canberra, 2010. Springer, Berlin.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1998.

I. Szita and A. Lörincz. The Many Faces of Optimism: A Unifying Approach. In *Proc. 12th International Conference (ICML'08)*, volume 307, Helsinki, Finland, 2008.

I. Szita, V. Gyenes, and A. Lőrincz. Reinforcement Learning with Echo State Networks. In *Proceedings of the 16th international conference on Artificial Neural Networks - Volume Part I*, ICANN'06, pages 830–839, Berlin, Heidelberg, 2006. Springer-Verlag.

J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A Monte Carlo AIXI Approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *Proceedings of the 17th international conference on Artificial neural networks (ICANN'07)*, pages 697–706, Berlin, Heidelberg, 2007. Springer-Verlag.

Appendix A. Table of constants

Experiment	α	*epochs*	*init-history*	*stop-explore*	*max-reward*	*anneal-temp*
Det Tmaze	0.1	500	20	100	400	1
Stoch Tmaze	0.1	500	100	100	400	1
Tiger	$1 \cdot 10^{-2}$	500	100	100	400	5
Locked Door	$1 \cdot 10^{-2}$	1,000	100	1,000	900	10

Table 1: The table lists the various constants used for each experiment. Common to all experiments were the maximum number of steps for a single annealing run capped at 50, the value of k in the exponential cooling scheme at 0.005, $\epsilon = 0.1$ and $\gamma = 0.99$. α is a parameter of the **Cost** that controls the balance between state and reward code-lengths, *anneal-temp* refers to the temperature T in the cooling schedule, *init-history* is the number of initial random actions performed by the agent, *stop-explore* is the epoch beyond which the agent no longer uses ϵ-exploration and *max-reward* is the value of the reward given to the garden-of-eden state in the extended MDP for all actions.

Appendix B. Algorithms

Algorithm 1: A high-level view of the generic ΦMDP algorithm.

Initialise ϕ ;
Input : Environment $Env()$;
Initialise history with observations and rewards from $t = init\_history$ random actions;
Initialise M to be the number of timesteps per epoch;
while *true* **do**
  $\phi = SimulAnneal(\phi, h_t)$;
  $s_{1:t} = \phi(h_t)$;
  $\pi = FindPolicy(s_{1:t}, r_{1:t}, a_{1:t-1})$;
  for $i = 1, 2, 3, ...M$ **do**
    $a_t \leftarrow \pi(s_t)$;
    $o_{t+1}, r_{t+1} \leftarrow Env(h_t, a_t)$;
    $h_{t+1} \leftarrow h_t a_t o_{t+1} r_{t+1}$;
    $t \leftarrow t + 1$;
  end
end
return $[\phi, \pi]$

Algorithm 2: Get current state given an observation sequence and a looping ST

getCurrentState(Observation sequence $o_{1:t}$);
currentNode = root;
$i = t$;
while *currentNode is not a state* **do**
    if *currentNode has a loop* **then**
        currentNode = node at the end of the loop;
    else
        currentNode = the o_i-th child of *currentNode*;
        $i = i - 1$;
    end
    if $i \leq 0$ **then**
        return s^{empty};
    end
end
return *currentNode*

Algorithm 3: getNeighbour() method for looping ST

Input: num_ obs : number of observations, statelist : list of states in current tree, looplist : list of loops in current tree
state = random state from current statelist;
Let c be a random number in {1,2,3,4};
if $c == 1$ *and (num_states > num_obs) and every sibling of the current state is also a state* **then**
    merge(*state*);
else if $c == 2$ *and (num_states* $> 2 \times$ *num_obs)* **then**
    if $uniform(0, 1) > 0.5$ *and looplist* $\neq$ *{}* **then**
        state = random state from looplist;
    end
    addLoop(*state*);
else if $c == 3$ *and looplist* $\neq$ {} **then**
    state = random state from looplist;
    removeLoop(state)
else
    split(state);
end

JMLR: Workshop and Conference Proceedings 24:25–42, 2012 10th European Workshop on Reinforcement Learning

Planning in Reward-Rich Domains via PAC Bandits

Sergiu Goschin SGOSCHIN@CS.RUTGERS.EDU
Ari Weinstein AWEINST@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Erick Chastain ERICKC@CS.RUTGERS.EDU
Rutgers University, Piscataway, NJ 08854 USA

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

In some decision-making environments, successful solutions are common. If the evaluation of candidate solutions is noisy, however, the challenge is knowing when a "good enough" answer has been found. We formalize this problem as an infinite-armed bandit and provide upper and lower bounds on the number of evaluations or "pulls" needed to identify a solution whose evaluation exceeds a given threshold r_0. We present several algorithms and use them to identify reliable strategies for solving screens from the video games *Infinite Mario* and *Pitfall!* We show order of magnitude improvements in sample complexity over a natural approach that pulls each arm until a good estimate of its success probability is known.

Keywords: Multi-armed bandits, Planning under uncertainty, Stochastic optimization

1. Introduction

Consider the following trivial problem. A huge jar of marbles contains some fraction ρ of black (success) marbles and the rest white (failure) marbles. We want to find a black marble as quickly as possible. If the black marbles are sufficiently plentiful in the jar, the problem is simple: Repeatedly draw marbles from the jar until a black one is found. The expected sample complexity is $\Theta(1/\rho)$. This kind of generate-and-test approach is simple, but can be extremely effective when solutions are common—for example, finding an *un*satisfying assignment for a randomly generated CNF formula is well solved with this approach.

The corresponding noisy problem is distinctly more challenging. Imagine the marbles in our jar will be used to roll through some sort of obstacle course and (due to weight or balance or size) some marbles are more successful at completing the course than others. If we (quickly) want to find a marble that navigates the obstacle course successfully at least $r_0 = 25\%$ of the time, how do we best allocate our test runs on the course? When do we run another evaluation of an existing marble and when do we grab a new one out of the jar? How do we minimize the (expected) total number of runs while still assuring (with high probability) that we end up with a good enough marble?

We formalize this problem as an infinite-armed bandit and provide a lower bound on the number of arm pulls needed to find an arm with payoff above r_0. We describe and analyze several algorithms that are close to this lower bound. We include comparisons of these algorithms using data from two well known video games—*Infinite Mario* and *Pitfall!*

2. Infinite-Armed Bandit Problem

We define an arm as a probability distribution (D_a) over possible reward values within a bounded range $[r_{\min}, r_{\max}]$. When an arm is *pulled*, it returns a reward value (sampled from D_a). One arm a is preferred to another a' if it has a higher expected reward value, $E_{r_a \sim D_a}[r_a] > E_{r_{a'} \sim D_{a'}}[r_{a'}]$. Arms are sampled from an *arm space* S, possibly infinitely large. The distribution D over the arm space defines an infinite-armed bandit problem IB(D).

We seek algorithms that take a reward level r_0 as input and attempt to minimize the number of pulls needed to identify an arm with expected value of r_0 or more. This *sample complexity* has a dependence on D and r_0, as it may be likely or unlikely to encounter an arm with high enough reward. Specifically, define $\rho = P_{a \sim D}(E_{r_a \sim D_a}[r_a] \geq r_0)$ as the probability of sampling a "good enough" arm. We assume the domain is reward rich—specifically, that ρ is bounded away from zero. By allowing the agent to aspire to any reward level, this definition of the performance measure is akin to the earlier work of Wang et al. [2008], which is constrained to finding the optimal reward value.

Formally, we define an (ϵ, δ, r_0)-correct algorithm ALG for an IB(D) problem to be an algorithm that after a number of samples $T(\epsilon, \delta, r_0, D)$ (that is finite with probability 1) returns an arm a with expected value $E[r_a] \geq r_0 - \epsilon$ with probability at least $1 - \delta$. This formalism is a variation of the PAC-Bandit model [Even-Dar et al., 2002] to an infinite number of arms.

Regarding the motivation of our model, we target a class of optimization problems where standard local search approaches fail: in particular, relations between arms are either not predictive of relations between their associated reward values or such relations can be ignored without sacrificing much in terms of the number of evaluations needed to get an approximately optimal solution. We view planning as an optimization problem, with every possible plan being an 'arm' in the infinite bandit model described above.

Our claim is that some apparently hard planning problems can be solved via a sampling and testing approach that cannot be solved by algorithms that depend on the existence of local structure for search. We documented this phenomenon in the video games *Infinite Mario* and *Pitfall!* where policies can be very similar but have vastly different outcomes (one different action in a long sequence can lead the agent to failure as opposed to successfully completing a level).

In Section 3, we introduce the problem by considering the simple case in which arms are deterministic, $P(E[r_a] = r_a) = 1$. In Sections 4 and 5, we address the more general case of arms with stochastic rewards. We will mostly focus on arms with a Bernoulli distribution D_a over rewards, but our bounds are extendable to arbitrary distributions with bounded support.

2.1. Related Work

The framework used in our work is closely related to several models from the multi-armed bandit literature. While the case of a finite number of arms is well understood [Auer et al., 2002], in the past few years, papers discussing bandits with infinitely many arms have appeared [Kleinberg et al., 2008; Bubeck et al., 2008]. Our work extends the PAC-Bandit setting [Even-Dar et al., 2002] to an infinite number of arms [Wang et al., 2008], but we replace the assumption about the probability of drawing a near optimal arm with a given

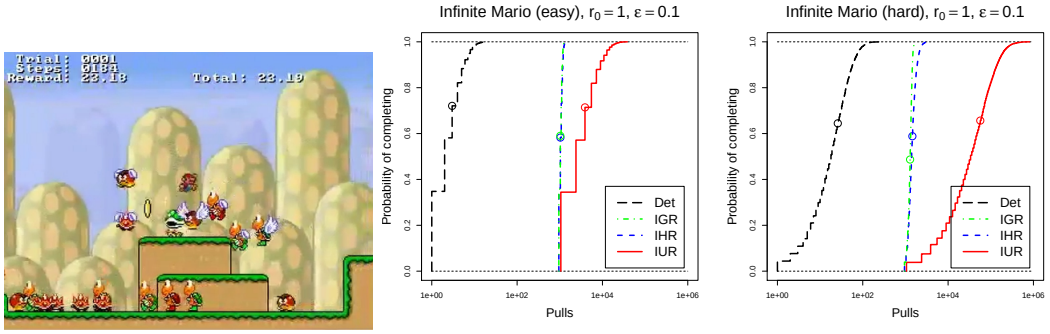


Figure 3.1: A screenshot of *Infinite Mario* and plots of the distribution of the sample complexity for an algorithm that pulls each arm once (x-axis log-scale). The distributions are plotted for 2 of the 50 *Infinite Mario* levels corresponding to the first (easy) and third (hard) quartiles. See the **Algorithms** section for more details.

target threshold. It is worth noting here that a typical assumption in the literature on continuous-armed bandits (one class of infinite-armed bandits) is that the structure in the arm space induces structure in the space of expected rewards of the arms. The mean-reward function is usually assumed to be Lipschitz and algorithms are created that take advantage of this smoothness assumption. In some cases, including our example problems, this assumption does not hold and algorithms that depend on it can fail.

Regarding our chosen performance measure, we depart from the often used cumulative regret setting and choose a setting that only requires an agent to have a good answer after some finite experimentation. This setting is related to some recent work [Bubeck et al., 2009; Audibert et al., 2010], but we chose the PAC-Bandit performance measure instead of the simple regret setting defined in the aforementioned work. Besides existing PAC-Bandit algorithms, we also draw algorithmic inspiration from the Hoeffding/Bernstein races framework [Maron and Moore, 1997; Heidrich-Meisner and Igel, 2009; Mnih et al., 2008], which we extend to our infinite-armed bandit setting, and from the empirical success of the Biased Robin algorithm from the Budgeted Learning framework [Madani et al., 2003], about which our analysis may offer some insight. Applicationwise, we target policy search and planning under uncertainty.

3. Experiment: *Infinite Mario*

Our first experiment used a version of *Infinite Mario* (a clone of the *Super Mario* video game, see the left panel of Figure 3.1) that was modified for the Reinforcement Learning Competition [Whiteson et al., 2010]. It was also used for other competitions [Togelius et al., 2010] and it is considered to be an interesting benchmark for planning and learning. The game is deterministic and gives us an opportunity to present a natural problem that illustrates the "reward richness" phenomenon motivating our work.

We treated starting screens in *Infinite Mario* as bandits, where each arm encodes an action sequence 50-steps long. In the experiments, the agent's goal was to reach a threshold

on the right side of the initial screen. The action set of the agent was restricted by removing the backward action (unnecessary for solving any level), resulting in 8 total actions and an arm space of size 8^{50}. Action sequences were tested in the actual game, assigning rewards of -1 if the agent was destroyed, 0 if it did not reach the goal in 50 steps, and a value of $100-t$, otherwise (where t was the number of steps taken before reaching the goal). Since the domain is deterministic, the agent simply sampled uniformly at random new arms until one was found with reward greater than 0. Sampling uniformly from the space of arms induces an unknown distribution D over the space of possible rewards, which is the distribution over arms described in section 2.

The average number of pulls needed to find a strategy for completing the first screen over a set of 50 levels ranged from 1 to 1000, with a median of 7.7 pulls and a mean of 55.7 (due to a few very difficult levels). Thus, testing just a handful of randomly generated action sequences was sufficient to find a successful trajectory in this game. The performance of the method is conveyed by the black (leftmost) lines in the plots in Figure 3.1 (the algorithms corresponding to the other curves are described in the next sections). The results show that nearly all screens were solved in well under 100 samples.

As an extension to this experiment, we 'chained' trajectories together to completely solve each of the 50 levels (as opposed to just treating the starting screens). Using a cap of 3000 pulls for each screen in a level, this simple method was able to complete 40 out of the 50 levels. (See the auxiliary material for links to videos of the discovered solutions.)

4. Sample Complexity Lower Bound

When the sampled arms are not deterministic, the problem of allocating pulls is more complex. The agent is faced with a choice between getting better accuracy estimates of previously sampled arms versus sampling new arms to find one with higher value. In the following, we state and prove a lower bound on the expected sample complexity of a correct algorithm for the case of Bernoulli arms.

Theorem 1 *Any (ϵ, δ, r_0)-correct algorithm for an $IB(D)$ problem has an expected sample complexity of at least $T(\epsilon, \delta, r_0, D) = \Omega(\frac{1}{\epsilon^2}(\frac{1}{\rho} + \log \frac{1}{\delta}))$.*

Due to space constraints, we leave the proof of the theorem for the auxiliary material (Appendix C). The key element of the proof is the use of Theorem 13 of Mannor et al. [2004], which states a lower bound on the expected sample complexity of any correct algorithm in the PAC-Bandit setting when the expected rewards of the arms are known up to a permutation.

5. Algorithms

When ρ is known, the infinite-armed bandit problem can be reduced to the classic PAC-Bandit setting and algorithms for that setting can be applied [Even-Dar et al., 2002; Mannor et al., 2004]. Specifically, if one samples a number of arms such that, with high probability, at least one has high expected reward (at least r_0) and then uses a version of Median Elimination [Mannor et al., 2004] to solve the resulting finite PAC-Bandit problem, the sample complexity bound will be $O(\frac{1}{\rho\epsilon^2} \log \frac{1}{\delta})$, roughly a factor of $\log \frac{1}{\delta}$ away from the lower bound. (Details available in Appendix F).

For the more general case when ρ is not known, this section introduces three new algorithms: one that is an incremental version of a naïve strategy [Even-Dar et al., 2002], one inspired by the Hoeffding Races framework [Maron and Moore, 1997; Heidrich-Meisner and Igel, 2009], and another that uses ideas from ballot-style theorems for random walks [Addario-Berry and Reed, 2008] to quickly reject unpromising arms.

All the algorithms we introduce have the structure of the **Generic Algorithm** (Algorithm 1): They sample an arm, make a bounded number of pulls for the arm, check if the arm should be accepted (and in this case, stop and return the arm) or rejected (sample a new arm from D and repeat). The decision rule for acceptance / rejection and when it can be applied is what differentiates the algorithms.

Algorithm 1:
GenericAlgorithm $(\epsilon, \delta, r_0, \mathit{RejectionFunction})$

$i = 1$, $\mathit{found} =$ **false**
for $i = 1, 2, \ldots$ **do**
  Sample a new arm $a_i \sim D$
  $\mathit{decision} = \mathit{RejectionFunction}(a_i, i, \epsilon, \delta, r_0)$
  if $\mathit{decision} = \mathit{ACCEPT}$ **then**
    return a_i
  end if
end for

5.1. Iterative Uniform Rejection (IUR)

Iterative Uniform Rejection (Algorithm 2) is an incremental version of the naïve strategy by Even-Dar et al. [2002]. The algorithm pulls an arm a fixed number of times to decide with high confidence if the arm has an expected reward less than or greater than $r_0 - \epsilon$. It samples arms in this manner until one with an estimated mean reward of at least $r_0 - \epsilon$ is found.

Algorithm 2: IterativeUniformRejection (ϵ, δ, r_0)

return $\mathit{GenericAlgorithm}(\epsilon, \delta, r_0, \mathit{UniformRejection})$

Function $\mathit{UniformRejection}(a, i, \epsilon, \delta, r_0)$

$n_0 = \frac{4}{\epsilon^2} \ln \frac{2i^2}{\delta}$
for $k = 1, 2, \ldots, n_0$ **do**
  Pull the arm to get reward $r_k \sim a$
end for
$\hat{r}_a = \frac{1}{n_0} \sum_{k=1}^{n_0} r_k$
if $\hat{r}_a < r_0 - \frac{\epsilon}{2}$ **then**
  return REJECT
end if
return ACCEPT

Theorem 2 *Algorithm* **Iterative Uniform Rejection** *is an* (ϵ, δ, r_0)*-correct algorithm for any IB*(D) *problem and its expected sample complexity is upper bounded by* $O(\frac{1}{\rho\epsilon^2} \log \frac{1}{\rho\delta})$.

The **IUR** algorithm is simple, correct, and achieves a bound close to the lower bound for the problem. We leave the proof of the theorem for the auxiliary material (Appendix A).

5.2. Iterative Hoeffding Rejection (IHR)

One problem with **IUR** is that it is very conservative in the sense of taking a large number of samples for each arm (with the dominant term being $\frac{1}{\epsilon^2}$). The algorithm does not take advantage of the fact that it may be possible to tell that an arm is highly unlikely to be better than $r_0 - \epsilon$ long before all n_0 pulls are performed. As a result, the algorithm wastes pulls deciding precisely *how* good or bad the arm is, when it just needs to know *whether* it is good or bad. **Iterative Hoeffding Rejection** (Algorithm 3) exploits the situation in which Δ_i, the difference between the expected reward of arm a_i and r_0, might be larger than ϵ, so an unpromising arm could be rejected before reach the decision threshold from **IUR**—an insight from the Hoeffding Races framework [Maron and Moore, 1997]. The main idea of the **IHR** algorithm is to maintain confidence intervals built using the Hoeffding bound around the empirical average for the sampled arm and to reject the arm as soon as the upper bound of the confidence interval drops below a certain threshold. If this threshold is not reached after a particular number of pulls, the arm is accepted.

Algorithm 3: IterativeHoeffdingRejection (ϵ, δ, r_0)

return *GenericAlgorithm*$(\epsilon, \delta, r_0,$ *HoeffdingRejection*$)$

Function HoeffdingRejection $(a, i, \epsilon, \delta, r_0)$

Let $\delta_0 = \frac{\delta}{2i^2}, j = 1$
$n_0 = \frac{4}{\epsilon^2} \ln \frac{1}{\epsilon\delta_0}$
for $j = 1, 2, ..., n_0$ **do**
  $r_j \sim a$; $\hat{r}_j^a = \frac{1}{j} \sum_{k=1}^{j} r_k$
  if $\hat{r}_j^a < r_0 - \sqrt{\frac{2\log(2j^2/\delta_0)}{j}}$ **then**
    return *REJECT*
  end if
  $j = j + 1$
end for
return *ACCEPT*

Theorem 3 *Algorithm* **Iterative Hoeffding Rejection** *is an* (ϵ, δ, r_0)*-correct algorithm for any IB*(D) *problem and its expected sample complexity is upper bounded by* $O(\frac{1}{\rho\epsilon^2} \log \frac{1}{\epsilon\rho\delta})$.

We leave the proof of the theorem for the auxiliary material (Appendix D). The analysis of the algorithm is tight in the worst case (consider the domain used to prove the lower bound in Theorem 1). Nevertheless, as we will show in the experiments section, the algorithm has a much better practical behavior than **IUR**. The reason can be understood by emphasizing

the differences between the arms in the upper bound for **IHR**. Define $\Delta_a = E[r_a] - r_0$ to be a random variable that encodes the difference between r_0 and the expectation of an arm sampled from D, and define Δ_- such that $\frac{1}{\Delta_-^2} = E[\frac{1}{\max(\epsilon^2, \Delta_a^2)} | \Delta_a < 0]$. ($\Delta_-$ is lower bounded by ϵ and it encodes the relevant difference for rejecting an arm if the arm has an expectated value smaller than r_0.) It can then be shown (see the proof in Appendix E) that:

Theorem 4 *The expected sample complexity of* **Iterative Hoeffding Rejection** *is upper bounded by* $O((\frac{1}{\epsilon^2} + \frac{1}{\rho\Delta_-^2}) \log \frac{1}{\epsilon\rho\delta})$ *with probability at least* $1 - \delta$.

For situations where Δ_- is larger than ϵ, the number of pulls needed to classify a 'bad' arm is actually much smaller (ignoring log factors, the difference is between $O(\frac{1}{\Delta_-^2})$ and $O(\frac{1}{\epsilon^2})$ pulls per arm). This difference is the reason why the algorithm has the potential to be much more useful then **IUR** in practice.

The algorithm can be improved by accepting an arm faster if the lower bound of the confidence interval for the empirical average of a particular arm becomes larger than r_0. Another immediate extension is to use Bernstein bounds [Mnih et al., 2008] instead of Hoeffding bounds to take advantage of the case where the distributions associated with each arm have low variance. We leave these straightforward improvements for an extended version of the paper.

5.3. Iterative Greedy Rejection (IGR)

Both **IUR** and **IHR** carefully decide whether an arm is good or bad before deciding to reject. As a consequence, with high probability, the first time they encounter a 'good' arm, they accept it. This strategy is reasonable in general, but it has one disadvantage: when the proportion of good arms is relatively low, these algorithms will spend a long time sampling and discarding bad arms. In some cases, a better strategy could be to reject faster—without being sure with high probability whether an arm is good or bad. This approach is permissible in our framework since failure to accept a good arm is only penalized in terms of sample complexity and does not compromise correctness. Related examples of empirically successful algorithms that quickly reject are Biased Robin [Madani et al., 2003] in the Budgeted Bandit setting and evolutionary algorithms in noisy settings [Fitzpatrick and Grefenstette, 1988].

We next describe an algorithm that implements such a strategy by constraining the empirical average of each sampled arm to stay above a certain threshold for it not to be rejected. While the worst-case bound we prove is a factor of $O(\frac{1}{\epsilon})$ worse than that of the other algorithms, the algorithm is strong experimentally for real data sets, and is still polynomial in the worst case.

Synthetic experiments that investigate various settings of the parameters (see Appendix G) indicate the upper bound is loose and point to a tighter upper bound that is similar to the one from Theorem 4 (with an advantage in some practical situations). We were unable to prove this bound though, and thus leave the improvement of the bound for **IGR** as an open problem.

Theorem 5 *Algorithm* **Iterative Greedy Rejection** *is an* (ϵ, δ, r_0)*-correct algorithm for any IB*(D) *problem and its expected sample complexity is upper bounded by* $O(\frac{1}{\rho\epsilon^3} \log \frac{1}{\epsilon\rho\delta})$*, if*

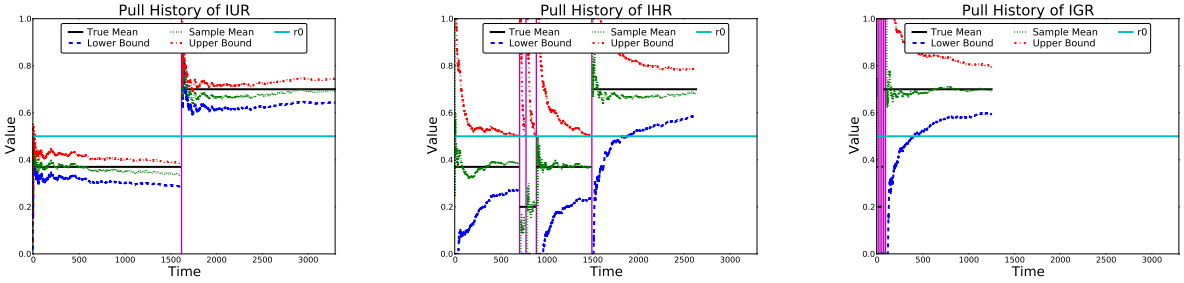


Figure 5.1: An illustration of the behavior of three algorithms in a simple domain.

Algorithm 4: IterativeGreedyRejection (ϵ, δ, r_0)

1: **return** *GenericAlgorithm*$(\epsilon, \delta, r_0,$ *GreedyRejection*$)$

Function GreedyRejection $(a, i, \epsilon, \delta, r_0)$

The function is identical to HoeffdingRejection with the exception of Line 5:
5′: **if** $\hat{r}_j^a < r_0 - \frac{\epsilon}{2}$ **then**

$r_0 > \epsilon$ (if $r_0 \leq \epsilon$, an algorithm can simply return the very first arm, which is guaranteed to be 'good' because its reward r satisfies $r \geq 0 \geq r_0 - \epsilon$.)

The proof can be found in Appendix B and it is one of the core contributions of this paper as it uses a new proving technique to upper bound the sample complexity.

The key idea of the proof is to interpret the evolution of the empirical average for a good arm as a random walk and then apply a ballot-style theorem [Addario-Berry and Reed, 2008] to bound the probability that the average will always be higher than a fixed threshold. Doing so allows us to lower bound the probability of accepting a good arm, which is the key to upper bounding the expected sample complexity.

One possible looseness in the analysis comes from ignoring the fact that in non-worst case scenarios, bad arms will be rejected before $n_{\max}(i)$ samples (which is why this strategy is successful in practice).

6. Illustrating the Algorithms

Figure 5.1 illustrates the behavior of all three algorithms in a simple setting where arms can take on three different values (0.20, 0.37, and 0.70) with equal probability ($r_0 = 0.5$, $\epsilon = 0.1$, $\delta = 0.01$). Thus, the goal is to recognize when a 0.70 arm is drawn. We note that this experiment is not meant to compare the algorithms empirically, but only to give insights about their different strategies for a simple example.

The first plot shows the behavior of IUR. Every time it draws an arm, it uses 1600 pulls to accurately estimate its payoff before deciding whether to accept or reject. In the plot, we show the algorithm's estimate of the mean of each arm it draws as it accumulates more data. We also plot a fixed interval of $\epsilon/2$ around this mean, which is the confidence interval of the estimate after 1600 pulls.

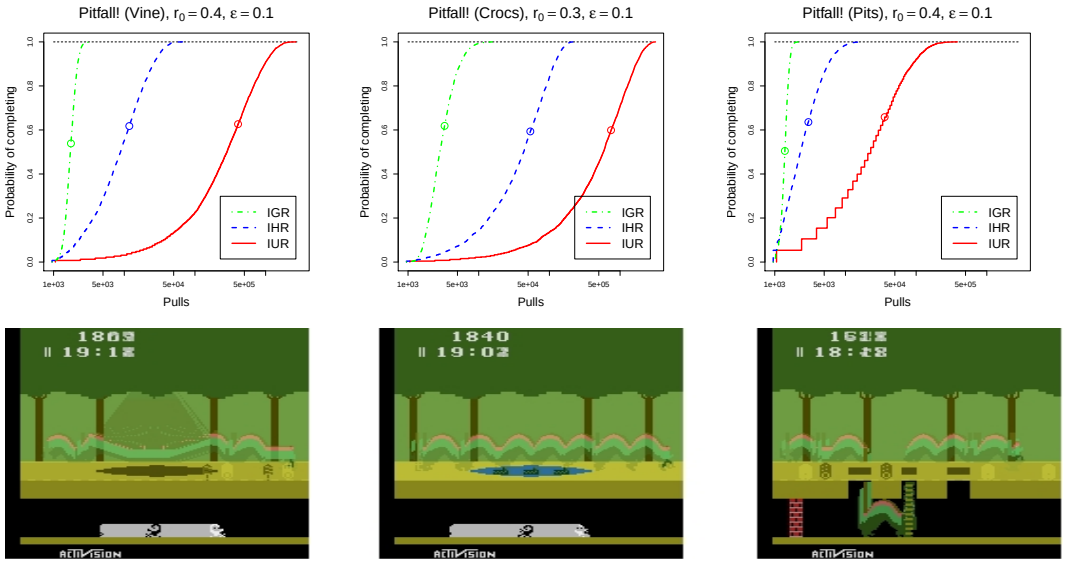


Figure 6.1: Plot of distribution of the sample complexity (pulls needed) for **IGR** (Iterative Greedy Rejection), **IHR** (Iterative Hoeffding Rejection) and **IUR** (Iterative Uniform Rejection) over a set of 5000 repetitions. The distributions are plotted for 3 different *Pitfall!* levels (shown along with a representation of a successful policy in the lower half of the figure). Average sample complexity for each algorithm is marked with a circle. All experiments used $\delta = 0.01$.

The second plot depicts the behavior of IHR, which maintains confidence intervals on the sample mean throughout. If the upper confidence interval drops below r_0, the arm is rejected. The result in the experiment is that **IHR** is able to reject 4 arms in the time it takes **IUR** to reject 1.

The third plot provides the analogous illustration of IGR. Note that this algorithm rejects an arm immediately if it fails on the first pull, and the sample mean must remain above r_0 long enough to verify the arm's payoff. The plot shows that the algorithm is able to test many different arms very quickly, occasionally discarding good arms. Ultimately, it settles in on a high scoring arm and evaluates it exhaustively.

7. Experiment: *Pitfall!*

Originally developed for the Atari 2600, *Pitfall!* is a game where the objective is to guide the protagonist through the jungle collecting treasure while avoiding items that harm him. In our experiments, interaction with the game was done via an emulator [JSt, 2008], which was modified to allow for software control. In the experiment, the agent's goal was defined simply as arriving at the right side of the screen on the top tier (some levels can be finished on the lower tier). The evaluation used the same 8 actionsfrom the *Infinite Mario* experiment. Stochasticity was added by randomly changing the joystick input to a centered joystick with no button press 5% of the time.

In this game, constructing a policy as a mapping from states to actions is difficult because it is unclear exactly what state representation to use. The Atari 2600 has 128 bytes of RAM, which means the actual size of the state space can be 8^{128}, much too large to effectively plan in directly. Treating the game as a collection of objects greatly simplifies the problem and has been used in *Pitfall!* for learning the dynamics of the first screen and navigating it successfully [Diuk et al., 2008], but requires prior domain knowledge to define what kind of interactions can occur between objects.

Because the issue of state in *Pitfall!* is problematic, one approach to planning in this domain is to not factor in state at all but to execute action sequences (which are policies) blindly (conditioned only on time, as opposed to state). The search space for sequences of 500 actions is 8^{500} possible plans. However, on average far fewer than 8^4 of the possible sequences actually need to be sampled uniformly at random before a successful one is found. This result is surprising, as the more difficult screens do not tolerate errors of more than a couple of pixels of placement. The success indicates that, like *Infinite Mario*, *Pitfall!* is reward rich. Figure 6.1 illustrates the results of running the three algorithms presented in this paper on the *Pitfall!* levels with stochasticity introduced. In all three cases, the random-walk-based **IGR** outperformed the races-based **IHR**, which outperformed the highly conservative **IUR** by a very large margin. The percentage improvement was greatest for the most challenging levels (Vine and Crocs). The same pattern can also be seen in Figure 3.1, where the algorithms were used on a deterministic domain without modification. Note that the deterministic strategy used for *Infinite Mario* is not successful in *Pitfall!* because of the noise we introduced. The deterministic algorithm assumes an arm is good if it is successful on the first pull, which can be very misleading. In Crocs, for example, the deterministic strategy results in over 90% of the runs erroneously returning a bad arm.

Of all the advantages of the infinite-armed bandit algorithms discussed here, the most significant may be the weak assumptions that are made: the only requirement is the probability of sampling a good enough arm be nonzero. It is thus an important topic of future research to compare the strategies from this paper with local search strategies and planning algorithms that are designed to take advantage of relations between arms or policies [Kleinberg et al., 2008; Bubeck et al., 2008; Bubeck and Munos, 2010].

8. Conclusion

We introduced an infinite-armed bandit framework that is tailored to optimization problems to which local search cannot be applied. It is closely related to the finite PAC multi-armed bandit model. We presented an almost-tight lower bound and three algorithms that solve the problem and provided analyses (including a novel random-walk-based method) proving that these algorithms achieve polynomial sample complexity bounds. We showed how a decision maker can balance between allocating pulls to get high-accuracy estimates and sampling new arms to find ones with higher expected rewards.

The framework models applications where good solutions are plentiful—where a good arm can be found by random sampling. It was shown that some non-trivial planning problems (such as two encountered in established video games) can be solved handily by exploiting this insight, even in the face of stochastic outcomes.

References

Jstella project, atari 2600, 2008. URL http://jstella.sourceforge.net/.

L. Addario-Berry and B. A. Reed. Ballot theorems, old and new. In *Horizons of Combinatorics*. Springer Berlin Heidelberg, 2008.

Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. Best arm identification in multi-armed bandits. In *COLT*, 2010.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *ML*, 2002.

Sébastien Bubeck and Rémi Munos. Open loop optimistic planning. In *Proceedings of the 23rd Annual Conference on Learning Theory (COLT)*, 2010.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. Online optimization in x-armed bandits. In *NIPS*, 2008.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *ALT*, 2009.

Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *ICML*, 2008.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *COLT*, 2002.

J. Michael Fitzpatrick and John J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 1988.

Verena Heidrich-Meisner and Christian Igel. Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *ICML*, 2009.

Olav Kallenberg. Ballot theorems and sojourn laws for stationary processes. *The Annals of Probability*, 1999.

Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *ACM Aymposium on Theory of Computing*, 2008.

Omid Madani, Daniel J. Lizotte, and Russell Greiner. Budgeted learning, part 1: The multi-armed bandit case. Technical report, University of Alberta, 2003.

Shie Mannor, John N. Tsitsiklis, Kristin Bennett, and Nicolò Cesa-Bianchi. The sample complexity of exploration in the multi-armed bandit problem. *JMLR*, 2004.

Oded Maron and Andrew Moore. The racing algorithm: Model selection for lazy learners. In *Artificial Intelligence Review*, 1997.

Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. Empirical bernstein stopping. In *ICML*, 2008.

Lajos Takacs. *Combinatorial methods in the theory of stochastic processes*. Wiley New York, 1967.

Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 mario AI competition. In *IEEE CEC 2010*, 2010.

Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In *NIPS*, 2008.

Shimon Whiteson, Brian Tanner, and Adam White. The reinforcement learning competitions. *AI Magazine*, 2010.

Appendix A. Proof of Theorem 2

Proof

Sample Complexity. We will first show that there is a constant lower bound on the probability of the algorithm stopping, which will in turn help us show the expected sample complexity is finite. Let A_i be the event of accepting the i'th sampled arm ($i \in 1, 2, ...$), conditioned on rejecting the first $i-1$ arms. Let N be a random variable that stands for the number of arms sampled until the algorithm returns an arm, and SC be a random variable that stands for the sample complexity. Note that $P(A_i) = P(\text{accept arm } a_i | a_i \text{ is 'good'})P(a_i \text{ is 'good'}) + P(\text{accept arm } a_i | a_i \text{ is 'bad'})P(a_i \text{ is 'bad'}) \geq P(\text{accept arm } a_i | a_i \text{ is 'good'})P(a_i \text{ is 'good'}) \geq (1 - \frac{\delta}{2i^2})\rho \geq \frac{\rho}{2}, \forall i \in 1, 2, \ldots$ (where the third inequality holds due to an application of the Hoeffding bound). Thus $E[N] \leq \frac{2}{\rho}$ (by the properties of the geometric distribution, where A_i stands for "success").

The expected sample complexity is $E[SC] = E[\sum_{i=1}^{N} \frac{4}{\epsilon^2} \log \frac{2i^2}{\delta}] \leq \frac{4}{\epsilon^2} E[N \log \frac{2N^2}{\delta}] \leq \frac{8}{\epsilon^2}(E[N \log N] + \frac{1}{\rho} \log \frac{2}{\delta})$ (with the right hand side of the first equality determined by the Hoeffding bound). The expectation for sample complexity is taken with respect to both sampling the arms from D and noise in the pulls themselves. Now, $E[N \log N] \leq \sqrt{E[N^2]E[\log^2 N]} \leq \sqrt{\frac{4}{\rho^2} E[\log^2 N]}$ (where the first inequality is Cauchy-Schwarz and the second is due to the properties of the geometric distribution). To bound $E[\log^2 N]$ we use the fact that the function $\log^2(x)$ is concave for $x \geq 3$ (assuming a natural logarithm) and we apply Jensen's inequality to get $E[\log^2 N] \leq \log^2 E[N] \leq \log^2 \frac{2}{\rho}$.

So, for $E[N] \geq 3$, $E[N \log N] \leq \frac{2}{\rho} \log \frac{2}{\rho}$. For $E[N] < 3$, $E[N \log N] \leq E[N^2] < 9$. As a consequence, $E[N \log N] \leq \max(9, \frac{2}{\rho} \log \frac{2}{\rho})$ and the bound follows.

Correctness. The algorithm will stop (with probability 1, since its expected sample complexity is finite) and recommend either a 'good' arm (with reward $r \geq r_0 - \epsilon$) or a 'bad' one (reward $r < r_0 - \epsilon$). The failure probability is $P(\text{failure}) \leq P(\bigcup_{i \geq 1} \{\text{incorrect recommendation at step } i\}) \leq \sum_{i \geq 1} \frac{\delta}{2i^2} \leq \delta$ (the second inequality follows via the Hoeffding bound given the number of samples $n_0(i)$ for each arm). ■

Appendix B. Proof of Theorem 5

Before we give the actual proof we will restate for completeness (and to unify notation) Corollary 2.3 from Kallenberg [1999].

Theorem A 1 *[Kallenberg [1999]] Let $(Z_1, Z_2, ...)$ a finite or infinite, stationary sequence of random variables with values in $\mathbb{R}_+ = [0, \infty]$ and let $T_j = \sum_{i \leq j} Z_i$ and $\beta = E[Z_1]$. Then there exists a random variable σ, uniform over $(0, 1)$ (and independent of Z_i) such that:*

$$P_{\sigma, Z_i, i \geq 1}[\sup_{j>0} \frac{T_j}{j} \leq \frac{\beta}{\sigma}] = 1.$$

Now we can prove theorem 5.

Proof We note that while the proof is done for Bernoulli arms, it is extendable to arbitrary distributions with bounded support. We use the notation from Theorem 2 and we will only discuss the sample complexity (the correctness follows similarly to the other algorithms). As mentioned, the main challenge, given the aggressiveness of the rejection procedure, is to get a positive lower bound on the probability of accepting a good arm. Let B be the event of accepting an arm if the expected reward associated with that arm is r_0 (the bound follows immediately for all arms with $r \geq r_0$, since the probability of acceptance will be at least as large as for r_0). Define $X = \text{Bernoulli}(r_0)$ as a Bernoulli distributed random variable.

Define $Y = \frac{X - r_0 + \epsilon/2}{1 - r_0 + \epsilon/2}$ as an affine transformation of X. Then, let $\alpha = E[Y] = \frac{\epsilon/2}{1 - r_0 + \epsilon/2} \geq \frac{\epsilon}{2}$ (since $r_0 > \epsilon$). Since $Y = 1$ with probability r_0 and $Y = \frac{-r_0 + \epsilon/2}{1 - r_0 + \epsilon/2} < 0$ with probability $1 - r_0$, we can interpret the series $\{S_j\}$ (with $S_j = \sum_{i=1}^{j} Y_i$, with Y_i being i.i.d. samples of Y, and implicitly X_i being i.i.d. samples of X) as a random walk.

We will now make two simplifying assumptions (and then describe at the end of the proof how to remove them). We assume: (1) $r_0 - \frac{\epsilon}{2} \geq \frac{1}{2}$ and (2) $\frac{-r_0 + \epsilon/2}{1 - r_0 + \epsilon/2} \in \mathbb{Z}^-$ (the set of negative integers). Then, $\{S_j\}$ is a positively biased random walk on the integers with maximum step value 1. In this case, we can apply a classic ballot-style result that says that $P(S_j > 0, \forall j = 1, 2, ...) = \max(E[Y], 0) = \alpha$, for example, Theorem 3 from Addario-Berry and Reed [2008], which is based on a result by Takacs [1967].

But, $\alpha = P(S_j > 0, \forall j = 1, 2, ...) \leq P(S_j \geq 0, \forall j = 1, 2, ...) = P(\frac{\sum_{i=1}^{j} X_i}{j} \geq r_0 - \frac{\epsilon}{2}, \forall j = 1, 2, ...) = P(\hat{X}_j \geq r_0 - \frac{\epsilon}{2}, \forall j = 1, 2, ...) \leq P(\hat{X}_j \geq r_0 - \frac{\epsilon}{2}, \forall j = 1, 2, ..., n_{\max}(i))$ (where $\hat{X}_j$ is the empirical average after j samples and corresponds to $\hat{r}_j^a$ from Line 5′ of the algorithm). Thus, $P(B) \geq \alpha \geq \frac{\epsilon}{2}$.

So, as in Theorem 2, $P(A_i) \geq P(B)\rho \geq \frac{\epsilon\rho}{2}$. This fact implies $E[N] \leq \frac{2}{\epsilon\rho}$ and the proof for the expected sample complexity bound follows similarly to that of Theorem 3 with an extra $\frac{2}{\epsilon}$ factor in the bound that comes via the upper bound on $E[N]$.

To complete the proof, one needs to show that a 'bad' arm (with expected value smaller than $r_0 - \epsilon$) will be rejected after at most $n_{\max}(i)$ samples. This claim follows via the same application of the Hoeffding bound as for the previous algorithms (the probability that a 'bad' arm is accepted after $n_{\max}(i)$ samples is smaller than δ_0, which is enough to bound the probability of error for the entire execution of the algorithm).

To remove Assumptions 1 and 2, we will apply ballot-style theorems for random variables with real values. The result will then follow by applying Corollary 2.3 from Kallenberg [1999].

The goal of the last part of the proof is to complete the proof for IGR for the general case of random walks on the real numbers.

Let $Z = 1 - Y$ (where Y was defined in the main text as a transformation of the Bernoulli random variable X). A set of i.i.d. samples of X induces a set of i.i.d. samples of Y and implicitely of Z $(Z_1, Z_2, ...)$. But $(Z_1, Z_2, ...)$ is a stationary sequence of variables with $E[Z_1] = E[Z] = 1 - E[Y] = 1 - \alpha$. The support of Z is $\{0, \frac{1}{1-r_0+\frac{\epsilon}{2}}\} \subset \mathbb{R}_+$.

Let $T_j = \sum_{i \leq j} Z_i$. Then, the conditions for theorem 1 hold and so there exists a Uniform$(0, 1)$ random variable σ such that $P[\sup_{j>0} \frac{T_j}{j} \leq \frac{1-\alpha}{\sigma}] = 1$.

Let's note $V = \sup_{j>0} \frac{T_j}{j}$ and $W = \frac{1-\alpha}{\sigma}$ two transformed random variables of Z_i and σ respectively. We know that $P[V \leq W] = 1$ (a relation known under the name of absolute stochastic dominance or statewise stochastic dominance). It is known that this relation implies the usual notion of (first order) stochastic dominance (which states that a random variable Y stochastically dominates a random variable X if for all elements x in the support of X and Y, $P(Y > x) \geq P(X > x)$ or equivalently $P(Y \leq x) \leq P(X \leq x)$).

So since W stochastically dominates V in an absolute sense, it also dominates it in a first-order sense. We will pick $1 \in \mathbb{R}_+$ and it follows that $P(W \leq 1) \leq P(V \leq 1)$.

But $P(W \leq 1) = P_\sigma(\frac{1-\alpha}{\sigma} \leq 1) = P_\sigma(\sigma \geq 1 - \alpha) = 1 - (1 - \alpha) = \alpha$ (where the third equality follows immediately from the properties of the uniform distribution).

We've thus shown that $\alpha \leq P(V \leq 1)$. Then $\alpha \leq P(\sup_{j>0} \frac{T_j}{j} \leq 1) = P(\frac{T_j}{j} \leq 1, \forall j > 0) = P(\sum_{i \leq j}(1 - Y_i) \leq j, \forall j > 0) = P(S_j \geq 0, \forall j > 0)$. We know from the proof of theorem 5 in the main text that $P(S_j \geq 0, \forall j > 0) \leq P(\hat{X}_j \geq r_0 - \frac{\epsilon}{2}, \forall j = 1, 2, ..., n_{\max}(i))$. Thus the desired relation $(P(B) \geq \alpha)$ holds even when Assumptions 1 and 2 are removed, and we consider random walks with steps taking real values. The rest of the proof remains unchanged. ■

Appendix C. Proof of Theorem 1

We will shortly introduce the PAC-Bandit setting as its definition is needed for the proof. For a thorough introduction we refer the reader to Even-Dar et al. [2002].

The PAC-Bandit problem is defined as follows: Given n arms and two parameters ϵ and δ, stop in finite time with probability 1 and return an arm at most ϵ away from the arm with the highest expected reward among the n with probability at least $1 - \delta$. The lower bound we will use from Mannor et al. [2004] has the form $\Omega(\frac{1}{\epsilon^2}(n + \log \frac{1}{\delta}))$.

Proof We will use contradiction and assume there exists an (ϵ, δ, r_0)-correct algorithm ALG that solves any IB(D) problem with expected sample complexity $o(\frac{1}{\epsilon^2}(\frac{1}{\rho} + \log \frac{1}{\delta}))$. The goal is to show that ALG would imply a correct algorithm for the PAC-Bandit problem with expected sample complexity $o(\frac{1}{\epsilon^2}(n + \log \frac{1}{\delta}))$, which would contradict the known lower bound in the PAC-Bandit setting.

Let D be a categorical probability distribution with 2 values in its support: $0.5 - \epsilon$ (a bad arm) and $0.5 + \epsilon$ (a good arm) with probability mass $1 - x$ on the first value and x on the second. Let's choose an arbitrary $r_0 \in (0.5, 0.5 + \epsilon]$ (so that we follow the constraint that ρ is bounded away from zero). Then $\rho = x$. Now, define a PAC-Bandit problem as follows: assume we are given n arms, $n - 1$ of which have expected reward of $0.5 - \epsilon$ and one of which has expected reward of $0.5 + \epsilon$. To be precise, it is worth mentioning that we allow

the algorithms in the PAC-Bandit setting to resample arms and ignore any previous pulls taken for those arms (this actually makes the PAC-Bandit problem harder, so the lower bound still has to hold).

Let $x = \rho = \frac{1}{n}$. When we use ALG for the PAC-Bandit problem, each time the algorithm samples a new arm from the environment, it selects an arm uniformly at random, with replacement, from the n arms. Applying ALG, it will get the good arm with probability at least $1-\delta$ with an expected number of samples $o(\frac{1}{\epsilon^2}(\frac{1}{\rho} + \log\frac{1}{\delta})) = o(\frac{1}{\epsilon^2}(n + \log\frac{1}{\delta}))$, which contradicts the lower bound mentioned above (Theorem 13 Mannor et al. [2004]). ■

Appendix D. Proof of theorem 3

Proof Sample Complexity. We keep the same notation as in the proof of Theorem 2. We use $E[r_{a_i}]$ to represent the expected value associated with arm a_i, $\hat{r}_j^{a_i}$ the empirical average of a_i's rewards after its jth pull, and $CI(j) = \sqrt{\frac{2\log(2j^2/\delta_0)}{j}}$ the confidence interval for the empirical average at step j. Let $n_{\max}(i) = \frac{4}{\epsilon^2}\log\frac{1}{\epsilon\delta_0}$ be the maximum number of pulls for arm a_i.

Now, $P(A_i) = \rho(1 - P(\text{reject arm } a_i | \text{ arm } a_i \text{ is good})) = \rho(1 - P(\cup_{j=1}^{n_{\max}(i)}\{\hat{r}_j^{a_i} \notin [E[r_{a_i}] - CI(j), E[r_{a_i}] + CI(j)]\})) \geq \rho(1 - \sum_{j=1}^{n_{\max}(i)}\frac{\delta_0}{2j^2}) \geq \rho(1-\delta_0) \geq \frac{\rho}{2}$. So, as in Theorem 2, $E[N] \leq \frac{2}{\rho}$.

Since the sampling of each arm stops after at most $n_{max}(i)$ steps, $E[SC] \leq E[\sum_{i=1}^{N}\frac{4}{\epsilon^2}\log\frac{2i^2}{\epsilon\delta}]$ and then the sample complexity bound follows similarly to the proof of Theorem 2.

Correctness. The algorithm stops with probability 1 in finite time, and $P(\text{failure}) \leq \sum_{i\geq 1} P(\{\text{ incorrect recommendation at step } i\}) = \sum_{i\geq 1} P(\bigcup_{j=1}^{n_{\max}(i)}\{\hat{r}_j^{a_i} \notin [E[r_{a_i}] - CI(j), E[r_{a_i}] + CI(j)]\}) \leq \sum_{i\geq 1}\sum_{j\geq 1}\frac{\delta}{4i^2j^2} \leq \delta$. ■

Appendix E. Proof of Theorem 4

Proof Since this is a high probability statement, we can assume for the rest of the proof that we are in a situation where the algorithm commits no errors (which happens w.p. at least $1-\delta$ as it can be shown that, for the entire experiment, the algorithm fails w.p. at most δ). Let's define $SC(a)$ to be the same complexity of accepting or rejecting an arm a. Let's fix (for now) the total number of sampled arms to $N = n$, and fix an arm a, with $\Delta_a < 0$ (which we label as a 'bad' arm).

Then, using the Hoeffding bound, $SC(a) = \frac{4}{\max(\epsilon^2, \Delta_a^2)}\log\frac{2i^2}{\max(\epsilon, \Delta_a)\delta}$ (where i is the index of the arm among all n arms). Let's assume D is continuous (the discrete case is similar) and let's define $f(\Delta_a)$ to be the pdf of Δ_a. Then, since $a \sim D$, $E[SC(a)|a \text{ 'bad'}] = \int_{\Delta_a<0}\frac{4}{\max(\epsilon^2,\Delta_a^2)}\log\frac{2i^2}{\max(\epsilon,\Delta_a)\delta}f(\Delta_a)d\Delta_a \leq \log\frac{2n^2}{\epsilon\delta}\int_{\Delta_a<0}\frac{4}{\max(\epsilon^2,\Delta_a^2)}f(\Delta_a)d\Delta_a \leq \frac{4}{\Delta_-^2}\log\frac{2n^2}{\epsilon\delta}$.

So, $E[\text{samples from all 'bad' arms}|N=n] = \sum_{k=1}^{n}E[\text{samples from } k \text{ 'bad' arms}|N=n]P(k \text{ arms are bad}) \leq \frac{4}{\Delta_-^2}\log\frac{2n^2}{\epsilon\delta}\sum_{k=1}^{n}kP(k \text{ arms are bad}) = \frac{4(1-\rho)n}{\Delta_-^2}\log\frac{2n^2}{\epsilon\delta}$. Similarly, it can be shown that $E[\text{samples from all 'good' arms}|N=n] \leq \frac{4\rho n}{\epsilon^2}\log\frac{2n^2}{\epsilon\delta}$.

Then, for any N, $EB = E[\text{samples from all 'bad' arms}] \leq \sum_{n=1}^{\infty} \frac{4(1-\rho)n}{\Delta_-^2} \log \frac{2n^2}{\epsilon\delta} P(N = n) \leq \frac{4(1-\rho)}{\Delta_-^2} \log \frac{2}{\epsilon\delta} E[N] + \frac{8(1-\rho)}{\Delta_-^2} E[N \log(N)] \leq \frac{16(1-\rho)}{\rho\Delta_-^2} \log \frac{4}{\epsilon\rho\delta}$ (where the last inequality follows from the bounds for $E[N]$ and $E[N \log(N)]$ from Theorem 2). So, $EB = O(\frac{1}{\rho\Delta_-^2} \log(\frac{1}{\epsilon\rho\delta}))$.

Using a similar argument, one can show that $EG = E[\text{samples from all 'good' arms}] = O(\frac{1}{\epsilon^2} \log(\frac{1}{\epsilon\rho\delta}))$. Thus, $E[SC] = EB + EG = O((\frac{1}{\epsilon^2} + \frac{1}{\rho\Delta_-^2}) \log \frac{1}{\epsilon\rho\delta})$. ■

Appendix F. Known Probability to Get a "Good" Arm—Algorithms

The algorithms we introduced so far treat the scenario for which the concentration of rewards (ρ) is unknown. In this section, we will solve the problem for the case of known ρ by reducing it to the PAC-Bandit setting (that we introduced formally in Appendix C).

The high level strategy is to compute how many arms one needs to sample to get a "good" arm with high probability and then apply a PAC-Bandit algorithm to select the "good arm" from the sampled ones. Algorithm 5 implements this strategy. It is worth noting here that besides the standard algorithms (**Uniform Planning** (UP), **Sequential Elimination** (SE) or **Median Elimination** (ME) Even-Dar et al. [2002]), we can also apply an algorithm with a better upper bound due to our assumption that the domain is reward rich ($\rho > 0$)—the version of Median Elimination from Section 7.1 of Mannor et al. [2004] (which we label as **Median Elimination with Known Bias** (MEKB) from this point on). The last algorithm is built to take advantage of knowledge of the bias of the best arm.

Algorithm 5: PAC-Bandit Reduction $(\epsilon, \delta, r_0, \rho)$

1: Sample $n = \frac{1}{\rho} \log(\frac{2}{\delta})$ arms.
2: Execute a correct **PAC-Bandit Algorithm** with input $(\epsilon, \frac{\delta}{2}, n)$ on the n multi-armed bandit problem.
3: Return the output of the **PAC-Bandit Algorithm**

We denote as SC_{ALG} the expected sample complexity of an IB(D) algorithm that uses a PAC-Bandit algorithm ALG as a subroutine in Step 2.

Proof [Proof] **Correctness**. Let $a_1, a_2, ..., a_n \sim D$ i.i.d. Define event $A = \{\forall i \in \{1...n\}, E[a_i] < r_0\}$ (in words, A stands for the event that the expected value of all the sampled arms is smaller than r_0). Then, $P_{a_i \sim D, i \in \{1...n\}}(A) = (1-\rho)^n$. If we choose $n = \frac{1}{\rho} \log(\frac{2}{\delta})$, then $P(A) \leq \frac{\delta}{2}$.

When ALG is executed with parameters from Step 2 in Algorithm 5, it will fail with probability at most $\frac{\delta}{2}$. Thus, by the union bound on $P(A)$ and the failure probability of ALG, the algorithm will fail with probability at most δ and will otherwise return an arm with the desired properties. ■

The sample complexity obviously depends on the algorithm chosen. It is easy to show that, in the worst case, $SC_{UP} = O(\frac{1}{\rho\epsilon^2} \log \frac{1}{\delta} \log \frac{1}{\rho\delta})$, $SC_{ME} = O(\frac{1}{\rho\epsilon^2} \log^2 \frac{1}{\delta})$ and $SC_{MEKB} =$

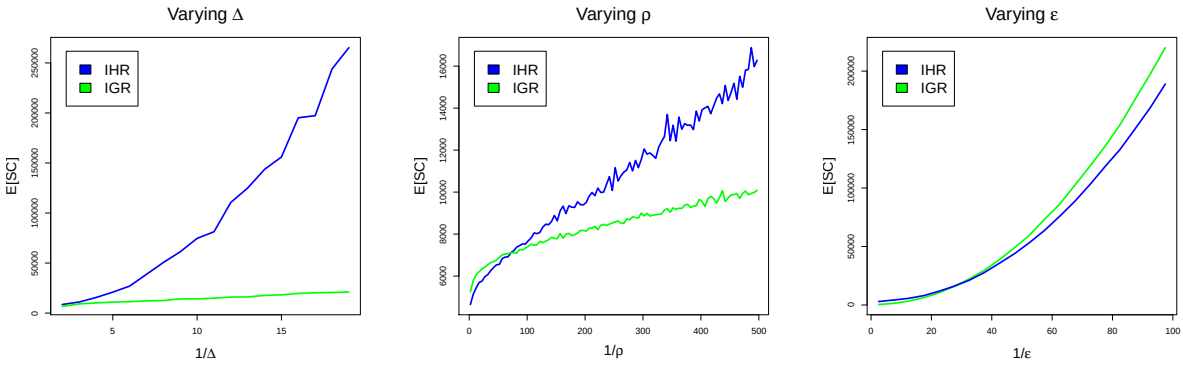


Figure G.1: Each data point in each graph is the average of 500 repetitions of the experiment. (a) In the left figure, we plot the dependence of the empirical average sample complexity (E[SC]) on $1/\Delta$. ϵ and ρ are fixed ($\epsilon = 0.05, \rho = 0.005$). (b) In the center figure, we plot the dependence of E[SC] on $1/\rho$. ϵ and Δ are fixed ($\epsilon = 0.05, \Delta = 0.4$). (c) In the right figure, we plot the dependence of E[SC] on $1/\epsilon$. ρ and Δ are fixed ($\rho = 0.005, \Delta = 0.4$). For all experiments, $\delta = 0.01$.

$O(\frac{1}{\rho\epsilon^2}\log\frac{1}{\delta})$. It can be observed that there is a gap of $O(\log\frac{1}{\delta})$ between the upper bound of MEKB and the lower bound from Theorem 1 (we leave the closing of this gap as an open problem)[1].

Appendix G. Experiments

G.1. *Pitfall!* and *Infinite Mario*

This section provides additional details about the experiments we ran for *Pitfall!* and *Infinite Mario*.

Four videos are included for *Pitfall!*. The experimental setting is described in the paper — it requires Harry to navigate from the left to the right side of the screen on the top tier without being killed or running out of time. Each video demonstrates one discovered trajectory, chosen as a representative of the others found. Each trajectory is run 20 times in each video. The locations of the videos are:

- Screen 3, Best Observed Policy, 3:11: http://www.youtube.com/watch?v=Jw-t7Ihe4Uc
- Screen 3, Worst Observed Policy, 3:36: http://www.youtube.com/watch?v=uAdXraDpUs0
- Screen 4, Best Observed Policy, 8:05: http://www.youtube.com/watch?v=r8Hr2Dc_NN0
- Screen 4, Worst Observed Policy, 7:06: http://www.youtube.com/watch?v=fOmAyGuXdvI

For *Infinite Mario*, we recorded videos of the performance of the simple strategy defined in the paper:

1. We remark here that while Median Elimination (and its variant MEKB) is the PAC-Bandit algorithm that offers the best sample complexity upper bound, it is not a practical algorithm.

- Success video (resulting from stitching together solutions to consecutive screens), 0:53: http://www.youtube.com/watch?v=tH5DRNrRS8I
- Partial success video, 0:49: http://www.youtube.com/watch?v=Wh8HGKIp7PQ
- First screens for 50 levels, 9:51: http://www.youtube.com/watch?v=tcJSQcVzRkc

Regarding the implementation, we took the *Infinite Mario* code from RLCompetition source code: http://code.google.com/p/rl-competition/ and RL-Glue code from RL-Glue source: http://glue.rl-community.org/wiki/Main_Page.

G.2. Synthetic Experiments

The role of this section is to further investigate the empirical sample complexity of IGR so as to give indications as to how the upper bound of IGR can be tightened and how does IGR compare with IHR in various scenarios.

We will use a simple but illustrative infinite bandit problem inspired by the sample complexity lower bound example. Let D (the probability distribution over the space of arms) be a categorical distribution with parameter Δ such that with probability ρ an arm is Bernoulli($\frac{1}{2} + \Delta$) (a 'good' arm) and with probability $1 - \rho$ is is Bernoulli($\frac{1}{2} - \Delta$) (a 'bad' arm). The ρ parameter encodes (as usual) the reward richness of the domain while Δ encodes how large is the difference between the good and the bad arms (and plays the role of Δ_- from section 5.2).

The three parameters of interest are obviously ϵ, Δ and ρ. In figure G.1, in each graph, two parameters have fixed values and the third is varied. We plot the dependence of the empirical average sample complexity on the inverse of each parameter (since the sample complexity bounds depend on $1/\epsilon$, $1/\Delta_-$, and $1/\rho$). We only compare IGR with IHR, since the empirical sample complexity of IUR is always significantly larger than the other two algorithms and thus does not offer any interesting insight.

In the left graph from Figure G.1 it is visible that **IGR** increasingly dominates **IHR** as the difference between the good and the bad arms increases. We think this is one of the two factors that determine the success of **IGR** in practice, as it provides fast rejection of bad arms for a variety of types of bad arms (for a fixed accuracy ϵ and concentration of good arms ρ). The behavior is consistent for other fixed values of ϵ and ρ.

In the center graph from Figure G.1 the dependency of **IGR** and **IHR** on ρ is interesting. For values of $\rho > 0.01$, **IHR** dominates, while when the concentration of good arms decreases, **IGR** starts to have a better sample complexity. The better dependency on ρ as it decreases is the second factor that makes **IGR** strong empirically.

In the graph on the right side of Figure G.1 the dependency on ϵ is plotted. The performance comparison between **IGR** and **IHR** is reversed as compared to the experiment where ρ was varied. **IGR** dominates for values of $\epsilon > 0.04$ while **IHR** starts dominating when ϵ decreases further.

While the experiments (b) and (c) in Figure G.1 show that **IGR** and **IHR** are not totally ordered in terms of sample complexity performance, it is intuitive that for practical applications (where ϵ is usually fixed) **IGR** tends to behave better. All three synthetic experiments indicate that the bound from Theorem 5 is loose and the true bound of **IGR** is actually closer to Theorem 4.

JMLR: Workshop and Conference Proceedings 24:43–57, 2012 10th European Workshop on Reinforcement Learning

Actor-Critic Reinforcement Learning with Energy-Based Policies

Nicolas Heess NHEESS@GATSBY.UCL.AC.UK

David Silver D.SILVER@CS.UCL.AC.UK

Yee Whye Teh Y.W.TEH@STATS.OX.AC.UK

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

We consider reinforcement learning in Markov decision processes with high dimensional state and action spaces. We parametrize policies using energy-based models (particularly restricted Boltzmann machines), and train them using policy gradient learning. Our approach builds upon Sallans and Hinton (2004), who parameterized value functions using energy-based models, trained using a non-linear variant of temporal-difference (TD) learning. Unfortunately, non-linear TD is known to diverge in theory and practice. We introduce the first sound and efficient algorithm for training energy-based policies, based on an actor-critic architecture. Our algorithm is computationally efficient, converges close to a local optimum, and outperforms Sallans and Hinton (2004) in several high dimensional domains.

1. Introduction

A major challenge in reinforcement learning is to find successful policies in problems with high-dimensional state or action spaces. In order to solve these problems effectively, it is advantageous to learn representations of the state and action spaces that enable a policy to achieve high performance. In this paper, we develop a framework for parameterizing richly structured policies, and for finding those with (locally) optimal performance.

We consider a class of policies based on *energy-based models* [LeCun et al., 2006], where the (negative) log probability of selecting an action is proportional to an energy function. Energy-based models have been widely studied in both supervised and unsupervised learning. They can learn deep, distributed representations of high-dimensional data (such as images) and model high-order dependencies, and here we will use them to directly parametrize representations over states and actions. We explore in detail a class of energy-based models called restricted Boltzmann machines (RBMs; Freund and Haussler, 1994; Welling et al., 2004). RBMs and conditional RBMs have been applied to a range of challenging tasks including multi-class classification [Larochelle and Bengio, 2008], collaborative filtering [Salakhutdinov et al., 2007], motion capture modeling [Taylor and Hinton, 2009], modeling of transformations in natural images [Memisevic and Hinton, 2010], and structured output prediction [Mnih et al., 2011].

To successfully use energy-based policies for reinforcement learning, it is necessary to address several challenges. First, the parameterized representations can be highly non-linear, leading to non-convex objectives with multiple optima. Second, it is often intractable to compute the partition function that normalizes the energy function into a probability

distribution. Third, evaluating a policy typically has high variance, since the performance depends on the rewards accumulated over a complete trajectory. Prior work [Sallans, 2002; Sallans and Hinton, 2004; Otsuka et al., 2010; Elfwing et al., 2010] partially addressed these issues using value-based reinforcement learning. The idea was to approximate the action-value function by the free energy of an energy-based model, and to train it by temporal-difference (TD) learning. However, this approach has a serious drawback: TD learning is known to diverge, both in theory and in practice, when using non-linear value functions [Tsitsiklis and Van Roy, 1997]. In addition, the policy is based on an arbitrary temperature which makes the algorithm hard to tune in practice.

We introduce a new approach to reinforcement learning with energy-based policies. The basic idea is to adjust the policy parameters to follow the gradient of the policy performance, using sample trajectories to obtain estimates of the gradient [Williams, 1992; Sutton et al., 1999; Baxter and Bartlett, 2001]. One major advantage of this approach is that it converges to a local optimum, even for non-linear policies. Simple policy gradient methods are sensitive to the parameterization of the policy, and also suffer from high variance in the gradient estimates. We address these shortcomings by following the *natural gradient*, which reduces the dependence of the performance of the policy gradient on the parameterization [Kakade, 2001]; and by using an actor-critic architecture, which uses an approximate value function to reduce the variance in the gradient estimates [Peters and Schaal, 2008; Bhatnagar et al., 2009]. We introduce two novel natural actor-critic algorithms for efficiently following the gradient of energy-based policies without resorting to explicit computation of the partition function. We apply our algorithms to learn RBM policies in a number of tasks, finding that they converge quickly to reasonable solutions in all tasks and consistently outperform Sallans and Hinton [2004], which sometimes even fails to converge.

2. Energy-based policies

We consider stationary Markov decision processes (MDPs) with bounded rewards and high dimensional state and action spaces denoted $\mathcal{S}$ and $\mathcal{A}$ respectively. Denote the state, action and reward at time t by s_t, a_t and r_t respectively, and the state transition probabilities and expected reward function by $P(s_{t+1}|s_t, a_t)$ and $R(s_t, a_t)$. We will consider stochastic and stationary polices described by conditional distributions over actions $\pi^\theta(a; s)$ parameterized by θ. We assume that given policy π^θ the MDP is ergodic with stationary distribution d^θ.

In this paper we consider energy-based policies which can be expressed as conditional joint distributions over actions a and a set of latent variables h:

$$\pi^\theta(a, h; s) = \frac{1}{Z(s)} e^{\phi(s,a,h)^\top \theta} \tag{1}$$

where $\phi(s, a, h)$ are a pre-defined set of features and $Z(s) = \sum_{a,h} \exp(\phi(s, a, h)^\top \theta)$ is the normalizing *partition function*. The policy itself is then obtained by marginalizing out h. The latent variables allow energy-based policies to parametrize complex non-linear and non-factorial relationships between actions and states, even though the underlying parameterization (1) is log linear in the features $\phi(s, a, h)$. For example, in a conditional restricted Boltzmann machine (RBM), the states s, actions a and latent variables h are all high-

dimensional binary vectors, and (1) is parameterized as:

$$\pi^\theta(a,h;s) = \tfrac{1}{Z(s)} e^{s^\top W_s h + a^\top W_a h + b_s^\top s + b_h^\top h + b_a^\top a} \tag{2}$$

where the parameters are matrices W_s, W_a and vectors b_s, b_a, b_h of appropriate dimensionalities. Marginalizing out h, we get a non-linearly parameterized policy:

$$F^\theta(s,a) = -b_s^\top s - b_a^\top a - \textstyle\sum_i \log(1+e^{s^\top W_{si} + a^\top W_{ai} + b_{hi}}), \qquad \pi^\theta(a;s) = \tfrac{1}{Z(s)} e^{-F(s,a;\theta)} \tag{3}$$

where i indices the latent variables, and W_{si}, W_{ai}, b_{hi} are parameters associated with latent variable h_i. The quantity $F(s,a;\theta)$ is called the *free energy*.

2.1. Free-energy Sarsa

Sallans and Hinton [2004] use the RBM free energy as a non-linear function approximator. Specifically, they approximate the action-value function $Q^\theta(s,a)$ of a discounted MDP using $-F^\theta(s,a)$, and select actions according to a Boltzmann exploration policy,

$$\pi^\theta(\mathbf{a};\mathbf{s},T) = \tfrac{1}{Z(\mathbf{s},T)} e^{-F(\mathbf{s},\mathbf{a};\theta)/T} \tag{4}$$

where the temperature T scales units of reward into units of probability: as $T \to \infty$ the policy becomes uniform; as $T \to 0$ it becomes greedy. The parameters of the RBM are updated using a non-linear version of the Sarsa algorithm, with time-varying step size β_t,

$$\begin{aligned} \delta_t &= r_{t+1} - \gamma F^{\theta_t}(s_{t+1}, a_{t+1}) + F^{\theta_t}(s_t, a_t) \\ \theta_{t+1} &= \theta_t - \beta_t \delta_t \nabla_\theta F^{\theta_t}(s_t, a_t), \end{aligned} \tag{5}$$

where δ_t is the TD error and γ is the discount factor of the MDP. We refer to this algorithm as energy-based Sarsa (ESARSA). Since Sarsa is a control algorithm based on TD learning, it may diverge with non-linear function approximations [Tsitsiklis and Van Roy, 1997]. Recently, convergent non-linear prediction algorithms based on gradient TD have been developed, however even these may still diverge when used for control [Maei et al., 2009].

2.2. Policy gradient

In this paper we consider an average reward formulation of MDPs. Our objective is to find parameters θ that (locally) maximize the *average reward* per time-step $J(\theta)$,

$$J(\theta) = \lim_{T\to\infty} \tfrac{1}{T}\, \mathrm{E}^\theta \left[\textstyle\sum_{t=1}^T r_t\right] = \mathrm{E}^\theta_{sa}\left[R(s,a)\right] \tag{6}$$

where E^θ denotes expectation under π^θ and E^θ_{sa} denotes expectation under the stationary distribution (where $s \sim d^\theta$ and $a|s \sim \pi^\theta(\cdot;s)$). The state-value function $V^\theta(s)$ and action-value function $Q^\theta(s,a)$ for policy π^θ are given by the differential reward from s (and a),

$$Q^\theta(s,a) = \sum_{t=1}^\infty \mathrm{E}^\theta\left[r_t - J(\theta) | s_0 = s, a_0 = a\right] \tag{7}$$

$$V^\theta(s) = \sum_{t=1}^\infty \mathrm{E}^\theta\left[r_t - J(\theta) | s_0 = s\right] = \sum_{a\in\mathcal{A}} \pi^\theta(s;a) Q^\theta(s,a) = \mathrm{E}^\theta_a[Q^\theta(s,a)] \tag{8}$$

where E_a^θ denotes expectation under $a|s \sim \pi^\theta(\cdot; s)$. The *advantage function* is the differential value for action a in state s:

$$A^\theta(s,a) = Q^\theta(s,a) - V^\theta(s) \tag{9}$$

Policy gradient methods update the policy parameters θ to follow the gradient of the average reward. The *policy gradient theorem* [Baxter and Bartlett, 2001; Sutton et al., 1999; Bhatnagar et al., 2009] provides the gradient of $J(\theta)$ with respect to θ,

$$\nabla_\theta J(\theta) = \mathrm{E}_{sa}^\theta \left[A^\theta(s,a) \nabla_\theta \log \pi^\theta(a;s) \right] \tag{10}$$

In practice, the advantage function A^θ is unknown and must be approximated. *Actor-critic* policy gradient algorithms estimate the advantage function, $\hat{A}^{\mathbf{w}} \approx A^\theta$, and update this estimate (the *critic*) in parallel with the policy parameters (the *actor*). If the parameterization of the approximation $\hat{A}^{\mathbf{w}}$ is *compatible* with the parameterization of the policy, then this approximation does not introduce any bias into the gradient direction [Sutton et al., 1999]. A parameterization is compatible if it satisfies

$$\nabla_{\mathbf{w}} \hat{A}^{\mathbf{w}}(s,a) = \nabla_\theta \log \pi^\theta(a;s) \tag{11}$$

That is, $\hat{A}^{\mathbf{w}}(s,a) = \psi^\theta(s,a)^\top \mathbf{w}$ where $\psi^\theta(s,a) = \nabla_\theta \log \pi^\theta(a;s)$. Furthermore, the parameters $\mathbf{w}$ should minimize the mean-squared-error (MSE),

$$\mathbf{w} = \operatorname*{argmin}_{\mathbf{w}'} \mathrm{E}_{sa}^\theta \left[\left(A^\theta(s,a) - \psi^\theta(s,a)^\top \mathbf{w}' \right)^2 \right] \tag{12}$$

This can be obtained by stochastic gradient updates:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t (\delta_t - \psi^{\theta_t}(s_t,a_t)^\top \mathbf{w}_t) \psi^{\theta_t}(s_t,a_t) \tag{13}$$

where θ_t is the current policy parameter, α_t is a step size and δ_t is an unbiased estimate of $A^\theta(s_t,a_t)$ (later).

The direction of the vanilla policy gradient is sensitive to reparameterizations of the policy that don't affect the action probabilities. *Natural* policy gradient algorithms [Kakade, 2001; Peters and Schaal, 2008] remove this dependence, by finding the direction that improves $J(\theta)$ the most for a fixed small amount of change in distribution (e.g. measured by KL divergence). The natural policy gradient $\nabla_\theta^{\mathrm{nat}} J(\theta)$ is defined by

$$\nabla_\theta^{\mathrm{nat}} J(\theta) = G_\theta^{-1} \nabla J(\theta), \qquad G_\theta = \mathrm{E}_{sa}^\theta \left[\nabla \log \pi(s,a) \nabla \log \pi(s,a)^T \right] \tag{14}$$

where G_θ is the Fisher information matrix. When using compatible function approximation, we find that,

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathrm{E}_{sa}^\theta \left[\nabla_\theta \log \pi^\theta(a;s) A^\theta(s,a) \right] \\ &= \mathrm{E}_{sa}^\theta \left[\nabla_\theta \log \pi^\theta(a;s) (\nabla_\theta \log \pi^\theta(a;s))^T \mathbf{w} \right] = G_\theta \mathbf{w} \end{aligned} \tag{15}$$

and the natural gradient simplifies to $\nabla_\theta^{\mathrm{nat}} J(\theta) = \mathbf{w}$. Policy parameters updates are then:

$$\theta_{t+1} = \theta_t + \beta_t \mathbf{w}_{t+1}, \tag{16}$$

The final ingredient is estimating the advantage function $A^\theta(s,a)$. One unbiased estimate is the TD error:

$$\delta_t = r_{t+1} - J(\theta) + V^\theta(s_{t+1}) - V^\theta(s_t) \tag{17}$$

which is used by Bhatnagar et al. [2009] to motivate their natural TD actor-critic (NATDAC) algorithm: They approximate the state-value function, and hence the TD error, using a linear function approximator $\hat{V}_\theta(s) \approx f(s)^T\mathbf{v}$, where $f(s)$ is a state-dependent feature vector and $\mathbf{v}$ are updated by TD learning:

$$\begin{aligned}\hat{J}_{t+1} &= (1-\zeta_t)\hat{J}_t + \zeta_t r_{t+1}\\ \delta_t &= r_{t+1} - \hat{J}_{t+1} + f(s_{t+1})^T\mathbf{v}_t - f(s_t)^T\mathbf{v}_t\\ \mathbf{v}_{t+1} &= \mathbf{v}_t + \alpha_t\delta_t f(s_t)\end{aligned} \tag{18}$$

Provided that the average reward, TD error and critic are updated on a slower time scale than for the actor, and step sizes are reduced at appropriate rates, NATDAC converges to a policy achieving near-local maximum average reward [Bhatnagar et al., 2009, Section 5].

3. NATDAC for energy-based policies

Energy-based policies pose a challenge to policy gradient methods. Both vanilla policy gradient updates and natural actor-critic updates require the computation of compatible features $\psi^\theta(s,a) = \nabla_\theta \log \pi^\theta(s,a)$. For an energy-based policy, this requires the computation of two expectations:

$$\begin{aligned}\psi^\theta(s,a) &= \nabla_\theta \log \pi^\theta(a;s) = \nabla_\theta \log \textstyle\sum_h e^{\phi(s,a,h)^\top\theta} - \nabla_\theta \log Z(s)\\ &= \mathrm{E}^\theta_h\left[\phi(s,a,h)\right] - \mathrm{E}^\theta_{a'h}\left[\phi(s,a',h)\right] \stackrel{def}{=} \phi^\theta(s,a) - \mathrm{E}^\theta_{a'}\left[\phi^\theta(s,a')\right].\end{aligned} \tag{19}$$

where E^θ_h is the expectation wrt the conditional of h given a and s while E^θ_{ah} is wrt both a and h. The first expectation, which we denote by $\phi^\theta(s,a)$, can be computed efficiently in a wide class of energy-based models, including RBMs. The second expectation involves a sum over all actions, and is intractable to compute in high-dimensional action spaces.

We now propose two incremental actor critic algorithms that avoid evaluating either this expectation or the partition function. The first algorithm *energy-based NATDAC* (ENATDAC) uses independent samples from the energy-based policy to approximate the intractable expectations. The second algorithm, *energy-based Q-value natural actor critic* (EQNAC) uses a deterministic approximation and has a lower computational requirement.

3.1. Energy-based NATDAC

For our first algorithm, we note that energy-based policies such as RBMs may be efficiently sampled, for example by blocked Gibbs sampling, without explicitly computing the partition function. Unbiased estimates of the update (13) can then be formed using two independent

unbiased estimates of the compatible features $\psi^\theta(s,a)$,

$$\hat{\psi}'_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{K}\sum_{k=1}^{K}\phi^{\theta_t}(s_t,a'_k), \qquad \hat{\psi}''_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{L}\sum_{l=1}^{L}\phi^{\theta_t}(s_t,a''_l)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\left(\delta_t - \hat{\psi}_t'^{\top}\mathbf{w}_t\right)\hat{\psi}''_t \tag{20}$$

where $a'_k, a''_l \sim \pi^{\theta_t}(\cdot;s)$ iid. Because (20) is equal in expectation to the critic update (13), the two time scale convergence proof for NATDAC [Bhatnagar et al., 2009, section 5.3] can be applied. Specifically, under our assumption that the critic is updated on a much slower time-scale than the actor, the policy θ_t and therefore ϕ^{θ_t} is effectively stationary during critic updates. As a result, the TD learning updates (20) are based on a function approximator that is linear in ϕ^{θ_t}, and do not suffer from the divergence issues faced by energy-based Sarsa.

One drawback of the NATDAC algorithm is that the quality of the state-value function approximator depends on the feature vector $f(s)$. Although the algorithm converges, the quality of the final policy may be significantly affected by the choice of features. In an energy-based framework, it is possible to marginalize over $\phi^\theta(s,a)$ to define an energy-based feature vector $f(s) = \mathrm{E}_a^\theta[\phi^\theta(s,a)]$. Again, this expectation is intractable to compute, but can be approximated by a third set of samples from the policy, $a'''_m \sim \pi^\theta(s,\cdot)$

$$\hat{f}_t = \frac{1}{M}\sum_{m=1}^{M}\phi^{\theta_t}(s_t,a'''_m) \tag{21}$$

Algorithm 4.1: ENATDAC

Input: $\hat{J}_0, \mathbf{v}_0, \mathbf{w}_0, \theta_0, s_0$
for $t = 0, 1, \ldots$ **do**
  $a_t, a'_{1\ldots K}, a''_{1\ldots L} \sim \pi^{\theta_t}(\cdot;s_t)$
  $s_{t+1} \sim P(\cdot|s_t,a_t)$
  $r_{t+1} = R(s_t,a_t)$
  $\hat{\psi}'_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{K}\sum_{k=1}^{K}\phi^{\theta_t}(s_t,a'_k)$
  $\hat{\psi}''_t = \phi^{\theta_t}(s_t,a_t) - \frac{1}{L}\sum_{l=1}^{L}\phi^{\theta_t}(s_t,a''_l)$
  $\hat{J_{t+1}} = (1-\zeta_t)\hat{J}_t + \zeta_t r_{t+1}$
  $\delta_t = \mathbf{r}_{t+1} - \hat{J}_{t+1} + \mathbf{v}^\top\hat{f}_{t+1} - \mathbf{v}^\top\hat{f}_t$
  $\mathbf{v}_{t+1} = \mathbf{v}_t + \alpha_t\delta_t f(s_t)$
  $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t(\delta_t - \hat{\psi}_t'^T\mathbf{w}_t)\hat{\psi}''_t$
  $\theta_{t+1} = \theta_t + \beta_t\mathbf{w}_{t+1}$
end for

Pseudocode for energy-based NATDAC is given in Algorithm 4.1. In summary, it uses an advantage actor-critic, based on TD learning with an energy-based feature vector, and using the sample approximation to the intractable expectations, and updates the policy by following the natural gradient.

3.2. Energy-based Q-value natural actor-critic

We now introduce our second algorithm, EQNAC, which is also based on a natural actor-critic approach. The key new idea is to note that, when using the compatible function approximator $\psi^\theta(s,a)$, the approximate advantage function $\hat{A}^{\mathbf{w}}(s,a)$ can be represented in terms of an approximate action-value $\hat{Q}^{\mathbf{w}}(s,a)$,

$$\hat{A}^{\mathbf{w}}(s,a) = \psi^\theta(s,a)^\top\mathbf{w} = \left(\phi^\theta(s,a) - \mathrm{E}_{a'}^\theta\left[\phi^\theta(s,a')\right]\right)^\top\mathbf{w} = \phi^\theta(s,a)^\top\mathbf{w} - \mathrm{E}_{a'}^\theta\left[\phi^\theta(s,a')\right]^\top\mathbf{w}$$
$$\overset{def}{=} \hat{Q}^{\mathbf{w}}(s,a) - \mathrm{E}_{a'}^\theta[\hat{Q}^{\mathbf{w}}(s,a')]. \tag{22}$$

This suggests that it is sufficient to estimate the action-value function $Q^\theta(s,a)$ by a linear approximator $\hat{Q}^{\mathbf{w}} = \phi^\theta(s,a)^\top \mathbf{w}$. This representation does not involve any intractable expectations. The EQNAC algorithm updates the critic parameters $\mathbf{w}$ by TD learning, and updates the policy parameters θ by following the natural gradient in the direction of $\mathbf{w}$. Pseudocode for EQNAC is shown in Algorithm 4.2. Again, for an effectively stationary policy the TD learning procedure is linear in ϕ^θ and therefore avoids the divergence issues encountered by energy-based Sarsa. Unlike ENATDAC, this algorithm does not require repeated sampling of actions to estimate the expectations. Furthermore, it only maintains two sets of parameters θ and $\mathbf{w}$, and does not require a state feature vector to be defined.

Algorithm 4.2: EQNAC

Input: $\hat{J}_0, \mathbf{w}_0, \theta_0, s_0$
$a_0 \sim \pi^{\theta_0}(\cdot; s_0)$
for $t = 0, 1, 2, \ldots$ **do**
  $r_{t+1} = R(s_t, a_t)$
  $s_{t+1} \sim P(\cdot|s_t, a_t)$
  $a_{t+1} \sim \pi^{\theta_t}(\cdot; s_t)$
  $\hat{J}_{t+1} = (1-\zeta_t)\hat{J}_t + \zeta_t r_{t+1}$
  $\delta_t = r_{t+1} - \hat{J}_{t+1} + \phi^{\theta_t}(s_{t+1}, a_{t+1})^\top \mathbf{w}_t - \phi^{\theta_t}(s_t, a_t)^\top \mathbf{w}_t$
  $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \delta_t \phi^{\theta_t}(s_t, a_t)$
  $\theta_{t+1} = \theta_t + \beta_t \mathbf{w}_{t+1}$
end for

4. Experiments

We evaluate our two algorithms, comparing them against ESARSA on three tasks below. As an additional comparison, we also considered a neural network (NN) with stochastic output units and the same structure as the RBM except that the hidden units were deterministic instead of stochastic. The NN was trained with NATDAC (details in supplemental material). The learning algorithms are sensitive to the choice of learning parameters. For all tasks and algorithms we therefore perform a grid search over different settings of the relevant parameters, and report results for the best setting.[1] In a cross-validation style setup, we first choose the parameters based on a preliminary parameter sweep, then fix the parameters and perform a final run for which we report the results. In all cases, we learn several policies with the same parameter settings but different random initializations. We obtain learning curves by evaluating each policy at regular intervals during learning, performing several hundred repeats of each task.[2] We use exact sampling for all algorithms in the stochastic policy and blocker tasks, and block Gibbs sampling for the octopus arm task. Note that our algorithms use an average reward formulation; whereas ESARSA uses a discounted reward formulation. For ESARSA on the blocker and octopus arm tasks γ is set to 1; for the stochastic policy task $\gamma = 0$.

The **stochastic policy** task [Sallans, 2002] is an example where a stochastic policy is required to deal with state aliasing, and serves to demonstrate that RBMs can model stochastic policies with high-dimensional action spaces. The action space consists of N binary variables and there are K *unobserved* states. Each of the unobserved states is associated with one particular configuration of the action variables. If the environment is in state k and the associated action is performed a reward of 10 is received and the environment

1. For ESARSA we consider initial learning rate, speed of decay of the learning rate, decay of the exploration temperature T (starting at 1), and variance of the Gaussian distribution used to randomly initialize the policy parameters at the beginning of learning. For the policy gradient algorithms we consider α_0, β_0, rate of decay for α, β, and scale of random initialization. We use $\zeta = \alpha$ in all experiments.
2. 500 actions for stochastic policy; 1000 epochs for blocker task; 100 epochs for octopus arm.

	N	K	prototypical actions
task 1	5	2	(0 0 1 0 1) (0 1 0 1 1)
task 2	6	3	(1 1 0 0 0 0) (0 0 1 1 0 0) (0 0 0 0 1 1)
task 3	8	3	(0 0 1 0 1 0 1 1) (0 1 0 1 1 1 0 0) (1 1 1 0 1 0 0 0)

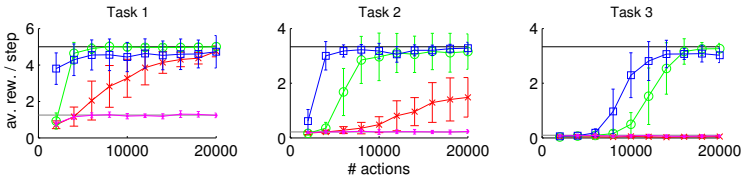
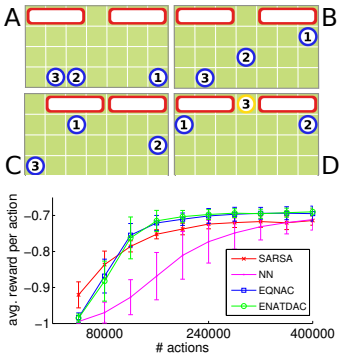


Figure 4.1: Stochastic policy task: The table shows the task-parameters for the three versions of the task. Results are shown for ENATDAC (green circles), EQNAC (blue squares), SARSA (red crosses), and NN (magenta). Black lines show average reward for optimal policy; light gray lines show rewards with independent actions.

Figure 4.2: Blocker task: The *top* plot illustrates several steps in a blocker game (adapted from Sallans and Hinton 2004). To win, the agents need to cooperate. Here, agents 1,2 force the blockers to split so that agent 3 can enter the end zone in the middle. The *bottom* plot shows, for each algorithm, mean and standard deviation of the average reward per step of the learned policies for 20 restarts with different random initialziations (see Fig. B.3 in the supplemental material for individual traces).



transitions into state $k+1$ (or into state 1 from K). Otherwise the state is unchanged and no reward is provided. Since the agent does not observe the environment state the optimal policy is to choose the K "good" action configurations with equal probability, achieving an average reward of $\frac{10}{K}$. A deterministic policy will get zero reward since it will get stuck in one of the states. Further, it is necessary to model *dependencies* between action variables: for independently drawn action variables the probability of generating a "good" configuration is very low. We consider three versions of the task with $N = 5, 6, 8$, and $K = 2, 3, 3$ (cf. Fig. 4.1). The first version was considered in Sallans [2002]. We used an RBM with 1 hidden unit for task 1 and 2 hidden units for tasks 2 and 3. Training was performed for 20000 actions. Results are shown in Fig. 4.1. The NN models each action dimension independently and therefore performs very poorly (see supplemental material for further discussion). In contrast, the RBM learns to represent these correlations via stochastic latent variables and performs very effectively. When training the RBM, ESARSA is only able to find the optimal policy for task 1, whereas EQNAC and ENATDAC find optimal policies in all cases.[3]

The **blocker task** (Sallans 2002; Sallans and Hinton 2004) is a multi-agent task in which agents have to reach the end zone at the top of a playing field while blockers try to stop

3. Although only the first 20000 actions are shown we allowed 50000 actions for ESARSA training. For task 2 additional training leads to a small improvement relative to the performance after 20000 actions but not for task 3 (see also Fig. B.1 in supplemental material).

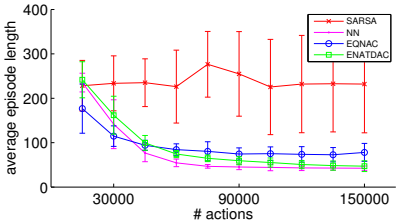


Figure 4.3: Octopus arm: Average number of steps required to hit the target. For each algorithm mean and std.-dev. for 10 restarts with different random initializations are shown. Figs. B.5,B.6 in the suppl. material show individual traces.

them. In each iteration of the game, each agent moves by one step in one of four directions, while the blockers behave in a deterministic manner, moving left or right to block the agents. We used a 7×4 board with 3 agents and 2 blockers as illustrated in Fig. 4.2. Blockers have a limited range of responsibility, with blocker 1 responsible for columns 1-4 and blocker 2 for columns 4-7. The state space consists of locations for each agent and for blocker (using 1-of-N encoding for each). A game is started by randomly placing agents at the bottom and the blockers at the top. It is played for 40 steps or until one of the agents reaches the end zone. A reward of -1 is given for each step prior to reaching the end zone, and a reward of 1 when one of the agents succeeds and the game is ended. We used RBMs with 16 hidden units, and learning was performed for 400000 actions. Results are shown in Fig. 4.2. EQNAC and ENATDAC consistently achieved good results. ESARSA and NN generally converged more slowly and found lower quality policies.

In this task we also experimented with several different state feature vectors $f(s)$, as required by ENATDAC (but not the other algorithms). We tested: randomly generated binary features (as e.g. Bhatnagar et al., 2009), choosing the dimensionality D of $f(s)$ to be lower than the number of states (which is 87808; we experimented with $D = 400, 1000$); raw feature encoding based on the state vector s directly; and the "energy based" feature vector described in section 3.1. We found that a poor choice of $f(s)$ can significantly reduce the asymptotic performance. The energy-based features performed very well, only equalled by large sets of carefully scaled random features.

Finally, we tested our algorithms on an **octopus arm** [Engel et al., 2005] task. The aim is to learn to control a simulated octopus arm to hit a target. The arm consists of C compartments and is attached to a rotating base. There are $8C + 2$ continuous state variables (x,y position/velocity of the nodes along the upper/lower side of the arm; angular position/velocity of the base) and $3C + 2$ action variables that control three muscles (dorsal, transversal, central) in each compartment as well as the clockwise and counter-clockwise rotation of the base. Previous work [Engel et al., 2005] simplified the high-dimensional action space using 6 "macro-actions" corresponding to particular patterns of muscle activations. Here, we do not make any such simplification; patterns of muscle activations are learnt by latent variables. Each muscle output was restricted to a binary activation. Below we present results for an arm with $C = 4$ compartments. The goal is to strike the target with any part of the arm. To direct exploration we provide a shaping reward at each time step: if the minimum distance between arm and target is decreased then a shaping reward of $+1$ is provided; if the distance is increased the reward is -1. The final reward for hitting the target is $+50$. An episode ends when the target is hit or after 300 steps.

We learned RBM-based policies with 32 stochastic hidden units, and an NN-based policy with 32 deterministic hidden units. We used energy-based features for (E)NATDAC as these are directly applicable in continuous-valued state spaces. Results for the octopus arm

are shown in Fig. 4.3. The policy-gradient approaches performed significantly better in this experiment. In particular, the RBM trained by ENATDAC, and the NN trained by NATDAC, both learnt good policies that were consistently able to hit the target. Some ENATDAC runs were trapped in suboptimal maxima where the arm moves only in one direction, thus taking a longer time to reach the target on some episodes (see Figs. B.5, B.6 in the supplemental material). The policies learned by EQNAC are slightly worse but the arm still learns to find and hit the target in the majority of cases. ESARSA performed relatively poorly. A video demonstration of the learnt octopus arm policies is available at: http://www.gatsby.ucl.ac.uk/~nheess/papers/ebrl/

5. Discussion

In our most challenging task, an energy-based policy was able to represent and learn, using ENATDAC, an effective control policy for a high-dimensional octopus arm. Unlike prior work, the latent variables *automatically* learnt to represent useful patterns of activations. In this task, a stochastic NN with deterministic hidden nodes was also able to solve the problem. This is not entirely unexpected as here a deterministic final policy is likely to be sufficient, while learning is simpler in the NN as it allows direct sampling of actions and the exact calculation of ψ^θ and $f(s)$. However, as demonstrated by the stochastic policy task, there are many domains in which partial observability, or an adversary, entails the need for highly structured stochastic policies that cannot be encoded by deterministic hidden nodes.

In our experiments the ESARSA algorithm did not achieve satisfactory results in the larger stochastic policy tasks, and also performed relatively poorly on the blocker and octopus arm tasks. The poor performance in some runs, and sensitivity to initial values, suggests that ESARSA is indeed suffering from divergence problems, presumably caused by the use of non-linear temporal-difference learning. In contrast, ENATDAC and EQNAC converged more robustly to a satisfactory solution in all tasks, suggesting that policy gradient algorithms may be more appropriate than value-based ones in this setting.

Our first algorithm, ENATDAC, is guaranteed to converge to a policy that achieves an average-reward that is close to local maximum. The algorithm requires the policy to be sampled many times: to select an action, to estimate the compatible features, and to construct the energy-based feature vector $f(s)$. However, we found empirically that a very small number of samples was sufficient to achieve good results: We used $K, L = 1$ in our experiments (cf. line 3 of Algorithm 4.1) except for the octopus arm where $K = L = 5$. Although the samples should be independent to ensure convergence, in practice the performance was unaltered by reusing the same samples.

Our second algorithm, EQNAC, uses a simple approximation to the advantage function. This algorithm does not require additional samples of the policy, beyond selecting the next action, making it more computationally efficient. It is also simpler than ENATDAC, using two parameter vectors rather than three. In practice, both algorithms performed well, achieving very similar performance on the blocker task, but with ENATDAC performing slightly better on the stochastic policy and more consistently on the octopus arm tasks.

In conclusion, reinforcement learning is particularly challenging in high-dimensional state and action spaces. Energy-based policies provide a promising architecture for addressing this challenge. We have demonstrated that prior work on value-based reinforcement learning,

using the ESARSA algorithm, is often unsatisfactory when using highly non-linear policies such as an RBM. We have introduced two novel algorithms, based on energy-based policy gradient methods, that are more robust and effective than ESARSA. ENATDAC is guaranteed to converge to a near-local maximum in average reward; EQNAC is also robust in practice, and requires less computation. Both algorithms outperformed ESARSA on several high dimensional tasks.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270327, and from the Gatsby Charitable foundation.

References

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *JAIR*, 15, 2001.

S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45: 2471–2482, 2009.

S. Elfwing, M. Otsuka, E. Uchibe, and K. Doya. Free-energy based reinforcement learning for vision-based navigation with high-dimensional sensory inputs. In *ICONIP*, 2010.

Y. Engel, P. Szabó, and D. Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. In *NIPS*, 2005.

Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical Report UCSC-CRL-94-25, University of California, Santa Cruz, 1994.

S. Kakade. A natural policy gradient. In *NIPS*, 2001.

H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. In *ICML*, 2008.

Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F.-J. Huang. A tutorial on energy-based learning. *Predicting Structured Data*. MIT Press, 2006.

H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *NIPS*, 2009.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22, 2010.

V. Mnih, H. Larochelle, and G. E. Hinton. Conditional restricted Boltzmann machines for structured output prediction. In *UAI*, 2011.

M. Otsuka, J. Yoshimoto, and K. Doya. Free-energy-based reinforcement learning in a partially observable environment. In *ESANN*, 2010.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71, 2008.

R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In *ICML*, 2007.

B. Sallans. *Reinforcement Learning for Factored Markov Decision Processes*. PhD thesis, Department of Computer Science, University of Toronto, 2002.

B. Sallans and G. E. Hinton. Reinforcement learning with factored states and actions. *JMLR*, 5, 2004.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

G. Taylor and G. E. Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In *ICML*, 2009.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42, 1997.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *NIPS*, 2004.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.

Appendix A. Neural Network

In order to provide a comparison with an alternative structured parameterization of a stochastic policy we considered a 2-layer neural network (NN). The network's structure is very similar to the RBM but it has deterministic hidden units and a feedforward archictecture.

For binary or continuous valued states s and binary action units a the state-conditional action distribution $\pi^\theta(a;s)$ defined by the NN is given as follows:

$$\pi^\theta(a;s) = \prod_i p(a_i|s), \tag{23}$$

$$p(a_i = 1|s) = \sigma(W_a h(s) + b), \tag{24}$$

$$h(s) = \sigma(W_s s + c), \tag{25}$$

where $h(s)$ is a vector of (deterministic) hidden units with activation between 0 and 1, and $\sigma(y) = 1/(1 + \exp(-y))$ is the sigmoid function that is applied element-wise. Weight matrices W_s, W_a, and bias vectors c, and b are the parameters of the network. The above architecture is easily generalized to discrete stochastic output units with more than two states.

This NN has the same number of parameters as the RBM and is able to model nonlinear dependence of the actions on the states. Unlike in a RBM, however, the stochastic output units are conditionally independent given the state. The network therefore fails to model correlations between the action variables. In particular, it is unable to represent multi-modal state-conditional action distributions as required for the the stochastic policy task.

In our experiments we trained the NN using NATDAC with the "energy-based" state features $f(s) = \mathrm{E}^\theta_a[\phi^\theta(s,a)]$ described in section 3.1 of the main text. Due to the independence of the action units (cf. eq. 23 above) for the NN the gradient of the log-policy $\nabla_\theta \log \pi^\theta(a;s) = \psi^\theta(s,a)$ and the energy-based state features $f(s)$ can be computed analytically and the approximations described in section 3.1 of the main text are not required.

It is worth noting that it is, in principle, possible to make the hidden units stochastic while maintaining a feed-forward architecture[4]. This would introduce correlations between the action variables. At the same time, however, it would also render the exact computation of e.g. ψ intractable (except in simple cases) due to the need to integrate – or sum – out the hidden variables.

Appendix B. Additional Experiments and Results

B.1. Stochastic policy task

Note that in the stochastic policy task there is no observed state. In this case the NN reduces to a set of biases that model the activation of each action unit independently. The RBM in contrast can model correlations between the action units and hence represent the multi-modal distribution over binary action vectors required for this task.

For instance, when the NN is applied to stochastic policy task 1 with $K = 2$ unobserved states the first and last action units a_1 and a_5 are effectively set deterministically to 1 and 0 respectively, but the remaining action units a_2, a_3, a_4 will have $p(a_i = 1) = 0.5$ for $i = 2, 3, 4$ after learning. Hence, the probability of drawing one of the two "good" actions is only $\frac{1}{8}$ instead of $\frac{1}{2}$ for the RBM (which is why the NN achieves a reward of only $10/8$ instead of $10/K$.

As explained in footnote 3 in the main text we allowed 50000 actions for learning for SARSA but only the performance after 20000 actions is shown in Fig. 1 in the main text (in line with the number of actions available to the other algorithms). Figure B.1 shows the full learning curves for SARSA for the stochastic policy task.

B.2. Grid world

The grid world task requires an agent to learn how to navigate to a final state. At the beginning of each trial the agent is placed at a random position in the environment. At each step he chooses one of the four actions

4. While the RBM corresponds to an *undirected* graphical model, a feedforward NN with stochastic hidden units would correspond to a *directed* graphical model.

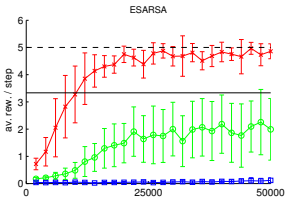


Figure B.1: Stochastic policy task: full learning curves for SARSA for all three versions of the task (red: task 1; green: task 2; blue: task 3). Same data as in Fig. 1 in the main text but all 50000 actions are shown. The dashed black line indicates the reward achieved by the optimal policy for task 1, the solid black line the optimal reward for tasks 2 and 3.

(N,S,E,W). The reward is -1 for all steps except if the agent reaches the end state, in which case it is 2. An episode ends when the agent reaches the end state or after a maximum of 20 actions. If an action cannot be performed, the agent remains at its current position. States and actions both use a 1-of-N encoding, and we also use a 1-of-N encoding for the state-value function approximation in ENATDAC, allowing it to represent the state-value function exactly. We used RBMs with 3 hidden units, and learning was performed for 40000 actions. The particular environment used in the experiments and the results obtained are shown in Fig. B.2. ENATDAC and EQNAC both find an optimal policy quickly and robustly, while ESARSA takes longer to converge to a policy, the policy found is usually not optimal, and the policy found is strongly dependent on the initialization of the policy parameters. Also, we found ESARSA to be much more sensitive to the settings of the learning parameters than our algorithms.

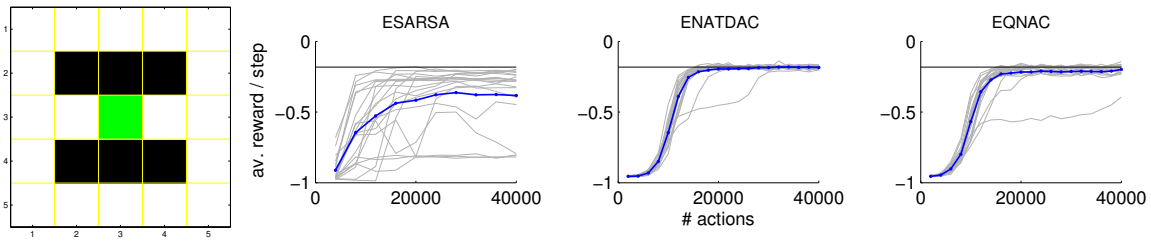


Figure B.2: 5×5 grid world: white squares are fields the agent can move to; black squares are walls; the end state is marked green. Results shown are for 20 runs. Gray curves indicate results for individual runs, blue curves show average.

B.3. Blocker Task

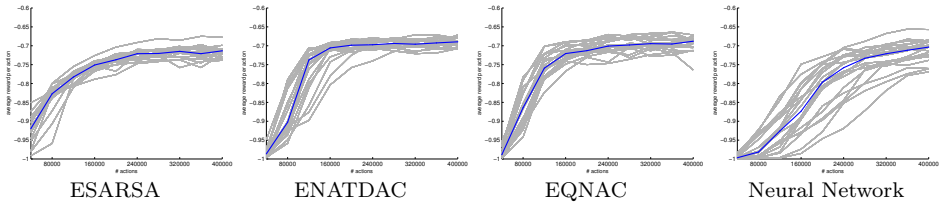
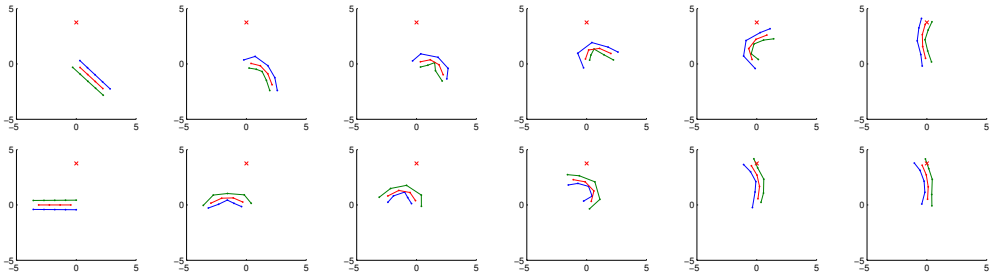
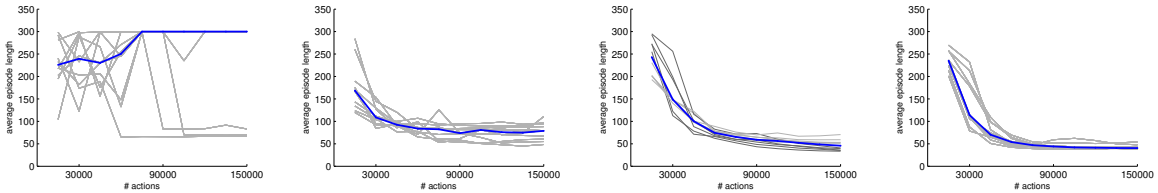


Figure B.3: Blocker task: Same data as in Fig. 4.2 in the main text but showing results for the 20 restarts with different random initializations individually for each algorithm: Gray lines show individual runs; thick blue line shows the median.



initial 5 steps 10 steps 20 steps 30 steps final

Figure B.4: Octopus arm hitting the target: Each row corresponds to a different run, starting from two different initial positions. The arm is shown in its initial position, after 5, 10, 20, and 30 steps, and after having hit the target. The target is shown by the red cross.



ESARSA EQNAC ENATDAC Neural Network

Figure B.5: Octopus arm: Average number of steps required to hit the target. Same data as in Fig. 4.3 in the main text but results are shown separately for the 10 restarts with different random initializations. The policies are evaluated every 15000 steps with 100 episodes. Gray lines correspond to individual runs, the median is shown in blue. For ENATDAC runs that got trapped in a local optimum (as explained in the text) are shown in light gray, others in dark gray.

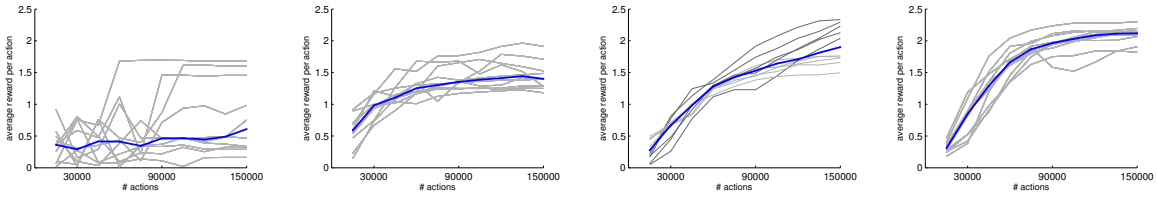
B.4. Octopus Arm

For the simulations we used the octopus arm simulator that is part of the RL Glue package[5]. We considered an arm with $C = 4$ compartments. Thus, the continuous state space was 34 dimensional, the binary action space 14 dimensional. (On the accompanying website[6] we also show results for an arm with $C = 6$ compartments, i.e. with a 50 dimensional state space, and a 20 dimensional action space.) Note that continuous valued states do note pose a problem for binary *conditional* RBMs.

At the beginning of each episode the arm is initialized at one of 4 random positions and the goal is then to hit the target as quickly as possible with any part of the arm. Fig. B.4 shows frames of two movies of the arm controlled by a learned policy hitting the target from two different initial positions. See the accompanying website for more results.

In Fig. B.6 we show as an alternative visualization of the results the average reward per step achieved with the learned policies. As pointed out in the main text ENATDAC occasionally got trapped in local optima, learning policies in which the arm rotated always in one direction independently of its initial position. With these policies the arm still hits the target reliably but requires a larger number of steps from some initial positions. This leads to a lower average reward per step at the end of learning. ENATDAC runs in which the arm learned suboptimal policies are shown in light gray in Figs. B.5, B.6.

5. http://glue.rl-community.org/wiki/Main_Page
6. http://www.gatsby.ucl.ac.uk/~nheess/papers/ebrl/



ESARSA EQNAC ENATDAC Neural Network

Figure B.6: Octopus arm: Alternative visualization of the results in Fig. B.5 (Fig. 4.3 in the main text). Average reward per step of the learned policies as a function of actions performed during learning. Results are shown for 10 restarts with different random initializations. Same format as in Fig. B.5

JMLR: Workshop and Conference Proceedings 24:59–75, 2012 10th European Workshop on Reinforcement Learning

Directed Exploration in Reinforcement Learning with Transferred Knowledge

Timothy A. Mann MANN23@TAMU.EDU
Yoonsuck Choe CHOE@TAMU.EDU
Department of Computer Science & Engineering
Texas A&M University

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

Experimental results suggest that transfer learning (TL), compared to learning from scratch, can decrease exploration by reinforcement learning (RL) algorithms. Most existing TL algorithms for RL are heuristic and may result in worse performance than learning from scratch (i.e., negative transfer). We introduce a theoretically grounded and flexible approach that transfers action-values via an intertask mapping and, based on those, explores the target task systematically. We characterize positive transfer as (1) decreasing sample complexity in the target task compared to the sample complexity of the base RL algorithm (without transferred action-values) and (2) guaranteeing that the algorithm converges to a near-optimal policy (i.e., negligible optimality loss). The sample complexity of our approach is no worse than the base algorithm's, and our analysis reveals that positive transfer can occur even with highly inaccurate and partial intertask mappings. Finally, we empirically test directed exploration with transfer in a multijoint reaching task, which highlights the value of our analysis and the robustness of our approach under imperfect conditions.

Keywords: Reinforcement Learning, Transfer Learning, Directed Exploration, Sample Complexity

1. Introduction

Transfer learning (TL) applied to reinforcement learning (RL) exploits knowledge gained while interacting with source tasks to learn faster in a target task [Lazaric, 2008; Taylor and Stone, 2009]. When TL is successful it is referred to as positive transfer, and when TL fails it is referred to as negative transfer. Previous research applying TL to RL has demonstrated its effectiveness in decreasing learning time [Selfridge et al., 1985; Taylor et al., 2007; Fernández et al., 2010], but there is a lack of theoretical understanding about when TL applied to RL will succeed or fail [Taylor and Stone, 2011]. Most TL+RL algorithms are heuristic in nature. This is in contrast with provably efficient single task RL algorithms such as R-MAX [Brafman and Tennenholtz, 2002; Kakade, 2003] and Delayed Q-learning [Strehl et al., 2006]. Lazaric and Restelli [2011] – a notable exception – has analyzed the sample complexity of TL from a batch RL perspective, but their work does not address exploration in the target task.

To learn efficiently, RL algorithms must balance between exploitation (executing a sequence of actions the algorithm has already learned will provide high long-term rewards) and exploration (trying uncertain actions to gain more information about the environment).

Exploration strategies can be broadly categorized as either undirected exploration or directed exploration [Thrun, 1992]. Undirected exploration is characterized by local, random selection of actions, while directed exploration, by contrast, uses global information to systematically determine which action to try. ϵ-greedy represents the most popular undirected exploration strategy, while "optimism in the face of uncertainty" (OFU) represents the most popular directed exploration strategy. OFU initially assumes that all actions result in higher value than may be true in the environment and greedily selects the action believed to give the highest reward (breaking ties arbitrarily). When the agent tries an action it samples its value distribution. If the sampled values are lower than expected, the algorithm lowers the action's estimated value and switches to another (possibly overestimated) action. Otherwise the algorithm sticks with its current action. In this way, the algorithm eventually settles on a nearly optimal action at every state (which results in a near-optimal policy [Kakade, 2003, Theorem 3.1.1]).

Previous research on TL+RL has almost exclusively studied undirected exploration [Taylor et al., 2007; Fernández et al., 2010] or batch transfer [Lazaric, 2008]. Although the sample complexity of exploration has been examined in the literature, to our knowledge, no previous work has formally analyzed sample complexity of exploration in the considerably more complex case of action-value transfer. Analyzing TL is more complicated because there are multiple learning algorithms and tasks that need to be considered. Our main innovation is to show how TL can be analyzed from a sample complexity perspective. We analyze action-value transfer via intertask mappings [Taylor and Stone, 2011] paired with the provably efficient Delayed Q-learning algorithm [Strehl et al., 2006] that uses the OFU exploration strategy to learn faster in the target task while avoiding optimality loss (i.e., positive transfer). Our approach has several advantages compared to previous TL approaches:

1. We can theoretically analyze our TL approach because we precisely define positive transfer in terms of sample complexity and optimality loss. The sample complexity of our approach can be compared to the sample complexity of single task RL algorithms (with respect to the target task).

2. Our analysis reveals that positive transfer can occur even when the distance between the optimal target task action-values and the transferred action-values is large.

3. Intertask mappings enable transfer between two tasks with different state-action spaces or when only a partial intertask mapping can be derived.

The rest of this paper is organized as follows: In section 2, we provide background on RL and TL. In section 3, we introduce α-weak admissible heuristics, which we use in section 4 to analyze the sample complexity and optimality loss of TL+RL. In section 5, we demonstrate the advantage of applying directed exploration to TL. In section 6 we discuss advantages and limitations of our approach and conclude in section 7.

2. Background

A Markov decision process (MDP) M is defined by a 5-tuple $\langle S, A, T, R, \gamma \rangle$ where S is a set of states, A is a set of actions, T is a set of transition probabilities $\Pr[s'|s,a]$ determining the probability of transitioning to a state $s' \in S$ immediately after selecting action $a \in A$

while in state $s \in S$, $R : S \times A \to \mathbb{R}$ assigns scalar rewards to state-action pairs, and γ is a discount factor that reduces the value of rewards distant in the future [Sutton and Barto, 1998]. We assume the transition probabilities T and the reward function R are unknown. The objective of most RL algorithms is to find a policy $\pi : S \to A$ that maximizes

$$Q^{\pi}_{M}(s_t, a_t) = E\left[R(s_t, a_t) + \sum_{\tau=t+1}^{\infty} \gamma^{\tau-t} R(s_\tau, \pi(s_\tau))\right], \quad (1)$$

the action-value for (s_t, a_t), which is the discounted, expected sum of future rewards from time t forward [Sutton and Barto, 1998]. The value function $V^{\pi}_{M}(s) = \max_{a \in A} Q^{\pi}_{M}(s, a)$. We denote the optimal value and action-value functions by V^{*}_{M} and Q^{*}_{M} (respectively) and assume that the reward function is bounded to the interval $[0, 1]$. Therefore, $0 \leq V^{*}_{M}(s) \leq \frac{1}{1-\gamma}$ for all states $s \in S$.

Sample complexity enables theoretical comparison between RL algorithms by measuring the number of samples required for an RL algorithm to achieve a learning objective. Given any MDP M, $\epsilon > 0$, and $\delta \in (0, 1]$, the *sample complexity of exploration* of an RL algorithm $\mathcal{A}$ is the number of timesteps t, such that, with probability at least $1 - \delta$,

$$V^{\mathcal{A}_t}_{M}(s_t) < V^{*}_{M}(s_t) - \epsilon \quad (2)$$

where s_t is the state and $V^{\mathcal{A}_t}_{M}$ denotes the value of $\mathcal{A}$'s policy at timestep t [Kakade, 2003]. Sample complexity that is polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $\frac{1}{1-\gamma}$, $N = |S|$, and $K = |A|$ is considered efficient. Algorithms that are provably efficient with respect to sample complexity of exploration are called PAC-MDP [Strehl et al., 2009].

A key component of algorithms with polynomial sample complexity is directed exploration. Whitehead [1991] demonstrated conditions where undirected exploration leads to exponential sample complexity, with respect to the number of states. Provably efficient algorithms such as R-MAX [Brafman and Tennenholtz, 2002; Kakade, 2003] and Delayed Q-learning [Strehl et al., 2006] use the OFU directed exploration strategy. Although the analysis in this paper can be extended to other provably efficient single task RL algorithms, we focus on Delayed Q-learning (DQL) for clarity. DQL has two parameters m and ϵ_1, where m controls the number of samples used during each update of a state-action pair and ϵ_1 controls how close to optimal the learned policy should be.

Previous research on TL+RL has primarily considered undirected exploration strategies [Taylor et al., 2007; Fernández et al., 2010] or batch RL, which does not consider the problem of exploration [Lazaric, 2008; Lazaric and Restelli, 2011]. If the transferred knowledge is not similar enough to any near-optimal policy, undirected exploration can lead to exponential sample complexity due to the same reasons it fails in single task RL. In this paper, we consider the transfer of action-values, which has been demonstrated to be effective in experiments [Selfridge et al., 1985; Taylor et al., 2007] with directed exploration. Before explaining our TL setup, we will find it useful to define a new structure called a weak admissible heuristic.

3. Action-value Initialization with Weak Admissible Heuristics

A good guess of the initial action-values can be useful in speeding up RL. A function $U : S \times A \to \mathbb{R}$ is an admissible heuristic if $Q^{*}(s, a) \leq U(s, a) \leq \frac{1}{1-\gamma}$ for all $(s, a) \in S \times A$.

Strehl et al. [2009] demonstrated that if the action-values are smaller than $\frac{1}{1-\gamma}$, then this initialization can decrease the sample complexity of exploration while maintaining PAC-MDP guarantees. Admissible heuristics provide valuable prior knowledge to PAC-MDP RL algorithms, but the specified prior knowledge does not need to be exact. This property will be useful in a TL setting for two reasons: (1) knowledge is estimated from a source task and (2) there is rarely an exact relationship between source and target tasks.

The admissible heuristic used by Strehl et al. [2009] is more restrictive than necessary. For example, Figure 3.1 shows an example where some of the initial action-values' estimates are below their corresponding optimal values, yet, following a simple OFU exploration strategy will converge to the near-optimal action b_2. Consider what would happen if the OFU exploration strategy is run on the example in Figure 3.1. First, the algorithm's initial policy would select action b_6 because the heuristic value (dotted box) is the highest. After selecting b_6 several times an update would occur decreasing the value associated with b_6 because its true value is much lower than the estimated value. Now action b_2 would be selected because it has the second highest estimated value and this estimate is very unlikely to drop below the value of any other action. So the algorithm would converge on the near-optimal action b_2. For our analysis of TL+RL, we would like to derive an extremely weak structure that will help characterize when TL will succeed and when it will fail. To help with this, we define the concept of a weak admissible heuristic.

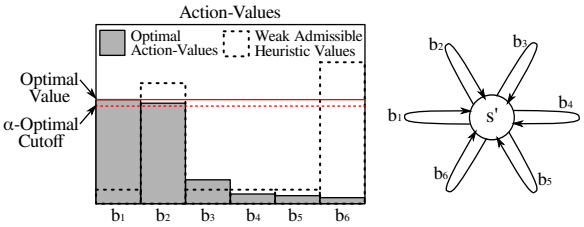


Figure 3.1: Weak admissible heuristic applied to a one-state task with six actions. The weak admissible heuristic only needs to optimistically initialize the action-value for a single nearly optimal action.

Definition 1 *A function $W : S \times A \to \mathbb{R}$ is an* **α-weak admissible heuristic** *(or just weak admissible heuristic) for MDP $M = \langle S, A, T, R, \gamma \rangle$, if for each $s \in S$, there exists $\tilde{a} \in A$ such that*

$$V^*(s) - \alpha \leq Q^*(s, \tilde{a}) \leq W(s, \tilde{a}) \leq \frac{1}{1-\gamma} \tag{3}$$

where α is the smallest non-negative value satisfying this inequality.

In other words, W is a weak admissible heuristic if at every state $s \in S$ at least one α-optimal action is mapped to an optimistic value, even though other action-values (including the optimal action) may be severely pessimistic. This is consistent with the situation in Figure 3.1, where an OFU exploration strategy converges to the near-optimal action b_2, with high probability, and not the optimal action b_1. In this case, the weak admissible heuristic implicitly eliminated several actions (including the optimal action b_1). This helped

reduce sample complexity because only one suboptimal action (b_6) was ever explored, and the algorithm still converges to a near-optimal policy.

Although weak admissible heuristics are considerably less constrained than the notion of admissible heuristic employed by Strehl et al. [2009], initializing DQL with a weak admissible heuristic (with small enough α) is sufficient to maintain PAC-MDP guarantees. For our analysis, the critical aspect of weak admissible heuristics is that they implicitly eliminate some state-action pairs from consideration (Lemma 1 below). This improves the efficiency of directed exploration because there are fewer state-action pairs to explore.

Lemma 1 *(State-action Pair Elimination) Let $\eta \geq 0$, W be an α-weak admissible heuristic for an MDP M, and $\mathcal{A}$ is a value-based RL algorithm with initial action-value estimates $\hat{Q}_0 = W$ such that for all timesteps $t \geq 1$, (1) $\mathcal{A}$ follows a greedy policy ($\mathcal{A}_t(s) = \arg\max_{a \in A} \hat{Q}_t(s,a)$), (2) Updates to $\hat{Q}_t(s,a)$ can only occur if (s,a) has been visited, and (3) If $W(s,a) \geq Q^*(s,a)$, then $\hat{Q}_t(s,a) \geq Q^*(s,a) - \eta$, then for all $(s,a) \in S \times A$ where $W(s,a) < V^*(s) - (\alpha + \eta)$, $\mathcal{A}$ will never explore (s,a) ($\mathcal{A}_t(s) \neq a$ at any timestep $t \geq 1$, where $\mathcal{A}_t$ denotes the policy of $\mathcal{A}$ at timestep t).*

Proof Since $\mathcal{A}$ follows a greedy policy with respect to $\hat{Q}_t$, a state-action pair (s,a) will only be selected if $\hat{Q}_t(s,a) = \max_{a' \in A} \hat{Q}_t(s,a')$. The proof is by induction on the timestep t. Suppose, without loss of generality, that $W(s,a) < V^*(s) - (\alpha + \eta)$.

Base Case ($t = 0$): By the definition of W there exists $\tilde{a}$ such that $W(s,\tilde{a}) \geq Q^*(s,\tilde{a}) \geq V^*(s) - \alpha > W(s,a)$. Thus $a \neq \arg\max_{a' \in A} W(s,a') = \arg\max_{a' \in A} \hat{Q}_0(s,a')$.

Induction Step: By assumption 3 the action-value estimate for $\hat{Q}_t(s,\tilde{a}) \geq Q^*(s,\tilde{a}) - \eta \geq (V^*(s) - (\alpha + \eta)) > W(s,a)$. Since (s,a) has not been tried yet no update to its action-value could have occurred (assumption 2), and will not be executed on timestep $t+1$ because $\hat{Q}_t(s,a) = W(s,a) < \hat{Q}_t(s,\tilde{a})$.

Therefore by induction, (s,a) will never be executed. ■

If W is a weak admissible heuristic, then Lemma 1 says that low valued state-action pairs will never be explored by the algorithm's policy. Condition 3 guarantees that the algorithm will never underestimate action-values (by more than η) that are initially optimistic. This combined with the weak admissible heuristic assumption allows us to guarantee that an α-optimal action will eventually be the action selected by the algorithm's greedy policy. DQL satisfies the algorithmic requirements of Lemma 1, with high probability.

Theorem 2 *Let $\alpha \geq 0$, $\epsilon > 0$, $\delta \in (0,1]$, and W be an α-weak admissible heuristic with respect to $M = \langle S, A, T, R, \gamma \rangle$. There exists $\epsilon_1 = O(\epsilon(1-\gamma))$ and $m = O\left(\frac{\ln(NK/\delta)}{\epsilon_1^2(1-\gamma)^2}\right)$, such that if $\mathcal{A}$ is an instance of the DQL algorithm initialized with parameters ϵ_1 and m, then $V^{\mathcal{A}_t}(s_t) \geq V^*(s_t) - (\epsilon + \frac{\alpha}{1-\gamma})$ on all but*

$$O\left(\frac{NK - X}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{NK}{\delta\epsilon(1-\gamma)}\right) \tag{4}$$

timesteps t, with probability at least $1-\delta$, where $X = \left|\left\{(s,a) \in S \times A \mid W(s,a) < V^*(s) - \alpha - \frac{\alpha}{1-\gamma}\right\}\right|$.

The proof of this theorem is similar to [Strehl et al., 2006, Theorem 1], which provides the following sample complexity bound for DQL without transferred knowledge:

$$O\left(\frac{NK}{\epsilon^4(1-\gamma)^8}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\ln\frac{NK}{\delta\epsilon(1-\gamma)}\right)$$

Our proof relies on Lemma 1 to demonstrate its dependence on $NK - X \leq NK$. See the Appendix for a proof of Theorem 2.

If $X > 0$ and α is small, then the sample complexity bound depends on $\tilde{O}(NK - X)$, which is smaller than the lower bound for any single task RL algorithm $\Omega(NK)$ with respect to the number of states and actions Strehl et al. [2009]. If $X = 0$ and $\alpha = 0$, then we restore the upper bound from [Strehl et al., 2006, Theorem 1]. Therefore, a weak admissible heuristic can help to decrease the sample complexity of exploration.

4. Transferring a Weak Admissible Heuristic

We use the concept of a weak admissible heuristic to analyze action-value transfer with the objective of learning to act near-optimally in the target task with sample complexity of exploration (Eq. 2) that is smaller than DQL without transferred knowledge. We denote the source task/MDP by M_{src} and the target task/MDP by M_{trg}. Consider the situation in Figure 4.1. First the agent learns action-values for the source task. Next, because the source task and the target task have a different number of actions, a function h called an intertask mapping (defined below) is used to relate action-values from the source task to the target task. Finally, notice that in Figure 4.1 the transferred action-values satisfy a weak admissible heuristic. In this section, we explore assumptions about the intertask mapping needed to ensure that the transferred action-values satisfy a weak admissible heuristic and how transfer influences sample complexity of exploration in the target task.

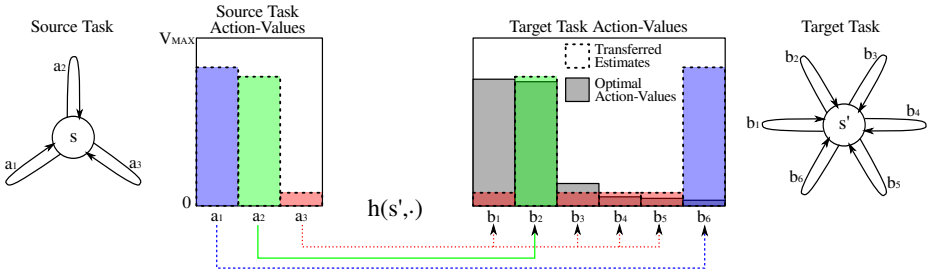


Figure 4.1: Transfer from a one-state source task with three actions to a one-state target task with six actions. Despite the transferred action-values severely underestimating the optimal action b_1 and severely overestimating the lowest valued action b_6, an OFU exploration strategy can still converge to a near-optimal policy (i.e., b_2).

There are two factors that affect positive transfer: (1) sample complexity and (2) optimality loss. Positive transfer occurs when the sample complexity of exploration in the target task is lower than the sample complexity of the base RL algorithm and no optimality loss

has occurred. Optimality loss occurs when transferred knowledge causes an RL algorithm to converge to a suboptimal policy along its current trajectory.

Typically, access to samples of the source task is less "expensive" than access to samples from the target task. For the purposes of this paper, we assume unrestricted access to a generative model for M_{src}. Using, for example, the PhasedValueIteration algorithm [Kakade, 2003] it is possible to learn arbitrarily accurate source task action-values with arbitrarily high confidence with polynomially many samples. Therefore, we will assume that the estimated source task action-values $\hat{Q}_{\text{src}}$ are ϵ_{src}-accurate.

If M_{src} and M_{trg} have different state-action spaces, then an intertask mapping $h : D \to S_{\text{src}} \times A_{\text{src}}$ is needed, where $D \subseteq S_{\text{trg}} \times A_{\text{trg}}$, to relate a subset of state-action pairs from the target task to state-action pairs in the source task. We assume that if $(s, a) \in D$, either there exists $(s, \tilde{a}) \in D$ such that

$$V^*_{\text{trg}}(s) - \alpha \leq Q^*_{\text{trg}}(s, \tilde{a}) \leq Q^*_{\text{src}}(h(s, \tilde{a})) \ , \tag{5}$$

which is analogous to (3) with $W(s, a) = Q^*_{\text{src}}(h(s, a))$ or there exists $(s, \tilde{a}) \notin D$ such that

$$V^*_{\text{trg}}(s) - \alpha \leq Q^*_{\text{trg}}(s, \tilde{a}) \tag{6}$$

in which case we can assign the value $W(s, \tilde{a}) = \frac{1}{1-\gamma}$. To transfer action-values we use

$$W(s, a) = \begin{cases} \min\left(\hat{Q}_{\text{src}}(h(s, a)) + \epsilon_{\text{src}}, \frac{1}{1-\gamma}\right) & \text{if } (s, a) \in D \\ \frac{1}{1-\gamma} & \text{otherwise} \end{cases} \tag{7}$$

to set initial action-value estimates given an intertask mapping h, and ϵ_{src}-accurate source task action-value estimates $\hat{Q}_{\text{src}}$. At every state at least one nearly optimal action is mapped to an action-value which overestimates the true action-value or not mapped at all. If a state-action pair is not in the domain D, then we assign the maximum possible value to ensure it is optimistically initialized. Under these assumptions the transferred action-values are an α-weak admissible heuristic.

Theorem 3 *Let $\epsilon > 0$, $\epsilon_{\text{src}} > 0$, $\delta \in (0, 1]$, $h : D \to S_{\text{src}} \times A_{\text{src}}$ be an intertask mapping from a subset of state-action pairs in M_{trg} to M_{src} satisfying (5) and (6), and $\hat{Q}_{\text{src}}$ are ϵ_{src}-accurate action-value estimates for M_{src}. There exists $\epsilon_1 = O(\epsilon(1-\gamma))$ and $m = O\left(\frac{\ln(NK/\delta)}{\epsilon_1^2(1-\gamma)^2}\right)$ such that if an instance $\mathcal{A}$ of the DQL algorithm with ϵ_1, m, and action-value estimates initialized by Eq. (7) is executed on M_{trg}, then $V^{\mathcal{A}_t}_{\text{trg}}(s) < V^*_{\text{trg}}(s) - (\epsilon + \frac{\alpha}{1-\gamma})$ occurs on at most*

$$O\left(\frac{NK - Y}{\epsilon^4(1-\gamma)^8} \ln\frac{1}{\delta} \ln\frac{1}{\epsilon(1-\gamma)} \ln\frac{NK}{\delta\epsilon(1-\gamma)}\right)$$

timesteps t, with probability at least $1 - \delta$, where $N = |S|$, $K = |A|$, and

$$Y = \left|\left\{(s, a) \in D \mid \hat{Q}_{\text{src}}(h(s, a)) < V^*_{\text{trg}}(s) - (\alpha + \frac{\alpha}{1-\gamma} + \epsilon_{\text{src}})\right\}\right|$$

is the number of state-action pairs that will never be explored.

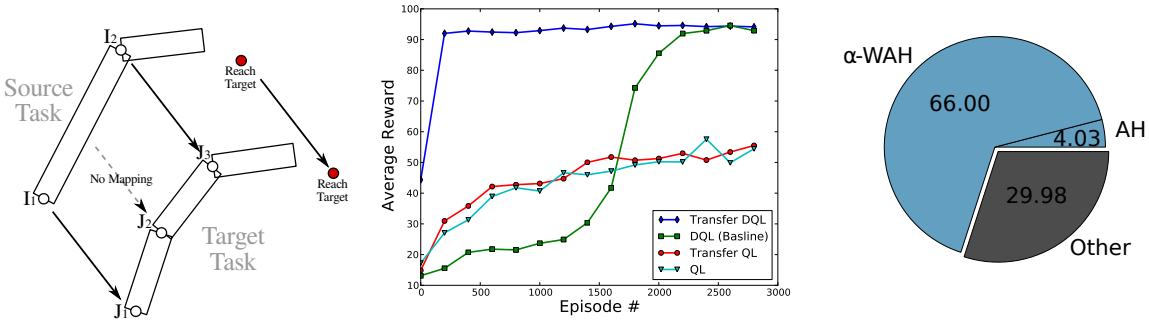
See Appendix for a proof of Theorem 3. The main importance of Theorem 3 is that we have reduced the analysis of action-value transfer to the analysis of learning with an α-weak admissible heuristic. Here, α controls optimality loss, and we can think of α (or $\alpha/(1-\gamma)$) as the error introduced by the intertask mapping h. If $\alpha \approx 0$ compared to ϵ, then there is little or no optimality loss compared to learning from scratch. In many cases, $\alpha = 0$ can be achieved, for example, if W turns out to be an admissible heuristic [Strehl et al., 2009]. However, if α is large, then the result of TL is likely to be poor in the worst case. Similar to X in Theorem 2, when Y is large the sample complexity of exploration in the target task decreases significantly compared to learning from scratch with DQL. Notice, however, that the sample complexity is never worse than learning from scratch. The main consideration should be identifying an intertask mapping that does not introduce much optimality loss. Thus, TL is characterized by optimality loss and sample complexity, and Theorem 3 helps to clarify this relationship.

5. Experiments and Results

Our experimental tasks were inverse kinematics problems (Figure 5.1a). In the source task M_{src}, the agent controlled a two-joint mechanical arm guiding its end-effector to one of four reach target locations. In the target task M_{trg}, the agent controlled a three-joint mechanical arm with noisy actuators by moving its end-effector to one of four reach target locations. In both tasks the state was represented by the target index and the joint locations. The actions encode whether to rotate each joint and in which direction. Actions only perturbed joints by a small amount. So a sequence of actions was needed to complete each task. In the target task there was a small probability (0.2) that the action applied to a joint would either leave the joint unchanged or overshoot the desired location. Because of the different number of joints, there is no one-to-one mapping between the state-action space of the source task and the state-action space of the target task. We defined an intertask mapping that relates the reach target index, and joints J_1 to I_1 and J_3 to I_2, leaving out joint J_2.

We compared DQL, which uses directed exploration, with and without transferred knowledge, and Q-learning (QL) using an ε-greedy exploration strategy with and without transferred knowledge. Figure 5.1b demonstrates that the transferred action-values enable DQL to quickly converge to a near-optimal solution, while DQL (with the same parameters) learning from scratch takes much longer to learn a good policy. Both QL conditions perform worse than the DQL conditions because local exploration is not efficient in the target task's large state-action space. The negligible difference between QL and Transfer QL is due to the fact that the transferred action-values are a very poor approximation to the optimal action-values.

DQL and Transfer DQL spend most of their timesteps in a small number of states (less than 50 out of 1,372). Figure 5.1c shows the average proportion of highly visited states where the transferred action-values satisfy the admissible heuristic (AH) and α-weak admissible heuristic criteria with $\alpha = 0.1$. Most of the transferred action-values at these highly visited states fall under α-WAH and AH conditions. Note that AH $\subseteq$ α-WAH. A small proportion do not satisfy α-WAH (i.e., Other). For the states satisfying α-WAH, we found that the policy learned by Transfer DQL selected good action choices (i.e., $Q^*_{\text{trg}}(s, \pi(s)) \geq V^*_{\text{trg}}(s) - \alpha$) almost 100% of the time. Interestingly, Transfer DQL performs well despite the fact that



(a) (b) (c)

Figure 5.1: (a) Transfer between inverse kinematic tasks where the source task controls a two-joint arm, and the target task controls a three-joint arm. (b) Average reward over 3,000 episodes in the target task for Delayed Q-learning without transfer (DQL (Baseline)), Delayed Q-learning with transferred action-values (Transfer DQL), Q-learning with transferred action-values (Transfer QL), and Q-learning without transfer (QL). Transfer DQL learns much more quickly than the compared algorithms. (c) Percentage of transferred action-values (out of highly visited states) satisfying the admissible heuristic (AH) and α-weak admissible heuristic (α-WAH) criteria. Note: AH $\subseteq$ α-WAH.

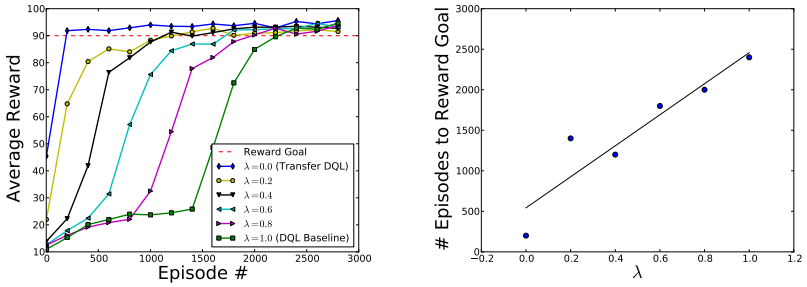
some of the highly visited states do not satisfy the α-WAH condition. This demonstrates the robustness of our TL approach in practice and is an important observation since we would not expect in practice to be able to rigerously guarantee (5) and (6) for all states. These result suggest that the weak admissible heuristic concept is important for understanding when transfer learning succeeds. On the other hand, in practice, guaranteeing that the transferred action-values satisfy the weak admissible heuristic criterion at every state is not necessary to achieve positive transfer.

We considered the impact of TL with partial intertask mappings by modifying the intertask mapping from the previous experiment. State-action pairs from its domain were randomly removed with probability λ. Figure 5.2 shows the average reward curve for Transfer DQL as λ is varied from 0.0 to 1.0. The training time has an approximately linear relationship with λ, increasing as λ increases.

6. Discussion

The main advantages of our approach are that it can be theoretically analyzed and that it works with several existing provably efficient algorithms, such as R-MAX, Modified R-MAX, and DQL. This approach may also work with future RL algorithms. We have shown that the transferred action-values do not need to be accurate (with respect to L_1-norm), and our analysis holds for MDPs with stochastic transitions and rewards.

The main limitation of our approach is that if the action-values do not satisfy a weak admissible heuristic with small α, then the final learned policy may be poor and the algorithm



(a) (b)

Figure 5.2: (*a*) Average reward for Transfer DQL as state-action pairs are removed from the intertask mapping's domain (with probability λ). (*b*) Increase in training time is approximately proportional to the probability that state-action pairs are removed.

will never recover. Another potential objection to our method is that there is no general way to obtain a good intertask mapping between tasks without solving both tasks. However, there are interesting special cases, for example, where an identity intertask mapping is used because the tasks only differ by their dynamics or reward structure.

7. Conclusion

We theoretically analyzed the combination of (1) action-value transfer via an intertask mapping with (2) directed exploration and found that the approach has several provable benefits. Only weak assumptions on the intertask mapping are necessary for positive transfer. Positive transfer can occur even when the distance between the transferred action-values and the optimal target task action-values is large. Although we have analyzed our TL approach using DQL, our analysis can be extended to other efficient algorithms such as R-MAX [Brafman and Tennenholtz, 2002] or Modified R-MAX [Szita and Szepesvári, 2010]. Finally, we demonstrated these advantages empirically on an inverse kinematics task. For states where the intertask mapping induced a weak admissible heuristic, DQL almost always learned to select a good action. Furthermore, our TL approach performed well even though a few highly visited states did not satisfy the weak admissible heuristic criteria, suggesting the approach is robust in practice. Although previous research has mostly paired undirected exploration with TL, we believe that directed exploration is an important mechanism for developing provably efficient TL+RL algorithms.

Acknowledgments

We acknowledge the EWRL reviewers and Matthew Taylor for their helpful feedback. This report is based in part on work supported by Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST) and the Texas A&M University Dissertation Fellowship.

References

Ronen I. Brafman and Moshe Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

Fernando Fernández, Javier Garcá, and Manuela Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58:866–871, 2010.

Sham Machandranath Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University College London, March 2003.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

Alessandro Lazaric. *Knowledge Transfer in Reinforcement Learning.* PhD thesis, Politecnico Di Milano, 2008.

Alessandro Lazaric and Marcello Restelli. Transfer from multiple MDPs. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1746–1754, 2011.

Oliver G. Selfridge, Richard Sutton, and Andrew Barto. Training and tracking in robotics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 670–672, 1985.

Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *Proceedings of the Twenty-third International Conference on Machine Learning (ICML-06)*, 2006.

Alexander L. Strehl, Lihong Li, and Michael Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

István Szita and Csaba Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

Mathew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

Matthew E. Taylor and Peter Stone. An introduction to inter-task transfer for reinforcement learning. *AI Magazine*, 32(1):15–34, 2011.

Sebastian Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University, January 1992.

Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical report, University of Rochester, 1991.

Appendix

In this appendix we provide proofs for Theorems 2 and 3. Throughout this appendix we assume that M is an MDP defined by $\langle S, A, T, R, \gamma \rangle$ and W is an α-weak admissible heuristic for some $\alpha > 0$, unless otherwise noted. To prove Theorem 2 we need to introduce several concepts and lemmas.

7.1. PAC-MDP Framework

Analysis of PAC-MDP algorithms typically depends on the notion of an induced MDP [Kakade, 2003; Strehl et al., 2006, 2009; Szita and Szepesvári, 2010]. We introduce a modified version of the induced MDP that depends on how action-values are initialized. We assume that the RL algorithm maintains action-values $\hat{Q}_t$ initialized by $W = \hat{Q}_0$. State-action pairs with optimistically initialized action-values (i.e., $W(s,a) \geq Q^*_M(s,a)$) are treated separately from state-action pairs with pessimistically initialized action-values (i.e., $W(s,a) < Q^*_M(s,a)$). This allows us to essentially ignore pessimistically initialized state-action pairs and focus on optimistically initialized state-action pairs.

Definition 2 *Let $M = \langle S, A, T, R, \gamma \rangle$ be an MDP, $\kappa \subseteq S \times A$, and W be an α-weak admissible heuristic with respect to M. The* **induced MDP** *with respect to M and κ is denoted by $M_\kappa = \langle S \cup \{z_{s,a} \mid (s,a) \notin \kappa\}, A, T_\kappa, R_\kappa, \gamma \rangle$ where*

$$T_\kappa(s'|s,a) = \begin{cases} T(s'|s,a) & \text{if } (s,a) \in \kappa \text{ and } W(s,a) \geq Q^*_M(s,a) \\ 1 & \text{if } ((s,a) \notin \kappa \text{ and } s' = z_{s,a}) \text{ or } (s \in \{z_{s,a} \mid (s,a) \notin \kappa\} \text{ and } s' = s) \\ 0 & \text{otherwise} \end{cases}$$

and

$$R_\kappa(s,a) = \begin{cases} R(s,a) & \text{if } (s,a) \in \kappa \text{ and } W(s,a) \geq Q^*_M(s,a) \\ (1-\gamma)\hat{Q}(s,a) & \text{otherwise} \end{cases}$$

where T_κ defines the transitions probabilities and R_κ defines the reward function of M_κ.

The purpose of this more complex version of the induced MDP is to obtain the following action-value function. Suppose that a value-based RL algorithm is initialized with action-values $W = \hat{Q}_0$, π_t is the policy followed by the RL algorithm at timestep t and $\kappa_t \subseteq S \times A$, then the induced MDP M_{κ_t} has action-values defined by

$$Q^{\pi_t}_{M_{\kappa_t}}(s,a) = \begin{cases} R(s,a) + \gamma \sum_{s' \in S} V^{\pi}_{M_{\kappa_t}}(s') & \text{if } (s,a) \in \kappa_t \text{ and } W(s,a) \geq Q^*_M(s,a) \\ \hat{Q}_t(s,a) & \text{if } (s,a) \notin \kappa_t \text{ or } W(s,a) < Q^*_M(s,a) \end{cases} \tag{8}$$

for all $(s,a) \in S \times A$. The induced MDP represents an idealized version of what the RL algorithm has learned. The set κ represents the number of state-action pairs that have been experienced enough to be considered "well modeled". In the standard analysis of Delayed Q-learning, the induced MDP M_κ becomes equivalent to M, if $\kappa \equiv S \times A$. In our analysis, we show instead that when it is difficult to escape from the set κ, then an ϵ-optimal policy in M_κ is an $\left(\epsilon + \frac{\alpha}{1-\gamma}\right)$-optimal policy in M.

Strehl et al. [2009] introduced a meta-PAC-MDP theorem that has been successfully used to analyze a number of RL algorithms. The next theorem is a minor generalization of [Strehl et al., 2009, Theorem 10].

Theorem 4 *[Strehl et al., 2009, Theorem 10] Let $\epsilon > 0$, $\delta \in (0,1]$, and $\mathcal{A}(\epsilon, \delta)$ be any value-based greedy learning algorithm such that, for every timestep t, $\mathcal{A}(\epsilon, \delta)$ maintains action-value estimates $\hat{Q}_t \leq \frac{1}{1-\gamma}$ and there exists a set κ_t of state-action pairs that depends on the agent's history up to timestep t. We denote $\max_{a \in A} \hat{Q}_t(s,a)$ by $\hat{V}_t(s)$ and assume that the set κ does not change unless an action-value is updated (i.e., $\kappa_t = \kappa_{t+1}$ unless, $\hat{Q}_t \neq \hat{Q}_{t+1}$) or an escape event occurs. Let M_{κ_t}*

be the known state-action MDP with respect to MDP M and W and π_t be the greedy policy followed by $\mathcal{A}_t$. Suppose that with probability at least $1-\delta$ the following conditions hold for all state-action pairs $s \in S$ and timesteps $t \geq 1$:

Condition 1: $\hat{V}_t(s) \geq V^*_M(s) - \beta$ *(optimism),*

Condition 2: $\hat{V}_t(s) - V^{\mathcal{A}_t}_{M_{\kappa_t}}(s) \leq \epsilon$ *(accuracy), and*

Condition 3: *the total number of updates of action-value estimates plus the number of times the escape event from κ_t can occur is bounded by $\zeta(\epsilon, \delta)$ (learning complexity).*

If $\mathcal{A}(\epsilon, \delta)$ is executed on M it will follow a $(3\epsilon + \beta)$-optimal policy on all but

$$O\left(\frac{\zeta(\epsilon,\delta)}{\epsilon^2(1-\gamma)^2}\ln\frac{1}{\delta}\ln\frac{1}{\epsilon(1-\gamma)}\right)$$

timesteps t with probability at least $1-2\delta$.

The key difference between this version of the theorem and the one reported in Strehl et al. [2009] is that we have separated the error variable β in condition 1 from the error variable ϵ in condition 2, which allows for a tighter optimality bound in our proof of Theorem 2. We omit the complete proof of Theorem 4 because our modification from [Strehl et al., 2009, Theorem 10] requires only minor changes to the proof. However, due to our modified notion of induced MDP, we discuss why the core inequalities hold without modification. The main objective is to argue that the policy followed by the algorithm is either a near-optimal policy in the MDP M or the algorithm's policy is likely to experience a state-action pair that is outside of the set of "well-modeled" experiences. Since the latter can only happen a finite number of times before there are no more "poorly modeled" state-action pairs (because our knowledge of the state-action pair improves with each experience), then eventually the algorithm must follow a near-optimal policy in M. In the proof of Strehl et al. [2009, Theorem 10], the argument starts by comparing the algorithm's value in M over a finite sequence of timesteps H to the same policy π_t in the induced MDP M_{κ_t} over H timesteps minus the probability of Z, an escape event will occur during H timesteps, times $\frac{2}{1-\gamma}$. In our definition of induced MDP the equation

$$V^{\mathcal{A}_t}_M(s,H) \geq V^{\pi_t}_{M_{\kappa_t}}(s,H) - \Pr[Z]\left(\frac{2}{1-\gamma}\right)$$

still holds because π_t always has a smaller value in the induced MDP while in κ_t and outside of κ_t the difference must be bounded by $\left(\frac{2}{1-\gamma}\right)$ since $\frac{1}{1-\gamma}$ is the maximum possible value in both M and M_{κ_t}. Furthermore, the policy π_t only escapes from κ_t with probability $\Pr[Z]$. This enables us to reproduce the core inequalities used by Strehl et al. [2009] to prove their Theorem 10:

$$\begin{aligned}
V^{\mathcal{A}_t}_M(s,H) &\geq V^{\pi_t}_{M_{\kappa_t}}(s,H) - \Pr[Z]\left(\tfrac{2}{1-\gamma}\right) \\
&\geq V^{\pi_t}_{M_{\kappa_t}}(s) - \epsilon - \Pr[Z]\left(\tfrac{2}{1-\gamma}\right) \\
&\geq \hat{V}_t(s) - 2\epsilon - \Pr[Z]\left(\tfrac{2}{1-\gamma}\right) \\
&\geq V^*_M(s) - \beta - 2\epsilon - \Pr[Z]\left(\tfrac{2}{1-\gamma}\right)
\end{aligned}$$

where the second step is obtained by choosing H (see Kearns and Singh [2002, Lemma 2]) large enough so that $V^{\pi_t}_{M_{\kappa_t}}(s) - V^{\pi_t}_{M_{\kappa_t}}(s,H) \leq \epsilon$. The third step is due to Condition 2 ($\hat{V}_t(s) - V^{\mathcal{A}_t}_{M_{\kappa_t}}(s) \leq \epsilon$), and the final step is due to Condition 1 ($\hat{V}_t(s) \geq V^*_M(s) - \beta$).

7.2. Delayed Q-learning Background & Lemmas

Similar to Strehl et al. [2009], our analysis of Delayed Q-learning will apply Theorem 4. The analysis of Delayed Q-learning mostly consists of finding appropriate values for arguments $m > 0$ and $\epsilon_1 > 0$ that depend on $\epsilon > 0$ and $\delta \in (0, 1]$, where increasing the value of m provides more statistical accuracy by averaging over more samples and decreasing ϵ_1 causes the algorithm to learn to act closer to optimal. Our analysis is similar to Strehl et al. [2006, 2009], but there are significant differences due to initializing the action-value estimates $\hat{Q}$ with an α-weak admissible heuristic.

We have assumed that the immediate rewards are bound to the interval $[0, 1]$. In Strehl et al. [2006], the Delayed Q-learning algorithm worked by initializing its action-values to $\frac{1}{1-\gamma}$ (the maximum possible action-value) and decreasing these estimates in a series of updates. Here the main difference is that Delayed Q-learning is initialized by $\hat{Q}_0 = W$. Initializing the action-value estimates with a weak admissible heuristic introduces several technical issues that are not addressed by Strehl et al. [2006] or Strehl et al. [2009]. Delayed Q-learning is called "delayed" because it updates action-value estimates in a series of batches of m samples from a state-action pair (s, a) before attempting to update (s, a)'s action-value estimate. It may collect many batches of m samples from each state-action pair. To know when to stop updating a state-action pair, Delayed Q-learning maintains a Boolean value for each state-action pair. These Boolean values are called the $LEARN$ flags.

Definition 3 *A* **batch** *of m samples*

$$AU(s, a) = \frac{1}{m} \sum_{i=1}^{m} \left(R_{k_i}(s, a) + \gamma \max_{a' \in A_{s'}} \hat{Q}_{k_i}(s', a') \right) \tag{9}$$

occurring at timesteps $k_1 < k_2 < \cdots < k_m$ for a state-action pair (s, a) consists of a sequence of m visits to (s, a) where the first visit occurs at a timestep k_1 corresponding to either the first timestep that (s, a) is visited by the algorithm or during the most recent prior visit to (s, a) at timestep $k' < k_1$, $LEARN_{k'}(s, a) = true$ and $l_{k'}(s, a) = m$ was true. A batch of samples is said to be completed on the first timestep $t > k_1$ such that $LEARN_t(s, a) = true$ and $l_t(s, a) = m$.

Delayed Q-learning has three notions of update:

1. Attempted Updates : An attempted update occurs when a batch of m samples has just been completed.
2. Update (or Successful Updates) : A successful update to a state-action pair (s, a) occurs when a completed batch of m samples at timestep t causes a change to the action-value estimates so that $\hat{Q}_t(s, a) \neq \hat{Q}_{t+1}(s, a)$.
3. Unsuccessful Updates : An unsuccessful update occurs when a batch of m samples completes but no change occurs to the action-values.

An attempted update to a state-action pair (s, a) at timestep t is successful if

$$\hat{Q}_t(s, a) - AU_t(s, a) \geq 2\epsilon_1 \tag{10}$$

the completed batch of samples is significantly lower ($< 2\epsilon_1$) than the current action-value estimate, and a successful update assigns

$$\hat{Q}_{t+1}(s, a) = AU_t(s, a) + \epsilon_1 \tag{11}$$

the completed batch of samples plus a small constant to the new action-value estimate. These rules guarantee that whenever a successful update occurs, the action-value estimates decrease by at least ϵ_1.

Lemma 5 *[Strehl et al., 2009, Lemma 19] No more than* $u = NK\left(1 + \frac{NK}{\epsilon_1(1-\gamma)}\right)$ *attempted updates can occurs during an execution of Delayed Q-learning on* M.

Lemma 5 bounds the total number of attempted updates that can possibly occur during the execution of Delayed Q-learning. This lemma is important because it allows us to bound the total probability that a failure event will occur.

The set

$$\kappa_t = \left\{(s,a) \in S \times A \mid \hat{Q}_t(s,a) - \left(R(s,a) - \gamma \sum_{s' \in S} T(s'|s,a)\hat{V}_t(s')\right) \leq 3\epsilon_1\right\} \tag{12}$$

consists of state-action pairs with small Bellman residual. Because the state-action pairs in κ_t have low Bellman error, Szita and Szepesvári [2010] has called Eq. (12) the "nice" set. The set κ_t is somewhat analogous to the known-state MDP used in the analysis of R-MAX. However, unlike R-MAX, the algorithm cannot determine which state-action pairs are actually in this set. It is used strictly for the purposes of analysis.

Let $\mathcal{X}_1$ denote the event that when Delayed Q-learning is executed on M, then every time k_1 when a new batch of samples for some state-action pair (s,a) begins, if $(s,a) \notin \kappa_{k_1}$ and the batch is completed at timestep k_m, then a successful update to (s,a) will occur at timestep k_m.

Lemma 6 *[Strehl et al., 2006, Lemma 1] If Delayed Q-learning is executed on* M *with parameters* m *and* ϵ_1 *where* m *satisfies*

$$m = \frac{1}{2\epsilon_1^2(1-\gamma)^2} \ln \frac{u}{\delta} \tag{13}$$

then event $\mathcal{X}_1$ *will occur, with probability at least* $1 - \delta$.

Lemma 6 establishes a value of m that is large enough to ensure that event $\mathcal{X}$ occurs with high probability.

The next lemma is a modified version of Strehl et al. [2006, Lemma 2]. The reason that this lemma needs to be modified is because not all of the action-value estimates maintained by Delayed Q-learning will be optimistic, since they are initialized according to the α-weak admissible heuristic W. Instead we show that the action-values that were initialized optimistically will remain approximately optimistic. In addition to the original purpose for this lemma, it also shows how Delayed Q-learning can be made to satisfy Condition 3 of Lemma 1, which we will use in our proof of Theorem 2.

Let $\mathcal{X}_2$ be the event that for all timesteps $t \geq 0$ and $(s,a) \in S \times A$, if $W(s,a) = \hat{Q}_0(s,a) \geq Q^*_M(s,a)$, then $\hat{Q}_t(s,a) \geq Q^*_M(s,a) - \frac{\alpha}{1-\gamma}$.

Lemma 7 *If Delayed Q-learning is initialized by the* α*-weak admissible heuristic* W *and is executed with* m *determined by (13), then event* $\mathcal{X}_2$ *occurs with probability at least* $1 - \delta$.

Proof We prove the claim by induction on the timestep t.

Base Case ($t = 0$): By our assumption if $W(s,a) \geq Q^*_M(s,a)$, then $\hat{Q}_0(s,a) = W(s,a) \geq Q^*_M(s,a) > Q^*_M(s,a) - \frac{\alpha}{1-\gamma}$.

Induction Step ($t > 0$): Suppose that for all timesteps $\tau = 1, 2, \ldots, t-1$, if $W(s,a) \geq Q^*_M(s,a)$, then $\hat{Q}_\tau(s,a) \geq Q^*_M(s,a) - \frac{\alpha}{1-\gamma}$. If during timestep $t-1$ no successful update occurs, then the action-value estimates are unchanged implying that $\hat{Q}_t(s,a) = \hat{Q}_{t-1}(s,a) \geq Q^*_M(s,a) - \frac{\alpha}{1-\gamma}$. On the other hand, if a successful update occurs during timestep $t-1$ at some state-action pair (s,a), then by the update rule given by (11)

$$\begin{aligned} \hat{Q}_t(s,a) &= AU(s,a) + \epsilon_1 \\ &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m}\left(R_{k_i}(s,a) + \gamma \hat{V}_{k_i}(s'_{k_i})\right) + \epsilon_1 \end{aligned}$$

where $\hat{V}_{k_i}(s) = \max_{a \in A} \hat{Q}_{k_i}(s,a)$ for $k_1 < k_2 < \cdots < k_m = t-1$. Notice that

$$\hat{V}_{k_i}(s) \geq V^*_M(s) - \alpha - \frac{\alpha}{1-\gamma} \tag{14}$$

for all $s \in S$ and $i = 1, 2, \ldots, m$, because the definition of an α-weak admissible heuristic guarantees that there exists $\tilde{a} \in A$ such that $W(s,\tilde{a}) \geq Q^*_M(s,\tilde{a}) \geq V^*_M(s) - \alpha$. This implies that $\hat{V}_{k_i}(s) = \max_{a \in A} \hat{Q}_{k_i}(s,a) \geq Q^*_M(s,\tilde{a}) - \frac{\alpha}{1-\gamma} \geq V^*_M(s) - \alpha - \frac{\alpha}{1-\gamma}$ for every state $s \in S$. Now we can plug (14) into the previous inequality to obtain

$$\begin{aligned} \hat{Q}_t(s,a) &= \tfrac{1}{m}\textstyle\sum_{i=1}^m \left(R_{k_i}(s,a) + \gamma \hat{V}_{k_i}(s'_{k_i})\right) + \epsilon_1 \\ &\geq \tfrac{1}{m}\textstyle\sum_{i=1}^m \left(R_{k_i}(s,a) + \gamma \left(V^*_M(s'_{k_i}) - \alpha - \tfrac{\alpha}{1-\gamma}\right)\right) + \epsilon_1 \\ &= \tfrac{1}{m}\textstyle\sum_{i=1}^m \left(R_{k_i}(s,a) + \gamma V^*_M(s'_{k_i})\right) - \tfrac{\alpha}{1-\gamma} + \epsilon_1 \end{aligned} \quad .$$

Now we define the random variables $J_i = \left(R_{k_i}(s,a) + \gamma V^*_M(s'_{k_i})\right)$ for $i = 1, 2, \ldots, m$ bound by the interval $\left[0, \frac{1}{1-\gamma}\right]$. By the Hoeffding inequality and our choice of m (13), $\frac{1}{m}\sum_{i=1}^m J_i \geq E[J_1] - \epsilon_1$ with probability at least $1 - \frac{\delta}{u}$. Thus

$$\begin{aligned} \hat{Q}_t(s,a) &\geq E[J_1] - \epsilon_1 + \epsilon_1 - \tfrac{\alpha}{1-\gamma} \\ &= E[J_1] - \tfrac{\alpha}{1-\gamma} \\ &= Q^*_M(s,a) - \tfrac{\alpha}{1-\gamma} \end{aligned}$$

We extend our argument over all u potential attempted updates using the union bound so that $\hat{Q}_t(s,a) \geq Q^*_M(s,a) - \frac{\alpha}{1-\gamma}$ holds over all timesteps $t \geq 0$, with probability at least $1 - u\frac{\delta}{u} = 1 - \delta$. ■

The next lemma bounds the number of timesteps that a state-action pair outside of the "nice" set κ can be experienced. This lemma is a modification of [Strehl et al., 2006, Lemma 4]. Our lemma decreases the number of times that a state-action pair that is not in κ can be experienced from $\frac{2mNK}{\epsilon_1(1-\gamma)}$ to $\frac{2m(NK-X)}{\epsilon_1(1-\gamma)}$, where X is the number of state-action pairs that are eliminated by initializing the action-value estimates with W. If X is large this represents a significant decrease in the number of experiences of state-action pairs outside of κ, which is the key factor in proving Theorem 2.

Lemma 8 *If events $\mathcal{X}_1$ and $\mathcal{X}_2$ occur, then $(s,a) \notin \kappa_t$ can be experienced on at most $\frac{2m(NK-X)}{\epsilon_1(1-\gamma)}$ timesteps t, where $X = \left|\left\{(s,a) \in S \times A \mid W(s,a) < V^*_M(s) - \alpha - \frac{\alpha}{1-\gamma}\right\}\right|$.*

Proof We start by applying Lemma 1. By its definition the Delayed Q-learning algorithm always follows a greedy policy (satisfying Condition 1 of Lemma 1) and cannot change an action-value estimate unless the corresponding state-action pair has been experienced (satisfying Condition 2 of Lemma 1). Since event $\mathcal{X}_2$ occurs, this satisfies Condition 3 of Lemma 1. Therefore by Lemma 1, under the above assumptions, Delayed Q-learning will only visit $NK - X$ state-action pairs.

The result follows from the proof of [Strehl et al., 2006, Lemma 4] over $NK - X$ state-action pairs rather than all NK state-action pairs. ■

7.3. Proof of Theorem 2

Now we are ready to prove Theorem 2.

Proof (of Theorem 2) We proceed by applying Theorem 4. We select $\epsilon_1 = \frac{\epsilon(1-\gamma)}{3}$ and m according to (13) so that event $\mathcal{X}_1$ holds with probability at least $1-\delta$ (by Lemma 6) and event $\mathcal{X}_2$ holds with probability at least $1-\delta$ (by Lemma 7).

Suppose that events $\mathcal{X}_1$ and $\mathcal{X}_2$ occur. Condition 1 ($\hat{V}_t(s) \geq V^*_M(s) - \beta$) of Theorem 4 is satisfied with $\beta = \frac{\alpha}{1-\gamma}$ (by Lemma 7). Now we argue that Condition 2 ($\hat{V}_t(s) - V^{\pi_t}_{M_{\kappa_t}}(s) \leq \epsilon$) of Theorem 4 is also satisfied. Notice that

$$\hat{V}_t(s) - V^{\pi_t}_{M_{\kappa_t}}(s) \leq \begin{cases} \gamma \sum_{s' \in S} T(s'|s,a) \left(\hat{V}_t(s') - V^{\pi_t}_{M_{\kappa_t}}(s') \right) + 3\epsilon_1 & \text{if } (s, \pi_t(s)) \in \kappa_t \\ 0 & \text{if } (s, \pi_t(s)) \notin \kappa_t \end{cases}$$

because $\hat{V}_t(s) - \left(R(s,a) + \gamma \sum_{s' \in S} \hat{V}_t(s') \right) \leq 3\epsilon_1$, whenever $(s, \pi_t(s)) \in \kappa_t$. This implies that $\hat{V}_t(s) - V^{\pi_t}_{M_{\kappa_t}}(s) \leq \frac{3\epsilon_1}{1-\gamma} = \frac{3(\epsilon(1-\gamma)/3)}{1-\gamma} = \epsilon$.

The number of timesteps that a state-action pair that is not in κ_t can be experienced is bounded in Lemma 8, which satisfies Condition 3 of Theorem 4. By our choice of m, events $\mathcal{X}_1$ and $\mathcal{X}_2$ occur with probability at least $1-2\delta$ (applying the union bound). Therefore the conditions of Theorem 4 are met with probability at least $1-2\delta$. By applying Theorem 4 and plugging in the values for the learning complexity, m, and ϵ_1, we obtain the fact that on all but

$$O\left(\frac{NK - X}{\epsilon^4 (1-\gamma)^8} \ln \frac{1}{\delta} \ln \frac{1}{\epsilon(1-\gamma)} \ln \frac{NK}{\delta \epsilon (1-\gamma)} \right)$$

timesteps, Delayed Q-learning follows an $\left(3\epsilon + \frac{\alpha}{1-\gamma}\right)$-optimal policy, with probability at least $1-3\delta$. We obtain our final result by setting $\epsilon \leftarrow \frac{\epsilon}{3}$ and $\delta \leftarrow \frac{\delta}{3}$, which modifies the bound by constant factors only. ■

7.4. Proof of Theorem 3

Proof (of Theorem 3) Let $s \in S_{\text{trg}}$ be a state in the target task. Because h is assumed to satisfy (5) and (6) there exists an action $\tilde{a} \in A_{\text{trg}}$ such that either (1) $(s, \tilde{a}) \in D$ and $V^*_{\text{trg}}(s) - \alpha \leq Q^*_{\text{trg}}(s, \tilde{a}) \leq Q^*_{\text{src}}(h(s, \tilde{a}))$ or (2) $(s, \tilde{a}) \notin D$ and $V^*_{\text{trg}}(s) - \alpha \leq Q^*_{\text{trg}}(s, \tilde{a})$.

In the first case ($(s, \tilde{a}) \in D$), Eq. (7) will assign the value

$$\begin{aligned} W(s, \tilde{a}) &= \min \left(\hat{Q}_{\text{src}}(h(s, \tilde{a})) + \epsilon_{\text{src}}, \tfrac{1}{1-\gamma} \right) \\ &\geq \hat{Q}_{\text{src}}(h(s, \tilde{a})) + \epsilon_{\text{src}} \\ &\geq (Q^*_{\text{src}}(h(s, \tilde{a})) - \epsilon_{\text{src}}) + \epsilon_{\text{src}} \\ &\geq Q^*_{\text{trg}}(s, \tilde{a}) \geq V^*_{\text{trg}}(s) - \alpha \end{aligned}$$

where the initial equality is due to the fact that $(s, \tilde{a}) \in D$ and values are transferred according to Eq. (7). The first step removes the minimum operation. The second step replaces the source task action-value estimates with the true source task action-values by subtracting ϵ_{src}. The final step is due to the fact that the intertask mapping satisfies (5). This inequality satisfies the α-weak admissible heuristic criteria for state s.

In the second case ($(s, \tilde{a}) \notin D$), $W(s, \tilde{a})$ is assigned the maximum value $\frac{1}{1-\gamma}$ which is greater than or equal to $Q^*_{\text{trg}}(s, \tilde{a})$ and $V^*_{\text{trg}}(s) - \alpha$. Therefore in both cases, the α-weak admissible heuristic criteria is satisfied at the state s. Since this argument holds for all states, then the transferred action-values induce an α-weak admissible heuristic with respect to the target task M_{trg}.

The remainder of proof follows from applying Theorem 2 to the target task with the transferred action-values taking the place of the α-weak admissible heuristic. ■

JMLR: Workshop and Conference Proceedings 24:77–88, 2012 10th European Workshop on Reinforcement Learning

Online Skill Discovery using Graph-based Clustering

Jan Hendrik Metzen JHM@INFORMATIK.UNI-BREMEN.DE
University of Bremen, Bremen, D-28359, Germany

Abstract

We introduce a new online skill discovery method for reinforcement learning in discrete domains. The method is based on the bottleneck principle and identifies skills using a bottom-up hierarchical clustering of the estimated transition graph. In contrast to prior clustering approaches, it can be used incrementally and thus several times during the learning process. Our empirical evaluation shows that "assuming dense local connectivity in the face of uncertainty" can prevent premature identification of skills. Furthermore, we show that the choice of the linkage criterion is crucial for dealing with non-random sampling policies and stochastic environments.

Keywords: Hierarchical Reinforcement Learning, Skill Discovery, Multi-task Learning

1. Introduction

Reinforcement Learning (RL) can provide autonomous agents with means to learn to improve their performance with experience. There exist theoretical guarantees for certain RL algorithms that the optimal policy for any discrete Markov Decision Process (MDP) is learned asymptotically. However, scaling RL to real-world problems with large or continuous state spaces remains a major challenge since the amount of experience the agent can collect is limited. One approach to alleviate this problem is Hierarchical RL [Barto and Mahadevan, 2003] which aims at dividing a problem into simpler subproblems, learning solutions for these subproblems, and encapsulate the acquired knowledge into so-called *skills* that can potentially be reused later on in the learning process. It has been shown that skills can help an agent to adapt to non-stationarity of the environment and to transfer knowledge between different but related tasks [Digney, 1998] and can increase the representability of the value function in continuous domains [Konidaris and Barto, 2009]. One of the major challenges in Hierarchical RL is to identify what might constitute a useful skill, i.e. how the problem should be decomposed. Skills should be reusable, distinct, and easy to learn. The task of identifying such skills is called *skill discovery* [Kirchner, 1995; Digney, 1996].

We propose the new online skill discovery method OGAHC which identifies densely connected regions in the state space using a constrained agglomerative hierarchical clustering of the estimated state transition graph. We show empirically in Section 4.1 that the proposed method is robust with regard to the agent's sampling policy and the stochasticity of the environment. In Section 4.2, we compare the proposed incremental, online skill discovery method to its non-incremental, offline counterpart. Furthermore, we show in Section 5 that using OGAHC for skill discovery in a Hierarchical RL agent outperforms related skill discovery approaches in a multi-task learning domain.

2. Background and Related Work

We adopt the *option framework* [Sutton et al., 1999], which formalizes skills as *options*. An option o consists of three components: the option's initiation set $I_o \subset S$ determining the states in which the option may be invoked, the option's termination condition $\beta_o : S \to [0, 1]$ specifying the probability of option execution terminating in a given state, and the option's policy π_o which defines the probability of executing an action in a state under option o. In the options framework, the agent's policy π may in any state s decide not solely to execute a primitive action but also to call any of the options for which $s \in I_o$. If an option is invoked, the option's policy π_o is followed until the option terminates according to β_o.

The option's policy π_o is defined relative to an option-specific reward function R_o that may differ from the global external reward function. Learning π_o given I_o, β_o, and R_o is denoted as *skill learning* and learning π given primitive actions and options is known as *compositional learning*. In this work we focus on *skill discovery*, i.e. choosing appropriate I_o, β_o, and R_o for a new option o. Autonomous skill discovery is very desirable since the quantities I_o, β_o, and R_o need not be predefined but can been identified by the agent itself and thus, skill discovery increase the agent's autonomy.

Most prior work on autonomous skill discovery is based on the concept of *bottleneck areas* in the state space. Informally, bottleneck areas have been described as the border states of densely connected areas in the state space [Menache et al., 2002] or as states that allow transitions to a different part of the environment [Şimşek and Barto, 2004]. A more formal definition is given by Şimşek & Barto [2009] which define bottleneck areas as those states which are local maxima of betweenness—a measure of centrality on graphs—on the transition graph. Once bottleneck areas have been identified, typically one (or several) skills are defined that try to reach this bottleneck from a local neighborhood of the bottleneck.

Since betweenness requires complete knowledge of the transition graph and is computationally expensive, several heuristics have been proposed to identify bottlenecks. One class of heuristics are *frequency-based approaches* that compute local statistics of states. Diverse density [McGovern and Barto, 2001] and relative novelty [Şimşek and Barto, 2004] have been considered in frequency-based heuristics. In diverse density, bottlenecks are identified as those states that are visited often on successful but not on unsuccessful trajectories. Relative novelty defines bottlenecks as those states that allow the agent to transition to an area in the state space that is otherwise difficult to reach from its current region. One disadvantage of frequency-based approaches is that they require repeated-sampling to obtain accurate estimates of the statistics and are thus not very sample-efficient.

An other class of heuristics that may be more sample-efficient are *graph-based approaches*. These heuristics are based on estimates of the transition graph and aim at partitioning this graph into subgraphs that are densely connected internally but only weakly connected with each other. Menache et al. [2002] propose a top-down approach for partitioning the global transition graph based on the max-flow/min-cut heuristic. Şimşek et al. [2005] follow a similar approach but partition local estimates of the global transition graph using a spectral clustering algorithm and use repeated sampling for identifying globally consistent bottlenecks. Due to the repeated sampling the approach shares some properties with the frequency-based approaches. Mannor et al. [2004] propose a bottom-up approach that partitions the global transition graph using agglomerative hierarchical clustering. The approach is offline, i.e. the

clustering can be executed only once and thus does not allow us to identify skills at different times during learning. This property limits the utility of the approach since there may be no single optimal point in time for skill discovery.

The main contribution of this work is a skill discovery method based on agglomerative clustering of the transition graph that shares some similarities with the work of Mannor et al. [2004]; however, the proposed method is online, i.e. can identify skills at any time.

3. Online Graph-based Agglomerative Hierarchical Clustering

We adopt the concept of identifying bottlenecks in the state space as basis for skill discovery. Informally, a set of states forms a bottleneck area if it lies on the boundary[1] of two densely connected areas of the state space that are only weakly connected mutually. We investigate approaches that identify such bottlenecks based on the sample transition graph using agglomerative clustering. We base skill discovery thus solely on the MDP's state transition probabilities $P^a_{ss'}$ and ignore the expected rewards $R^a_{ss'}$ since we consider $R^a_{ss'}$ to encode the task and aim at identifying task-independent skills that can be reused in different tasks.

3.1. Sample Transition Graph

For discrete environments, we construct the *sample transition graph* $G = (V, E, w)$, which is a directed, weighted graph that is used as estimate for the actual unknown transition graph, as follows: for a given set of n transitions $\{(s_i, a_i, s'_i)\}_{i=1...n}$ sampled from an MDP, we set $V = \{s_i\} \cup \{s'_i\}$ to the set of all states that have been visited and $E = \{(s_i, s'_i)\}$ to the set of all one-step transitions that have occurred. For each action $a \in A$, we add the attribute $N(s, a, s') = \sum_{i=1}^{n} \delta((s, a, s'), (s_i, a_i, s'_i))$ to any edge $(s, s') \in E$ and the attribute $N(s, a) = \sum_{i=1}^{n} \delta((s, a), (s_i, a_i))$ to any node $s \in V$ where $\delta(x, y) = 1$ if $x = y$ else 0.

Different choices for the edge weights $w(e)$ have been proposed. While Mannor et al. [2004] used essentially uniform edge weights $w_{uni}(e) = 1$, Şimşek et al. [2005] proposed the edge weights $w_{on}((s, s')) = \sum_a N(s, a, s')$, i.e. how often the transition $s \to s'$ has been observed under the sampling policy. While w_{uni} ignore both the stochasticity of the environment and the action preferences of the agent, w_{on} take both into account (i.e. it is "on-policy" with regard to the sampling policy). We argue that in order to identify properties of $P^a_{ss'}$ like bottlenecks, one should take the stochasticity into account but be independent of the sampling policy. Thus, we propose the edge weights $w_{off}((s, s')) = \sum_a N(s, a, s')/N(s, a)$, which are "off-policy" (note that $N(s, a, s')/N(s, a)$ is the maximum likelihood estimate of $P^a_{ss'}$ and thus independent of the sampling policy).

3.2. Linkage criteria

The purpose of the linkage criterion l is to give a quantitative judgment if the boundary of two connected, disjoint subgraphs $A, B \subset G$ forms a bottleneck in $G = (V, E, w)$. The larger the linkage value for the pair (A, B), the more evidence the criterion offers for a bottleneck between A and B. By choosing a threshold ψ, one can create a binary criterion for "bottleneckness" (by identifying a bottleneck between A and B if $l(A, B) > \psi$).

1. For a graph $G = (V, E)$, we define the boundary of two disjoint node sets $A, B \subset V$ as $b_G(A, B) = \{v \in V | \exists v^* \in V : (v, v^*) \in E \cap (A \times B \cup B \times A)\}$

Algorithm 1: Constrained agglomerative clustering

Input: graph $G = (V, E, w)$, constraint set C, linkage criterion l, threshold ψ
Initialize partition: $P = \{\{v\} | v \in V\}$
$C = C \cup$ **lambda** $p_1, p_2 : (p_1 \times p_2) \cap E \neq \emptyset$ # Merge only clusters p_i that are connected in G
while True **do**
  $M_c = \{(p_1, p_2) | (p_1, p_2) \in (P \times P) \wedge \bigwedge_{c \in C} c(p_1, p_2)\}$ # Merge-candidates that fulfill all constraints
  $p_1^*, p_2^* = \arg\min_{p_1, p_2 \in M_c} l(p_1, p_2)$ # Find merge candidates with minimal linkage
  if $l(p_1^*, p_2^*) > \psi$: **return** P # No more densely connected clusters → return clustering
  $P = (P \setminus \{p_1^*, p_2^*\}) \cup \{p_1^* \cup p_2^*\}$ # Merge p_1^* and p_2^*

As base for linkages, we define the (directed) *connectivity mass* c_m of subgraphs A and B as $c_m(A, B) = \sum_{e \in E \cap (A \times B)} w(e)$. The connectivity mass gets large for large, densely connected subgraphs with big edge weights. Mannor et al. [2004] proposed a linkage criterion for bottleneck identification in transition graphs that is defined as follows (using our notation and $|A|$ being the number of vertices in subgraph A): $M(A, B) = \frac{\min(|A|, |B|) \log(\max(|A|, |B|))}{c_m(A,B) + c_m(B,A)}$. The linkage is based on uniform edge weights such that the denominator is equal to the number of edges between A and B in G. The linkage thus assigns large values to subgraphs of similar sizes that are only weakly interconnected.

Şimşek et al. [2005] have proposed the $\hat{N}_{cut}$ criterion which is defined as follows (using our notation): $\hat{N}_{cut}(A, B) = \frac{c_m(A,B) + c_m(B,A)}{c_m(A,V) + c_m(B,A)} + \frac{c_m(B,A) + c_m(A,B)}{c_m(B,V) + c_m(A,B)}$. The $\hat{N}_{cut}$ criterion is an approximation of the sum of probabilities that the agent transitions in one time step from a state in subgraph A to a state in subgraph B or vice versa. The authors used a top-down spectral clustering algorithm to find graph cuts that minimize $\hat{N}_{cut}$; in order to use it as a linkage criterion where large values indicate a bottleneck, we shall use $-\hat{N}_{cut}$. The authors used the criterion with on-policy edge weights; we claim that it should rather be used with off-policy weights (see Section 4.1).

3.3. Constrained Agglomerative Clustering

In order to identify all bottlenecks of G, we determine a partition P^* of minimal cardinality of the graph nodes into disjoint clusters p_i such that neither of the subgraphs induced by the p_i contains a bottleneck. By construction, the boundary of any connected pair of these subgraphs forms a bottleneck (otherwise, the two subgraphs would have been merged because of the minimal cardinality objective). Formally: $P^* = \arg\min_{P \in \mathcal{P}(V)} |P| \quad \text{s.t.} \max_{p_i \in P, q_i \subset p_i} l(p_i \setminus q_i, q_i) \leq \psi$, with $\mathcal{P}(V)$ being the set of all possible partitions of V. Alternatively, we may also fix the number of clusters to k and identify a partition P^* with $|P^*| = k$ such that the maximal intra-cluster linkage l is minimized: $P^* = \arg\min_{|P|=k} \max_{p_i \in P, q_i \subset p_i} l(p_i \setminus q_i, q_i)$.

Since finding the optimal solutions for these problems is $\mathcal{NP}$-hard, we use an approximate approach similar to the one proposed by Mannor et al. [2004] which is based on agglomerative hierarchical clustering (see Algorithm 1). This algorithm starts by assigning each node into a

Algorithm 2: OGAHC

Input: linkage criterion l, parameters ρ, ψ, m
Initialize: partition $P = \emptyset$, graph $G = (\emptyset, \emptyset)$,
while True **do**
  $(s_1, a_1, s_2, \ldots, a_{m-1}, s_m) = \text{ACT}(\text{AGENT}, \text{ENV})$ # Act for $m-1$ steps and observe trajectory
  $\text{UPDATE}(G, (s_1, a_1, s_2, \ldots, a_{m-1}, s_m))$ # Update transition graph with trajectory
  $G' = \text{SMOOTH}(G, \rho)$ # Pseudo transitions for under-explored nodes
  $C = \emptyset$ # Constraints for keeping partitions consistent
  for $(p_A, p_B) \in P \times P$ with $p_A \neq p_B$ **do**
    # Must not merge two clusters with elements that had not been merged in last iteration
    $C = C \cup \{\textbf{lambda}\ p_1, p_2 : (p_1 \cup p_2) \cap p_A = \emptyset \vee (p_1 \cup p_2) \cap p_B = \emptyset\}$
  end for
  $P = \text{CAC}(G', C, l, \psi)$ # Partition G' using constrained agglomerative clustering (CAC)

separate cluster and afterwards merges greedily the pair of clusters that has minimal linkage among all pairs of clusters connected in G until no pair remains with a linkage below ψ. Additionally, the algorithm allows the specification of further constraints that determine which clusters may be merged (see Section 3.4).

3.4. Online Clustering

The clustering approach presented in Section 3.3 is an offline approach, i.e. it can be executed only once when "enough" transitions have been gathered to estimate the sample transition graph. The specific point in time when clustering is performed presents a trade-off: on the one hand, one would like to perform the clustering as early as possible in order to maximize the impact of the discovered skills during learning; on the other hand, performing the clustering too early may result in spurious clusters that are due to considerable deviations of the estimated transition graph from the true transition graph. It would thus be highly desirable to use an online clustering algorithm instead, which can be invoked several times during learning. In order to make such an online clustering algorithm useful for skill discovery, it has to fulfill the following properties: a) Subsequent executions of the clustering should be consistent, i.e. bottlenecks identified in one invocation of the clustering should persist in subsequent ones. Otherwise, skills corresponding to these bottlenecks would need to be deleted (causing an undesirable loss of learned knowledge) or to be modified to a new target, which would require re-learning and might have a detrimental effect onto higher-level policies which are based on these skills. b) The less reliable the estimate of the transition graph in a certain area of the state spaces is, the less likely the clustering should identify bottlenecks in this area.

To address these issues, we propose the *Online Graph-based Agglomerative Hierarchical Clustering* (OGAHC) method (see Algorithm 2). Its main contributions are the following: a) In order to guarantee consistency of subsequent clusterings, an increasing set of constraints that must be obeyed by the clustering is maintained. These constraints ensure that no two nodes that have been assigned to different clusters are assigned to the same cluster in

subsequent clusterings. b) In order to prevent premature identification of bottleneck areas, OGAHC builds on a heuristic that may be framed as "assume dense local connectivity in the face of uncertainty". Technically, instead of working on the maximum likelihood estimate G of the transition graph, the clustering is performed on a modified transition graph $G' = \text{SMOOTH}(G, \rho)$. In G', for each $N(s,a) < \rho$, $N(s,a,s')$ is incremented by $(\rho - N(s,a))/k$ for any s' that is among the $k = \max(5, \rho)$ nearest neighbors. The nearest neighbors can be computed based on any measure of state similarity; if no such measure exists, one could alternatively use the k graph nodes with minimal distance to s in G. ρ is a free parameter of the algorithm that determines if the algorithm is more liberal, i.e. tends to identify bottlenecks early in the learning process despite the uncertainty in the sample transition graph, or—for larger values of ρ— more conservative. Note that sparsity properties of G—which correspond to a reduced runtime of the constrained agglomerative clustering algorithm because of less merge candidates—are maintained in G'.

The heuristic "assume dense local connectivity in the face of uncertainty" prevents bottlenecks from being identified prematurely since the linkage value is typically decreased by adding pseudo transitions and thus, fewer clusters are formed in early invocations of the algorithm. Over time, fewer pseudo transitions are added, the connectivity decreases, the typical linkage of subgraphs increases, and thus, more clusters are formed. Note that specifying the number of clusters k beforehand instead of ψ is not practical in online clustering since it does not allow the implicit increase of granularity of the clustering over time.

3.5. Skill Prototype Generation

Given the partitioning P obtained using OGAHC, one skill is generated for reaching each identified bottleneck area. We set the bottleneck area of two clusters A and B to the boundary of the corresponding subgraphs of G. The skill prototype $(I_{AB}, \beta_{AB}, R_{AB})$ that is generated for the bottleneck between A and B is then defined as follows:

$$I_{AB} = (A^* \cup B^*) \setminus b_G(A,B) \qquad \beta_{AB}(s) = 0 \text{ if } s \in I_{AB} \text{ else } 1$$
$$R_{AB}((s,a,r,s')) = r \text{ if } s' \in (A^* \cup B^*) \text{ else } r_p + r,$$

where $A^* = \{v \in V | \exists v' \in A : (v,v') \in E\}$, and r_p is a parameter of the algorithm that determines the penalty for failing to fulfill a skill's objective. Further skill prototypes are generated for reaching identified terminal states s_t from the adjacent clusters A. Note that the skill prototypes may change over time when A or B change because of re-clustering; however, a skill and its corresponding bottleneck area can never disappear.

4. Cluster Accordance Analysis

This section presents an empirical evaluation of the quality of the partitions generated by different clustering approaches and linkage criteria in 50 randomly generated MDPs. Each of the MDPs consists of 400 states and in each state, 4 actions are available. The stochasticity of the MDPs is controlled by the parameter $\chi \in [0,1]$, where larger values of χ correspond to increased stochasticity. Along with the MDPs, ground-truth partitions of the states consisting of 7 clusters have been generated. For more details, we refer to the supplementary material. For each MDP, an optimal policy has been computed offline and

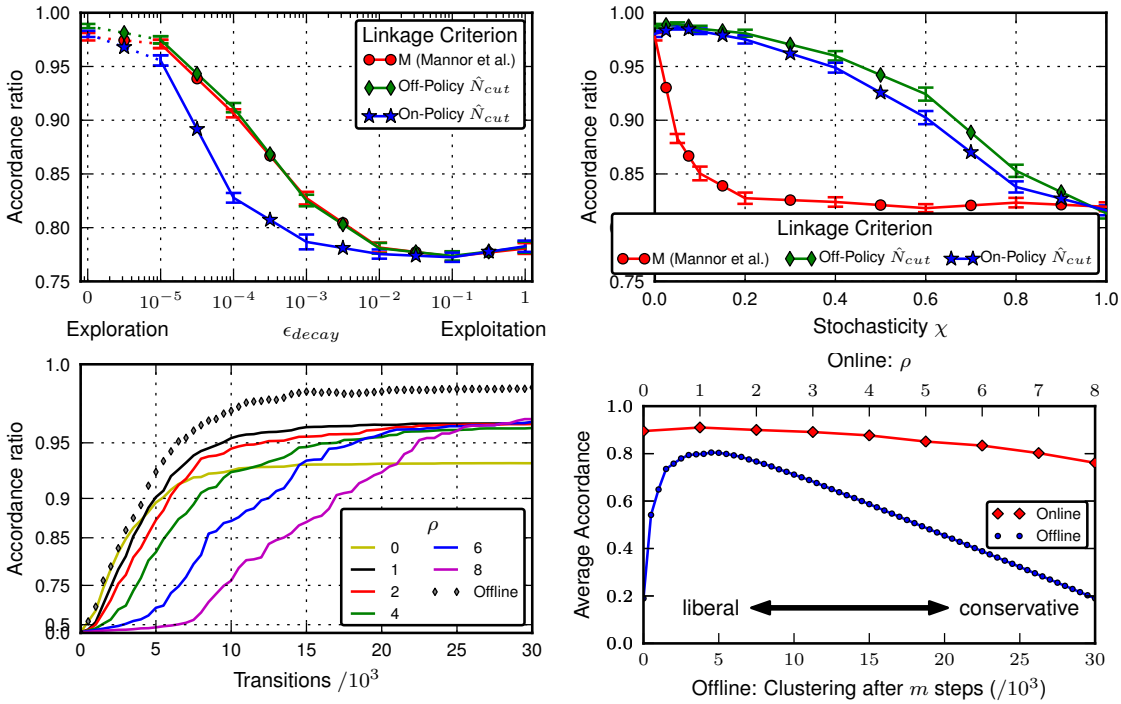


Figure 4.1: Upper row: Comparison of different linkage criteria for different degrees of exploration and different stochasticity of the environment. Lower row: Comparison of OGAHC (for different values of ρ) and offline clustering. Errorbars show the standard error of the mean.

a trajectory consisting of n transitions has been sampled by following the optimal policy ϵ-greedily. ϵ was set initially to 1 and then decayed after each step by the factor $1 - \epsilon_{decay}$. Based on the sampled transitions, a partition of the states is determined and compared to the ground truth partition. The agreement of the two partitions P_1 and P_2 is measured using the accordance ratio $acc(P_1, P_2) = \frac{1}{|S|^2} \sum_{s,\bar{s} \in S} \delta(\delta(P_1(s), P_1(\bar{s})), \delta(P_2(s), P_2(\bar{s})))$, i.e. the ratio of element-pairs on which P_1 and P_2 agree on assigning them to the same or to different clusters ($\delta(x, y) = 1$ if $x = y$ else 0). Statistical hypothesis testing has been conducted using Student's independent two-samples t-test.

4.1. Linkage criterion

In a first experiment, we compare different linkage criteria in an offline clustering setting for $\chi = 0$. To analyze the effect of non-uniform exploration of the agent, we have varied ϵ_{decay}. Large values of ϵ_{decay} correspond to an agent which starts to exploit more early and as a consequence, obtains a more biased estimate of the transition graph. The upper left plot in Figure 4.1 shows a comparison of the partitions obtained after $n = 25000$ transitions for the M-linkage and the $\hat{N}_{cut}$-linkage with both on-policy and off-policy edge weights as

discussed in Section 3.2. The main result shown in the figure is that the on-policy $\hat{N}_{cut}$ linkage obtains significantly worse partitions than the other two linkages for intermediate values of ϵ_{decay} ($p < 0.008$ for $\epsilon_{decay} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$). This can be attributed to the fact that the on-policy $\hat{N}_{cut}$ linkage bases its partitioning not solely on the environment's transition probabilities but also on the agent's action selection which is undesirable if the agent selects action non-uniformly. In a second experiment ($n = 25000$, $\epsilon_{decay} = 0$), we have varied the stochasticity χ of the environment. The upper right plot in Figure 4.1 shows that the M-linkage obtains significantly worse results than the off-policy $\hat{N}_{cut}$ linkage if the environment gets slightly non-deterministic ($p < 0.0001$ for $0.05 \leq \chi \leq 0.8$). This is due to the use of uniform edge weights in the M-linkage; these weights do not allow to differentiate between probable and less probable transitions, which is apparently important for skill discovery in non-deterministic environments. In summary, the results show that the off-policy $\hat{N}_{cut}$-linkage is the most robust linkage criterion; the following experiments have been conducted with this linkage accordingly.

4.2. Online Graph-Clustering

In this subsection, we compare the OGAHC algorithm with its offline counterpart (essentially the approach proposed by Mannor et al. [2004]). We have used the same 50 MDPs as before and set $n = 30000$, $\epsilon_{decay} = 0$, $\chi = 0$, and $\psi = -0.075$. For OGAHC, the clustering has been updated every 500 steps and different choices of ρ have been evaluated. The offline clustering has been performed after $m \in \{500, 1000, \ldots, 30000\}$ transitions, taking all m transitions that have been acquired so far into account at once.

The lower left plot in Figure 4.1 shows how the accordance ratio of the identified partitions changes over time. It can be seen that smaller values of ρ achieve higher accordance ratios in the early phase since they tend to identify clusters more early. However, these clusters are potentially suboptimal since they are based on a sample transition graph that was estimated based on a small number of transitions and has thus a high sampling error. Accordingly, it can be seen that for e.g. $\rho = 0$ the accordance ratio plateaus on a lower level than for larger values of ρ. Comparing OGAHC to offline clustering, one can see that for any number of transitions, the online clustering tends to be slightly worse than the offline clustering obtained at that point in time. This can either be due to the enforced consistency with prior clusterings or to the influence of ρ.

However, the clusterings of OGAHC can be refined over time while the clusterings of the offline approach are fixed. Thus, the performance of the two method should not only be compared at a single point in time. The lower right plot of Figure 4.1 shows the accordance ratio averaged over the 30000 steps for different values of ρ and m. For both small and large values of m, the average accordance of the offline clustering is low. For small m, the average accordance is low because the clustering is based on only few transitions and cannot be improved later on; for large values of m, the average accordance is low because there is no clustering at all for a long initial period. Intermediate values ($m \approx 4500$) obtain a higher average accordance of approx. 0.81. In contrast, the online clustering depends less on the specific choice of ρ. The optimal value for ρ is 1, which results in an average accordance of approx. 0.91; however, for any value $0 \leq \rho \leq 6$, OGAHC achieves a significantly higher average accordance than the offline clustering for $m = 4500$ ($p < 0.019$). Thus, even without

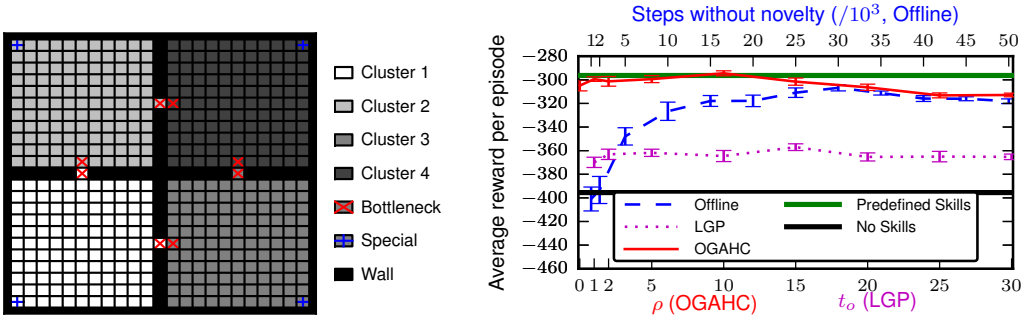


Figure 5.1: Left: Structure, ground-truth partitioning, and bottlenecks of the multi-task maze world. Right: Reward per episode (averaged over the first 1000 episodes) for different skill discovery strategies. ρ and t_o are varied over the same value range (bottom x-axis). See supplementary material for detailed learning curves.

fine-tuning ρ, the online clustering can outperform the offline clustering with optimally chosen m with regard to the average clustering quality. In the next section, we will investigate if the same holds true for the learning performance of a Hierarchical RL agent using OGAHC for skill discovery compared to agents using other approaches.

5. Multi-task Learning

In this section, we evaluate the utility of the OGAHC method within a Hierarchical RL agent in a multi-task learning scenario. The scenario is a 23×23 maze world consisting of four rooms with four "special" states (see Figure 5.1). In each time step, the agent obtains a reward of -1. The agent has to learn to solve 12 different tasks in this environment; each task is defined by a pair (s_0, s_t) of special states where s_0 is the start state and s_t is the goal state of the task. In each episode, a task is chosen at random and the specific task is communicated to the agent as a state space dimension. This dimension is used by the agent for policy learning but ignored during skill discovery and skill learning. The acquired skills can thus be transferred between tasks and may give a useful exploration bias for the agent.

The agent uses a 2-level hierarchy: The lower level consists of the acquired skills and a special option discussed below. On the upper level, the agent may choose among these skills but not choose a primitive action directly, i.e. the action space is abstracted not augmented. 1-step intra-option Q-Learning [Precup, 2000] is used for skill learning; no experience replay is conducted to avoid intermixing the contributions of skill discovery and experience replay (see Jong et al. [2008] for a discussion). A special low-level option prototype is provided to the agent: this option can be invoked in any state, terminates in any state with probability $\beta = 0.01$, and uses the external reward signal. This option allows the agent thus to learn a monolithic global policy. The reasons for this option are twofold: on the one hand, it guarantees that the agent can learn a global optimal policy eventually despite the abstraction of the action space. On the other hand, it makes the agent less susceptible to the broken

exploration symmetry that is caused by temporal abstraction (see [Jong et al., 2008]) than pure augmentation of the action space.

We compare OGAHC to the Offline Clustering approach proposed by Mannor et al. [2004] and the Local Graph Partitioning (LGP) approach by Şimşek et al. [2005]. As baseline, we evaluate the agent using no skill discovery (i.e. learning a monolithic policy using the "special" option), and the agent that is provided with the ground-truth partitioning (see Figure 5.1) from the beginning. $|\psi|$ has been set to 0.075 for all approaches. For LGP, the required hit-ratio has been set to $t_p = 0.2$, the option lag to $l_o = 20$, the window length to $h = 1000$, and the update frequency to 250. These parameters have been chosen based on preliminary experiments. The discount factor of the agent has been set to $\gamma = 0.99$, the learning rate to $\alpha = 1.0$, the value functions have been initialized optimistically to 0.0, and the penalty for failing to fulfill a skill's objective has been set to $r_p = -100$. For OGAHC, the parameter ρ has been varied between 0 and 30, for LGP the minimum number of state observations t_o has been varied between 1 and 30, and the Offline Clustering has been conducted after m steps in which no novel state has been encountered (where m has been varied between 5000 and 50000). Each setting has been evaluated 15 times for 1000 episodes.

The results of the experiment are depicted in Figure 5.1. The agent provided with predefined skill prototypes obtains an average reward per episode of $\bar{r}_e = -296.3 \pm 1.6$ and the agent using no skills of $\bar{r}_e = -395.5 \pm 0.9$. The maximal gain of average reward that can be achieved by biasing exploration using skills is thus approx. +100. Using LGP for skill discovery does not achieve an average reward of more than $\bar{r}_e = -350$ for any choice of t_o. Closer inspection showed that for small values of t_o, the resulting partitioning was far from being optimal while for larger values of t_o, the skills have been introduced too late to provide a useful exploration bias. Using the Offline Clustering approach, the optimal time for performing the clustering is after $m^* = 30000$ steps without observing any novel state which results in $\bar{r}_e = -306.8 \pm 2.6$. Thus, performing Offline Clustering at the right time does already realize 90% of the possible reward gain. However, for the same reasons as in Section 4.2 the choice of m is crucial: performing clustering too early results in suboptimal partitions while performing it too late limits the benefits of the acquired skills for learning.

The OGAHC approach achieves $\bar{r}_e > -305$ for any choice of $\rho \leq 15$ and $\bar{r}_e = -294.5 \pm 2.1$ for $\rho = 10$. The approach allows thus to acquire more than 90% of the possible reward gain (and thus more than obtained by Offline Clustering for any choice of m) for a broad range of values for ρ, making the specific choice of ρ less important. For the optimal choice $\rho^* = 10$, OGAHC achieves an average reward that is on par with what can be obtained with the predefined skill prototypes. This can be explained by the observation that for this choice of ρ, OGAHC discovers skills that are close to the optimal ones very early during learning (typically during the first 3 episodes). Since skill learning takes several episodes, the predefined skill prototypes offer the agent an efficient exploration bias starting after approx. 5 episodes; thus OGAHC can be on a par with the predefined skill prototypes. In summary, the results show that using OGAHC for skill discovery allows to identify reusable skills at an early stage of learning without requiring to fine-tune the parameter ρ.

6. Conclusion and Future Work

We have presented the novel online skill discovery method OGAHC which is based on identifying bottleneck areas in the state transition graph by means of an agglomerative hierarchical clustering. Our empirical studies suggest that the approach can reliably identify useful skills in a very early stage of learning and allows the agent thus to explore the environment more efficiently than with other skill discovery methods or without utilizing skills at all. Future work is to extend the approach to MDPs with large and continuous state spaces and to evaluate it on real-world problems.

References

Supplementary material. URL http://www.informatik.uni-bremen.de/~jhm/files/ewrl_2012_ogahc_supplement.pdf.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, October 2003.

B. L. Digney. Emergent hierarchical control structures: Learning Reactive/Hierarchical relationships in reinforcement environments. In *From Animals to Animats: The 4th Conference on Simulation of Adaptive Behavior.*, page 363–372, Cambridge, MA, 1996. MIT Press.

B. L. Digney. Learning hierarchical control structures for multiple tasks and changing environments. In *Proceedings of the 5th Conference on the Simulation of Adaptive Behavior*, pages 321–330. MIT Press, 1998.

N. K. Jong, T. Hester, and P. Stone. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 299–306, 2008.

F. Kirchner. Automatic decomposition of reinforcement learning tasks. In *Proceedings of the AAAI 95 Fall Symposium Series on Active Learning*, pages 56–59, Cambridge, MA, USA, 1995. AAAI Press.

G. Konidaris and A. G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, volume 22, pages 1015–1023, 2009.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 560–567, 2004.

A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the 18th International Conference on Machine Learning*, pages 361–368, 2001.

I. Menache, S. Mannor, and N. Shimkin. Q-Cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, pages 295–306, 2002.

D. Precup. *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 2000.

Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, pages 751–758, 2004.

Ö. Şimşek and A. G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems (NIPS)*, volume 22, pages 1497–1504, 2009.

Ö. Şimşek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 816–823. ACM, 2005.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

JMLR: Workshop and Conference Proceedings 24:89–101, 2012 10th European Workshop on Reinforcement Learning

An Empirical Analysis of Off-policy Learning in Discrete MDPs

Cosmin Păduraru COSMIN@CS.MCGILL.CA
Doina Precup DPRECUP@CS.MCGILL.CA
Joelle Pineau JPINEAU@CS.MCGILL.CA
Gheorghe Comănici GCOMAN@CS.MCGILL.CA
McGill University, School of Computer Science, Montreal, Canada

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

Off-policy evaluation is the problem of evaluating a decision-making policy using data collected under a different behaviour policy. While several methods are available for addressing off-policy evaluation, little work has been done on identifying the best methods. In this paper, we conduct an in-depth comparative study of several off-policy evaluation methods in non-bandit, finite-horizon MDPs, using randomly generated MDPs, as well as a Mallard population dynamics model [Anderson, 1975] . We find that un-normalized importance sampling can exhibit prohibitively large variance in problems involving look-ahead longer than a few time steps, and that dynamic programming methods perform better than Monte-Carlo style methods.

1. Introduction

One of the core competencies of most intelligent decision-making agents is the ability to properly evaluate their decision-making strategy. In a reinforcement learning context, this is the policy evaluation problem, which involves the estimation of the expected return associated with a policy. The ideal method for evaluating policies is to apply them in practice, observe the return, and estimate the expected return (and its uncertainty) using this data. However, if data is expensive (or rare), or the number of policies to evaluate is large, this may be infeasible. A popular alternative is to evaluate the decision-making policy of interest (called the *target policy*) using data collected under a different, *behaviour policy*. This method is known as *off-policy policy evaluation*. Off-policy learning has been used in a range of applications, such as energy systems [Hannah and Dunson, 2011], robotics [Riedmiller, 2005], clinical studies [Pineau et al., 2009], and tax collection [Abe et al., 2010].

We focus on two types of off-policy estimators: model-based [Sutton and Barto, 1998; Mannor et al., 2007], and importance sampling weighting of the returns [Precup et al., 2000; Robins et al., 2000; Murphy, 2005]. Several methods for continuous MDPs that would fall in neither category, such as LSTD, can be shown to reduce to a model-based estimator in the discrete setting [Boyan, 2002]. The tree backup algorithm proposed in Precup et al. [2000] also becomes, in expectation, equivalent to a model-based estimator. One class of algorithms which cannot be easily pegged into one of these categories is gradient-based temporal-difference methods [Sutton et al., 2009]. However, since these algorithms are de-

signed specifically to handle infinite-horizon problems in which function approximation is necessary, we will not discuss them in this paper. Three of the estimators we compare have been proposed in the literature. The other two (per-step importance sampling and normalized per-step importance sampling) are related to existing methods, but have not been studied in the form we propose.

Previous comparative studies for off-policy estimators considered single-step contextual bandit problems [Kang et al., 2007; Dudik et al., 2011]. We present an empirical study for finite-horizon discrete MDPs with arbitrary horizon length. We use a set of randomly generated MDPs, similar to those used by Castronovo and Ernst [2012], and a simulated model of the Mallard population dynamics, first proposed by Anderson [1975] and subsequently used by Fonnesbeck [2005].

2. Finite-horizon MDPs

A finite-horizon MDP is defined as a tuple $\langle S, A, P, R\rangle$, where S is a set of states; A is a set of actions; $P : S \times A \times S \to [0, 1]$ is the transition model, with $P_{sa}^{s'}$ denoting the conditional probability of a transition to state s' given current state s and action a; $R : S \times A \to [0, 1]$ is the reward function, with R_{sa} denoting the immediate expected reward for state s and action a. A policy $\pi : S \times A \to [0, 1]$ specifies how decisions are made. In a finite MDP, the model can be represented using matrices $P \in \mathbb{R}^{|S\times A|\times|S|}$ and $R \in \mathbb{R}^{|S\times A|}$. Similarly, policies can also be represented as block-diagonal matrices $\pi \in \mathbb{R}^{|S|\times|S\times A|}$.

The value of a policy π for a decision horizon of length K is defined as:

$$V_K^\pi(s) = E[r_0 + r_1 + \cdots + r_K | s_0 = s] = \sum_a \pi_{sa} \left(R_{sa} + \sum_{s'} P_{sa}^{s'} V_{K-1}^\pi(s') \right).$$

or, if we consider V_K^π to be the vector of all state values,

$$V_K^\pi = \pi(R + PV_{K-1}^\pi) = \cdots = \sum_{k=0}^{K} (\pi P)^k \pi R. \tag{1}$$

3. Off-policy value function estimation

In practice, the model of the MDP is usually unknown, so Eq. (1) cannot be applied directly. Furthermore, the user might not be able to obtain trajectories using policy π. In natural resource management, in particular, implementing a policy is not only expensive but also potentially unrealistic, given that interesting time horizons may span decades. Instead, the user may have access to data gathered under some existing policy b (or possibly, under several known policies from different geographic regions or different periods). Off-policy value function estimation methods are designed to deal with this case.

All estimators used in this paper are summarized in Table 1. We will now describe the notation and context for each of them.

Importance sampling [Rubinstein, 1981] is a technique for sampling from one distribution by weighting the samples generated from another distribution. It has been proposed as an off-policy estimator for MDPs both in reinforcement learning [Precup et al., 2000] and in the

	Un-normalized	Normalized
Per-trajectory importance sampling	$\hat{V}_K^{PTIS}(s) = \frac{1}{n_s}\sum_{i=1}^{n_s}\left(\prod_{j=1}^{K}\frac{\pi(s_{i:j},a_{i:j})}{b(s_{i:j},a_{i:j})}\right)\sum_{l=1}^{K} r_{i:l}$	Instead of n_s, divide by $\sum_{i=1}^{n_s}\left(\prod_{j=1}^{K}\frac{\pi(s_{i:j},a_{i:j})}{b(s_{i:j},a_{i:j})}\right)$
Per-step importance sampling	$\hat{V}_k^{PSIS}(s) = \frac{1}{n_s}\sum_a \frac{\pi_{sa}}{b_{sa}}\sum_{i\in B(s,a)}[r_i+\hat{V}_{k-1}^{PSIS}(s'_i)]$	Instead of n_s, divide by $\sum_{i\in B(s)}\frac{\pi(s,a_i)}{b(s,a_i)} = \sum_a n_{sa}\frac{\pi_{sa}}{b_{sa}}$
Model-based	$\hat{V}_K^{MB}(s) = \sum_a \pi_{sa}\left(\hat{R}_{sa} + \sum_{s'}\hat{P}_{sa}^{s'}\hat{V}_{K-1}^{MB}(s')\right) = \sum_{k=0}^{K}(\pi\hat{P})^k\pi\hat{R}$	

Table 1: Off-policy estimators for discrete MDPs. **All methods are consistent, but (un-normalized) per-trajectory importance sampling is the only one that is unbiased.**

clinical trial literature [Robins et al., 2000], where it is called inverse probability weighting. Importance sampling methods typically assume that the behaviour policy used to collect the data, denoted b, is known, and that $\pi_{sa} > 0 \implies b_{sa} > 0$. The naive implementation of importance sampling for off-policy evaluation weighs entire trajectories. This existing estimator is the (un-normalized) *per-trajectory importance sampling* (PTIS) in Table 1. It is computed based on n_s trajectory fragments of length K that start from state s in the batch, where the i^{th} trajectory is denoted by:

$$(s_{i:0} = s, a_{i:0}, r_{i:0}, s_{i:1}, a_{i:1}, r_{i:1}, ..., s_{i:K}, a_{i:K}, r_{i:K}),$$

The weights in the importance sampling estimator can be scaled to the $[0, 1]$ interval by normalizing over their sum. This is seen as a way to reduce estimator variance, and leads to the *normalized per-trajectory importance sampling estimator* in Table 1, which we will denote by $\hat{V}^{PTIS-N}$. This is also an existing estimator [Precup et al., 2000; Murphy, 2005].

In order to avoid the variance introduced by weighting entire trajectories, we introduce *per-step importance sampling* as an alternative. These estimators are very similar to the per-decision importance sampling [Precup et al., 2000]. However, we present them for state values and finite-horizon problems with batch data, rather than for action values and episodic problems with on-line data. Consider the more general setting where a sample i is composed of start state s_i, action a_i generated from b at s_i, and (r_i, s'_i) the response of the model (P, R) at (s_i, a_i). The (un-normalized) *per-step importance sampling estimator*, shown in Table 1, uses n_s, n_{sa}, and $n_{sas'}$ to denote the sizes of the subsets restricted by the start state s, action a and/or next state s', and $B(s)$ and $B(s, a)$ to denote the subsets of samples for which the

start state and/or action choice is s, a. If $n_s = 0$, there is no data at this state, so we have to pre-define $\hat{V}_k^{PSIS}(s)$. We also construct a normalized version, which we denote by $\hat{V}_K^{PSIS-N}$.

Model-based MDP estimators construct approximations $\hat{P}$ and $\hat{R}$ of the transition and reward model, and then use standard methods such as dynamic programming to compute the value function for the estimated model. For discrete MDPs, consistent estimators of the model are given by:

$$\hat{R}_{sa} = \frac{1}{n_{sa}} \sum_{i \in B(s,a)} r_i, \qquad \hat{P}_{sa}^{s'} = \frac{n_{sas'}}{n_{sa}}. \tag{2}$$

Similarly to per-step importance sampling, we have to use a pre-defined value if $n_s = 0$ or $n_{sa} = 0$. The finite-horizon value function can then be estimated using the approximate model $\hat{R}, \hat{P}$.

This estimator is intuitive and has a long history. However, we are only aware of one work on its statistical properties, by Mannor et al. [2007]. For infinite-horizon discrete MDPs with discounting, Mannor et al. [2007] compute second-order approximations for the bias and variance of the model-based estimator, and examine its empirical performance on a discretized version of a catalog ordering problem.

Note that the MB estimator can be expressed in the same form as the per-step importance sampling estimators, but with an estimate of b as surrogate:

$$\hat{V}_k^{MB}(s) = \frac{1}{n_s} \sum_a \frac{\pi_{sa}}{n_{sa}/n_s} \sum_{i \in B(s,a)} \left(r_i + \hat{V}_{k-1}^{MB}(s_i') \right). \tag{3}$$

Results from [Rubinstein, 1981] can be used to show that both PTIS and PTIS-N are consistent estimators. PTIS is also unbiased; however, normalization can introduce bias, while typically reducing variance. The dynamic programming estimators (PSIS, PSIS-N and MB) are also consistent. This can be proven using Slutsky's theorem [Dudewicz and Mishra, 1988], by noting that all of them can be written in the form

$$\hat{V}_K = \sum_{k=0}^{K} (\pi Z \hat{P})^k \pi Z \hat{R}, \tag{4}$$

where Z is a diagonal matrix with entries

$$Z_{sa}^{PSIS} = (n_{sa}/n_s)/b_{sa} \qquad Z_{sa}^{PSIS-N} = (n_{sa}/W(s))/b_{sa} \qquad Z_{sa}^{MB} = 1,$$

with $W(s)$ denoting the normalization term for PSIS-N.

4. Empirical results

In this section we study the empirical performance of the different off-policy estimators on two domains: a simulated model of a natural resource management problem, and a set of randomly generated MDPs. Note that, for the model-based method, we use a default reward value $R_{sa} = 0$ for the state-action pairs (s, a) for which $n_{sa} = 0$ (unless otherwise specified), and a default transition model that self-loops ($P_{sa}^s = 1$).

4.1. Mallard population model

Anderson's model is formulated as an MDP with yearly time increments, two-dimensional state, and continuous actions [Anderson, 1975]. The state variables are the adult population N_t and the number of ponds P_t (both expressed in millions), while the action H_t represents the proportion of animals to be harvested in year t. The state transitions are defined by the following equations:

$$N_{t+1} = N_t(1 - 0.37e^{2.78H_t}) + \left(\frac{1}{12.48}P_t^{0.851} + \frac{0.519}{N_t}\right)^{-1}(1 - 0.49e^{0.9H_t})$$
$$P_{t+1} = -2.76 + 0.391P_t + 0.233\epsilon_t$$

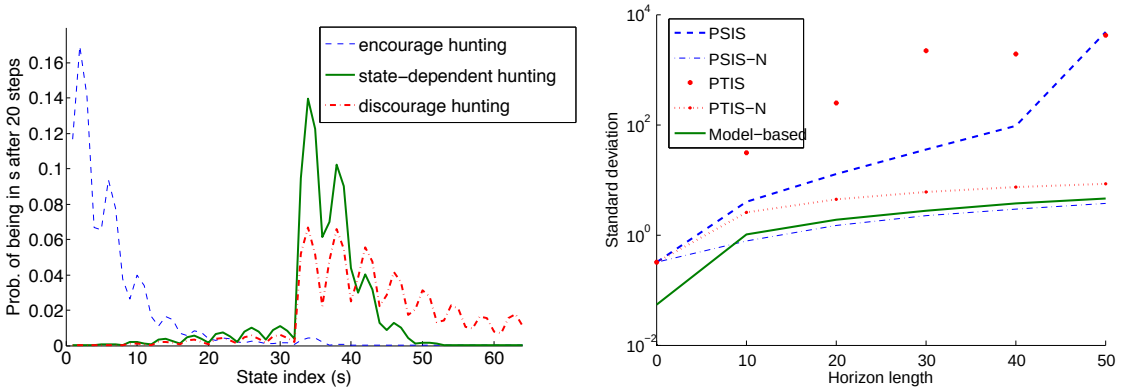
where $\epsilon_t \sim N(16.46, 4.41)$ is a normally distributed random variable describing the amount of precipitation during year t (in inches). The reward is defined as the number of birds harvested in a given year, computed as

$$R(N_t, P_t, H_t) = H_t\left(0.92N_t + \left(\frac{1}{12.48}P_t^{0.851} + \frac{0.519}{N_t}\right)^{-1}\right).$$

Anderson constructed and validated this model based on real data about the evolution of the Mallard population. For more details, including model justification, we refer the reader to [Anderson, 1975].

For our experiments, we used a discretized version of the model. Since the states where the bird population is close to 0 are particularly important, we used a discretization with higher resolution in that region of the state space. More precisely, we divided N_t into intervals of length 2 when $N_t > 2$, and length 0.25 when $N_t \leq 2$. P_t was divided into four intervals of unit length. We also assumed that state features are bounded, so $N_t \in [0, 17]$ and $P_t \in [0, 4]$. This resulted in 64 states. We generated 10 million transitions from the original MDP by sampling starting states uniformly randomly; then, we used the data to estimate a transition matrix and reward function for the discrete MDP. The transition function was estimated using maximum likelihood estimation, whereas the reward function was defined as a Gaussian for each discretized interval, with its mean and variance estimated from the generated data. This produced the MDP that we used as "ground truth"; that is, we investigated how well our methods estimate the value function for this discretized MDP.

We considered three policies, all selecting from two actions: a_1 representing $H_t = 0$ and a_2 representing $H_t = 0.3$. The first policy, which we call *discourage hunting*, selects a_1 with probability 0.8 and a_2 with probability 0.2 in every state. The second policy, which we call *state-dependent hunting*, prescribes reduced hunting when the mallard population or the number of ponds is low, and larger amounts of hunting otherwise; more precisely, it selects a_1 with probability 0.8 in the discrete states corresponding to $[0, 12] \times [0, 1]$, $[0, 8] \times [1, 2]$, $[0, 4] \times [2, 3]$, and $[0, 2] \times [3, 4]$, and a_2 with probability 0.8 for the rest of the state space. The third policy, called *encourage hunting*, selects a_2 with probability 0.8 in all states. Throughout the experiments, we used *discourage hunting* as the behaviour policy, and used the discrete state corresponding to $N_t = 7$ and $P_t = 1.5$ as the starting state s_0. Each batch of training data was generated as a single, uninterrupted trajectory starting in s_0, with actions selected according to the behaviour policy. Hence, the number of samples in a



(*a*) Probabilities of being in different states after 20 steps, under the three different policies

(*b*) Standard deviations of the three estimators based on a 500-sample trajectory with *state-dependent hunting* as the target.

Figure 4.1: State probabilities (left) and standard deviations of three estimators (right). Results averaged over 100, 000 runs. Note the logarithmic scale for the y axis in the right panel.

batch is the length of this trajectory. We present results estimating the value of the start state s_0 under all policies. Unless otherwise specified, the results are averages over 1000 batches.

As seen in Figure 1(a), the three policies tend to visit different regions of the state space. In particular, the distribution of states under *encourage hunting* leads predominantly to states corresponding to low population numbers, which is very different from the other two policies. Intuitively, this discrepancy should make estimating the value function for *encourage hunting* particularly challenging, given that *discourage hunting* is used as the behaviour policy. This intuition is confirmed by our empirical results.

Figure 4.1(*b*) contains a plot of the standard deviations of the different estimators when *state-dependent hunting* is the target policy. Even for a target policy that induces a state distribution fairly close to the one under the behaviour policy, PSIS and PTIS can have very large variance for horizons longer than a few time steps.

For the remainder of this section, we further investigate the performance of PSIS-N, PTIS-N, and MB. We examine the performance of these three estimators in terms of bias and root mean squared error (RMSE), when either *state-dependent hunting* or *encourage hunting* is the target policy.

For a particular horizon, we can examine the methods' performance as a function of the amount of data available, as illustrated in Figures 4.2 and 4.3. As expected, the performance of both methods improves when increasing the size of the batch, although much slower if the target policy is very different from the behaviour policy.

PTIS-N exhibits the poorest performance in all settings. The performance difference is most striking when *encourage hunting* is the target policy. *Encourage hunting* has the reversed action selection probabilities from the behaviour policy, and it induces a completely

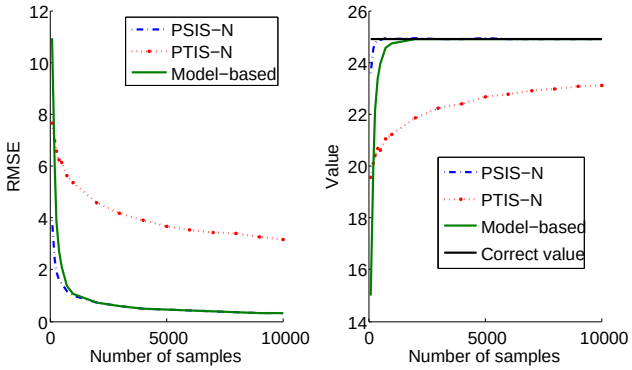
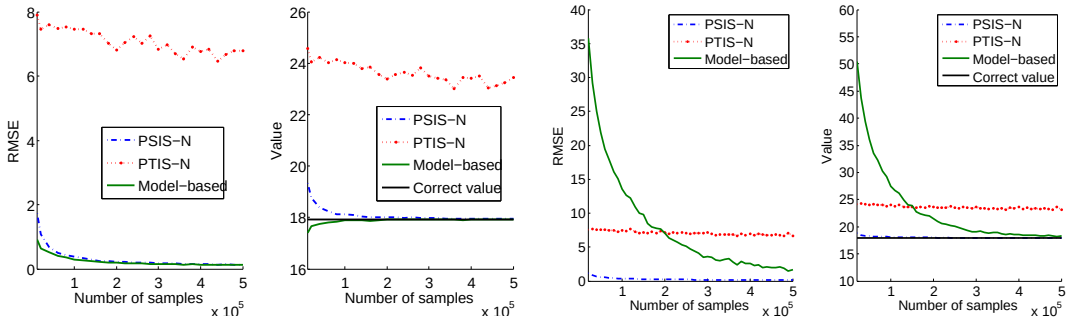


Figure 4.2: RMSE and bias of the 20-step value function estimate for various sample sizes, using *state-dependent hunting* as the target policy.



(*a*) Default MB reward = 0. (*b*) Default MB reward = 10.

Figure 4.3: RMSE and bias of the 20-step value function estimate for various sample sizes, using *encourage hunting* as the target policy. The only difference between the two graphs is the default value used by the model-based method for R_{sa} when $n_{sa} = 0$. Note that the x axis is different from Figure 4.2, allowing for batch sizes of up to 500,000 samples.

different state distribution (as seen in Figure 4.1). This suggests that PTIS-N's performance is particularly weak when the problem is highly off-policy.

The poor performance of per-trajectory methods is particularly interesting, given that they are commonly used as a method for evaluating treatment effects from clinical trials [Robins et al., 2000; Murphy, 2005]. We conjecture that this happens because of at least two reasons. First, the existing evaluations of multi-stage treatments are based on clinical trials that typically have very short horizon lengths (two and three stage trials are common), and for such short horizons the difference between the methods' performance is not as large. Second, epidemiologists tend to include information about previous treatment in the state space, making the set of states accessible in k steps different for all k. In such a setup,

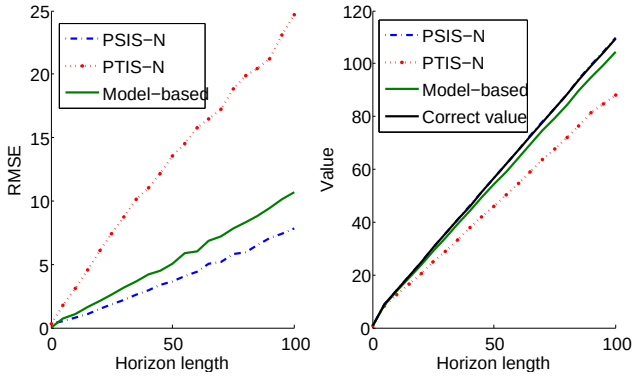


Figure 4.4: RMSE and bias for various horizons and batches of 500 samples each, using *state-dependent hunting* as the target policy. For the bias graph, the line for PSIS-N overlaps that for the correct value.

methods that take advantage of the Markov property (such as PSIS-N and MB) may offer fewer benefits.

PSIS-N performed strongly in all settings. The only time that the model-based method outperformed PSIS-N was when *encourage hunting* was the target policy, and the default value used by the model-based method for the state-action pairs (s, a) for which $n_{sa} = 0$ was $R_{sa} = 0$ (Figure 4.3(a)). The good performance of the model-based method in this setting is likely to be due to the fact that the reward for most states encountered under *encourage hunting* is actually very close to zero, because *encourage hunting* depletes the animal population. When a different default value was used ($R_{sa} = 10$, Figure 4.3(b)), the model-based method performed noticeably worse, due to the increased bias induced by using a default reward that was further from the true value.

Figure 4.4 illustrates how the length of the horizon affects performance. The performance of all methods degrades as the horizon increases. This is expected, as increasing the horizon while maintaining the same number of samples means that we effectively have fewer samples per time step, which increases the variance. However, the ranking of the methods' performance remains the same regardless of the horizon. We have observed a similar phenomenon when using *encourage hunting* as a target policy.

4.2. Randomly generated MDPs

In this section, we present experiments on a set of randomly generated MDPs. The experiments in the previous section indicated that the model-based method may have reduced performance due to high bias when there are zero samples for some of the state-action pairs. Since increasing the total number of available actions increases the probability of having no samples for a state-action pair, we use the random MDPs to illustrate how the bias of the model-based method is affected by the number of available actions.

The randomly generated MDPs are similar to those used by Castronovo and Ernst [2012]. Each of the MDPs has 20 states and 5 actions, except for one experiment where we varied the number of actions. The transition function is generated by randomly selecting, for each

state-action pair (s, a), 10% of the states as successor states, generating a uniform random variable in $N(s') \in [0, 1]$ for each of the successor states s', and then normalizing to obtain the transition probabilities: $P_{sa}^{s'} = \frac{N(s')}{\sum_{s'' \in Succ(s,a)} N(s'')}$.

For states one through 10, the reward for state-action pair (s, a) is equal to zero with 0.9 probability and to a number chosen uniformly randomly in $[0, 1]$ otherwise. For states 11 through 20, the probabilities are reversed (zero with probability 0.1 and a uniform number with probability 0.9). This is slightly different from Castronovo and Ernst [2012] - they used the first reward model (the one we used for states 1-10) as a prior for generating deterministic rewards at all the states in the MDP. The starting state is state 1.

We used a uniform behaviour policy that selected each action with equal probability at all states. The target policy was one that ascribed 60% of the probability mass to the first action, with the rest spread equally among the other actions.

Similar to the previous experiment, we computed the bias and RMSE of the different estimators with respect to the correct value function. We sampled 10 different MDPs, and for each of the MDPs we generated 1000 batches. The results we will present are averaged over the resulting 10000 batches.

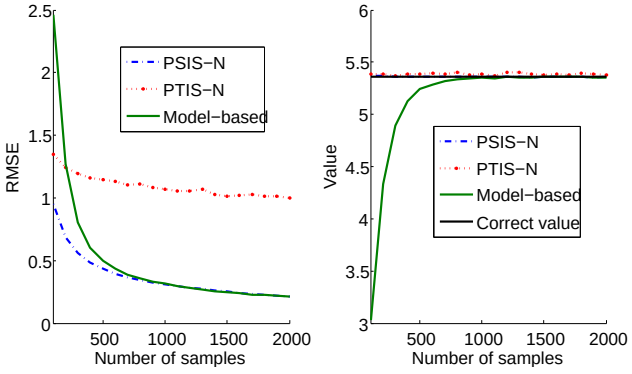
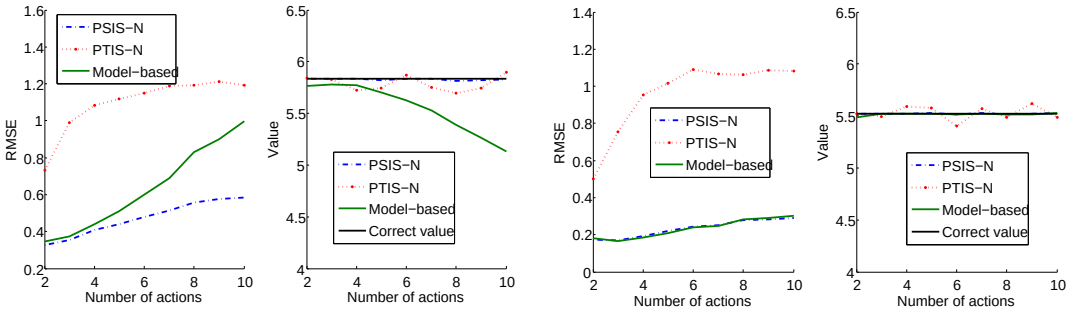


Figure 4.5: RMSE and bias of the 20-step value function estimate for various sample sizes on the random MDPs.

Figure 4.5 illustrates the performance of the methods as a function of the sample size on a 20-step problem. The results are similar to those observed on the mallard domain, with PSIS-N having the best performance, the model-based estimator having high RMSE for small sample sizes due to bias induced by having to use default parameter values when $n_{sa} = 0$, and PTIS-N being very slow to converge.

In order to emphasize the effect that having to use default parameter values when $n_{sa} = 0$ has on the model-based method, we conducted an experiment where we varied the number of actions for our random MDPs. Our hypothesis was that the probability that some n_{sa} is zero will increase as the number of actions increases, and therefore the bias of the model-based method will increase as well. The results, displayed in Figure 4.6, illustrate this phenomenon. For small sample sizes (Figure 4.6(a)), the bias of the model-based method increases steeply as the number of available actions increases. In contrast, PSIS-N appears to be more robust to changes in the size of the action set. For larger sample sizes, $P(n_{sa} = 0)$



(*a*) Batch size = 500 transitions. (*b*) Batch size = 2000 transitions.

Figure 4.6: RMSE and bias of the 20-step value function estimate for various action set sizes on the random MDPs. The two graphs differ in the number of samples in the each batch.

decreases, and all methods are affected by the change in the size of the action set in a similar way (Figure 4.6(*b*)).

5. Discussion

We studied several off-policy learning algorithms, including two new estimators, PSIS and PSIS-N, that are per-step versions of importance sampling which take advantage of Markov assumptions about the model. We briefly discussed the estimators' bias and consistency, and presented a detailed empirical analysis of their performance in a case study pertaining to the management of an animal species. We found that the model-based estimator and the normalized per-step estimator (PSIS-N) performed particularly well. We also found that the model-based estimator can suffer from significant bias if no samples are available for some of the state-action pairs, particularly for problems with many available actions.

We emphasize that the importance sampling estimators require a fixed and known behaviour policy. If the behaviour policy is instead estimated from data, we obtain the model-based estimator (as shown). Cases in which the data is gathered according to multiple behaviour policies (e.g. gathered from different geographic locations), could also be easily incorporated in the estimators by appropriate weighting of the different data batches.

The bias and variance of the off-policy estimators were illustrated through the empirical results. From a theoretical standpoint, there are challenges in providing a formal analysis of the weighted estimators in the sequential case (horizon > 1). This is an interesting area for future work, though we expect it may be difficult to obtain closed-form expressions for these quantities.

As shown in our experiments, it is crucial to assess values for longer time horizons, as the horizon impacts the value of a policy, as well as the ordering of policies. Our results suggest that the horizon length should also be an important factor when choosing an estimator. Some decision-making domains, notably in medicine, deal with relatively short horizons, and in those cases estimators such as PTIS, which have large variance over long horizons

but are unbiased, may be preferable. In domains with longer decision horizons, estimators such as PSIS-N tend to have lower error (though the error increases with horizon length).

One limitation of this work is that the empirical domains are discrete, simulated environments. Additional experiments with real data would undoubtedly make the analysis more compelling. However, gathering real data under a new policy in domains such as natural resource management tends to be expensive, or even impossible. Therefore, ground truth (V^π in our case) is difficult to establish, potentially rendering comparative analyses less meaningful.

In continuous MDPs, off-policy learning can be applied, but generates further complications. In particular, the discrepancy between the probability of a trajectory under the behaviour and the target policy can lead to divergence. Several algorithms have been proposed in order to account for trajectory distribution discrepancies. Precup et al. [2001] use importance sampling weights to correct for the probability of reaching a specific point in a trajectory. The resulting estimators are consistent (in the space of representable value functions) but tend to have high variance. Sutton et al. [2009] address the problem of off-policy learning from on-line data. The main idea is to estimate a secondary set of parameters (in addition to those describing the value function), which are used to stabilize the value function weights and prevent divergence. In discrete MDPs, however, all these estimators are more conservative and hence less sample-efficient than those on which we focused.

Acknowledgements

Funding for this research was provided by the National Institutes of Health (grant R21 DA019800) and the Natural Sciences and Engineering Council Canada (Discovery Grant program).

References

Naoki Abe, P. Melville, C. Pendus, C.K. Reddy, D.L. Jensen, V.P. Thomas, J.J. Bennett, G.F. Anderson, B.R. Cooley, M. Kowalczyk, and Others. Optimizing debt collections using constrained reinforcement learning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2010.

D.R. Anderson. Optimal exploitation strategies for an animal population in a Markovian environment: a theory and an example. *Ecology*, 56(6):1281–1297, 1975.

J.A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

M. Castronovo and D. Ernst. Learning Exploration / Exploitation Strategies for Single Trajectory Reinforcement Learning. In *10th European Workshop on Reinforcement Learning*, 2012.

Edward J. Dudewicz and Satya N. Mishra. *Modern Mathematical Statistics*. Wiley, New York, NY, 1988.

M Dudik, J Langford, and L Li. Doubly Robust Policy Evaluation and Learning. In *International Conference on Machine Learning*, 2011.

C.J. Fonnesbeck. Solving dynamic wildlife resource optimization problems using reinforcement learning. *Natural Resource Modeling*, 18(1):1–40, 2005.

L Hannah and D Dunson. Approximate Dynamic Programming for Storage Problems. In *International Conference on Machine Learning*, 2011.

JDY Kang, J L Schafer, Anastasios a Tsiatis, and Marie Davidian. Demystifying double robustness: a comparison of alternative strategies for estimating a population mean from incomplete data. *Statistical Science*, 22(4):569–573, January 2007.

S. Mannor, D. Simester, P. Sun, and J. N. Tsitsiklis. Bias and Variance Approximation in Value Function Estimates. *Management Science*, 53(2):308–322, February 2007.

S A Murphy. An experimental design for the development of adaptive treatment strategies. *Statistics in medicine*, 24(10):1455–81, May 2005.

J Pineau, A Guez, R D Vincent, G Panuccio, and M Avoli. Treating epilepsy via adaptive neurostimulation: a reinforcement learning approach. *International journal of neural systems*, 19(4):227–40, August 2009.

D Precup, RS Sutton, and S Singh. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.

Doina Precup, RS Sutton, and S Dasgupta. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.

M Riedmiller. Neural fitted Q-iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the European Conference on Machine Learning*, volume 3720. Springer, 2005.

J M Robins, M a Hernán, and B Brumback. Marginal structural models and causal inference in epidemiology. *Epidemiology*, 11(5):550–60, September 2000.

Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. Wiley, New York, NY, 1981.

R S Sutton and A G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvari, and Eric Wiewiora. Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation. In *International Conference on Machine Learning*, 2009.

JMLR: Workshop and Conference Proceedings 24:103–116, 2012 10th European Workshop on Reinforcement Learning

Evaluation and Analysis of the Performance of the EXP3 Algorithm in Stochastic Environments

Yevgeny Seldin SELDIN@TUEBINGEN.MPG.DE
Max Planck Institute for Intelligent Systems, Tübingen, Germany
University College London, London, UK

Csaba Szepesvári SZEPESVA@UALBERTA.CA
University of Alberta, Edmonton, Canada

Peter Auer AUER@UNILEOBEN.AC.AT
Montanuniversität Leoben, Austria

Yasin Abbasi-Yadkori ABBASIYA@UALBERTA.CA
University of Alberta, Edmonton, Canada

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

EXP3 is a popular algorithm for adversarial multiarmed bandits, suggested and analyzed in this setting by Auer et al. [2002b]. Recently there was an increased interest in the performance of this algorithm in the stochastic setting, due to its new applications to stochastic multiarmed bandits with side information [Seldin et al., 2011] and to multiarmed bandits in the mixed stochastic-adversarial setting [Bubeck and Slivkins, 2012]. We present an empirical evaluation and improved analysis of the performance of the EXP3 algorithm in the stochastic setting, as well as a modification of the EXP3 algorithm capable of achieving "logarithmic" regret in stochastic environments.

1. Introduction

Multiarmed bandits are the simplest instance of the exploration-exploitation trade-off problem, which is the basic question in reinforcement learning. There exist two main variants of this problem: stochastic and adversarial. In stochastic multiarmed bandit problems the rewards for playing each arm are generated independently from unknown distributions corresponding to each arm [Auer et al., 2002a]. In adversarial multiarmed bandit problems a sequence of rewards is generated for each arm by an adversary before the game starts [Auer et al., 2002b].

The performance of algorithms for multiarmed bandits is usually evaluated in terms of regret bounds. In the stochastic setting the regret bounds control the difference between the reward obtained by the algorithm and the expected reward that would have been obtained if the algorithm would have played the best arm in all rounds of the game. In the adversarial setting the regret bounds control the difference between the reward obtained by the algorithm and the reward that could have been obtained if the algorithm would have played in all rounds the arm corresponding to the best rewards sequence. In both stochastic and adversarial environments we can talk about in-expectation and high-probability regret bounds.

In-expectation regret bounds are concerned with the expected regret of the algorithm (with respect to its internal randomization and, in the stochastic case, the stochasticity of the environment). High-probability regret bounds provide stronger high-probability guarantees on individual roll-outs of the game. This difference is important to keep in mind when comparing different results.

There exist stochastic and deterministic strategies for stochastic multiarmed bandits. Deterministic strategies are based on computing upper confidence bounds for each of the arms at each round of the game and playing the arm with the highest upper confidence bound. The simplest algorithm from this family is UCB1 algorithm of Auer et al. [2002a], which uses Hoeffding's inequality for computing the upper confidence bounds. Several improvements of this algorithm were proposed, which use tighter concentration inequalities for computing the upper confidence bounds and/or more careful algorithms [Audibert and Bubeck, 2009; Audibert et al., 2009; Auer and Ortner, 2010; Garivier and Cappé, 2011; Maillard et al., 2011].

Stochastic algorithms for stochastic multiarmed bandits include Thompson sampling [Thompson, 1933; Chapelle and Li, 2011; Kaufmann et al., 2012] and EwS algorithm of Maillard [2011]. Stochastic policies have some practical advantages over deterministic UCB-type algorithms. In particular, they are easier to apply in the situation of delayed feedback [Chapelle and Li, 2011].

The most well-known algorithm for adversarial multiarmed bandits is the EXP3.P suggested by Auer et al. [2002b]. In each round of the game EXP3.P picks an arm according to a Gibbs distribution based on upper confidence bounds for each arm, and plays it.

Most analyses of algorithms for stochastic multiarmed bandits provide in-expectation regret bounds. Typically these bounds achieve regret of the form $O(\sum_{\{a:\Delta(a)>0\}} \frac{\ln t}{\Delta(a)})$, termed "logarithmic" regret, where $\Delta(a)$ is the gap between the expected reward of arm a and the expected reward of the "best" arm (the one corresponding to the highest expected reward). In the adversarial case EXP3.P yields $\tilde{O}(\sqrt{KT})$ high-probability regret bound, where the time horizon T is assumed to be known and $\tilde{O}$ hides some logarithmic factors [Auer et al., 2002b]. Unknown time horizons are treated by using the "doubling trick".

There is another algorithm for adversarial multiarmed bandits suggested in Auer et al. [2002b], called EXP3, that picks arms according to a Gibbs distribution based on empirical importance-weighted rewards of the arms instead of using upper confidence bounds. To be more precise, the Gibbs distribution is mixed in a carefully designed proportion with a uniform distribution, which performs the exploration.

We note that there is an important distinction between EXP3 and stochastic algorithms for stochastic multiarmed bandits mentioned earlier (Thompson sampling and EwS): EXP3 is based on importance-weighted sampling, whereas the other two algorithms are based on unweighted rewards. Importance-weighted sampling provides certain advantages when moving to more complex problems, such as stochastic multiarmed bandits with side information [Seldin et al., 2011]. It is also used as a part of the strategy in mixed adversarial and stochastic settings [Bubeck and Slivkins, 2012]. This motivates our study of the performance of EXP3 in stochastic environments.

Seldin et al. [2012] observed that the performance of EXP3 in stochastic environments is comparable to the performance of UCB1 in the "initial phase" of the game (the "initial phase" corresponds to $t < \ln(\Delta)^2/\Delta^4$, where Δ is the minimal positive gap). However,

EXP3 cannot keep up with UCB1 after the "initial phase" and the paper provided only suboptimal high-probability $\tilde{O}(K^{1/3}t^{2/3})$ regret bound for the algorithm. We note that Auer et al. [2002b] provided a $\tilde{O}(\sqrt{KT})$ in-expectation regret bound for EXP3 in the adversarial setting, but no high-probability statement could be derived. The question of whether a high-probability $\tilde{O}(\sqrt{Kt})$ regret bound can be derived in the stochastic setting remained open. Although there is a high-probability $\tilde{O}(\sqrt{KT})$ regret bound for the EXP3.P algorithm in the adversarial setting (which directly implies that the same bound holds in the stochastic setting), Seldin et al. [2012] showed that in practice in the stochastic setting EXP3.P is significantly inferior to EXP3, which provided the motivation for a deeper study of EXP3.

1.1. Summary of the Main Contributions

In Theorem 1 we provide a $\tilde{O}(\sqrt{KT})$ high-probability regret bound for application of the EXP3 algorithm in stochastic environments. We note that we analyze EXP3 algorithm with time-varying learning rate, which is a more elegant approach than the doubling trick.

Theorem 1 is based on Lemma 2, which shows that in stochastic environments the empirical importance-weighted rewards of all arms are "well-behaved" with high probability. We note that in adversarial environments the empirical importance-weighted rewards are not "well-behaved" and additional tools are required for their control, such as EXP3.P.

We also propose and analyze a modification of the EXP3 algorithm that we name EXP3ELM, which is based on a combination of EXP3 with action elimination (see Algorithm 2 box). EXP3ELM performs identically to EXP3 in the "initial phase" of the game and comparably to UCB1 after the "initial phase". In Theorem 3 we prove a "logarithmic" regret bound for EXP3ELM.

We also provide an empirical evaluation of the performance of EXP3 and EXP3ELM.

2. Problem Setting and Definitions

Let $\mathcal{A}$ be a set of K actions (arms) and let $a \in \mathcal{A}$ denote the actions. Denote by $R(a)$ the expected reward of action a and let $p(\cdot|a)$ be the unknown reward distribution underlying a. We assume that the support of $p(\cdot|a)$ is contained in the [0,1] interval. Let π_t be a distribution over $\mathcal{A}$ that is played at round t of the game (a policy). Let $\{A_1, A_2, \dots\}$ be the sequence of actions played, such that A_t is distributed according to π_t and π_t is computed based on the history of past observations, $\mathcal{H}_{t-1} \equiv (A_1, \dots, A_{t-1}, R_1, \dots, R_{t-1})$, where $R_1, R_2, \dots$ is the sequence of observed rewards, so that R_t is distributed according to $p(\cdot|A_t)$.

For $t \geq 1$ and $a \in \{1, \dots, K\}$ define a set of random variables R_t^a (the importance weighted rewards) and their cumulative sum $\hat{R}_t(a)$ as:

$$R_t^a \equiv \frac{R_t}{\pi_t(a)} \mathbb{I}_{\{A_t=a\}} = \begin{cases} \frac{1}{\pi_t(a)} R_t, & \text{if } A_t = a; \\ 0, & \text{otherwise;} \end{cases} \qquad \hat{R}_t(a) \equiv \sum_{\tau=1}^{t} R_\tau^a.$$

Note that $\mathbb{E}[R_t^a|\mathcal{H}_{t-1}] = R(a)$ and $\mathbb{E}[\hat{R}_t(a)] = tR(a)$.

Let $a^* \equiv \arg\max_a R(a)$ be the "best" action in the game (if there are multiple "best" actions, pick any of them arbitrarily). We define the regret and empirical regret of an action

$t = 0$; $\varepsilon_0 = \frac{1}{K}$; $\forall a$: $\hat{R}_0(a) = 0$.
for $t = 1, 2, \ldots$ **do**
  $\varepsilon_t = \min\left\{\frac{1}{K}, \sqrt{\frac{\ln K}{Kt}}\right\}$.
  $\forall a$: $\tilde{\rho}_t(a) = (1 - K\varepsilon_t)\frac{e^{\varepsilon_{t-1}\hat{R}_{t-1}(a)}}{\sum_{a'} e^{\varepsilon_{t-1}\hat{R}_{t-1}(a')}} + \varepsilon_t$.
  Draw A_t according to $\tilde{\rho}_t$ and play it.
  Observe reward R_t.
  $\forall a$: $\hat{R}_t(a) = \hat{R}_{t-1}(a) + \frac{R_t}{\tilde{\rho}_t(a)}\mathbb{I}_{\{a=A_t\}}$.
end

Algorithm 1: EXP3.

Input: Confidence parameter δ.
Initialization: $t = 0$; $\varepsilon_0 = \frac{1}{K}$; $\mathcal{A}_0 = \mathcal{A}$; $B = 4(e-2)\left(2\ln K + \ln\frac{2}{\delta}\right)$; $\forall a$: $\hat{R}_0(a) = 0$;
for $t = 1, 2, \ldots$ **do**
  $\varepsilon_t = \min\left\{\frac{1}{K}, \sqrt{\frac{\ln K}{Kt}}\right\}$.
  $\forall a \in \mathcal{A}_t$:
  $\tilde{\rho}_t(a) = (1 - |\mathcal{A}_t|\varepsilon_t)\frac{e^{\varepsilon_{t-1}\hat{R}_{t-1}(a)}}{\sum_{a' \in \mathcal{A}_t} e^{\varepsilon_{t-1}\hat{R}_{t-1}(a')}} + \varepsilon_t$.
  Draw $A_t \in \mathcal{A}_t$ according to $\tilde{\rho}_t$ and play it.
  Observe reward R_t.
  $\forall a$: $\hat{R}_t(a) = \hat{R}_{t-1}(a) + \frac{R_t}{\tilde{\rho}_t(a)}\mathbb{I}_{\{a=A_t\}}$.
  $\forall a \in \mathcal{A}_t$: $V_{R_t}(a) = V_{R_{t-1}}(a) + \frac{1}{\tilde{\rho}_t(a)}$.
  $\mathcal{A}_t = \mathcal{A}_{t-1} \setminus$
  $\left\{a : \hat{R}_t^{max} - \hat{R}_t(a) > \sqrt{B(V_{R_t}(\hat{a}_t^*) + V_{R_t}(a))}\right\}$.
end

Algorithm 2: EXP3ELM.

a by:

$$\Delta(a) \equiv R(a^*) - R(a), \qquad \hat{\Delta}_t(a) \equiv \hat{R}_t(a^*) - \hat{R}_t(a).$$

Note that $\hat{R}_t(a) - tR(a)$ is a martingale. The variance of this martingale satisfies:

$$\sum_{\tau=1}^{t} \mathbb{E}\left[(R_\tau^a - R(a))^2 \middle| \mathcal{H}_{\tau-1}\right] = \sum_{\tau=1}^{t} \mathbb{E}\left[(R_\tau^a)^2 \middle| \mathcal{H}_{\tau-1}\right] - tR(a)^2 \le \sum_{\tau=1}^{t} \pi_t(a)\frac{(R_t)^2}{\pi_t(a)^2} = \sum_{\tau=1}^{t} \frac{1}{\pi_\tau(a)},$$

where we used the fact that $R_t \le 1$. Let $V_{R_t}(a) \equiv \sum_{\tau=1}^{t} \frac{1}{\pi_\tau(a)}$ be the upper bound on the variance of the martingale $\hat{R}_t(a) - tR(a)$.

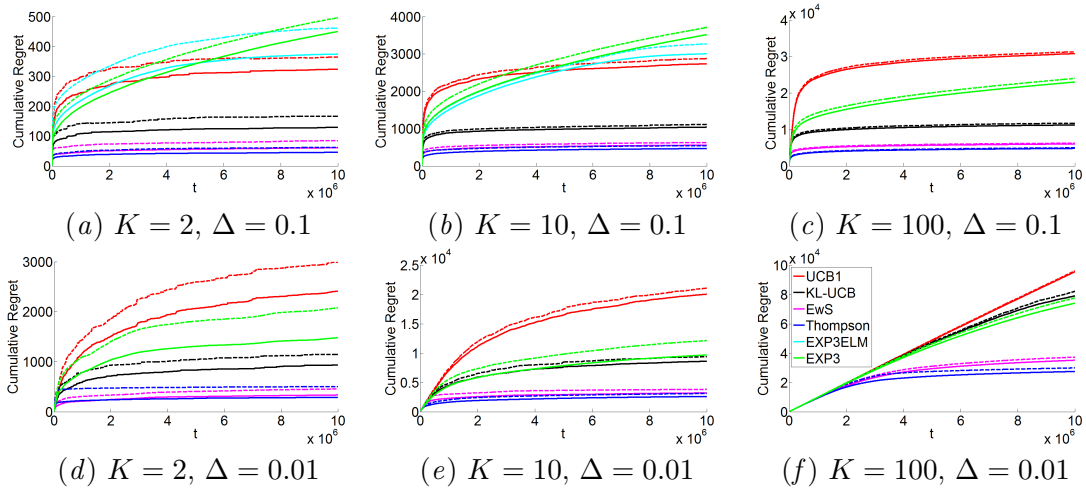
Finally, define the highest empirical reward at round t and the empirically best arm at round t as:

$$\hat{R}_t^{max} \equiv \max_a \hat{R}_t(a), \qquad \hat{a}_t^* \equiv \arg\max_a \hat{R}_t(a).$$

3. Algorithms

In this section we present two learning algorithms that we are using in this paper. The first algorithm is a minor modification of the EXP3 algorithm of Auer et al. [2002b]. The difference is that the learning rate ε_t, which is equal to the exploration rate, is changing with time.

The EXP3 algorithm keeps sampling "bad" actions at the rate of $\varepsilon_t = \sqrt{\frac{\ln K}{Kt}}$ and, therefore, its regret is $\Omega(\sqrt{Kt})$ and it cannot compete with UCB1 after the "initial phase" of the game. This shortcoming is fixed in the EXP3ELM algorithm in the Algorithm 2 box. EXP3ELM maintains a set $\mathcal{A}_t$ of "active" actions. Initially, $\mathcal{A}_0 = \mathcal{A}$ and the actions are withdrawn from the active set as soon as it becomes evident with high probability that they are suboptimal.



(a) $K = 2$, $\Delta = 0.1$ (b) $K = 10$, $\Delta = 0.1$ (c) $K = 100$, $\Delta = 0.1$

(d) $K = 2$, $\Delta = 0.01$ (e) $K = 10$, $\Delta = 0.01$ (f) $K = 100$, $\Delta = 0.01$

Figure 4.1: **Comparison of EXP3, EXP3ELM, UCB1, KL-UCB, EwS, and Thompson sampling.** The legend in figure (f) corresponds to all the figures. Solid lines represent the mean performance over the experiments and dashed lines represent the mean plus one standard deviation over the ten repetitions of the corresponding experiment. In figures (c) - (f) the number of rounds was insufficient for arms elimination to take place, therefore, the behavior of EXP3ELM is identical to EXP3 and EXP3ELM is omitted from the figures. Enlarged version of the figure is provided in the supplementary material.

4. Experimental Results

Before we dive into the analysis of the algorithms we present several experimental results. We consider stochastic multiarmed bandit problem with Bernoulli rewards. For all the suboptimal arms the rewards are Bernoulli with bias 0.5 and for the single best arm the reward is Bernoulli with bias $0.5 + \Delta$. We run the experiments with $K = 2$, $K = 10$, and $K = 100$, and $\Delta = 0.1$ and $\Delta = 0.01$ (in total, six combinations of K and Δ). We run each game for 10^7 rounds and make ten repetitions of each experiment.

We compare EXP3 and EXP3ELM algorithms presented in the previous section with UCB1 [Auer et al., 2002a], KL-UCB [Garivier and Cappé, 2011; Maillard et al., 2011] (we use the version of the algorithm suggested in Garivier and Cappé [2011] with constant $c = 3$), EwS [Maillard, 2011], and Thompson sampling [Thompson, 1933; Kaufmann et al., 2012]. From the experiments we see that EXP3 is superior to UCB1 in the "initial phase" of the game and EXP3ELM is identical to EXP3 in the "initial phase" and comparable to UCB1 after the "initial phase". The KL-UCB algorithm is a much stronger competitor. Nevertheless, for the hardest "needle in a haystack" problems ($\Delta = 0.01$ and $K = 10$ and 100) EXP3 performs comparably and, in the hardest case, even slightly better. EwS and Thompson sampling are clear leaders, however, we remind that EXP3 is based on importance-weighted sampling and its study in the stochastic setting is of independent interest.

5. Analysis

In this section we present an analysis of the EXP3 and EXP3ELM algorithms. For the rigorous analysis we consider a slightly modified version of the algorithms that we name EXP3C and EXP3ELMC. The modification reduces the learning rate by a factor of $1/\sqrt{C \ln K}$ (namely, we take $\varepsilon_t = \min\left\{1/\sqrt{CKt},\ 1/(2K)\right\}$), where C is defined in Theorem 1. We point out that C depends on the confidence parameter δ, but not on the time horizon. In practice, the reduction of the learning rate reduces the performance. The elimination rule in the modified EXP3ELMC algorithm uses a slightly weaker deterministic upper bound on the variance of the martingales:

$$\mathcal{A}_t = \mathcal{A}_{t-1} \setminus \left\{a : \hat{R}_t^{max} - \hat{R}_t(a) > 2\sqrt{CKt}\right\}. \tag{1}$$

We let $\hat{R}_t(EXP3C) \equiv \sum_{\tau=1}^{t} R_\tau$ denote the reward of EXP3C after t rounds. Similarly, $\hat{R}_t(EXP3ELMC)$ is the cumulative reward of EXP3ELMC after t rounds.

Theorem 1 *Let $\delta \in (0,1)$ and define $C \equiv 4(e-2)\left(\ln K + \ln \frac{4}{\delta}\right) e^2$. Then with probability greater than $1-\delta$ the regret of EXP3C is bounded as*:

$$tR(a^*) - \hat{R}_t(EXP3C) \le (1+\ln K)\sqrt{CKt} + 2(3e-4)\sqrt{\frac{Kt}{C}} + (e-2)\sqrt{\frac{1}{2}t \ln \frac{2}{\delta}} + 2K. \tag{2}$$

The key element in the proof of Theorem 1 are inequalities (3) and (4) in the following lemma, which show that the empirical rewards of all actions in EXP3 are "well-behaved" and that the empirical reward of the best action a^* always stays "at the top". (The proofs are provided at the end of this section.)

Lemma 2 *In the EXP3C algorithm with probability greater than $1 - \frac{\delta}{2}$, for all t*:

$$\hat{R}_t^{max} \le tR(a^*) + \sqrt{CKt}, \tag{3}$$

$$\hat{R}_t(a^*) \ge tR(a^*) - \sqrt{CKt}, \tag{4}$$

$$V_{R_t}(a^*) \le C'Kt, \tag{5}$$

$$\frac{1}{\tilde{\rho}_t(a^*)} \le C'K, \tag{6}$$

$$\text{If } \hat{R}_{t-1}(a) \ge (t-1)R(a^*) - \sqrt{CK(t-1)}, \text{ then } \frac{1}{\tilde{\rho}_t(a)} \le C'K, \tag{7}$$

where $C' = e^2$ and C is defined in Theorem 1.

It is important to note that in the EXP3.P algorithm for the adversarial case the control over empirical importance-weighted rewards is achieved by adding the variance of the rewards to the exponent of the Gibbs distribution in $\tilde{\rho}_t$. As it follows from Lemma 2, in the stochastic case the empirical importance-weighted rewards are controlled directly.

Finally, we show that EXP3ELM achieves "logarithmic" regret ($\ln t$ term does not appear explicitly in the bound, but it is replaced by $\ln(1/\delta)$ in the definition of C). The proof of this theorem is provided in the supplementary material.

Theorem 3 *Let $\delta \in (0,1)$ and C as defined in Theorem 1, let RHS(2) be the right hand side of (2), then with probability greater than $1-\delta$ the regret of EXP3ELMC is bounded as*:

$$tR(a^*) - \hat{R}_t(EXP3ELMC) \leq \min\left\{RHS(2),\ 4CK\sum_a \frac{1}{\Delta(a)}\right\}.$$

5.1. Proofs

Let $\rho_t(a) \equiv \frac{e^{\varepsilon_{t-1}\hat{R}_{t-1}(a)}}{Z(\rho_t)}$ denote the non-smoothed version of $\tilde{\rho}_t$, where $Z(\rho_t) \equiv \sum_a e^{\varepsilon_{t-1}\hat{R}_{t-1}(a)}$ is the normalization factor. Also, let $\rho_t^{\varepsilon}(a) \equiv \frac{e^{\varepsilon\hat{R}_{t-1}(a)}}{Z(\rho_t^{\varepsilon})}$ and $Z(\rho_t^{\varepsilon}) \equiv \sum_a e^{\varepsilon\hat{R}_{t-1}(a)}$ denote the corresponding quantities, where instead of the "native" value ε_{t-1} we use a different ε.

We use the following two results from prior work. The first result from Auer et al. [2002b] relates the rewards R_t to the logarithm of the fraction of normalization coefficients. Note that in the definition of ρ_t we have ε_{t-1}, so $Z(\rho_t^{\varepsilon_{t-2}})$ corresponds to ρ_t with ε from the definition of ρ_{t-1}.

Lemma 4 ([Auer et al., 2002b])

$$R_t \geq (1-K\varepsilon_t)\frac{1}{\varepsilon_t}\ln\frac{Z(\rho_t^{\varepsilon_{t-2}})}{Z(\rho_{t-1})} - (e-2)\varepsilon_t\frac{1}{\tilde{\rho}_t(A_t)} \tag{8}$$

The second result from [Maillard, 2011, Section 3.2] bounds the sum of logarithms of the fractions of normalization coefficients.

Lemma 5 ([Maillard, 2011])

$$\sum_{\tau=1}^{t}\frac{1}{\varepsilon_\tau}\ln\frac{Z(\rho_\tau^{\varepsilon_{\tau-2}})}{Z(\rho_{\tau-1})} \geq \hat{R}_t(a^*) - \frac{\ln K}{\varepsilon_t}.$$

We also prove the following lemma that takes care of the last term in (8). Lemma 5 together with Lemma 6 allow us to work with a varying learning rate.

Lemma 6 *With probability greater than $1-\frac{\delta}{2}$, for all t:*

$$\sum_{\tau=1}^{t}\varepsilon_\tau\frac{1}{\tilde{\rho}_\tau(A_\tau)} \leq 2\sqrt{\frac{Kt}{C}} + \sqrt{\frac{1}{2}t\ln\frac{2}{\delta}}.$$

Proof of Lemma 6 We have $\mathbb{E}\left[\frac{1}{\tilde{\rho}_t(A_t)}\middle|\mathcal{H}_{t-1}\right] = \sum_a \tilde{\rho}_t(a)\frac{1}{\tilde{\rho}_t(a)} = K$, which implies that $\sum_{\tau=1}^{t}\varepsilon_\tau\left(\frac{1}{\tilde{\rho}_\tau(A_\tau)} - K\right)$ is a martingale. Since $\varepsilon_t\frac{1}{\tilde{\rho}_t(A_t)} \in (0,1]$, we have by Hoeffding-Azuma's inequality, with probability greater than $1-\frac{\delta}{2}$:

$$\sum_{\tau=1}^{t}\varepsilon_\tau\frac{1}{\tilde{\rho}_\tau(A_\tau)} \leq K\sum_{\tau=1}^{t}\varepsilon_\tau + \sqrt{\frac{1}{2}t\ln\frac{2}{\delta}} \leq 2\sqrt{\frac{Kt}{C}} + \sqrt{\frac{1}{2}t\ln\frac{2}{\delta}}.$$

By martingale stopping argument, similar to the one used in the proof of Bernstein's inequality (see supplementary material), the statement holds for all t simultaneously. ■

Now we are ready to prove Theorem 1. (The proof of Lemma 2 is provided below.)

Proof of Theorem 1 Summing over t the two sides of Lemma 4 and applying Lemmas 5, 2, and 6, we obtain with probability greater than $1-\delta$:

$$\begin{aligned}
\sum_{\tau=1}^{t} R_t &\geq \sum_{\tau=1}^{t}(1-K\varepsilon_\tau)\frac{1}{\varepsilon_\tau}\ln\frac{Z(\rho_\tau^{\varepsilon_\tau-2})}{Z(\rho_{\tau-1})} - (e-2)\sum_{\tau=1}^{t}\varepsilon_\tau\frac{1}{\tilde{\rho}_\tau(A_\tau)} \\
&= \sum_{\tau=1}^{t}\frac{1}{\varepsilon_\tau}\ln\frac{Z(\rho_\tau^{\varepsilon_\tau-2})}{Z(\rho_{\tau-1})} - K\sum_{\tau=1}^{t}\varepsilon_\tau\frac{1}{\varepsilon_\tau}\ln\frac{Z(\rho_\tau^{\varepsilon_\tau-2})}{Z(\rho_{\tau-1})} - (e-2)\sum_{\tau=1}^{t}\varepsilon_\tau\frac{1}{\tilde{\rho}_\tau(A_\tau)} \\
&\geq \hat{R}_t(a^*) - \frac{\ln K}{\varepsilon_t} - 2(e-1)K\sum_{\tau=1}^{t}\varepsilon_\tau - 2(e-2)\sqrt{\frac{Kt}{C}} - (e-2)\sqrt{\frac{1}{2}t\ln\frac{2}{\delta}} \qquad (9) \\
&\geq tR(a^*) - \sqrt{CKt} - \sqrt{CKt}\ln K - 4(e-1)\sqrt{\frac{Kt}{C}} - 2(e-2)\sqrt{\frac{Kt}{C}} - (e-2)\sqrt{\frac{1}{2}t\ln\frac{2}{\delta}} - 2K \qquad (10) \\
&= tR(a^*) - (1+\ln K)\sqrt{CKt} - 2(3e-4)\sqrt{\frac{Kt}{C}} - (e-2)\sqrt{\frac{1}{2}t\ln\frac{2}{\delta}} - 2K,
\end{aligned}$$

where in (9) we applied Lemmas 5 and 6 and used the following upper bound on $\frac{1}{\varepsilon_t}\ln\frac{Z(\rho_t^{\varepsilon_t-2})}{Z(\rho_{t-1})}$, which follows from Lemma 4 (by definition $\varepsilon_t \leq \frac{1}{2}$)

$$\frac{1}{\varepsilon_t}\ln\frac{Z(\rho_t^{\varepsilon_t-2})}{Z(\rho_{t-1})} \leq \frac{1}{1-K\varepsilon_t}\left(R_t + (e-2)\varepsilon_t\frac{1}{\tilde{\rho}_t(A_t)}\right) \leq 2(1+(e-2)) = 2(e-1),$$

and in (10) we applied Lemma 2. (By the union bound, Lemmas 2 and 6 hold simultaneously with probability greater than $1-\delta$.) ■

Proof of Lemma 2 We prove the lemma by induction. For $t=1$ all the claims of the lemma hold. We assume that the claims hold for $t-1$ and show that this implies that they also hold for t.

We start with the proof of (6). If $\rho_t(a^*) \geq \frac{1}{K}$ we are done. Otherwise:

$$\begin{aligned}
\frac{1}{\tilde{\rho}_t(a^*)} \leq \frac{1}{\rho_t(a^*)} &= \frac{\sum_{a'} e^{\varepsilon_{t-1}\hat{R}_{t-1}(a')}}{e^{\varepsilon_{t-1}\hat{R}_{t-1}(a^*)}} \\
&\leq \sum_{a'} e^{\varepsilon_{t-1}\left(\hat{R}_{t-1}^{max} - (t-1)R(a^*) + \sqrt{CK(t-1)}\right)} \leq Ke^{2\varepsilon_{t-1}\sqrt{CK(t-1)}} = C'K,
\end{aligned}$$

where we used the fact that $\hat{R}_{t-1}(a') \leq \hat{R}_{t-1}^{max}$ for all a' and the induction assumption $\hat{R}_t(a^*) \geq tR(a^*) - \sqrt{CKt}$.

Inequality (5) follows from (6) and inequality (4) follows from (5) by Bernstein's inequality (see supplementary material).

Now we prove (7). We note that the proof of (6) was based on the induction assumption (4) for $t-1$. In (7) we assumed that $\hat{R}_{t-1}(a) \geq (t-1)R(a^*) - \sqrt{CK(t-1)}$ and using this assumption the proof is identical.

It is now left to prove (3). If $\hat{R}_{t-1}(a) < (t-1)R(a^*) - \sqrt{CK(t-1)}$ then

$$\hat{R}_t(a) = \hat{R}_{t-1}(a) + R_t^a \leq (t-1)R(a^*) - \sqrt{CK(t-1)} + \frac{1}{\varepsilon_t} \leq tR(a^*) + \sqrt{CKt}.$$

Otherwise, we are in the case $\hat{R}_{t-1}(a) \geq (t-1)R(a^*) - \sqrt{CK(t-1)}$. Let

$$\ell(t) \equiv \max_{\tau} \left\{ \tau : \begin{array}{c} \tau<t \text{ and} \\ \hat{R}_{\tau-1}(a)<(\tau-1)R(a^*)-\sqrt{CK(\tau-1)} \\ \text{and } \hat{R}_{\tau}(a)\geq\tau R(a^*)-\sqrt{CK\tau} \end{array} \right\}.$$

$\ell(t)$ is the last time before t when $\hat{R}_\tau(a)$ crosses the $\tau R(a^*) - \sqrt{CK\tau}$ line from below. If $\hat{R}_\tau(a)$ always stays above this line, define $\ell(t) \equiv 1$.

Let $\hat{R}_{t_1}^{t_2}(a) \equiv \sum_{\tau=t_1+1}^{t_2} R_\tau^a$ be the sub-sum of the rewards of arm a from time t_1+1 to time t_2. By our assumption and the definition of $\ell(t)$ we have $\hat{R}_\tau(a) \geq (t-1)R(a^*) - \sqrt{CK(t-1)}$ for all $\ell(t) \leq \tau < t$. Furthermore, $\hat{R}_{\ell(t)}^t - (t-\ell(t))R(a)$ is a martingale and, by (7), the variance of this martingale satisfies

$$\sum_{\tau=\ell(t)+1}^{t} \mathbb{E}\left[(\hat{R}_\tau^a - R(a))^2 | \mathcal{H}_{\tau-1}\right] \leq \sum_{\tau=\ell(t)+1}^{t} \frac{1}{\tilde{\rho}_\tau(a)} \leq C'K(t-\ell(t)).$$

Therefore, by Bernstein's inequality:

$$\hat{R}_{\ell(t)}^t \leq (t-\ell(t))R(a) + \sqrt{CK(t-\ell(t))} \leq (t-\ell(t))R(a^*) + \sqrt{CKt}.$$

As well,

$$\hat{R}_{\ell(t)}(a) = \hat{R}_{\ell(t)-1}(a) + \frac{1}{\varepsilon_{\ell(t)}} \leq (\ell(t)-1)R(a^*) - \sqrt{CK(\ell(t)-1)} + \frac{1}{\varepsilon_{\ell(t)}} \leq \ell(t)R(a^*).$$

Putting it together we obtain:

$$\hat{R}_t(a) = \hat{R}_{\ell(t)}(a) + \hat{R}_{\ell(t)}^t(a) \leq \ell(t)R(a^*) + (t-\ell(t))R(a^*) + \sqrt{CKt} = tR(a^*) + \sqrt{CKt}.$$

■

5.2. Comparison with Other Regret Bounds

We stress out that our results in Theorems 1 and 3 and Lemma 2 are high-probability results, as opposed to in-expectation analysis that is provided for most other algorithms for stochastic multiarmed bandits, so the comparison is not completely straightforward. In Theorem 3 we obtain an $O((\sqrt{Kt})\ln K)$ regret in the "initial phase" of the game and $O(K \ln K \sum_a \frac{1}{\Delta(a)})$ after the "initial phase" of the game. The bound for the "initial phase" can be compared with $O(\sqrt{KT}\frac{\ln(K \ln K)}{\sqrt{\ln K}})$ in-expectation regret bound for an improved version of UCB [Auer

and Ortner, 2010]. We lose slightly less than a factor of $\sqrt{\ln K}$, however, we achieve a stronger high-probability statement. We also note that the $\sqrt{\ln K}$ factor can be slightly reduced by tuning C. The regret bound after the "initial phase" can be compared with the $O(\ln t \sum_a \frac{1}{\Delta(a)})$ in-expectation regret bound for the UCB1 algorithm [Auer et al., 2002a]. We do not have the $\ln t$ factor since we eliminate arms, but this factor will come back if we replace the elimination by more intelligent exploration schedule. On the other hand, we have $K \ln K$ factor in our bound, but, as in the case with the "initial phase", we provide a stronger high-probability statement.

It is also interesting to compare our result with the analysis of EXP3 and EXP3.P algorithms for adversarial multiarmed bandits in Auer et al. [2002b]. In Theorem 1 we provide a high-probability $O(\ln K \sqrt{Kt})$ regret bound for EXP3 in the stochastic setting. This can be compared with $O(\sqrt{KT \ln K})$ in-expectation regret bound for EXP3 and $O(\sqrt{KT \ln(KT)})$ high-probability regret bound for EXP3.P in the adversarial setting. (We note that Auer et al. [2002b] assume that the time horizon T is known and tune the learning rate based on this knowledge. A similar in-expectation regret bound for application of EXP3 with time-varying learning rate to adversarial bandits is provided by Maillard [2011].) The main message of Theorem 1 and Lemma 2 is that in the stochastic setting we can apply Gibbs sampling based on the empirical rewards instead of Gibbs sampling based on upper confidence bounds used by EXP3.P algorithm for adversarial environments.

6. Discussion

We presented a high-probability analysis of the EXP3 algorithm in stochastic environments and designed a modification of EXP3 that performs identically to EXP3 in the "initial phase" of the game and comparably to UCB1 after the "initial phase". The main distinction of the proposed algorithm from existing algorithms for stochastic multiarmed bandits is that it is based on importance-weighted sampling. We provided a comprehensive study of the behavior of importance-weighted sampling in stochastic environments.

Acknowledgments

We would like to thank Odalric-Ambrym Maillard for helpful discussions and useful references. This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886, and by the European Community's Seventh Framework Programme (FP7/2007-2013), under grant agreement N^o270327. This publication only reflects the authors' views.

References

Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Jean-Yves Audibert and Sébastien Bubeck. Minimax policies for adversarial and stochastic bandits. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, 2009.

Jean Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration-exploitation trade-off using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 2009.

Peter Auer and Ronald Ortner. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1-2):55–65, 2010.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47, 2002a.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal of Computing*, 32(1), 2002b.

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings on the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Sébastien Bubeck and Aleksandrs Slivkins. The best of both worlds: stochastic and adversarial bandits. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, 2012.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

David A. Freedman. On tail probabilities for martingales. *The Annals of Probability*, 3(1), 1975.

Aurélien Garivier and Olivier Cappé. The KL-UCB algorithm for bounded stochastic bandits and beyond. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, 2011.

Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An optimal finite time analysis. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)*, 2012.

Odalric-Ambrym Maillard. *Apprentissage Séquentiel: Bandits, Statistique et Renforcement.* PhD thesis, INRIA Lille, 2011.

Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. A finite-time analysis of multi-armed bandits problems with Kullback-Leibler divergences. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, 2011.

Yevgeny Seldin, Peter Auer, François Laviolette, John Shawe-Taylor, and Ronald Ortner. PAC-Bayesian analysis of contextual bandits. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Yevgeny Seldin, Nicolò Cesa-Bianchi, Peter Auer, François Laviolette, and John Shawe-Taylor. PAC-Bayes-Bernstein inequality for martingales and its application to multiarmed bandits. *JMLR Workshop and Conference Proceedings*, 26, 2012.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25, 1933.

Supplementary Material

Appendix A. Proof of Theorem 3

For the proof of Theorem 3 we need one small lemma.

Lemma 7

$$\text{if } \hat{R}_{t-1}(a) \geq \hat{R}_{t-1}^{max} - 2\sqrt{CK(t-1)}, \text{ then } \frac{1}{\tilde{\rho}_t(a)} \leq C'K \tag{11}$$

and for all actions in the active set $\mathcal{A}_t$ at time t, we have $V_{R_t}(a) \leq C'Kt$.

Proof of Lemma 7 We start with the proof of (11). If $\rho_t(a) \geq \frac{1}{K}$ we are done. Otherwise,

$$\begin{aligned}\frac{1}{\tilde{\rho}_t(a)} \leq \frac{1}{\rho_t(a)} &= \frac{\sum_{a'} e^{\varepsilon_{t-1}\hat{R}_{t-1}(a')}}{e^{\varepsilon_{t-1}\hat{R}_t(a)}} \\ &\leq \sum_{a'} e^{\varepsilon_{t-1}\left(\hat{R}_{t-1}^{max} - \hat{R}_{t-1}^{max} + 2\sqrt{CK(t-1)}\right)} = \sum_{a'} e^{2\varepsilon_{t-1}\sqrt{CK(t-1)}} = C'K,\end{aligned}$$

where we used that fact that $\hat{R}_{t-1}(a') \leq \hat{R}_{t-1}^{max}$ for all a' and the assumption $\hat{R}_{t-1}(a) \geq \hat{R}_{t-1}^{max} - 2\sqrt{CK(t-1)}$. Finally, as long as an action a is in the active set, the precondition of (11) holds and, therefore, we obtain $V_{R_t}(a) = \sum_{\tau=1}^{t} \frac{1}{\tilde{\rho}_\tau(a)} \leq C'Kt$. ■

Proof of Theorem 3 By inequalities (3) and (4) and the definition of the elimination rule, with probability greater than $1-\delta$ the best action a^* is never eliminated by EXP3ELMC.

Now we bound the number of times that suboptimal actions are played. As long as action a is active, by Bernstein's inequality and the bound on $V_{R_t}(a)$ in Lemma 7, with probability greater than $1-\delta$ we have $\hat{R}_t(a) \leq tR(a) + \sqrt{CKt}$. On the other hand, $\hat{R}_t^{max} \geq \hat{R}_t(a^*) \geq tR(a^*) - \sqrt{CKt}$. Thus, with probability greater than $1-\delta$:

$$\hat{R}_t^{max} - \hat{R}_t(a) \geq t\Delta(a) - 2\sqrt{CKt}. \tag{12}$$

By the elimination rule (1), an action a is eliminated, at the latest, when the right hand side of (12) is greater than $2\sqrt{CKt}$, which means that with probability greater than $1-\delta$ each suboptimal action is eliminated after at most $\frac{4CK}{\Delta(a)^2}$ rounds. The cumulative regret for playing action a with regret $\Delta(a)$ over $\frac{4CK}{\Delta(a)^2}$ rounds is $\frac{4CK}{\Delta(a)}$. Summing over the actions we obtain the result of the theorem. ■

Appendix B. Bernstein's Inequality

We used the following form of Bernstein's inequality, which is based on a fairly standard martingale stopping argument [Freedman, 1975; Abbasi-Yadkori et al., 2011].

Theorem 8 (Bernstein's Inequality) *Let $X_1, X_2, \ldots$ be a martingale difference sequence, such that $|X_t| \leq \alpha_t$ for an increasing deterministic sequence $\alpha_1, \alpha_2, \ldots$ with probability 1. Let $M_t \equiv \sum_{\tau=1}^{t} X_\tau$ be martingale. Let $\bar{V}_1, \bar{V}_2, \ldots$ be a sequence of deterministic upper bounds on the variance $V_t \equiv \sum_{\tau=1}^{t} \mathbb{E}[X_\tau^2 | X_1, \ldots, X_{\tau-1}]$ of the martingale M_t, such that $\bar{V}_t$-s satisfy $\sqrt{\frac{\ln \frac{2}{\delta}}{(e-2)\bar{V}_t}} \leq \frac{1}{\alpha_t}$. Then with probability greater than $1-\delta$ for all t:*

$$|M_t| \leq 2\sqrt{(e-2)\bar{V}_t \ln \frac{2}{\delta}}.$$

The theorem is based on the following lemma [Freedman, 1975; Beygelzimer et al., 2011].

Lemma 9 *For M_t and V_t defined in Theorem 8 and $\lambda_t \in [0, \frac{1}{\alpha_t}]$:*

$$\mathbb{E}\left[e^{\lambda_t M_t - (e-2)\lambda_t^2 V_t}\right] \leq 1.$$

Proof of Theorem 8. Define the stopping time $t^\star$:

$$t^\star \equiv \min\left\{t : |M_t| > 2\sqrt{(e-2)\bar{V}_t \ln \frac{2}{\delta}}\right\}.$$

Define the stopped martingale difference sequence by $X_t^{t^\star} \equiv X_t \mathbb{I}_{t \leq t^\star}$. We have $|X_t^{t^\star}| \leq \alpha_t$ for all t. Clearly, the stopped martingale difference sequence is also a martingale difference sequence. Define by $M_t^{t^\star} \equiv \sum_{\tau=1}^{t} X_t^{t^\star}$ the stopped martingale and by $V_t^{t^\star}$ its conditional variance process. Clearly, $V_t^{t^\star} \leq \bar{V}_{t^\star}^{t^\star}$ for all t. Hence, we have:

$$\mathbb{E}\left[e^{\lambda_{t^\star} M_t^{t^\star} - (e-2)\lambda_{t^\star}^2 \bar{V}_{t^\star}}\right] \leq \mathbb{E}\left[e^{\lambda_{t^\star} M_t^{t^\star} - (e-2)\lambda_{t^\star}^2 V_t^{t^\star}}\right] \leq 1.$$

From here, by Markov's inequality, with probability greater than $1 - \frac{\delta}{2}$:

$$\lambda_{t^\star} M_t^{t^\star} - (e-2)\lambda_{t^\star}^2 \bar{V}_{t^\star} \leq \ln \frac{2}{\delta} + \ln \mathbb{E}\left[e^{\lambda_{t^\star} M_t^{t^\star} - (e-2)\lambda_{t^\star}^2 \bar{V}_{t^\star}}\right] \leq \ln \frac{2}{\delta}.$$

By applying the same argument to the negative martingale $-M_t$ and taking $\lambda_{t^\star} \equiv \sqrt{\frac{\ln \frac{2}{\delta}}{(e-2)\bar{V}_{t^\star}}}$ we obtain that with probability greater than $1-\delta$:

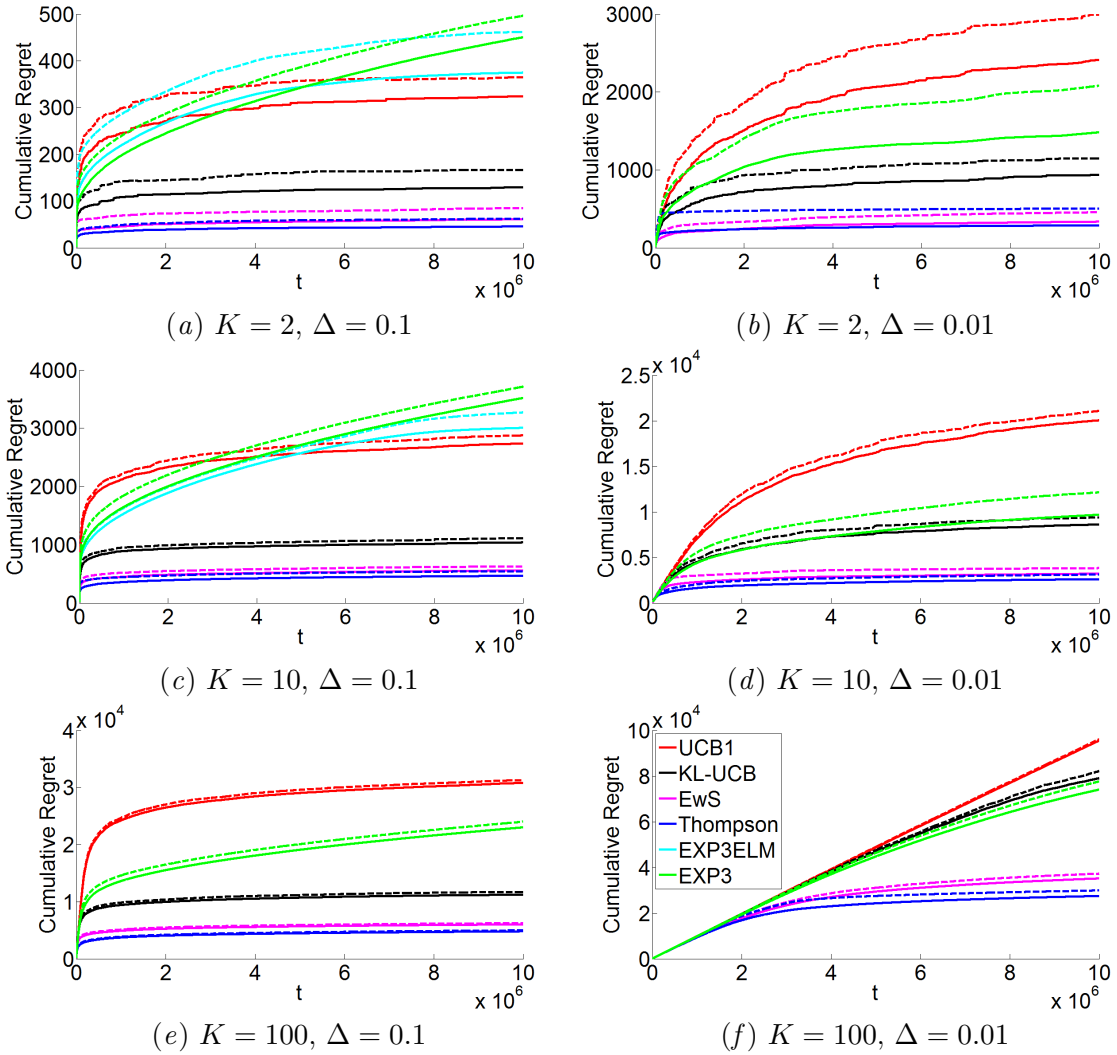
$$|M_t^{t^\star}| \leq \frac{\ln \frac{2}{\delta}}{\lambda_{t^\star}} + (e-2)\lambda_{t^\star}^2 \bar{V}_{t^\star} = 2\sqrt{(e-2)\bar{V}_{t^\star} \ln \frac{2}{\delta}}$$

Finally, we have:

$$\mathbb{P}\left\{\exists t : |M_t| > 2\sqrt{(e-2)\bar{V}_t \ln \frac{2}{\delta}}\right\} = \mathbb{P}\left\{t^\star < \infty\right\} = \mathbb{P}\left\{|M_{t^\star}^{t^\star}| > 2\sqrt{(e-2)\bar{V}_{t^\star} \ln \frac{2}{\delta}},\ t^\star < \infty\right\}$$
$$\leq \mathbb{P}\left\{|M_{t^\star}^{t^\star}| > 2\sqrt{(e-2)\bar{V}_{t^\star} \ln \frac{2}{\delta}}\right\} \leq \delta$$

■

Appendix C. Enlarged Version of Figure 4.1



(a) $K = 2$, $\Delta = 0.1$ (b) $K = 2$, $\Delta = 0.01$

(c) $K = 10$, $\Delta = 0.1$ (d) $K = 10$, $\Delta = 0.01$

(e) $K = 100$, $\Delta = 0.1$ (f) $K = 100$, $\Delta = 0.01$

Figure C.1: **Enlarged version of Figure 4.1.** The columns in this figure correspond to the rows in Figure 4.1. Please, see the caption of Figure 4.1 for explanations.

JMLR: Workshop and Conference Proceedings 24:117–129, 2012 10th European Workshop on Reinforcement Learning

Gradient Temporal Difference Networks

David Silver D.SILVER@CS.UCL.AC.UK
Department of Computer Science, CSML, University College London, London, WC1E 6BT

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

Temporal-difference (TD) networks (Sutton and Tanner, 2004) are a predictive representation of state in which each node is an *answer* to a *question* about future observations or questions. Unfortunately, existing algorithms for learning TD networks are known to diverge, even in very simple problems. In this paper we present the first sound learning rule for TD networks. Our approach is to develop a true gradient descent algorithm that takes account of all three roles performed by each node in the network: as state, as an answer, and as a target for other questions. Our algorithm combines gradient temporal-difference learning (Maei et al., 2009) with real-time recurrent learning (Williams and Zipser, 1994). We provide a generalisation of the Bellman equation that corresponds to the semantics of the TD network, and prove that our algorithm converges to a fixed point of this equation.

1. Introduction

Representation learning is a major challenge faced by machine learning and artificial intelligence. The goal is to automatically identify a representation of state that can be updated from the agent's sequence of observations and actions. *Predictive state representations* (PSRs) provide a promising approach to representation learning. Every state variable is observable rather than latent, and represents a specific prediction about future observations. The agent maintains its state by updating its predictions after each interaction with its environment. It has been shown that a small number of carefully chosen predictions provide a sufficient statistic for predicting all future experience; in other words that those predictions summarise all useful information from the agent's previous interactions [Singh et al., 2004]. The intuition is that an agent which can effectively predict the future will be able to act effectively within its environment, for example to maximise long-term reward.

Temporal-difference networks (TD networks) are a type of PSR that may ask *compositional* predictions, not just about future observations, but about future state variables (*i.e.* predictions of predictions). This enables TD networks to operate at a more abstract level than traditional PSRs. However, temporal-difference networks suffer from a major drawback: the learning algorithm may diverge, even in simple environments.

The main contribution of this paper is to provide a sound and non-divergent learning rule for TD networks and related architectures. Unlike previous TD network learning rules, our approach is based on a true gradient descent algorithm. It uses *gradient temporal-difference* (GTD) learning to account for the compositions of predictions into the future; and backpropagation through time (BPTT) [Rumelhart et al., 1986] or *real-time recurrent learning* (RTRL) [Williams and Zipser, 1989] to correctly account for the full history of previous state variables.

2. Gradient Temporal-Difference Learning

A key problem in reinforcement learning is to estimate the total discounted reward from a given state, so as to evaluate the quality of a fixed policy. In this section we consider a Markov reward process with state space $\mathcal{S}$, transition dynamics from state s to s' given by $\mathcal{T}_{s,s'} = \mathbb{P}[s'|s]$, reward r given by $\mathcal{R}_s = \mathbb{E}[r|s]$, and discount factor $0 < \gamma < 1$. The *return* counts the total discounted reward from time t, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, and the *value function* is the expected return from state s, $V_s = \mathbb{E}[R_t \mid s_t = s]$. The Bellman equation defines a recursive relationship, $V = \mathcal{R} + \gamma \mathcal{T} V$. This equation has a unique fixed point corresponding to the true value function.

In general, the transition dynamics $\mathcal{T}$ and reward function $\mathcal{R}$ are unknown. Furthermore, the number of states $|\mathcal{S}|$ is often large, and therefore it is necessary to approximate the true value function V_s using a function approximator $V_s = f(s, \theta)$ with parameters θ. These parameters can be updated by stochastic gradient descent, so as to minimise the mean squared error between the value function and the return, $\Delta\theta_t = \alpha(R_t - V^\theta_{s_t}) \nabla_\theta V^\theta_{s_t}$, where α is a scalar step-size parameter. However, the return is high variance and also unknown at time t. The idea of *temporal-difference* (TD) learning is to substitute the return R_t with a lower variance, one-step estimate of value, called the *TD target*: $r_t + \gamma V^\theta_{s_{t+1}}$. This idea is known as *bootstrapping* [Sutton, 1988], and leads to the TD learning update, $\Delta\theta_t = \alpha \delta_t \nabla_\theta V^\theta_{s_t}$, where δ_t is the *TD error* $\delta_t = r_t + \gamma V^\theta_{s_{t+1}} - V^\theta_{s_t}$. Temporal-difference learning is particularly effective with a linear function approximator $V^\theta_s = \phi(s)^\top \theta$. Unfortunately, for non-linear function approximation, or for off-policy learning, TD learning is known to diverge [Tsitsiklis and Van Roy, 1997; Sutton et al., 2009].

Temporal-difference learning is not a true gradient descent algorithm, because it ignores the derivative of the TD target. *Gradient temporal-difference* (GTD) learning addresses this issue, by minimising an objective function corresponding to the error in the Bellman equation, by stochastic gradient descent. This objective measures the error between value function V^θ and the corresponding target given by the Bellman equation $\mathcal{R} + \gamma \mathcal{T} V^\theta$. However, the Bellman target typically lies outside the space of value functions that can be represented by function approximator f. It is therefore projected back into this space using a projection Π^θ. This gives the mean squared projected Bellman error (MSPBE), $J(\theta) = ||V^\theta - \Pi^\theta(\mathcal{R} + \gamma \mathcal{T} V^\theta)||^2_\rho$, where the squared norm $||\cdot||^2_\rho$ is weighted by the stationary distribution $\rho(s)$ of the Markov reward process. To ensure tractable computation when using a non-linear function approximator f, the projection operator Π^θ is a linear projection onto the tangent space of V^θ, $\Pi^\theta = \Phi_\theta (\Phi_\theta^\top D \Phi_\theta)^{-1} \Phi_\theta^\top D$, where D is the diagonal matrix $D = diag(\rho)$; and Φ_θ is the tangent space of V^θ, where each row is a value gradient $\phi(s) = (\Phi_\theta)_{s,\cdot} = \nabla_\theta V^\theta_s$.

The *GTD2* algorithm minimises the MSPBE by stochastic gradient descent. The MSPBE can be rewritten as a product of three expectations,

$$J(\theta) = \mathbb{E}[\phi(s)\delta]^\top \mathbb{E}\left[\phi(s)\phi(s)^\top\right]^{-1} \mathbb{E}[\phi(s)\delta] \tag{1}$$

which can be written as $J(\theta) = \mathbb{E}[\phi(s)\delta]^\top w_\theta$ where $w_\theta = \mathbb{E}\left[\phi(s)\phi(s)^\top\right]^{-1} \mathbb{E}[\phi(s)\delta]$. The GTD2 algorithm simultaneously estimates $w \approx w_\theta$ by stochastic gradient descent; and also minimises $J(\theta)$ by stochastic gradient descent, assuming that the current estimate is correct,

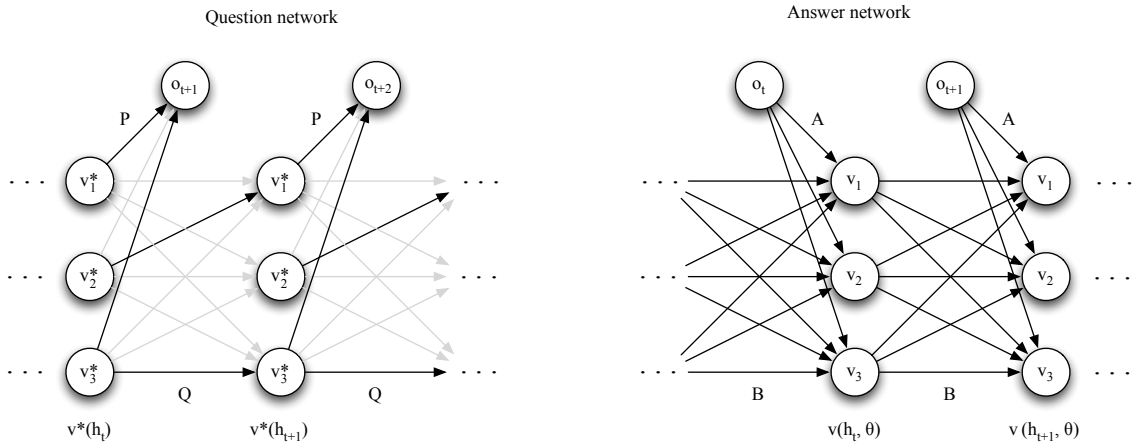


Figure 3.1: A TD network with 3 predictions and 1 observation. The question network gives the semantics of the predictions: v_1^* is the expected observation at the next time-step; v_2^* is the expected observation after two time-steps; and v_3^* is the expected sum of future observations. The structure of the question network is specified by weight matrices P, Q; black edges have weight 1 and grey edges have weight 0. The answer network specifies the mechanism by which answers to these questions are updated over time. Its weight matrices A, B are adjusted so that $v \approx v^*$.

i.e. that $w = w_\theta$. This gives the following updates, applied at every time-step t with step-size parameters α and β,

$$\psi_t = (\delta_t - \phi(s_t)^\top w)\nabla^2_\theta V_s w \tag{2}$$

$$\Delta\theta_t = \alpha\left(\phi(s_t) - \gamma\phi(s_{t+1}))(\phi(s_t)^\top w\right) - \psi_t \tag{3}$$

$$\Delta w_t = \beta\phi(s_t)\left(\delta_t - \phi(s_t)^\top w\right) \tag{4}$$

The GTD2 algorithm converges to a local minimum of $J(\theta)$, even when using non-linear function approximation [Maei et al., 2009] or off-policy learning [Sutton et al., 2009]. The *TDC* algorithm minimises the same objective but using a slightly different derivation,

$$\Delta\theta_t = \alpha\left(\phi(s_t)\delta - \gamma\phi(s_{t+1})(\phi(s_t)^\top w)\right) - \psi_t \tag{5}$$

3. Temporal Difference Networks

We focus now on the uncontrolled case in which an agent simply receives a time series of m-dimensional observations o_t at each time-step t. A history h_t is a length t sequence of observations, $h_t = o_1 ... o_t$. We assume that histories are generated by a Markov chain with a probability $1 - \gamma$ of terminating after every transition, to give a history of length τ. The Markov chain has (unknown) transition probabilities $T_{h,ho} = \gamma\mathbb{P}\left[o_{t+1} = o \mid h_t = h, \tau > t\right]$,

with zero probability for all other transitions. This Markov chain has a well-defined distribution $d(h)$ over the set $\mathcal{H}$ of all histories, $d(h_t) = \mathbb{P}[\tau = t, o_1 ... o_t = h_t]$.

A *state representation* is a function $v(h)$ mapping histories to a vector of state variables $v_i(h)$. For a state representation to be usable online, it must be *incremental* – *i.e.* the state can be updated from one observation to the next, $v(h_{t+1}) = f(v(h_t), o_{t+1})$. A *predictive state representation* (PSR) is a state representation in which all state variables are predictions of future events. The *target* of a prediction is a function of future observations, $z_t = g(o_{t+1}, o_{t+2}, ...)$. The *true answer* for a prediction is the expected value of the target, given a history of observations h, $v^*(h) = \mathbb{E}[z_t \mid h_t = h]$. The PSR learns an estimated *answer* $v(h, \theta) \approx v^*(h)$ to each prediction, by adjusting parameters θ. The answer vector $v(h_t, \theta)$ is the state representation used by the agent at time-step t. For example, in the original PSR, g was a vector of indicator functions over future observations, and f was a linear function [Singh et al., 2004].

A temporal-difference (TD) network is comprised of a *question network* and an *answer network*. The question network defines the semantics of the compositional predictions, i.e. how a target depends on subsequent targets. We focus here on *linear question networks*, where the target is a linear combination of the subsequent target and observation, $z_t = Po_{t+1} + Qz_{t+1}$. Linear question networks satisfy a *Bellman network equation*,

$$v^*(h_t) = \mathbb{E}[z_t \mid h_t] = \mathbb{E}[Po_{t+1} + Qz_{t+1} \mid h_t] = \mathbb{E}[Po_{t+1} + Qv^*(h_{t+1}) \mid h_t] \tag{6}$$

The simplest predictions are grounded directly in observations, for example the target might be the observation at the next time-step, $v_1^*(h) = \mathbb{E}[o_{t+1} \mid h_t = h]$. A compositional prediction might be the expected value at time-step $t+1$ of the expected observation at time-step $t+2$, $v_2^*(h) = \mathbb{E}[v_1^*(h_{t+1}) \mid h_t = h]$. Compositional questions can also be recursive, so that questions can be asked about temporally distant observations. For example, a value function can be represented by a prediction of the form $v_3^*(h) = \mathbb{E}[o_t + v_3^*(h_{t+1}) \mid h_t = h]$, where o_t can be viewed as a reward, and v_3^* is the value function for this reward.

The answer network is a non-linear representation of state containing n state variables. Each state variable $v_i(h)$ represents the estimated answer to the ith prediction. The answers are updated incrementally by combining the state $v(h_{t-1})$ at the last time-step – the previous answers – with the new observation o_t. In the original TD network paper, the answers were represented by a recurrent neural network, although other architectures are possible. In this case, $v(h_t, \theta) = \sigma(Ao_t + Bv(h_{t-1}))$ where $\sigma(x)$ is a non-linear activation function with derivative $\sigma'(x)$; $\theta = [A, B]$ is an $n \times (m + n)$ weight matrix; and $v(h_0) := 0$ by definition.

The key idea of temporal-difference networks is to train the parameters of the answer network, so that the estimated answers approximate the true answers as closely as possible. One way to measure the quality of our answers is using the Bellman network equation, i.e. to seek answers $v(h, \theta)$ that (approximately) satisfy this equation. One approach to solving the Bellman network equation is by *bootstrapping*, i.e. by using the right hand side of the equation, $Po_{t+1} + Qv(h_{t+1}, \theta)$, as a surrogate for the true value function $v^*(h_t)$. The parameters of the answer network can then be adjusted online by gradient descent, so as to

minimise the mean squared error, $MSE(\theta) = \mathbb{E}\left[\sum_{i=1}^{n} \left(v_i^*(h_t) - v_i(h_t, \theta)\right)^2\right]$,

$$\begin{aligned} -\frac{1}{2}\nabla_\theta MSE(\theta) &= \mathbb{E}\left[\sum_{i=1}^{n} \left(v_i^*(h_t) - v_i(h_t, \theta)\right) \nabla_\theta v_i(h_t, \theta)\right] \\ &= \mathbb{E}\left[\phi(h_t, \theta)\left(v^*(h_t) - v(h_t, \theta)\right)\right] \end{aligned} \tag{7}$$

where $\phi(h, \theta)$ is a matrix combining the gradients for each answer,

$$\phi(h, \theta) = [\nabla_\theta v_1(h, \theta), ..., \nabla_\theta v_n(h, \theta)] \tag{8}$$

Sampling the gradient gives a stochastic gradient descent algorithm,

$$\begin{aligned} \Delta\theta &= \alpha\phi(h_t, \theta)\left(v^*(h_t) - v(h_t, \theta)\right) \\ &\approx \alpha\phi(h_t, \theta)\left(Po_{t+1} + Qv(h_{t+1}, \theta) - v(h_t, \theta)\right) \\ &= \alpha\phi(h_t, \theta)\delta(h_{t+1}, \theta) \end{aligned} \tag{9}$$

where α is a scalar step-size; and $\delta(h_{t+1}, \theta)$ is the *TD error*,

$$\delta(h_{t+1}, \theta) := Po_{t+1} + Qv(h_{t+1}, \theta) - v(h_t, \theta) \tag{10}$$

However, Sutton et al. did not in fact follow the full recurrent gradient, but rather a computationally expedient one-step approximation to this gradient,

$$\begin{aligned} &\nabla_\theta v_i(h_t) = \nabla_\theta \sigma(A_{i,:}o(h_t) + B_{i,:}v(h_{t-1}))_i \\ &\frac{\partial v_i(h_t)}{\partial A_{j,k}} \approx \begin{cases} o_k(h_t)\sigma'(v_i(h_t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \Delta A_{i,k} = \alpha\delta_i(h_{t+1}, \theta)o_k(h_t)\sigma'(v_i(h_t)) \\ &\frac{\partial v_i(h_t)}{\partial B_{j,k}} \approx \begin{cases} v_k(h_{t-1})\sigma'(v_i(h_t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \qquad \Delta B_{i,k} = \alpha\delta_i(h_{t+1}, \theta)v_k(h_{t-1})\sigma'(v_i(h_t)) \end{aligned} \tag{11}$$

Like simple recurrent networks [Elman, 1991], this *simple TD network* learning rule only considers direct connections from previous state variables $v(h_{t-1})$ to current state variables $v(h_t)$, ignoring indirect contributions from previous state variables $v(h_1), ..., v(h_{t-2})$.

Simple temporal-difference networks have been extended to incorporate histories [Tanner and Sutton, 2005a]; to learn over longer time-scales using a TD(λ)-like algorithm [Tanner and Sutton, 2005b]; to use action-conditional or option-conditional questions [Sutton et al., 2005]; to automatically discover question networks [Makino and Takagi, 2008] and to use hidden nodes [Makino, 2009].

Unfortunately, it is well known that the simple TD network learning rule may lead to divergence, even in seemingly trivial problems such as a 6-state "cycle world" [Tanner, 2005]. This is because each node in a TD network may perform up to three different roles. First, a node may act as state, i.e. the state variable $v_i(h_{t-1})$ at the previous time-step $t-1$ is used to construct answers $v_j(h_t)$ at the current time-step t. Second, a node may act as an answer, i.e. $v_j(h_t)$ provides the current answer for the jth prediction at time t. Third, a node may act as a target, i.e. $v_k(h_{t+1})$ may be predicted by a predecessor answer $v_j(h_t)$. The simple

TD network learning rule only follows the gradient with respect to the first of these three roles, and is therefore not a true gradient descent algorithm. Specifically, if an answer is selfishly updated to reduce its error with respect to its own question, this will also change the state representation, which could result in greater overall error. It will also change the TD-targets, which could again result in greater overall error. This last issue is well-known in the context of value function learning, causing TD learning to diverge when using a non-linear value function approximator [Tsitsiklis and Van Roy, 1997]. In the remainder of this paper, we develop a sound TD network learning rule that correctly takes account of all three roles.

4. Bellman Network Equation

The standard Bellman equation provides a recursive one-step relationship for one single prediction: the value function. The main contribution of this paper is to generalise the Bellman equation to linear question networks (see equation 6). We can then find answer network parameters θ that solve, as effectively as possible, the Bellman network equation. We now make this notion precise.

In our Markov chain, there is one "state" for each distinct history. The Bellman network equation provides a recursive relationship between the answers $v_i(h)$ for every prediction i and every history h. To represent all of these values, we define an *answer matrix* $V^\theta_{h,i} = v_i(h, \theta)$, which is the $\infty \times n$ matrix of all answers given parameters $\theta = [A, B]$.

Each prediction is weighted by a non-negative *prediction weight* c_i indicating the relative importance of question i. Each history h is weighted by its probability $d(h)$. We define a weighted squared norm $||.||_{c,d}$ that is weighted both by the prediction weight c_i and by the history distribution $d(h)$,

$$||V||^2_{c,d} = \sum_{h \in \mathcal{H}} d(h) \sum_{i=1}^{n} c_i V^2_{h,i} \tag{12}$$

We define $\Pi(V)$ to be the non-linear projection function that finds the closest answer matrix V^θ to the given matrix V, using the weighted squared norm $||.||^2_{c,d}$,

$$\Pi(V) = \operatorname*{argmin}_\theta ||V^\theta - V||^2_{c,d} \tag{13}$$

The question network represents the relationship between true answers. However, the true expectations will not typically be representable by any parameter vector θ, and therefore we must project them back into the representable space of answer networks,

$$V^\theta = \Pi(T(OP^\top + V^\theta Q^\top)) \tag{14}$$

where O is the $\infty \times m$ matrix of last observations, $O_{h_t,i} = o_t$; P and Q are the parameters of the question network; and T is the history transition matrix. In practice, the non-linear projection Π is intractable to compute. We therefore use a local linear approximation Π_θ that finds the closest point in the tangent space to the current answer matrix V^θ. The tangent space is represented by an $\infty \times (m+n)n$ matrix Φ_θ of partial derivatives. Each row

of the tangent space corresponds to a particular combination of history h and prediction i, with a row index that we denote by $h \circ i$. Each column j corresponds to a parameter θ_j of the answer network, $(\Phi_\theta)_{h\circ i,j} = \frac{\partial V^\theta_{h,i}}{\partial \theta_j}$. The linear projection operator Π_θ projects an answer matrix V onto tangent space Φ_θ,

$$\Pi_\theta \overline{V} = \Phi_\theta (\Phi_\theta^\top G \Phi_\theta)^{-1} \Phi_\theta^\top G \overline{V} \tag{15}$$

where the operator $\overline{V}$ represents a reshaping of an $\infty \times n$ matrix V into a $\infty \times 1$ column vector with one row for each combined history and prediction $h \circ i$. G is a diagonal matrix that gives the combined history distribution and prediction weight for each history and prediction $h \circ i$, $G_{h\circ i, h\circ i} = c_i d(h)$. This linear projection leads to the *projected Bellman network equation*,

$$\overline{V^\theta} = \Pi_\theta \overline{T(OP^\top + V^\theta Q^\top)} \tag{16}$$

5. Gradient Temporal-Difference Network Learning Rule

Our approach to solving the linear TD network equation is based on the nonlinear gradient TD learning algorithm of Maei *et al.* (2009). Like this prior work, we solve our Bellman-like equation by minimising a mean squared projected error by gradient descent. However, we must extend Maei *et al.* in three ways. First, TD networks include *multiple* value functions, and therefore we generalise the mean-squared error to use the projection operator Π_θ defined in the previous section. Second, each answer in a TD networks may depend on multiple interrelated targets, whereas TD learning only considers a single TD target $r_t + \gamma V^\theta_{s_{t+1}}$. Third, we must generalise from a finite state space to an infinite space of histories. We proceed by defining the *mean squared projected Bellman network error* (MSPBNE), $J(\theta)$, for answer network parameters A. This objective measures the difference between the answer matrix and the expected TD-targets at the next time-step, projected onto the tangent space,

$$\begin{aligned} J(\theta) &= ||\Pi_\theta \overline{T(OP^\top + V^\theta Q^\top) - V^\theta}||^2_G = ||\Pi_\theta \Delta_\theta||^2_G = \Delta_\theta^\top \Pi_\theta^\top G \Pi_\theta \Delta_\theta = \Delta_\theta^\top G^\top \Pi_\theta \Delta_\theta \\ &= \Delta_\theta^\top G^\top \Phi_\theta (\Phi_\theta^\top G \Phi_\theta)^{-1} \Phi_\theta^\top G \Delta_\theta = \left(\Phi_\theta^\top G \Delta_\theta\right)^\top \left(\Phi_\theta^\top G \Phi_\theta\right)^{-1} \left(\Phi_\theta^\top G \Delta_\theta\right) \end{aligned} \tag{17}$$

where $\Delta_\theta := \overline{T(OP^\top + V^\theta Q^\top) - V^\theta}$ is the *TD network error*, a column vector giving the expected one-step error for each combined history and prediction $h \circ i$. Each of these three terms can be rewritten as an expectation over histories,

$$\begin{aligned} \Phi_\theta^\top G \Phi_\theta &= \sum_{h\in\mathcal{G}} \sum_{i=1}^{n} \phi_i(h,\theta) c_i d(h) \phi_i(h,\theta)^\top \\ &= \mathbb{E}\left[\phi(h,\theta) C \phi(h,\theta)^\top\right] \end{aligned} \tag{18}$$

$$\begin{aligned} \Phi_\theta^\top G \Delta_\theta &= \Phi_\theta^\top G \left(\overline{T(OP^\top + V^\theta Q^\top) - V^\theta}\right) \\ &= \sum_{h\in\mathcal{G}} \sum_{i=1}^{n} \phi_i(h,\theta) c_i d(h) \left(\sum_{h'\in\mathcal{G}} T_{h,h'} \delta_i(h',\theta)\right) \\ &= \mathbb{E}\left[\phi(h,\theta) C \delta(h',\theta)\right] \end{aligned} \tag{19}$$

where C is the diagonal prediction weight matrix $C_{ii} = c_i$. The MSPBNE can now be written as a product of expectations,

$$w_\theta = \mathbb{E}\left[\phi(h,\theta)C\phi(h,\theta)^\top\right]^{-1}\mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta)\right]$$
$$J(\theta) = \mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta)\right]^\top w_\theta \tag{20}$$

The *GTD network* learning rule updates the parameters by gradient descent, so as to minimise the MSPBNE. In the appendix we derive the gradient of the MSPBNE,

$$\psi(\theta, w_\theta) = \sum_{i=1}^{n} c_i \left(\delta_i(h',\theta) - \phi_i(h,\theta)^\top w_\theta\right)\nabla^2_\theta V_{h,i} w_\theta \tag{21}$$

$$-\frac{1}{2}\nabla_\theta J(\theta) = \mathbb{E}\left[-\nabla_\theta\delta(h',\theta)C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right] \tag{22}$$

From Equation 10 we see the derivative of the TD error is a linear combination of gradients,

$$-\nabla_\theta\delta(h',\theta) = \phi(h,\theta) - \phi(h',\theta)Q^\top \tag{23}$$

As in GTD2, we separate the algorithm into an online linear estimator for $w \approx w_\theta$, and an online stochastic gradient descent update that minimises $J(\theta)$ assuming that $w = w_\theta$,

$$\Delta\theta = \alpha\left[\left(\phi(h,\theta) - \phi(h',\theta)Q^\top\right)C(\phi(h,\theta)^\top w) - \psi(\theta, w)\right] \tag{24}$$

$$\Delta w = \beta\phi(h,\theta)C(\delta(h',\theta) - \phi(h,\theta)^\top w) \tag{25}$$

This *recurrent GTD network* learning rule is very similar to the GTD2 update rule for non-linear function approximators (Equations 1 to 3), where states have been replaced by histories, and where we sum the GTD2 updates over all predictions i, weighted by the prediction weight c_i.

We can also rearrange the gradient of the MSPBNE into an alternative form,

$$\begin{aligned}-\frac{1}{2}\nabla_\theta J(\theta) &= \mathbb{E}\left[\phi(h,\theta)C\phi(h,\theta)^\top w_\theta - \phi(h',\theta)Q^\top C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right] \\ &= \mathbb{E}\left[\phi(h,\theta)C\delta(h',\theta) - \phi(h',\theta)Q^\top C\phi(h,\theta)^\top w_\theta - \psi(\theta, w_\theta)\right]\end{aligned} \tag{26}$$

This gives an alternative update for θ, which we call the *recurrent TDC network* learning rule, by analogy to the non-linear TDC update [Maei et al., 2009].

$$\Delta\theta = \alpha\left[\phi(h,\theta)C\delta(h',\theta) - \phi(h',\theta)Q^\top C(\phi(h,\theta)^\top w) - \psi(\theta, w)\right] \tag{27}$$

The first term is identical to recurrent TD network learning when $c_i = 1, \forall i$ (see equation 9); the remaining two terms provide a correction that ensures the true gradient is followed.

Finally, we note that the components of the gradient matrix $\phi(h,\theta)$ are *sensitivites* of the form $\frac{\partial v_i(h)}{\partial A_{j,k}}$ and $\frac{\partial v_i(h)}{\partial B_{j,k}}$, which can be calculated online using the real-time recurrent learning algorithm (RTRL) [Williams and Zipser, 1989]; or offline by backpropagation through time [Rumelhart et al., 1986]. Furthermore, the Hessian-vector product required by $\psi(\theta, w)$

can be calculated efficiently by Pearlmutter's method [Pearlmutter, 1994]. As a result, if RTRL is used, then the computational complexity of the recurrent GTD (or TDC) network learning is equal to the RTRL algorithm, *i.e.* $O(n^2(m+n)^2)$ per time-step. If BTPP is used, then the computational complexity of recurrent GTD (or TDC) network learning is $O(T(m+n)n^2)$ over T time-steps, which is n times greater than BPTT because the gradient must be computed separately for every prediction. In practice, automatic differentiation (AD) provides a convenient method for tracking the first and second derivatives of the state representation. Forward accumulation AD performs a similar computation, online, to real-time recurrent learning [Williams and Zipser, 1989]; whereas reverse accumulation AD performs a similar computation, offline, to backpropagation through time [Werbos, 2006].

6. Outline Proof of Convergence

We now outline a proof of convergence for the GTD network learning rule, closely following Theorem 2 of [Maei et al., 2009]. For convergence analysis, we augment the above GTD network learning algorithm with an additional step to project the parameters back into a compact set $C \subset \mathbb{R}^{(m+n)n}$ with a smooth boundary, using a projection $\Gamma(\theta)$. Let K be the set of asymptotically stable equilibria of the ODE (17) in [Maei et al., 2009], i.e. the local minima of $J(\theta)$ modified by Γ.

Theorem 1 *Let $(h_k, h'_k)_{k\geq 0}$ be a sequence of transitions drawn i.i.d. from the history distribution, $h_k \sim d(h)$, and transition dynamics, $h'_k \sim P_{h,\cdot} \mid h_k$. Consider the augmented GTD network learning algorithm, with a positive sequence of step-sizes α_k and β_k such that $\sum_{k=0}^{\infty} \alpha_k = \infty, \sum_{k=0}^{\infty} \beta_k = \infty, \sum_{k=0}^{\infty} \alpha_k^2 < \infty, \sum_{k=0}^{\infty} \beta_k^2 < \infty$ and $\frac{\alpha_k}{\beta_k} \to 0$ as $k \to \infty$. Assume that for each $\theta \in C$, $\mathbb{E}(\sum_{i=1}^{n} \phi_i(h,\theta)\phi_i(h,\theta)^\top)$ is nonsingular. Then $\theta_k \to K$, with probability 1, as $k \to \infty$.*

The proof of this theorem follows closely along the lines of Theorem 2 in [Maei et al., 2009], using martingale difference sequences based on the recurrent GTD network learning rule, rather than the GTD2 learning rule. Since the answer network is a three times differentiable function approximator, the conditions of the proof are met.[1]

7. Network Architecture

A wide variety of state representations based on recurrent neural network architectures have been considered in the past. However, these architectures have enforced constraints on the roles that each node can perform, at least in part to avoid problems during learning. For example, simple recurrent (SR) networks [Elman, 1991] have two types of nodes: *hidden* nodes are purely state variables, and *output* nodes are purely answers; no nodes are allowed to be targets. TD networks [Sutton and Tanner, 2004] have one type of node, representing both state variables and answers; no nodes are allowed to be purely a state variable (hidden) or purely an answer (output). Like SR networks, SR-TD networks [Makino, 2009] have both hidden nodes and output nodes, but only hidden nodes are allowed to be targets.

1. We have omitted some additional technical details due to using an infinite state space.

Clearly there are a large number of network architectures which do not follow any of these constraints, but which may potentially have desirable properties.

By defining a sound learning rule for arbitrary question networks, we remove these constraints from previous architectures. Recurrent GTD networks can be applied to any architecture in which nodes take on any or all of the three roles: state, answer and target. To make node i a hidden node, the corresponding prediction weight c_i is set to zero. To make node i an output node, the outgoing edges of the answer network $A_{i,j}$ should be set to zero for all j. Recurrent GTD networks also enable many intermediate variants, not corresponding to prior architectures, to be explored.

8. Empirical Results

We consider a simple *cycle world*, corresponding to a cyclic 6-state Markov chain with observation $o = 1$ generated in state s_1 and observation $o = 0$ generated in the five other states. We used a simple depth 6 "chain" question network containing M-step predictions of the observation: $v_1^*(h_t) = \mathbb{E}[o_{t+1} \mid h_t]$, $v_{i+1}^*(h_t) = \mathbb{E}[v_i^*(h_{t+1}) \mid h_t]$ for $i \in [1, 6]$; and a fully connected recurrent answer network with logistic activation functions. This domain has been cited as a counter-example to TD network learning, due to divergence issues in previous experiments [Tanner, 2005].

We used forward accumulation AD [Rump, 1999] to estimate the sensitivity of the answers to weight parameters. We used a grid search to identify the best constant values for α and β. Weights in $\theta = [A, B]$ were initialised to small random values, and w was initialised to zero. On each domain, we compared the simple TD network learning rule (Equation 11) [Sutton and Tanner, 2004], with the recurrent GTD and TDC network learning rule (Equations 25). 10 learning curves were generated over 50,000 steps, starting from random initial weights; performance was measured by the mean squared observation error (MSOE) in predicting observations at the next time-step. The results are shown in Figure 8.1. The simple TD network learning rule performs well initially, but is attracted into an unstable divergence pattern after around 25,000 steps; whereas GTD and TDC network learning converge robustly on all runs.

9. Conclusion

TD networks are a promising approach to representation learning that combines ideas from both predictive representations and recurrent neural networks. We have presented the first convergent learning rule for TD networks. This should enable the full promise of TD networks to be realised. Furthermore, a sound learning rule enables a wide variety of new architectures to be considered, spanning the full spectrum between recurrent neural networks and predictive state representations.

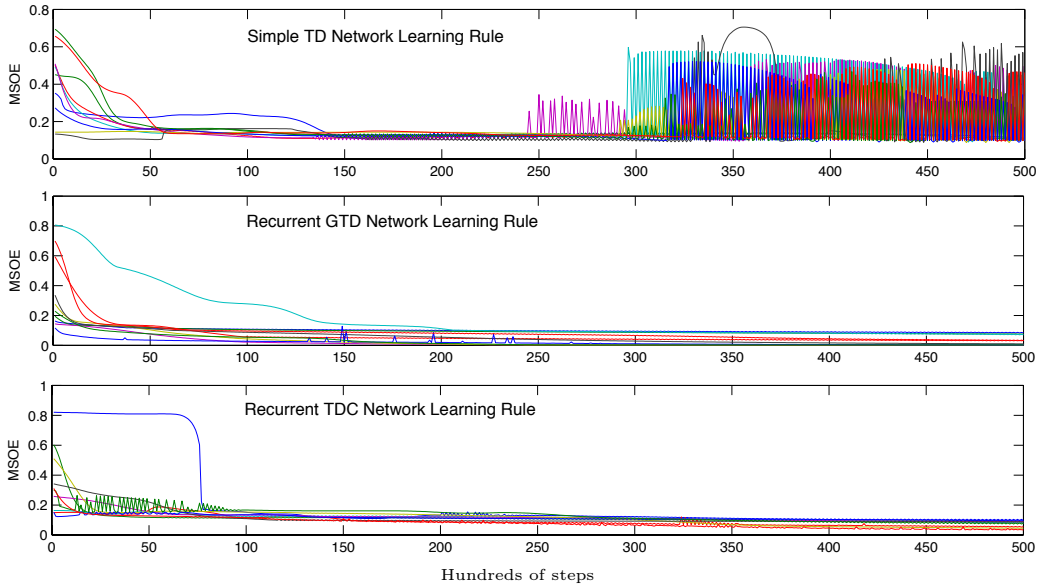


Figure 8.1: Empirical convergence of TD network learning rules on a 6 state cycle world. 10 learning curves are shown for each algorithm, starting from random weights.

References

Jeffrey L. Elman. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7:195–225, 1991.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Rich Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *Advances in Neural Information Processing Systems 22*, pages 1204–1212, 2009.

Takaki Makino. Proto-predictive representation of states with simple recurrent temporal-difference networks. In *26th International Conference on Machine Learning*, page 88, 2009.

Takaki Makino and Toshihisa Takagi. On-line discovery of temporal-difference networks. In *25th International Conference on Machine Learning*, pages 632–639, 2008.

Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1): 147–160, 1994.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, pages 318–362. MIT Press, 1986.

Siegfried Rump. INTLAB - INTerval LABoratory. In *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, 1999.

Satinder P. Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *20th Conference in Uncertainty in Artificial Intelligence*, pages 512–518, 2004.

R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3 (9):9–44, 1988.

Richard S. Sutton and Brian Tanner. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, 2004.

Richard S. Sutton, Eddie J. Rafols, and Anna Koop. Temporal abstraction in temporal-difference networks. In *Advances in Neural Information Processing Systems 18*, 2005.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *26th Annual International Conference on Machine Learning*, page 125, 2009.

Brian Tanner. Temporal-difference networks. Master's thesis, University of Alberta, 2005.

Brian Tanner and Richard S. Sutton. Temporal-difference networks with history. In *19th International Joint Conference on Artificial Intelligence*, pages 865–870, 2005a.

Brian Tanner and Richard S. Sutton. Td(lambda) networks: temporal-difference networks with eligibility traces. In *22nd International Conference on Machine Learning*, pages 888–895, 2005b.

J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674Ð690, 1997.

Paul Werbos. Backwards differentiation in ad and neural nets: Past links and new opportunities. In *Automatic Differentiation: Applications, Theory, and Implementations*, volume 50, pages 15–34. Springer Berlin Heidelberg, 2006.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Appendix A. Gradient of the MSPBNE

The gradient of the MSPBNE can be derived by an extension of Maei *et al.* (2009), making an equivalent assumption that $\mathbb{E}[P]\,\phi(h)C\phi(h)^\top$ is non-singular in a neighbourhood around A. We recall the definition of the MSPBNE $J\theta$), and formulate the derivative in terms of the term w_θ,

$$
\begin{aligned}
w_\theta &= \mathbb{E}\left[P\right]\phi(h)C\phi(h)^\top{}^{-1}\mathbb{E}\left[P\right]\phi(h)C\delta(h',\theta) \\
J(\theta) &= \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
-\frac{1}{2}\nabla_\theta J(\theta) &= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&\quad -\frac{1}{2}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&\quad +\frac{1}{2}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]\mathbb{E}\left[\phi(h)C\phi(h)^\top\right]^{-1}\mathbb{E}\left[\phi(h)C\delta(h',\theta)\right] \\
&= -\nabla_\theta \mathbb{E}\left[\phi(h)C\delta(h',\theta)\right]^\top w_\theta + \frac{1}{2}w_\theta^\top \nabla_\theta \mathbb{E}\left[\phi(h)C\phi(h)^\top\right]w_\theta \\
&= -\nabla_\theta \mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\delta_i(h')\right]^\top w_\theta + \frac{1}{2}w_\theta^\top \nabla_\theta \mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\phi_i(h,\theta)^\top\right]w_\theta \\
&= -\mathbb{E}\left[\sum_{i=1}^n c_i\phi_i(h,\theta)\nabla_\theta\delta_i(h')\right]^\top w_\theta - \mathbb{E}\left[\sum_{i=1}^n c_i\nabla_\theta\phi_i(h,\theta)\delta_i(h')\right]^\top w_\theta + \mathbb{E}\left[\sum_{i=1}^n c_i w_\theta^\top\phi_i(h,\theta)\nabla_\theta\phi_i(h,\theta)^\top w_\theta\right] \\
&= -\mathbb{E}\left[\nabla_\theta\delta(h',\theta)C\phi_i(h,\theta)^\top w_\theta\right] - \mathbb{E}\left[\sum_{i=1}^n c_i\nabla_\theta^2(V(h)w_\theta)\left(\delta_i(h') - \phi_i(h,\theta)^\top w_\theta\right)\right] \\
&= -\mathbb{E}\left[\nabla_\theta\delta(h',\theta)C\phi_i(h,\theta)^\top w_\theta + \psi(\theta,w_\theta)\right]
\end{aligned}
$$

where $\psi(\theta,w_\theta) = \sum_{i=1}^n c_i\nabla_\theta^2(V(h)w_\theta)\left(\delta_i(h',\theta) - \phi_i(h,\theta)^\top w_\theta\right)$

JMLR: Workshop and Conference Proceedings 24:131–141, 2012 10th European Workshop on Reinforcement Learning

Semi-Supervised Apprenticeship Learning

Michal Valko MICHAL.VALKO@INRIA.FR
Mohammad Ghavamzadeh MOHAMMAD.GHAVAMZADEH@INRIA.FR
Alessandro Lazaric ALESSANDRO.LAZARIC@INRIA.FR
INRIA Lille - Nord Europe, team SequeL, France

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

In apprenticeship learning we aim to learn a good policy by observing the behavior of an expert or a set of experts. In particular, we consider the case where the expert acts so as to maximize an unknown reward function defined as a linear combination of a set of state features. In this paper, we consider the setting where we observe many sample trajectories (i.e., sequences of states) but only one or a few of them are *labeled* as experts' trajectories. We investigate the conditions under which the remaining *unlabeled* trajectories can help in learning a policy with a good performance. In particular, we define an extension to the max-margin inverse reinforcement learning proposed by Abbeel and Ng [2004] where, at each iteration, the max-margin optimization step is replaced by a semi-supervised optimization problem which favors classifiers separating clusters of trajectories. Finally, we report empirical results on two grid-world domains showing that the semi-supervised algorithm is able to output a better policy in fewer iterations than the related algorithm that does not take the unlabeled trajectories into account.

1. Introduction

In traditional reinforcement learning, agents are rewarded for achieving certain states and try to learn a policy that maximized the rewards cumulated over time. Constructing such a reward can be not only tedious but a reward could also be difficult to encode explicitly. The goal of apprenticeship learning [Ng and Russell, 2000] is to learn a good behavior by observing the behavior of an expert (usually a human). In fact, in many applications it is easier to collect observations of experts' behaviors rather than encoding a suitable reward function to obtain the desired behavior. The effectiveness of this approach to learn non-trivial behaviors has already been demonstrated on complex tasks such as inverted helicopter navigation [Ng et al., 2004], ball-in-a-cup game [Boularias et al., 2011], or learning highway driving behavior [Abbeel and Ng, 2004; Levine et al., 2011].

A typical scenario for apprenticeship learning is to have an expert performing a few optimal (or close to the optimal) trajectories and use these to learn a good policy. In this paper, we consider a different setting where we also have access to many other trajectories, for which we do not know whether they are actually generated by an expert or not. For instance, in learning a driving behavior we might have recorded many driving patterns but only a few has been thoroughly analyzed and labeled as *expert* trajectories. We refer to this setting as *semi-supervised apprenticeship learning*.

Since the beginning of apprenticeship learning [Ng and Russell, 2000], several different approaches have been proposed, including inverse reinforcement learning [Abbeel and Ng, 2004], max-margin planning [Ratliff et al., 2006], or maximum entropy inverse reinforcement learning [Ziebart et al., 2008]. In this paper, we address the semi-supervised setting by an extension of the (max-margin) inverse reinforcement learning proposed by Abbeel and Ng [2004]. We assume that the *"good"* and *"bad"* trajectories are well enough separated in some feature space defined over the state space. Therefore, a decision boundary that separates good and bad trajectories, should not cross the dense regions of the data. This distributional assumption is typical in semi-supervised learning [Zhu, 2008] and it is often referred to as *cluster assumption*. Such an assumption enables us to use semi-supervised support vector machines [Bennett and Demiriz, 1999] and to take advantage of the unlabeled trajectories when learning a policy. This assumption can be verified if some policies (and therefore trajectories) are more natural given the semantics of the problem.

The IRL method of Abbeel and Ng [2004] is an iterative algorithm, where in each step a new reward is generated. In each step we use a MDP solver to compute the optimal policy given that reward. The goal of IRL is to find a policy (or a mixture thereof) that would match the expected feature counts of the expert (i.e., the probability with which the agent achieves different states). The new reward is generated as a solution to a max-margin classification, where the expert's feature expectations are considered as positives and the feature expectation of the generated policies as negatives. In the method we propose, we aim to reduce the number of iterations of this procedure and therefore reduce the number of calls to the MDP solver. We do it by augmenting the feature expectations of the expert's and generated policies with the feature expectations of unlabeled trajectories. The aim is to improve the max-margin classification learning and discover better policies in fewer iterations. Furthermore, if the expert performer does not achieve the maximum reward already, the unlabeled trajectories may help us discover a better performing policy.

2. Background

Let MDP\R denote a finite state Markov decision process without a reward function: (S, A, T, γ, D), where S is a finite set of states, A is a set of actions, T is a set of transition probabilities, γ is a discount factor, and D is the initial state distribution from which a starting state s_0 is sampled. We assume that we are given a vector of state features $\phi : S \to [0,1]^k$. We also assume the existence of a *true* reward function $R^\star$ defined as a *linear* combination of the features ϕ, i.e., $R^\star(s) = \boldsymbol{w}^\star \cdot \phi(s)$, where $\|\boldsymbol{w}^\star\|_1 \leq 1$ is the true weight reward vector. The value function of a policy π is defined as the expected sum of discounted rewards, i.e., $V^\pi(s) = \mathbb{E}\left[\sum_t \gamma^t R(s_t)\right]$, thus the expected value of a policy π w.r.t. to the initial distribution D becomes

$$\mathbb{E}_{s_0 \sim D}[V^\pi(s_0)] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right] = \boldsymbol{w} \cdot \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi\right] = \boldsymbol{w} \cdot \boldsymbol{\mu}(\pi),$$

where $\boldsymbol{\mu}(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi\right]$ is the vector of *feature expectations* given policy π. We also assume that we can access an MDP solver which, given a reward function, outputs a policy that maximizes it. The main input of an IRL method is a set of m expert trajectories. Let a trajectory $\tau_i = (s^{(i)}_{E,1}, \ldots, s^{(i)}_{E,T_i})$ be a sequence of T_i states generated by the expert

Algorithm 1: IRL: Inverse reinforcement learning [Abbeel and Ng, 2004]

Input: ε, expert trajectories $\{\tau_i\}_{i=1}^m$
Estimate $\hat{\boldsymbol{\mu}}_E \leftarrow \frac{1}{m}\sum_{i=1}^m \sum_{t=0}^{T_i} \gamma^t \phi(s_{E,t}^{(i)})$
Randomly pick $\pi^{(0)}$ and set $i = 1$
repeat
  $t^{(i)} \leftarrow \max_{\|\boldsymbol{w}\|_2 \leq 1} \min_{j<i} \ \boldsymbol{w} \cdot (\hat{\boldsymbol{\mu}}_E - \hat{\boldsymbol{\mu}}^{(j)})$
  Let $\boldsymbol{w}^{(i)}$ be the one attaining this maximum
  $\pi^{(i)} \leftarrow \text{MDPSolver}(R = \boldsymbol{w}^{(i)} \cdot \boldsymbol{\phi})$
  Estimate $\hat{\boldsymbol{\mu}}^{(i)} \leftarrow \boldsymbol{\mu}(\pi^{(i)})$
  $i \leftarrow i + 1$
until $t^{(i)} \leq \varepsilon$

policy π_E starting from state $s_{E,1}^{(i)}$. Consequently, the input of an IRL algorithm is a set $\{\tau_i\}_{i=1}^m$. Given the set of trajectories we can compute an empirical estimate of their feature expectations as

$$\widehat{\boldsymbol{\mu}}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i}\gamma^t \phi(s_{E,t}^{(i)}).$$

The goal of IRL algorithm of Abbeel and Ng [2004] is to find a policy $\tilde{\pi}$, such that $\|\boldsymbol{\mu}(\tilde{\pi}) - \boldsymbol{\mu}_E\|_2 \leq \varepsilon$. In such case, for any $\|\boldsymbol{w}\|_1 \leq 1$:

$$\left|\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)|\pi_E\right] - \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t)|\tilde{\pi}\right]\right| = |\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\mu}(\tilde{\pi}) - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\mu}_E| \leq \|\boldsymbol{w}\|_2 \, \|\boldsymbol{\mu}(\tilde{\pi}) - \boldsymbol{\mu}_E\|_2 = \varepsilon,$$

since $\|\boldsymbol{w}\|_2 \leq \|\boldsymbol{w}\|_1 \leq 1$. This implies that if we find a policy $\tilde{\pi}$ whose feature expectations are ε-close to the ones of the expert, then the performance of $\tilde{\pi}$ would also be ε-close to the performance of the expert, no matter the actual value of the true reward vector $\boldsymbol{w}^*$. Algorithm 1 of Abbeel and Ng [2004] is therefore trying to find such $\tilde{\pi}$. Notice that the algorithm terminates whenever the margin $t^{(i+1)}$ falls below a given threshold ε given as input to the algorithm. As a result,

$$\forall \boldsymbol{w}, \text{ where } \|\boldsymbol{w}\|_2 \leq 1, \quad \exists i \ \text{ s.t. } \ \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\mu}^{(i)} \leq \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\mu}_E - \varepsilon. \tag{1}$$

We can then construct a *mixture* policy $\tilde{\pi}$, such that for any $\boldsymbol{w}$, the feature expectations $\boldsymbol{\mu}(\tilde{\pi})$ are at most ε away from the feature expectations $\boldsymbol{\mu}_E$ of the expert,

$$\boldsymbol{\mu}(\tilde{\pi}) = \min_{\boldsymbol{\mu}} \|\boldsymbol{\mu}_E - \boldsymbol{\mu}\|_2 \ \text{ s.t. } \ \boldsymbol{\mu} = \sum_i \lambda_i \boldsymbol{\mu}^{(i)} \ \text{ and } \ \sum_i \lambda_i = 1. \tag{2}$$

This corresponds to a mix of the policies computed during the execution of Algorithm 1 with the mixing weights $\boldsymbol{\lambda}$. The resulting mixture policy can be executed by sampling from $\{\pi^{(i)}\}_i$ at the beginning and then acting according to $\pi^{(i)}$. For such a mixture, it is guaranteed that $\|\boldsymbol{\mu}_E - \boldsymbol{\mu}(\tilde{\pi})\|_2 \leq \varepsilon$. However, having a resulting policy being a mixture of several policies instead of a single policy is often seen as a drawback of the IRL method [Ratliff et al., 2006; Ziebart et al., 2008].

3. Semi-supervised inverse reinforcement learning

The optimization step for $\boldsymbol{w}$ of Algorithm 1 described in Section 2 is equivalent to support vector machines (SVM) by Vapnik [1995], if we associate a label $+1$ with the expert's feature expectations $\boldsymbol{\mu}_E$, and a label -1 with the feature expectations of the policies computed so far: $\{\boldsymbol{\mu}(\pi^{(j)}) : j = 0 \dots (i-1)\}$. As Abbeel and Ng [2004] point out, the optimization can be solved by any quadratic program solver or a specialized SVM solver.

In the following, we describe how to improve the previous IRL algorithm when we have access to many other trajectories but without knowing whether they are generated by an expert or not. Specifically, unlabeled trajectories can help us to find $\boldsymbol{w}$ that separates the expert feature counts from the non-expert ones faster. We describe how to use semi-supervised SVMs for this purpose.

Besides the standard *hinge loss* $\mathcal{L}(f, \boldsymbol{x}, y) = \max\{1 - y\,|f(\boldsymbol{x})|\,, 0\}$ where $(\boldsymbol{x}, y)$ is a labeled sample and f is a generic classifier, semi-supervised support vector machines (SVMs) also uses the *hat loss* $\widehat{\mathcal{L}}(f, \boldsymbol{x}) = \max\{1 - |f(\boldsymbol{x})|\,, 0\}$ on unlabeled data [Bennett and Demiriz, 1999] and define the optimization problem

$$\hat{f} = \min_f \left[\sum_{i \in L} \mathcal{L}(f, \boldsymbol{x}_i, y_i) + \gamma_l \|f\|^2 + \gamma_u \sum_{i \in U} \widehat{\mathcal{L}}(f, \boldsymbol{x}_i) \right], \tag{3}$$

to compute the max-margin decision boundary $\hat{f}$ that avoids dense regions of data, where L and U are the sets of labeled and unlabeled samples respectively, γ_l is the cost penalty as in standard SVMs, and γ_u weighs the influence of unlabeled examples in U. No matter what the label of an unlabeled example would be, the hat loss $\widehat{\mathcal{L}}$ penalizes functions f with a small distance to unlabeled examples. The hat loss, however, makes the optimization problem (3) non-convex. As a result, it is hard to solve the problem optimally and most of the work in this field has focused on approximations. A comprehensive review of these methods was done by Zhu [2008]. In our experiments, we make use of the transductive SVMs by Joachims [1999a].

The objective of IRL is to find such mixture policy that would be close to the expert's one in the feature space. IRL builds this mixture iteratively with every new policy getting closer. In the early rounds however only few policies are available. This is where we can take advantage of the unlabeled trajectories. Assuming that the expert and non-expert policies are separated in the feature space, optimizing the objective (3) can result in $\boldsymbol{w}$ that is very close to $\boldsymbol{w}^\star$ and in turn can generate better performing policy sooner. In other words, our algorithm in the beginning looks for a policy that would be optimal for $\boldsymbol{w}$ suggested by the unlabeled data. As the more new policies are generated, they outweigh this bias and the final mixture policy would be close to the expert's in the feature space for any possible $\boldsymbol{w}$. Figure 3.1 illustrates how can unlabeled trajectories help the learning when the cluster assumption is satisfied. In the figure, the true reward is the vertical line, the feature expectations of the expert is the big red dot on the bottom left and the feature expectation of the generated trajectory (after the first iteration) is the big blue dot on the top right. The remaining dots correspond to the feature expectations of the unlabeled trajectories. Semi-supervised learning objective gets penalized when the learned reward (decision boundary) crosses the dense regions of the feature space and therefore can recover a reward close to the true one.

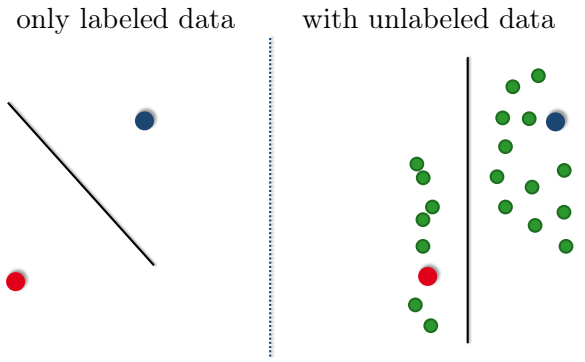


Figure 3.1: Cluster assumption of *semi-supervised learning*. **Left:** supervised learning. **Right:** semi-supervised learning (with unlabeled trajectories)

Specifically, the optimization problem (3) for linear functions f, where the experts feature expectation are labeled as $+1$ and all the feature expectations from the generated policies labeled as -1, simplifies to:

$$\min_{\boldsymbol{w}}\Big(\max\{1 - \boldsymbol{w}^\mathsf{T}\hat{\boldsymbol{\mu}}_E, 0\} + \gamma_l \left\|\boldsymbol{w}\right\|_2 + \sum_{j<i} \max\{1 + \boldsymbol{w}^\mathsf{T}\hat{\boldsymbol{\mu}}^{(j)}, 0\} + \gamma_u \sum_{u\in U} \max\{1 - \left|\boldsymbol{w}^\mathsf{T}\hat{\boldsymbol{\mu}}_u\right|, 0\}\Big)$$

Algorithm 2 shows the pseudocode of our SSIRL algorithm. At the beginning, we compute the empirical estimates $\hat{\boldsymbol{\mu}}_u$ for each unlabeled trajectory (or a set of m_u trajectories, if several trajectories belong to the same performer). After the optimization step for $\boldsymbol{w}$, which corresponds to (3), we normalize $\boldsymbol{w}$ such that $\left\|\boldsymbol{w}\right\|_2 = 1$, in order to preserve the approximation properties of the IRL algorithm. Note that since both SSIRL and IRL algorithms iteratively create new policies, eventually the unlabeled trajectories will be outweighed and, in the limit, the performance of the generated mixture policies will be similar. However, with the informative set of unlabeled trajectories present during the learning, SSIRL is able to recover $\boldsymbol{w}^\star$ faster. Therefore, it may need less iterations than IRL and consequently smaller number of calls to the MDP solver.

After the execution of SSIRL algorithm, we need to perform the same optimization procedure (Equation 2) as for the IRL algorithm to get the mixing weights $\boldsymbol{\lambda}$. However, we would like to stress an important point about policy mixing with SSIRL: Even though the feature expectations from the unlabeled trajectories can speed up the learning we do not know the policies that generated them. This has two consequences. First, these feature expectations cannot be used for finding the final mixture after the algorithm has converged. Consequently, the final mixture policy still consists of policies related to labeled trajectories. Second, the distance between the unlabeled feature counts (i.e. the margin t) cannot be taken into account for the stopping criterion. Otherwise, it could for example happen that we have an unlabeled feature expectations which are ε close to $\boldsymbol{\mu}_E$. The algorithm would terminate immediately but no policy could be actually generated. Therefore, the benefit of SSIRL is that a better $\boldsymbol{w}$ is likely to be found earlier and consequently a better policy is *generated* for that $\boldsymbol{w}$.

Algorithm 2: SSIRL: Semi-supervised inverse reinforcement learning

Input: ε, γ_l, γ_u

expert trajectories $\{s_{E,t}^{(i)}\}$

unlabeled trajectories from U performers $\{s_{u,t}^{(i)}\}$

estimate $\hat{\boldsymbol{\mu}}_E \leftarrow \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T_i}\gamma_l^t\phi(s_{E,t}^{(i)})$

for $u = 1$ **to** U **do**

estimate $\hat{\boldsymbol{\mu}}_u \leftarrow \frac{1}{m_u}\sum_{i=1}^{m_u}\sum_{t=0}^{T_{u,i}}\gamma^t\phi(s_{u,t}^{(i)})$

end for

randomly pick $\pi^{(0)}$ and set $i \leftarrow 1$

repeat

$$\boldsymbol{w}^{(i)} \leftarrow \min_{\boldsymbol{w}}\Big(\max\{1-\boldsymbol{w}^\intercal\hat{\boldsymbol{\mu}}_E,0\}+\gamma_l\left\|\boldsymbol{w}\right\|_2 + \sum_{j<i}\max\{1+\boldsymbol{w}^\intercal\hat{\boldsymbol{\mu}}^{(j)},0\}+\gamma_u\sum_{u\in U}\max\{1-\left|\boldsymbol{w}^\intercal\hat{\boldsymbol{\mu}}_u\right|,0\}\Big)$$

$\boldsymbol{w}^{(i)} \leftarrow \boldsymbol{w}^{(i)}/\left\|\boldsymbol{w}^{(i)}\right\|_2$

$\pi^{(i)} \leftarrow \mathrm{MDP}(R=(\boldsymbol{w}^{(i)})^\intercal\boldsymbol{\phi})$

estimate $\hat{\boldsymbol{\mu}}^{(i)} \leftarrow \boldsymbol{\mu}(\pi^{(i)})$

$t^{(i)} \leftarrow \min_i \boldsymbol{w}^\intercal(\hat{\boldsymbol{\mu}}_E - \hat{\boldsymbol{\mu}}^{(i)})$

$i \leftarrow i+1$

until $t^{(i)} \le \varepsilon$

4. Experiments

The purpose of this section is to show that with unlabeled trajectories, SSIRL can find a good policy in less iterations than IRL, thus requiring fewer calls to the MDP solver. For the comparison we use the variations of the gridworld domain of Abbeel and Ng [2004].

4.1. Gridworld

In the first set of experiments, we use 64×64 gridworlds. The agent has 4 actions (*north*, *west*, *south*, *east*) with 70% chance of success and 30% chance of taking a random different action. The grid is divided into 64 macrocells of size 8×8. These macrocells define the features: for each macrocell i and state s there is a feature ϕ_i such that $\phi_i(s) = 1$ if the state s belongs to the macrocell i and 0 otherwise. The rewards are randomly generated such that the $\boldsymbol{w}^\star$ is sparse, which gives rise to interesting policies [Abbeel and Ng, 2004]. The discount factor γ is set to 0.99.

To simulate the clustering assumption in this domain, we sample the trajectories as follows. We sample $\boldsymbol{w}^\star$ for the expert and a different $\boldsymbol{w}'$ for the non-expert performer. The estimated feature expectations $\hat{\boldsymbol{\mu}}_E$ for the expert were computed by simulating the policy optimizing $\boldsymbol{w}^\star$. The first half of the unlabeled trajectories corresponds to different simulations of the same expert policy. The other half of trajectories was simulated from a policy optimizing $\boldsymbol{w}'$. Therefore, this setup also shows how SSIRL can address *sampling errors*, when the expert trajectory is a noisy sample from the optimal policy.

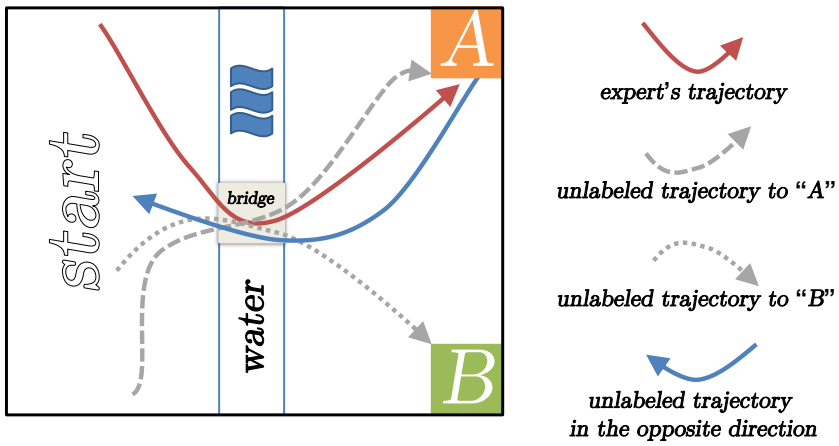


Figure 4.1: Two goals domain

4.2. Two goals navigation

The domain illustrated in Figure 4.1 defines a more natural scenario where clustered feature expectations arise in practice. It consists of a starting section on the left, two places of interest ("A" and "B") in the corners on the right, and a river with a bridge in the middle. The task is to navigate from the start to the one of the two places of interest while avoiding stepping into the river.

The state space, actions, and features are defined as in the previous domain of gridworlds. However, the true reward (unknown to a learner) is set to 0.01 for the feature corresponding to the place "A" and to -0.005 for the features corresponding to the river except for the bridge where it is set to zero (as well for the other regions). The expert demonstrates a trajectory from the start to the place "A". The first half of the unlabeled trajectories consists also from demonstrations navigating from the start to "A" which can be thought of as additional well performing demonstrations.

We perform two experiments with this domain in which we provide different set of trajectories as the other half of unlabeled trajectories. In the first one, the other half of the trajectories are demonstrations from the start to "B" instead. In the second experiment, the other half of demonstrations are trajectories in the reversed direction, i.e., from "A" to the start. Due to the design, one can expect that the both sets of trajectories form two clusters in the feature space of the macrocells. We choose these trajectories such that some features can be helpful (as passing through the bridge) and some not (as going to a different goal or in an opposite direction).

4.3. Implementation details

As an MDP solver, we use a standard policy iteration method. For the optimization of $\boldsymbol{w}$ both in IRL and SSIRL algorithm we use SVMlight implementation of Joachims [1999a],

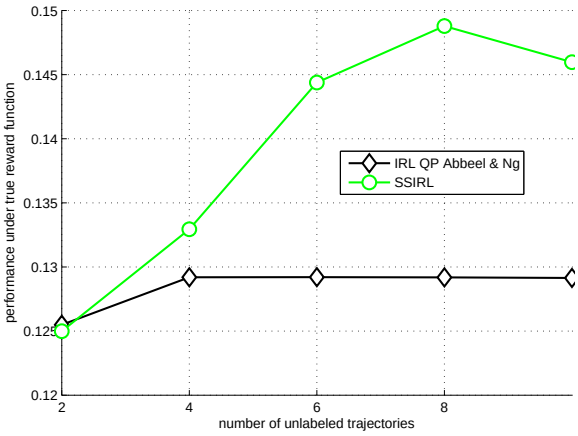


Figure 4.2: Performance of the mixture policies under the true reward after 10 iterations of both algorithms as a function of the number of extra unlabeled trajectories.

which is capable of both supervised and semi-supervised (transductive) SVM learning. Learning parameters γ_l, γ_u were set to the default values. To calculate the mixture coefficients $\boldsymbol{\lambda}$ of the mixture policy (Eq. 2) we use the quadratic program solver CVX by Grant et al. [2006].

4.4. Results

Figures 4.2 and 4.3 show the results of the *gridworld* experiments. Both the true reward function and the reward function for the non-experts are sampled randomly at the beginning of each trial. In Figure 4.2 we varied the number of unlabeled trajectories for our semi-supervised SSIRL method and compared it to IRL. Figure 4.2 shows the performance of the final mixture policy under the true reward (unknown to both algorithms) after 10 iterations[1] of both IRL and SSIRL. The results are averaged over 10 simulations[2]. The improvement levels off after 6 iterations. In Figure 4.3 we fixed the number of unlabeled trajectories to 10 and observe the performance under the true reward function in a single run as we iterate IRL and SSIRL. In this case, IRL converged to the expert's feature counts in few iterations. On the other hand, notice that SSIRL what was able in the begging discover even better mixture policy than the expert. We also show the performance of the optimal policy given the true reward (Figure 4.3, red squares). After the fifth iteration, performance SSIRL drops as the unlabeled trajectories are outweighed by the newly generated policies. As mentioned before, as we increase the number of iterations SSIRL converges to IRL. However, as Figure 4.3 shows, a better policy may be discovered much sooner.

1. Note again, that since number of unlabeled trajectories is finite, after many iterations both IRL and SSL converge to an approximately same policy.
2. The performance of IRL is obviously independent of the number of unlabeled trajectories. The small fluctuations in this and the following plots are due to averaging over different 10 simulations each time. This is to assure that the both algorithms are given the same (noisy) sample of the expert's feature counts.

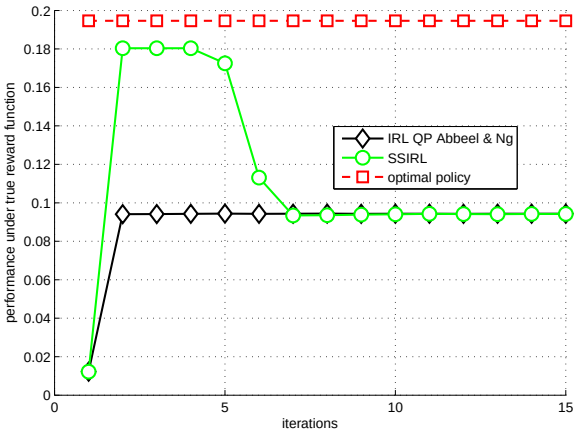


Figure 4.3: **Gridworld** - Performance of the mixture policies under the true reward (unknown to both algorithms). The true reward function and the reward function for unlabeled trajectories are chosen randomly and the graph shows the performance of the mixture policy. The performance of the optimal policy is shown by the dashed line with red squares.

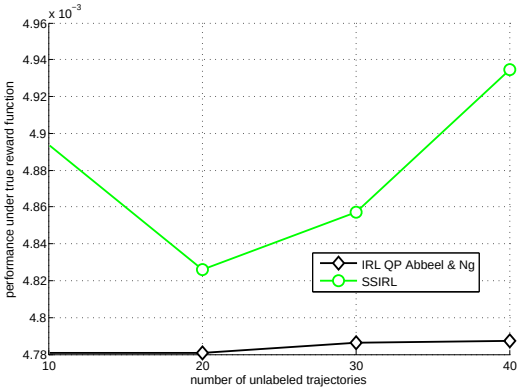


Figure 4.4: **Two goals navigation** - Performance after 5 iterations of both algorithms as a function of the number of extra unlabeled trajectories. First half of the unlabeled trajectories goes from start to “A", the other half goes to “B”.

The results for the both setups in *two goals navigation* domain are shown in Figures 4.4 and 4.5. In both cases we vary the number of the unlabeled trajectories given to SSIRL and observed how it affects the performance under the true reward. The figures display the performance of the mixture policies under the true reward after 5 iterations of both IRL and SSIRL. We can observe the improvement of the learned mixture policies in the case of SSIRL, which is typically greater when more unlabeled trajectories are provided.

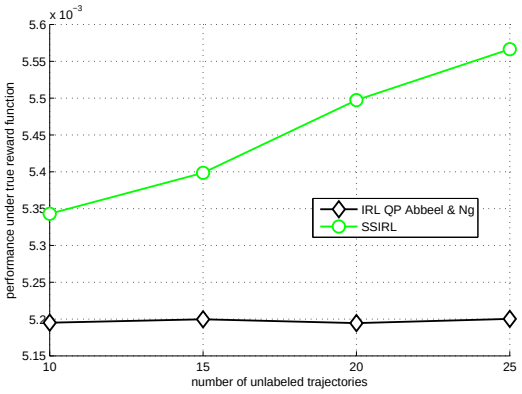


Figure 4.5: **Two goals navigation** - Performance after 5 iterations of both algorithms as a function of the number of extra unlabeled trajectories. First half of the unlabeled trajectories goes from start to "A", the other half goes the opposite direction.

5. Conclusion

We presented a semi-supervised IRL algorithm for apprenticeship learning. The algorithm combines the max-margin IRL algorithm of Abbeel and Ng [2004] and transductive SVMs [Joachims, 1999b]. Our SSIRL algorithm leverages the cluster assumption in the feature expectations space of the provided trajectories. To our best knowledge this is the first work that makes use of the unlabeled trajectories in the apprenticeship learning.

One of the limitation of the SSIRL is that the expected feature counts of provided trajectories should comply with the clustering assumption. Otherwise, including them in the optimization maybe not helpful at all, as shown by Singh et al. [2008]. Moreover, our method inherits the well known limitations of IRL [Abbeel and Ng, 2004], such as the assumption of the linearity for the reward in the feature space or only a mixture policy as the output of the algorithm. Finally, since the stopping criterion for IRL is defined as closeness to the expert trajectory in the feature space, other stopping criteria need to be used to recognize that the policy found by SSIRL is performing better.

In the future, we plan to enhance other methods with unlabeled trajectories, such as max-margin planning or maximum entropy inverse reinforcement learning. Moreover, we plan to investigate leveraging manifold assumption, which is also well studied in the semi-supervised learning [Zhu et al., 2003].

Acknowledgments

This research work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the "contrat de projets état region 2007–2013", and by PASCAL2 European Network of Excellence. The research leading to these results has also received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 270327 (project CompLACS).

References

Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st international conference on machine learning*, 2004.

Kristin Bennett and Ayhan Demiriz. Semi-Supervised Support Vector Machines. In *Advances in Neural Information Processing Systems 11*, pages 368–374, 1999.

Abdeslam Boularias, Jens Kober, and Jan Peters. Model-free inverse reinforcement learning. In *Proceedings of Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined Convex Programming and CVX. *Review Literature And Arts Of The Americas*, C(3):1–26, 2006.

Thorsten Joachims. Making large-Scale SVM Learning Practical. *Advances in Kernel Methods Support Vector Learning*, pages 169–184, 1999a.

Thorsten Joachims. Transductive Inference for Text Classification using Support Vector Machines. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, San Francisco, CA, USA, 1999b.

Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. In *NIPS*, pages 1–9, 2011.

Andrew Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Inverted Autonomous Helicopter Flight via Reinforcement Learning. In *International Symposium on Experimental Robotics*, 2004.

Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In Jorge Pinho De Sousa, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. *Proceedings of the 23rd ICML*, 3(10), 2006.

Aarti Singh, Robert D Nowak, and Xiaojin Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in Neural Information Processing Systems 21*, 2008.

Vladimir N Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

Xiaojin Zhu. Semi-Supervised Learning Literature Survey. Technical Report 1530, University of Wisconsin-Madison, 2008.

Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *Proceedings of the 20th International Conference on Machine Learning*, pages 912–919, 2003.

Brian Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. *Proc AAAI*, 2008.

JMLR: Workshop and Conference Proceedings 24:143–153, 2012 10th European Workshop on Reinforcement Learning

An investigation of imitation learning algorithms for structured prediction

Andreas Vlachos ANDREAS.VLACHOS@CL.CAM.AC.UK
Computer Laboratory, University of Cambridge, UK.

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

In the imitation learning paradigm algorithms learn from expert demonstrations in order to become able to accomplish a particular task. Daumé III et al. [2009] framed structured prediction in this paradigm and developed the search-based structured prediction algorithm (SEARN) which has been applied successfully to various natural language processing tasks with state-of-the-art performance. Recently, Ross et al. [2011] proposed the dataset aggregation algorithm (DAGGER) and compared it with SEARN in sequential prediction tasks. In this paper, we compare these two algorithms in the context of a more complex structured prediction task, namely biomedical event extraction. We demonstrate that DAGGER has more stable performance and faster learning than SEARN, and that these advantages are more pronounced in the parameter-free versions of the algorithms.

Keywords: Real-world Applications, Imitation Learning, Natural Language Processing, Structured Prediction.

1. Introduction

Imitation learning algorithms aim at learning controllers from demonstrations by human experts [Schaal, 1999; Abbeel, 2008; Syed, 2010]. Unlike standard reinforcement learning algorithms [Sutton and Barto, 1996], they do not require the specification of a reward function by the practitioner. Instead, the algorithm observes a human expert perform a series of actions to accomplish the task in question and learns a policy that "imitates" the expert with the purpose of generalizing to unseen data. These actions have dependencies between them, since earlier ones affect the input to the following ones and the algorithm needs to handle the discrepancy between the actions of the expert in the demonstration during training and the actions predicted by the learned controller during testing. Imitation learning algorithms have been applied successfully to a variety of domains and tasks including autonomous helicopter flight [Coates et al., 2008] and statistical dialog management [Syed and Schapire, 2007].

For structured prediction tasks in natural language processing (NLP) the output space of an instance is a structured group of labels [Smith, 2011]. For example, in part-of-speech tagging the output for a sentence is a sequence of part-of-speech tags, or in handwriting recognition the output is a sequence of characters. Structures can often be more complex than sequences, for example in syntactic dependency parsing the output space for a sentence is a set of labeled edges spanning the words. While learning methods such as Conditional Random Fields [Lafferty et al., 2001] and Markov Logic Networks [Domingos and Lowd,

2009] have been developed for structured prediction, they are usually tailored to particular structures so that they can perform parameter learning and inference efficiently. As a result, a common approach that performs well in practice is to decompose structured prediction into multiple classification tasks, in which each label of the structured output is predicted by a classifier.

Under this prism, learning for structured prediction can be viewed as learning a controller whose actions are to output each of the labels of the structured output. Similar to the controllers in reinforcement learning, these actions have dependencies between them (e.g. in part-of-speech tagging, determiners are commonly followed by nouns instead of verbs, or in handwriting recognition some character sequences are more likely than others) which must be taken into account in order to achieve good performance. The training signal commonly provided is a set of labeled instances produced by a human expert, which is akin to the human demonstrations in the imitation learning paradigm. Daumé III et al. [2009] proposed an imitation learning algorithm[1], search-based structured prediction (SEARN), that reduces the problem of learning a model for structured prediction into learning a set of classifiers. This reduction enables SEARN to tackle structured prediction tasks with complex output spaces, and it has been applied successfully to a variety of tasks including summarization [Daumé III et al., 2009] and biomedical event extraction [Vlachos and Craven, 2011].

In this work, we investigate a novel imitation learning algorithm proposed by Ross et al. [2011], dataset aggregation (DAGGER) that also reduces the problem of learning structured prediction to classification learning. It was compared to SEARN on learning video game-playing agents and handwriting recognition and was shown to be more stable and have faster learning while achieving state-of-the-art performance.

In this paper we make the following contributions. We present SEARN and DAGGER in a unified description, highlighting the connections between imitation learning and structured prediction. We then compare them in the context of biomedical event extraction [Kim et al., 2011], a task in which the structure of the output is more complex than a sequence of labels, and confirm the aforementioned advantages of DAGGER over SEARN which are more pronounced in the parameter-free versions of the algorithms. Furthermore, we explore the effect of the learning rate on the balance between precision and recall achieved by the algorithms. We believe that these contributions are relevant to applications of imitation learning algorithms to other structured prediction tasks, as well as to the development and evaluation of imitation learning algorithms.

2. Imitation learning algorithms for structured prediction

SEARN and DAGGER form the structured output prediction of an instance s as a sequence of T actions $\hat{y}_{1:T}$ made by a learned policy H. Each action $\hat{y}_t$ can use features from s and all previous actions $\hat{y}_{1:t-1}$, thus exploiting possible dependencies. The number of actions taken for an instance is not defined in advance but it depends on the actions chosen.

Algorithm 1 presents the training procedure for SEARN and DAGGER. Both algorithms require a set of labeled training instances $\mathcal{S}$ and a loss function ℓ that compares structured

1. SEARN infers rewards using the loss function and the labeled instances, therefore it is better described as an apprenticeship learning algorithm [Syed, 2010]. Note though that this distinction between imitation and apprenticeship learning is not consistent among authors [Abbeel, 2008].

Algorithm 1: Imitation learning training

Input: training data $\mathcal{S}$, *expert policy* $\pi^\star$, loss function ℓ, learning rate β, CSC learner *CSCL*
Output: policy H_N

1 Examples $E = \emptyset$
2 **for** $i = 1$ **to** N **do**
3 $p = (1-\beta)^{i-1}$
4 current policy $\pi = p\pi^\star + (1-p)H_{i-1}$
5 **if** SEARN **then**
6 Examples $E = \emptyset$
7 **for** s ***in*** S **do**
8 Predict $\pi(s) = \hat{y}_{1:T}$
9 **for** $\hat{y}_t$ ***in*** $\pi(s)$ **do**
10 Extract features $\Phi_t = f(s, \hat{y}_{1:t-1})$
11 **foreach** *possible action* y_t^j **do**
12 **if** SEARN **then**
13 Predict $y\prime_{t+1:T} = \pi(s|\hat{y}_{1:t-1}, y_t^j)$
14 **else**
15 Predict $y\prime_{t+1:T} = \pi^\star(s|\hat{y}_{1:t-1}, y_t^j)$
16 Estimate $c_t^j = \ell(\hat{y}_{1:t-1}, y_t^j, y\prime_{t+1:T})$
17 Add (Φ_t, c_t) to E
18 Learn a policy $h_i = CSCL(E)$
19 **if** SEARN **then**
20 $H_i = \beta \sum_{j=1}^{i} \frac{(1-\beta)^{i-j}}{1-(1-\beta)^i} h_j$
21 **else**
22 $H_i = h_i$

output predictions of instances in $\mathcal{S}$ against the correct output for them. In addition, an *expert policy* $\pi^\star$ must be specified which is a function that returns the optimal action $\hat{y}_t$ for the instances in the training data, which is akin to an expert demonstrating the task. An action is optimal when it minimizes the loss over the instance given the previous actions $\hat{y}_{1:t-1}$ assuming that all future actions $\hat{y}_{t+1:T}$ are also optimal. $\pi^\star$ is typically derived from the labeled training instances (e.g. in handwriting recognition $\pi^\star$ returns the correct character for each position) and it must be able to deal with mistaken $\hat{y}_{1:t-1}$. Both algorithms output a learned policy H that is a classifier, which unlike the expert policy $\pi^\star$, it can generalize to unseen data. Finally, the learning rate β and a cost sensitive classification (CSC) learner (*CSCL*) must be provided. In CSC each training instance has a vector of misclassification costs associated with it, thus rendering some mistakes on some instances to be more expensive than others [Domingos, 1999].

Each training iteration of both algorithms begins by setting the probability p (line 3) of using $\pi^\star$ in the current policy π. In the first iteration only $\pi^\star$ is used but in later iterations

π becomes stochastic as for each action we use $\pi^\star$ with probability p and the learned policy from the previous iteration h_{i-1} with probability $1-p$ (line 4). Then π is used to predict each instance s in the training data (line 8). For each s and each action $\hat{y}_t$, a CSC example is generated (lines 10-17). The features Φ_t are extracted from s and the previous actions $\hat{y}_{1:t-1}$ (line 10) and are desinged to be good predictors of the current action $\hat{y}_t$. For example, in part-of-speech tagging, commonly used features include the word whose part of speech label we are predicting, the words preceding it and following it, as well as the label predicted for the previous word. The cost for each possible action y_t^j is estimated by predicting the remaining actions $y\prime_{t+1:T}$ in s using either π or $\pi^\star$ (line 13 or 15) and calculating the loss incurred given that action w.r.t. the correct output using ℓ (line 16). The features for each timestep together with the costs for each possible action at that timestep (Φ_t, c_t) form one CSC example (line 17). The CSC examples obtained from all the training instances are used by a CSC learning algorithm to learn a policy h_i (line 18) which is combined with the previously learned ones to form the new policy H_i.

In each iteration, the algorithm predicts the instances in the training data and estimates the cost of each action. This procedure is commonly referred to as inverse reinforcement learning [Abbeel and Ng, 2004], since unlike standard reinforcement learning, an expert policy is given but we try to learn the reward for each action (cost). Note that the learned policy from the previous iteration is used in generating the CSC examples in predicting each training data instance (line), as well as estimating the cost for each action in the case of SEARN (line 13). The degree to which it is used depends on the probability p set in the beginning of each iteration. By gradually decreasing the use of the expert policy in the current policy, both algorithms adapt the learned policy to its own predictions.

The main algorithmic difference between SEARN and DAGGER is in the learning of the classifiers h_i in each iteration and in combining them into a policy H_i. Under SEARN, each h_i is learned using only the CSC examples generated in iteration i (line 6) and is then combined with the classifiers learned in the previous iterations $h_{1:i-1}$ according to the learning rate β (line 20). On the other hand, DAGGER learns h_i using CSC examples from iterations $1:i$ and uses it as the learned policy H_i (line 22). Thus DAGGER can combine the training signal obtained from all iterations more flexibly, which results in faster learning and more stable performance compared to SEARN.

Another difference between the two algorithms is that DAGGER uses the expert policy $\pi^\star$ to predict the remaining actions in $y\prime_{t+1:T}$.[2] This approach to costing had been proposed by Daumé III et al. [2009] in the context of SEARN, referred to as optimal approximation.

The learning rate β determines how fast the current policy π moves away from $\pi^\star$. A special case is obtained when $\beta = 1$, also referred to as pure policy iteration or parameter-free. In this case, $\pi^\star$ is used only in the first iteration to reproduce the correct output and π only uses only the learned policy from the previous iteration H_{i-1}. Furthermore, the classifier combination under SEARN becomes the same as the one of DAGGER, i.e. only the most recently learned classifier is used. In this setting the algorithms cannot query $\pi^\star$ after the first iteration, thus $\pi^\star$ does not need to handle mistakes in previous actions since all actions are optimal in the first iteration. Furthermore, the predictions in lines 8 and 12 become deterministic. However, relying only on the learned policy for action prediction

2. This fact was pointed out by a reviewer as it was not mentioned by Ross et al. [2011].

beyond the first iteration (note that the cost estimation still uses the correct output via the loss function) renders the learning harder as the algorithms are given less supervision.

3. Tackling biomedical event extraction with imitation learning

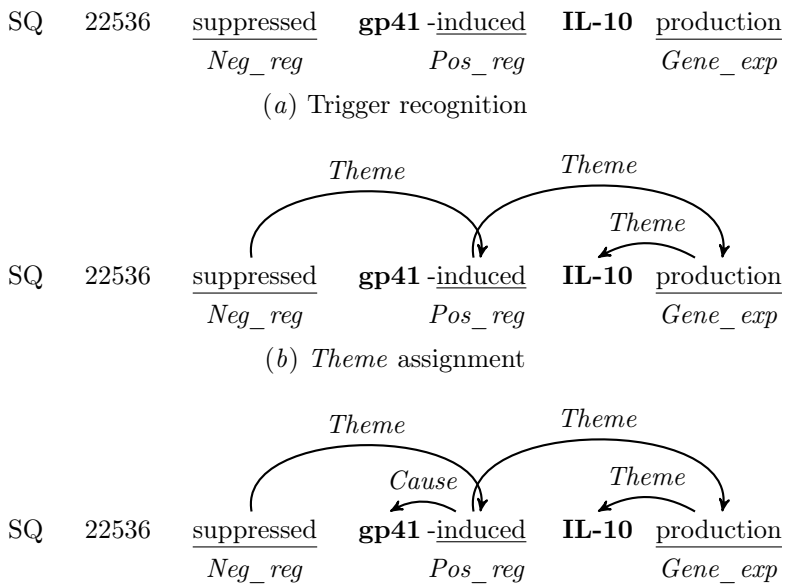
Vlachos and Craven [2012] developed a biomedical event extraction approach trained with Searn that achieved the second-best reported performance on the data from the recent BioNLP 2011 shared task (BioNLP11ST) [Kim et al., 2011]. Therefore, we decided to use the BioNLP11ST setup to compare empirically Searn and DAgger. In this section we describe briefly the task and the approach, but the interested reader is referred to the article by Vlachos and Craven [2012] for more details.

The term biomedical event extraction is used to refer to the task of extracting descriptions of actions and relations among one or more entities from the biomedical literature. In BioNLP11ST each event consists of a trigger and one or more arguments, the latter being proteins or other events. Protein names are annotated in advance and any token in a sentence can be a trigger for one of the nine event types. Depending on their event types, triggers are assigned theme and cause arguments. In an example demonstrating the complexity of the task, given the passage "...SQ 22536 suppressed gp41-induced IL-10 production in monocytes", systems should extract the three appropriately nested events listed in Figure 3.1(*d*). Performance is measured using Recall, Precision and F-score over complete events, i.e. the trigger, the event type and the arguments all must be correct in order to obtain a true positive.

In our approach, we treat each sentence independently and decompose event extraction in four stages: trigger recognition, theme assignment, cause assignment and event construction (Fig. 3.1).[3] Apart from the last one which is rule-based, each stage has its own module to perform the classification needed. The basic features used are extracted from the lemmatization and the syntactic parse of the sentence. Furthermore, we extract structural features for each action from the previous ones, for example the trigger recognition label of the previous token is used as a feature to predict the label for the current token.

The loss function sums the number of false positive and false negative events, following the task evaluation. The expert policy for a sentence is derived from the correct events contained in the training data and returns the action that minimizes the loss over the sentence given the previous actions and assuming that all future actions are optimal. In the first iteration it returns the actions required to reproduce the correct events in the sentence. In subsequent iterations, in order to deal with mistakes in previous actions, it avoids assigning arguments to incorrectly tagged triggers and avoids using incorrect events as arguments of other events. It it important to note that the same events can be obtained using different sequences of actions and that the correct events in the training data specify only one such sequence. For example in Figure 3.1, token "SQ" could have been tagged as a trigger without assigning any arguments to it and we ould still obtain the correct events. In order to resolve this ambiguity, we restrict the expert policy to return only the the triggers that are necessary to produce the correct events.

3. While different task decompositions are possible, this 4-stage decomposition is the most commonly used one among the systems participating in BioNLP11ST.



(*a*) Trigger recognition

(*b*) *Theme* assignment

(*c*) *Cause* assignment

ID	type	*Trigger*	*Theme*	*Cause*
E1	*Neg_reg*	suppressed	*E2*	
E2	*Pos_reg*	induced	*E3*	gp41
E3	*Gene_exp*	production	IL-10	

(*d*) Event construction

Figure 3.1: The stages of our biomedical event extraction system.

The classifiers learned for trigger recognition, theme assignment and cause assignment, are unlikely to be able to replicate to the expert policy due to the difficulty of the tasks. In such cases, the optimal approximation method for costing (line 15 in Alg. 1) is unlikely to estimate the cost of each action correctly, since the actions predicted by the learned policy are likely to differ from the ones returned by the expert policy. Therefore, in our experiments we use the SEARN-style cost estimation (line 13 in Alg. 1) for both SEARN and DAGGER. In order to restrict the effects of the stochasticity of this approach, we use the *focused costing* method [Vlachos and Craven, 2011], in which the cost estimation for an action takes into account only the part of the output graph connected with that action, thus limiting the part of the structured output considered for this purpose.

The structure of the output space for the biomedical event extraction decomposition described above is a sequence of tags defining triggers and their event types, combined with a directed acyclic graph in which vertices correspond to triggers or proteins and edges represent argument assignments. Thus it is more complex than the handwriting recognition task that was used in the comparison performed by Ross et al. [2011] which is a sequential tagging task. Also, it is not amenable to several commonly used structured prediction methods which rely on the output structure being a sequence or a tree in order to perform inference efficiently. Furthermore, incorrect actions can prohibit other correct actions from

being taken, e.g. if a token is incorrectly predicted not to be a trigger, it is impossible to assign arguments to it. Again, this is unlike sequential tagging tasks, where an incorrect prediction of a token does not prohibit the correct prediction of the remaining ones. Both algorithms under comparison use the learned policies from previous iterations in order to generate training examples (line 9 in Alg 1), therefore incorrectly predicted actions can inhibit the algorithm from reaching regions of the output space that could provide useful CSC examples. Thus, the learning rate which determines how frequently the expert policy is queried is likely to be more important for biomedical event extraction than it was in the comparisons of Ross et al. [2011].

4. Experiments

In our experiments we run Searn and DAgger for 12 training iterations and perform CSC learning using the online passive-aggressive (PA) algorithm [Crammer et al., 2006]. BioNLP11ST comprises three datasets – training, development and test – which consist of five full articles each and 800, 150 and 260 abstracts respectively. We extract features from the output of the syntactic parser by McClosky [2010] as provided by the shared task organizers [Stenetorp et al., 2011]. The use of this publicly available resource allows for easy replication of our experiments. While the correct output is provided for the training and development datasets, evaluation on the test dataset is only possible once per day via a webserver in order to maintain the fairness of comparisons between systems. However, since we focus on comparing the algorithms in terms of stability and learning speed, we report results on the development set.

Initially we compare Searn and DAgger with learning rate equal to one. Figure 4.1(a) shows that the performance of DAgger peaks after 6 iterations beyond which it remains stable. On the other hand, the performance of Searn oscillates between high recall/low precision and low recall/high precision iterations (Figures 4.1(b) and 4.1(c)). In particular, in the first iteration Searn learns theme and cause assignment components given correctly identified triggers only (the expert policy only returns those), but the trigger recognition component learned returns many incorrect ones, thus resulting in high recall and low precision. This behaviour is reversed in the second iteration, in which the theme and cause assignment components are learned so that they can accommodate for incorrectly recognized triggers, but at the same time the trigger recognition component becomes extremely conservative. This pattern holds for subsequent iterations, albeit progressively less pronounced. In contrast, DAgger can combine training signal from both iterations, thus its performance improves faster and avoids such oscillating behaviour. The unstable behaviour of Searn affects the training time as well, since in the high recall/low precision iterations the algorithm needs to consider many more actions during the cost-sensitive example generation steps (lines 8-17 in Alg. 1).

Figure 4.1(b) shows a substantial drop in recall for both algorithms in the second iteration. This is due to the learned policy in the first iteration being unable to replicate the correct output without using the expert policy. This issue did not emerge in the experiments of Ross et al. [2011], but as explained in Section 3, event extraction is likely to be affected by it. Using slower learning rates ameliorates this problem (Figure 4.1(e)) and renders Searn

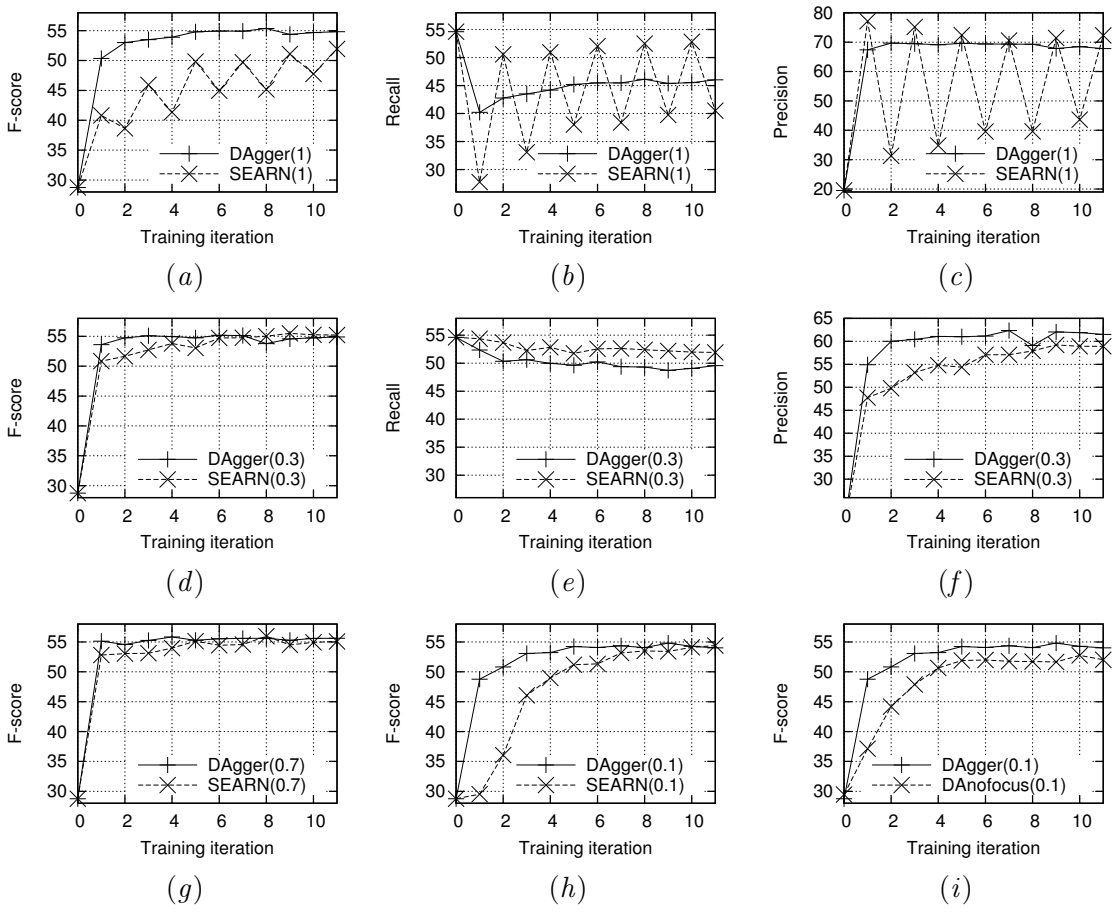


Figure 4.1: Development dataset results for DAGGER(β) and SEARN(β) with various learning rates.

more stable, but DAGGER still learns faster for a range of learning rates (0.7, 0.3 and 0.1 in Figures 4.1(d), 4.1(g) and 4.1(h) respectively).

Even though slower learning rates improve the performance for both SEARN and DAGGER, the improvement for the latter is not as dramatic (about 1.5 points in F-score). As discussed in Sec. 2, when $\beta < 1$ action costing becomes stochastic, which can result in unreliable estimates. In our experiments we used the *focused costing* approach proposed by Vlachos and Craven [2011], who reported that it improved the performance of SEARN by 4 F-score points. In Figure 4.1(i) we compared the performance of DAGGER with and without focused costing and we show that even though focused costing results in faster learning, the difference in terms of F-score is smaller, approximately 2 points. In other words, DAGGER is more robust w.r.t. the choice of action costing method. Also note that using DAGGER with $\beta = 1$ avoids introducing stochasticity in action costing, while unlike SEARN it remains stable, thus it is more likely to be applicable to other structured prediction tasks.

5. Conclusions - Future work

In this paper we compared two imitation learning algorithms for structured prediction, Searn and DAgger. We presented them in a unified description and evaluated them on biomedical event extraction. We found that DAgger is more stable and learns faster, while being more robust with respect to the choice of learning rates and action costing. These advantages are more pronounced in the parameter-free versions of the algorithms which avoid stochastic cost estimates and need simpler expert policy definitions. Finally, we assessed the effect of the learning rate in complex structured prediction tasks in which mistaken predictions can inhibit imitation learning algorithms from exploring useful parts of the training data. Our contributions should be relevant to applications of imitation learning to other structured prediction tasks.

In future work, we will apply imitation learning algorithms to other complex structured prediction tasks. Furthermore, we would like to explore and compare against other structured prediction frameworks such as structured prediction cascades [Weiss and Taskar, 2010] and output space search [Doppa et al., 2012] that also rely on reductions of structured prediction learning to simpler problems.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SpaceBook project www.spacebook-project.eu).

References

Pieter Abbeel. *Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control*. PhD thesis, Department of Computer Science, Stanford University, 2008.

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, pages 46–53, 2004.

Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine learning*, pages 144–151, 2008.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75:297–325, 2009.

Pedro Domingos. Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 155–164. Association for Computing Machinery, 1999.

Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 1st edition, 2009. ISBN 1598296922, 9781598296921.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Output space search for structured prediction. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.

Jin-Dong Kim, Yue Wang, Toshihisa Takagi, and Akinori Yonezawa. Overview of the Genia Event task in BioNLP Shared Task 2011. In *Proceedings of the BioNLP 2011 Workshop Companion Volume for Shared Task*, pages 7–15, 2011.

John D. Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of 18th International Conference in Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc., 2001.

David McClosky. *Any domain parsing: Automatic domain adaptation for natural language parsing*. PhD thesis, Department of Computer Science, Brown University, 2010.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *14th International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, June 1999.

Noah A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May 2011.

Pontus Stenetorp, Goran Topić, Sampo Pyysalo, Tomoko Ohta, Jin-Dong Kim, and Jun'ichi Tsujii. BioNLP Shared Task 2011: Supporting Resources. In *Proceedings of the BioNLP 2011 Workshop Companion Volume for Shared Task*, 2011.

Richard Sutton and Andrew Barto. *Reinforcement learning: An introduction*. MIT press, 1996.

Umar Syed. *Reinforcement learning without rewards*. PhD thesis, Department of Computer Science, Princeton University, 2010.

Umar Syed and Robert E. Schapire. Imitation learning with a value-based prior. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 384–391, 2007.

Andreas Vlachos and Mark Craven. Search-based structured prediction applied to biomedical event extraction. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 49–57. Association for Computational Linguistics, 2011.

Andreas Vlachos and Mark Craven. Biomedical event extraction from abstracts and full papers using search-based structured prediction. *BMC Bioinformatics*, 13(suppl. 11):S5, 2012.

David Weiss and Ben Taskar. Structured prediction cascades. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

JMLR: Workshop and Conference Proceedings 24:155–166, 2012 10th European Workshop on Reinforcement Learning

Rollout-based Game-tree Search Outprunes Traditional Alpha-beta

Ari Weinstein AWEINST@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Sergiu Goschin SGOSCHIN@CS.RUTGERS.EDU
*

Editor: Marc Peter Deisenroth, Csaba Szepesvári, Jan Peters

Abstract

Recently, rollout-based planning and search methods have emerged as an alternative to traditional tree-search methods. The fundamental operation in rollout-based tree search is the generation of trajectories in the search tree from root to leaf. Game-playing programs based on Monte-Carlo rollouts methods such as "UCT" have proven remarkably effective at using information from trajectories to make state-of-the-art decisions at the root. In this paper, we show that trajectories can be used to prune more aggressively than classical alpha-beta search. We modify a rollout-based method, FSSS, to allow for use in game-tree search and show it outprunes alpha-beta both empirically and formally.

1. Introduction

Rollout-based planning is an approach that has been highly successful in game-tree search. This class of methods attempts to build estimates of the quality of available actions by sampling values from the reward and transition functions of a generative model, repeatedly beginning execution at some start state and ending at a terminal state. UCT [Kocsis and Szepesvári, 2006] is an example of this method that was initially designed for planning in standard stochastic Markov decision processes (MDPs), but was later extended to find approximate solutions in minimax trees. Since its introduction, UCT has seen significant success in the game of Go, and was the first computer agent to achieve master level play in the 9x9 variant [Gelly and Silver, 2008].

While UCT does produce state-of-the-art results in Go, it has been proven to perform very poorly when searching trees with particular properties, due to highly aggressive pruning [Coquelin and Munos, 2007; Walsh et al., 2010]. Likewise, UCT has not been successful in chess, a result attributed to poor performance of the algorithm in the presence of "search traps," which are common in chess but not in Go [Ramanujan et al., 2011].

Exact minimax search algorithms do not suffer from this behavior. Alpha-beta [Edwards and Hart, 1961] is one such branch-and-bound algorithm which computes the exact minimax value of a search tree. While it has been shown that alpha-beta is optimal in terms of the number of leaves explored [Pearl, 1982], the proof holds only for directional, or depth-first algorithms.

* This research was supported in part by National Science Foundation DGE-0549115.

One of the earliest best-first algorithms provably better than alpha-beta in terms of state expansions was the *best-first* SSS* [Stockman, 1979]. This algorithm, however, requires exponential time operations. Framing the algorithm differently resolved this issue: MTD($+\infty$) performs state expansions in the same order as SSS*, but is computationally tractable [Plaat et al., 1996].

In this paper, forward-search sparse sampling [Walsh et al., 2010], or FSSS, is extended for use in minimax search, resulting in the algorithm FSSS-Minimax. In its original formulation for MDPs, FSSS conducts rollouts and prunes subtrees that cannot contribute to the optimal solution. Additionally, it does so in a principled manner that allows it to avoid the previously mentioned problems of UCT. In the setting of game-tree search, we will prove that FSSS-Minimax dominates (is guaranteed to expand a subset of the states expanded by) alpha-beta, while still computing the correct value at the root. Even though the guarantees of FSSS-Minimax, SSS*, and MTD($+\infty$) are the same with respect to alpha-beta, there are important distinctions. SSS* and MTD($+\infty$) are two different approaches that produce identical policies, while FSSS-Minimax follows a different policy in terms of state expansions.

Next, the minimax setting where planning takes place, as well as the original formulation of FSSS, is discussed. From there, the extension of FSSS to minimax trees is presented. After its introduction, FSSS-Minimax is empirically compared to alpha-beta and MTD($+\infty$)Ȧ formal comparison of the two algorithms follows, proving FSSS-Minimax expands a subset of states expanded by alpha-beta. The discussion section addresses shortcomings of FSSS, and proposes future work.

2. Setting

Search is performed on a tree $G = \langle S, A, R, T \rangle$. This tuple is composed of a set of states S, a set of actions A, a reward function R, and a transition function T. All functions in G are deterministic. Each state s has a value $V(s)$, which is conditioned on the rules dictated by the three types of states in the tree. The state types are: max where the agent acts, min where the opponent chooses, and leaves where no further transitions are allowed. The goal of a max state s is to select actions that maximize $V(s)$ by choosing a path to a leaf l that maximizes $R(l)$, and min states must minimize this value. The value of the tree, $V(G)$ is equal to the value of the root of G, $V(G.root)$

The following functions over states $s \in S$ are defined: $\min(s)$, $\max(s)$, and $\text{leaf}(s)$ which return Boolean values corresponding to membership of the state to a class of states. Because it simplifies some of the descriptions, min and max states have a field defined as f. For a max state s, $s.f = \max$, and likewise for min states.

The transition function, T, is defined only for min and max states, and actions $a \in A$, where $T(s, a) \rightarrow s' \in S$. For leaves, T is undefined. The reward function $R(s) \rightarrow \mathbb{R}$ is defined only for leaves.

2.1. FSSS-Expectimax

The original formulation of FSSS, which we call FSSS-Expectimax, is designed for searching a tree that consists of expect (an additional type of state where a stochastic transition occurs) and max states [Walsh et al., 2010]. Like all FSSS algorithms, FSSS-Expectimax

explores the tree while conducting rollouts from the root to a leaf, maintaining upper and lower bounds (U and L) on the value of each state in the tree. The ultimate goal is to have the the lower bound L and upper bounds U meet at the root. Rollouts in FSSS-Expectimax proceed by checking the function to choose a next state. In max states, the child is the one with the highest upper bound. In expect states, the outcome which contributes most to the difference between L and U is chosen. Previously published results show that this approach computes the correct value of the tree in finite time.

After the rollout, information from the expanded leaf is propagated up in the following manner: when a leaf is expanded, its upper and lower bounds are set to its reward. From there, for a max parent s, and updated child s', if $L(s')$ or $U(s')$ are maximal values amongst its siblings, the bounds of the parent are updated. For an expectation parent s, L and U are updated based on the weighted averages of value over all observed children. This process continues up the tree until either an update fails to modify a state value, or $L(G.root)$ and $U(G.root)$ have been updated, at which point a new rollout begins.

3. FSSS-Minimax

The most commonly used method to solve minimax trees is the alpha-beta algorithm, which prunes subtrees of the currently visited state s, based on whether further information from s can contribute to the solution. It does so by maintaining variables α and β, which correspond to the range of values at the current state that could impact the value at the root, based on the expansions done to that point. Alpha-beta search is described in Algorithm 1, but is written to highlight its similarities to Algorithm 2 (our main contribution) so storage of V is not technically necessary.

When exploring a state where the inequality $\alpha < \beta$ is not maintained, search is terminated from that state, as any further information from the subtree rooted at that state could not possibly influence $V(G)$. This simple rule is quite effective, and there are cases where the algorithm visits the minimum number of possible states [Knuth and Moore, 1975]. As mentioned, however, alpha-beta's optimality in terms of pruning is only over the class of depth-first search algorithms [Pearl, 1982]. Because rewards only exist in the leaves the setting described in this paper, we define performance in terms of the number of leaves expanded. The algorithm introduced next is a best-first algorithm that never expands more leaves than alpha-beta.

In Algorithm 2, we introduce the extension of FSSS to minimax trees, FSSS-Minimax. For the sake of brevity, beyond this point we will use FSSS to refer to FSSS-Minimax. The description that follows is for max states, but the behavior in min states is symmetric. Like alpha-beta, FSSS uses top-down variables α and β to guide pruning, but unlike alpha-beta, it also computes lower and upper bounds on the minimax value bottom up, which are maintained in the variables L and U which are maintained and updated during all rollouts on the game tree. Children effectively have L and U clipped by the α and β values of the parent. The bounds are constrained in this manner because, just like in alpha-beta, values outside the range of (α, β) cannot influence the value at the root.

After a child is identified to be visited, the α and β bounds must be adjusted based on the bounds of its siblings to avoid unnecessary calculation further down in the tree. In alpha-beta, α'—the α value used at a child state—is set to be the maximum value amongst

Algorithm 1: Alpha-Beta

begin `Solve`(G)
1 | **return** `Search` $(G.root, -\infty, \infty)$
end
begin `Search`$(s,\ \alpha,\ \beta)$
2 | **if** *leaf*(s) **then**
3 | | $V(s) \leftarrow R(s)$
4 | | **return**
| **end**
5 | $\alpha' \leftarrow \alpha$
6 | $\beta' \leftarrow \beta$
7 | **for** $i^* \leftarrow 1...|A|$ **do**
8 | | **if** $\max(s)$ **then**
9 | | | $v \leftarrow \max_{i<i^*} V(T(s, a_i))$
10 | | | $\alpha' \leftarrow \max(\alpha, v)$
| | **end**
11 | | **if** $\min(s)$ **then**
12 | | | $v \leftarrow \min_{i<i^*} V(T(s, a_i))$
13 | | | $\beta' \leftarrow \min(\beta, v)$
| | **end**
14 | | **if** $\alpha' \geq \beta'$ **then**
15 | | | **break**
| | **end**
16 | | `Search` $(T(s, a_{i^*}), \alpha', \beta')$
| **end**
17 | $V(s) \leftarrow (s.f)_i V(T(s, a_i))$
end

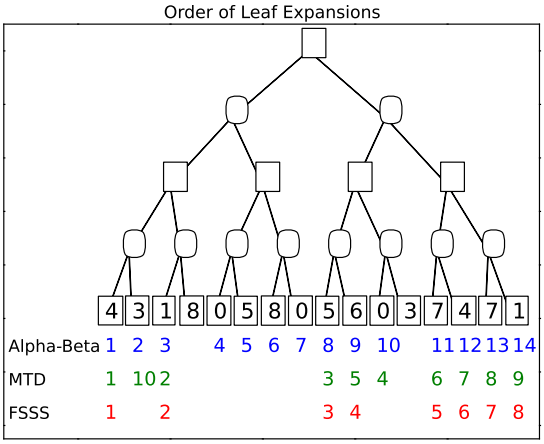


Figure 3.1: Order of expansions performed by alpha-beta, MTD($+\infty$), and FSSS. Rewards are rendered in the leaves. Max states are square, min states are circular.

α and the siblings that have been searched (those to its left). Because FSSS has partial information about all siblings, α' is set to the maximum of α and the *upper bounds* of the siblings of the child to be visited.

The modification of α' and β' by ϵ (which must be a value smaller than the possible differences in leaf values) on lines 18 and 24 of Algorithm 2, has no direct parallel in alpha-beta. As will be proved later, there is always an unexpanded leaf at the end of the trajectory followed by FSSS. For this to hold, it is necessary to have both a non-point range of possible values, as well as overlap between the ranges (α, β) and (L, U). Therefore, if $\alpha = U$, or $\beta = L$, the range of values considered becomes a point value of the state, making it look prematurely closed. Because of this, slightly incrementing β (or decrementing α) allows FSSS to still consider a range of values, without impacting the correctness.

In Figure 3.1, an illustration of the leaf expansions performed by alpha-beta, MTD($+\infty$) and FSSS for a small tree is presented. Leaves have integer rewards as labeled, and actions are ordered left to right. In this small example, alpha-beta expands 14 leaves, MTD($+\infty$) expands 10, and FSSS expands 8. The minimax value is 4, which comes from the leaf third from the right. As a depth-first method, alpha-beta can only choose to ignore leaves, but must proceed strictly from left to right. MTD($+\infty$) and FSSS, as best-first methods, can explore the leaves "out of order," which helps to grant them more effective pruning. (The numbers beneath the leaves denote the expansion order of the leaves for each algorithm.)

This example demonstrates the difference between MTD($+\infty$)and FSSS, which occurs at the fourth leaf expansion. Because of the initialization of the null-window in MTD($+\infty$), the first iteration performs what is called a "max-expansion" of the tree. In a max-expansion, max states expand all their children, but min states only expand the first (leftmost) child. Therefore, in a tree of this height, MTD($+\infty$) will always perform the first 4 expansions in this manner independent of the values encountered at the leaves.

Algorithm 2: FSSS-Minimax

begin `Solve`(G)
1 | **while** $L(G.root) \neq U(G.root)$ **do**
2 | | `Search` $(G.root, -\infty, \infty)$
| | **end**
3 | | **return** $L(G.root)$
end
begin `Search`(s, α, β)
4 | | **if** *leaf*(s) **then**
5 | | | $L(s) \leftarrow U(s) \leftarrow R(s)$
6 | | | **return**
| | | **end**
7 | | i^*, α', $\beta' \leftarrow$ Traverse (s, α, β)
8 | | `Search` $(T(s, a_{i^*}), \alpha', \beta')$
9 | | $L(s) \leftarrow (s.f)_i\ L(T(s, a_i))$, $U(s) \leftarrow (s.f)_i\ U(T(s, a_i))$
end
begin `Traverse`(s, α, β)
10 | | **for** $i \leftarrow 1...|A|$ **do**
11 | | | $U'(s, a_i) \leftarrow \min(\beta, U(T(s, a_i)))$, $L'(s, a_i) \leftarrow \max(\alpha, L(T(s, a_i)))$
| | | **end**
12 | | $\alpha' \leftarrow \alpha$, $\beta' \leftarrow \beta$
13 | | **if** $\max(s)$ **then**
14 | | | $i^* \leftarrow \text{argmax}_i\ U'(s, a_i)$
15 | | | $v \leftarrow \max_{i \neq i^*} U'(s, a_i)$
16 | | | $\alpha' \leftarrow \max(\alpha, v)$
17 | | | **if** $\alpha' = U'(s, a_{i^*})$ **then**
18 | | | | $\alpha' \leftarrow \alpha' - \epsilon$
| | | | **end**
| | | **end**
19 | | **if** $\min(s)$ **then**
20 | | | $i^* \leftarrow \text{argmin}_i\ L'(s, a_i)$
21 | | | $v \leftarrow \min_{i \neq i^*} L'(s, a_i)$
22 | | | $\beta' \leftarrow \min(\beta, v)$
23 | | | **if** $\beta' = L'(s, a_{i^*})$ **then**
24 | | | | $\beta' \leftarrow \beta' + \epsilon$
| | | | **end**
| | | **end**
25 | | **return** i^*, α', β'
end

FSSS, on the other hand, is only forced to perform the first 3 expansions regardless of the observed rewards. In all cases, the algorithm will also traverse to the grandparent of the third leaf to be expanded on the following trajectory. From that state, FSSS traverses based on the results of the first two expansions. Here, because neither had a reward greater than 5, FSSS need not expand the fourth leaf expanded by MTD($+\infty$).

4. Empirical Comparison

In this section, we compare alpha-beta, MTD($+\infty$), and FSSS in two classes of synthetic game trees. All trees have binary decisions and randomly selected rewards in the leaves. Because rewards only exist in the leaf states, the metric we report is the number of leaf states expanded, and in the experiments the number of leaves (and therefore the height of the trees) varies. Children of a state are initially explored in fixed left-to-right order in all cases, and ties are broken in favor of the left child. The height h of the tree is counted starting at 0, which is just the root, so there are 2^h leaves in a tree.

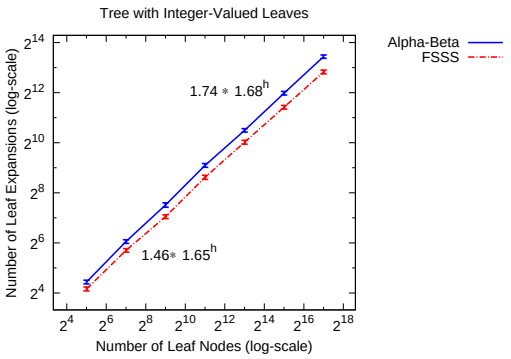
In the first experiment, rewards are integers sampled uniformly from roughly -2^{31} to 2^{31}. The results presented in Figure 4.1(a) show the number of leaves expanded by alpha-beta and FSSS when searching trees of a varying size. Note that both axes are log-scale, so the slight difference in the slope of the two lines indicates a difference in the base determining exponential growth. Using the height h, generating fit curves to the data yielded the functions $1.74 \cdot 1.68^h$ for alpha-beta and $1.46 \cdot 1.65^h$ for FSSS, which fit within the error bounds of the empirical data at almost all points. In this case, the number of expansions of FSSS and MTD($+\infty$) are not statistically significantly different, so only the expansions of FSSS are rendered. Between FSSS and MTD($+\infty$), the algorithm that expands the fewest leaves varies from tree to tree, so neither algorithm dominates the other.

We performed a second experiment to provide a clearer picture of the relative strengths of FSSS and MTD($+\infty$) in the context of trees with binary (0 or 1) valued leaves. Specifically, we measured the proportion of such trees where FSSS or MTD($+\infty$) expanded strictly fewer leaves than alpha-beta. (Neither algorithm ever expands *more* leaves than alpha-beta.)

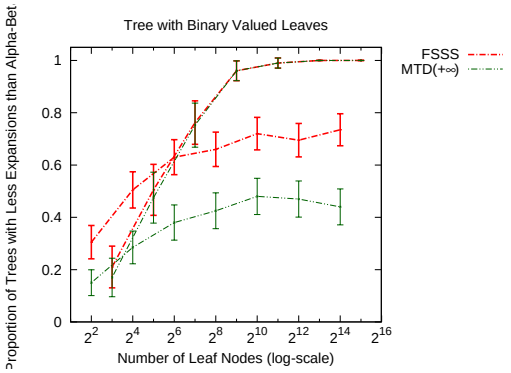
The results of this experiment are shown in Figure 4.1(b) (higher values indicate more pruning). Two things are worth mentioning. First, the results varied greatly based on the parity of the tree height—we plot performance for odd height and even height trees separately in the graph. A similar finding has been reported previously [Plaat et al., 1996]. Second, although the figure shows that FSSS expands less nodes than alpha-beta more often than MTD($+\infty$), in many cases the difference between the actual number of leaves expanded by FSSS and MTD($+\infty$)was extremely small even on the largest trees, partially because of the small range of leaf values.

5. Formal Comparison

The previous section showed that FSSS outprunes alpha-beta on a set of randomly generated game trees. This section proves a stronger result—FSSS will never expand a state that alpha-beta does not. Therefore, the total number of leaves expanded by FSSS is upper bounded by the expansions made by alpha-beta.



(a) Difference in number of leaf expansions between alpha-beta and FSSS.



(b) Fraction of trees where FSSS and MTD(+∞) expand strictly fewer states than Alpha-Beta.

Figure 4.1: Emprical comparison of FSSS and alpha-beta.

First, we argue that FSSS finds $V(G)$ and that it terminates because it never revisits an expanded leaf.

Theorem 1 (FSSS) *FSSS never attempts to enter a closed state, and its bounds are correct.*

This proof will be by induction. When execution begins from the root, if $L(G.root) = U(G.root)$, then calculation is complete. At each stage during a rollout, if s is a leaf, that leaf is expanded and the ranges are updated accordingly. Otherwise, in the rollout, $L(s), U(s)$ are the bounds on the value of state s, k is the branching factor ($k = |A|$), and $s_1, ..., s_k$ are the children of state s, reachable by corresponding actions $\{a_1, ..., a_k\} = A$. $L(s_1), ..., L(s_k)$ and $U(s_1), ..., U(s_k)$ are the lower and upper bounds on the children.

Recall that $L(s) = \max_i L(s_i)$, $U(s) = \max_i U(s_i)$ for max states and $L(s) = \min_i L(s_i)$, $U(s) = \min_i U(s_i)$ for min states. At the start of the rollout, 3 conditions hold:

1. $L(s) \neq U(s)$, because rollouts end when the bounds at the root match.
2. $L(s) < \beta$, because $\beta = \infty$. (If $L(s) = \beta = \infty$, then $U(s) = \infty$, contradicting Condition 1.)
3. $\alpha < U(s)$, because $\alpha = -\infty$. (If $U(s) = \alpha = -\infty$, then $L(s) = -\infty$, contradicting Condition 1.)

We now consider the recursive step when a child s_{i^*} is selected for traversal.

5.1. Condition 1: $L(s_{i^*}) \neq U(s_{i^*})$

Proof Proof by contradiction, for the case when s is a max state. Assume $L(s_{i^*}) = U(s_{i^*})$. Note that $L(s_{i^*}) \leq L(s)$ and $L(s) < \beta$. So, $L(s_{i^*}) < \beta$ and $U(s_{i^*}) < \beta$. Since $U(s_{i^*}) = \max_i \min(\beta, U(s_i))$ and $\beta > U(s_{i^*})$, then $U(s_{i^*}) = \max_i U(s_i) = U(s)$. In addition, $L(s) = \max_i L(s_i)$, so it follows that $L(s) = L(s_{i^*}) = U(s_{i^*}) = U(s)$. But, $L(s) \neq U(s)$, by the inductive hypothesis, so the contradiction implies $L(s_{i^*}) \neq U(s_{i^*})$. The same argument goes through with the appropriate substitutions if s is a min state. ■

5.2. Condition 2: $L(s_{i^*}) < \beta'$

Proof If s is a max state, $\beta' = \beta$. We have $L(s) < \beta$ by our induction hypothesis. We also have $L(s_{i^*}) \leq L(s)$ because $L(s)$ is computed to be the largest of the $L(s_i)$ values.

If s is a min state, $i^* = \text{argmin}_i \max(\alpha, L(s_i))$. Then, $\beta' = \min_{i \neq i*} \max(\alpha, L(s_i))$. We know $L(s_{i^*}) \leq \beta'$ because the range of values in the min that defines β' is more restrictive than the one that defines i^*. If $L(s_{i^*}) < \beta'$, the result is proved. If $L(s_{i^*}) = \beta'$, the algorithm increases β' infinitesimally, which also completes the result. ■

5.3. Condition 3: $\alpha' < U(s_{i^*})$

In this case, the arguments here are the same as for Condition 2, with the appropriate substitutions.

5.4. FSSS and Alpha-Beta

Now that we have established that FSSS makes consistent progress toward solving the tree, we show that all of the computation takes place within the parts of the tree alpha-beta has not pruned.

Theorem 2 *FSSS only visits parts of the tree that alpha-beta visits.*

Proof We proceed by showing that, for any state s visited by alpha-beta, FSSS will not traverse to a child state that alpha-beta prunes. Given the base case—that FSSS and alpha-beta both start at the root—it follows that FSSS will remain within the set of states visited by alpha-beta.

Let α and β be the values of the corresponding variables in FSSS when the current state is s. (We assume s is a max state. The reasoning for min states is analogous.) Let $\hat{\alpha}$ and $\hat{\beta}$ be the α and β values in alpha-beta at that same state. The "primed" versions of these variables denote the same values at a child state of s. Note that the following invariant holds: $\hat{\alpha} \leq \alpha$ and $\beta \geq \hat{\beta}$. That is, the α and β values in FSSS lie inside those of alpha-beta. To see why, note that this condition is true at $G.root$. Now, consider the computation of $\hat{\alpha}'$ in alpha-beta. It is the maximum of $\hat{\alpha}$ and all the values of the child states to its left.

Compare this value to that computed in Lines 15 and 16 of Algorithm 2. Here, the value α' passed to the selected child state is the maximum of α (an upper bound on $\hat{\alpha}$ by the inductive hypothesis) and the upper bounds of the values of *all* the other child states (not just those to its left). Because α' is assigned the max over a superset of values compared to in alpha-beta, α' must be an upper bound on $\hat{\alpha}'$.

The only case left to consider is what happens in Lines 17 and 18 of Algorithm 2. In this case, if the α' value has been increased and now matches the clipped upper bound of the selected child state, α' is decremented a tiny amount—one that is guaranteed to be less than the difference between any pair of leaf values. Since this value is increased (almost) up to β, it remains an upper bound on $\hat{\alpha}'$.

For max states, the values of β do not change in either algorithm. Thus, if the α and β range for FSSS lies inside the α and β range for alpha-beta at some state, it will also lie inside for all the states visited along a rollout to a leaf. Therefore, FSSS remains within the

subtree not pruned by alpha-beta. ■

Putting the two theorems together, as a rollout proceeds, FSSS always keeps an unexpanded leaf that alpha-beta expands beneath it, and FSSS at most expands the same states expanded by alpha-beta.

6. Discussion

In terms of related work, FSSS is similar to the Score Bounded Monte-Carlo Tree Search algorithm [Cazenave and Saffidine, 2010]. The major difference between it and FSSS is the lack of α and β values to guide search of the game tree, which is significant because without them does not dominate alpha-beta. We also note there are other algorithms provably better than alpha-beta not mentioned, as we are most concerned with the relationship between FSSS and alpha-beta; further comparisons would make valuable future work.

While FSSS is guaranteed to never expand more leaves than alpha-beta, the best-first approach comes at a cost. In terms of memory requirements, alpha-beta only needs to store α, β, and the path from the root to the current state being examined, which has cost logarithmic in the size of the tree. FSSS, on the other hand, must store upper and lower bounds for all states it has encountered, which is linear in the number of states visited. Fortunately, it has been shown that memory is not a limiting factor when performing game-tree search even on "large" games such as chess and checkers [Plaat et al., 1996].

Similarly, alpha-beta is also superior to FSSS in terms of computational cost. In the worst case, nearly all the states in the tree must be expanded to compute $V(G)$. For alpha-beta, processing is done in one depth-first traversal of the tree, which has cost linear in the size of the tree. FSSS, however, must conduct a number of trajectories through the tree, visiting each leaf once. In this case, for a tree of height h, alpha-beta visits 2^{h+1} states, whereas FSSS visits states a total of $h2^h$ times. The costs associated with increased memory use and full rollouts did seem to manifest themselves in practice, as alpha-beta was faster than FSSS in all our experiments.

Although we did observe FSSS to be slower than alpha-beta, this may not hold in all domains. In our experiments, $R(s)$ is quickly queried, so it is not harmful to visit many leaves. In some domains, however, the method that assigns a value to a leaf is expensive to compute. While good heuristic functions are known for evaluating board states in chess, they do not yet exist for the game of Go. As a result, when attempting to evaluate the quality of a leaf (generally not end-game positions in practice), the game is played out to completion using a (potentially randomized) policy and the process may be conducted many times to obtain a reasonable estimate of value. In such cases where evaluating the quality of a leaf is expensive, FSSS may be faster in terms of actual processing time than alpha-beta.

In order to improve computation time, FSSS can benefit from refinements which we have not included in order to simplify the algorithm. In particular, value backups can stop, and rollouts can be restarted from, the deepest state in the tree visited in the last trajectory that did not have a change in its (L, U) bounds instead of from the root of the tree. This does not alter the policy of the algorithm.

Another point worth noting is that, in practice, search algorithms are generally combined with move ordering, iterative deepening, and evaluation functions—speed up techniques that

have not been addressed here. While these additions generally improve performance, they also violate assumptions made in terms of guarantees of state expansions between algorithms. For example, when incorporating these methods, it was shown that alpha-beta sometimes expands fewer states than MTD($+\infty$) Plaat et al. [1996]; we believe the same to be true for FSSS.

7. Future Work

There are a number of opportunities to extend FSSS. The original formulation of FSSS incorporates domain knowledge about possible values a state can achieve in G. As long as these values are admissible, pruning would be improved while proofs of correctness as well as dominance of alpha-beta would still be maintained. Additionally, FSSS can be extended for use in minimax domains that involve stochasticity, such as backgammon. Finally, an extension of FSSS to the anytime case would be useful when the game-tree cannot be solved due to time constraints.

Finally, FSSS as presented only terminates once the upper and lower bounds at the root become equal. If we are simply interested in the optimal action, as opposed to the value at the root, execution can be terminated once the lower bound on an action is greater than the upper bound on all other actions at the root. If computation has a hard limitation (in terms of time or number of samples requested), extension of the algorithm to function in an "anytime" manner, such as what UCT performs, also warrants investigation.

References

Tristan Cazenave and Abdallah Saffidine. Score bounded monte-carlo tree search. In *Computers and Games*, pages 93–104, 2010.

Pierre-Arnuad Coquelin and Remi Munos. Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*, pages 67–74, 2007.

D.J. Edwards and T.P. Hart. The alpha-beta heuristic. In *AI Memos (1959 - 2004)*, 1961.

Sylvain Gelly and David Silver. Achieving master level play in 9 x 9 computer go. In *Association for the Advancement of Artificial Intelligence*, pages 1537–1540, 2008.

Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. In *Artificial Intelligence*, volume 6, pages 293–326, 1975.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

Judea Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM*, 25:559–564, August 1982.

Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first fixed-depth minimax algorithms. In *Artificial Intelligence*, volume 87, pages 255 – 293, 1996.

Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On the behavior of uct in synthetic search spaces. In *ICAPS 2011 Workshop, Monte-Carlo Tree Search: Theory and Applications*, 2011.

George C. Stockman. A minimax algorithm better than alpha-beta? In *Artifical Intelligence 12*, pages 179–196, 1979.

Thomas J. Walsh, Sergiu Goschin, and Michael L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Association for the Advancement of Artificial Intelligence*, 2010.